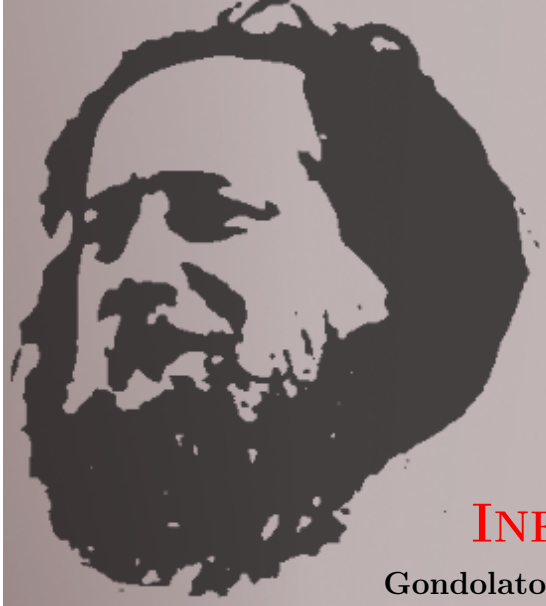


2013. Április

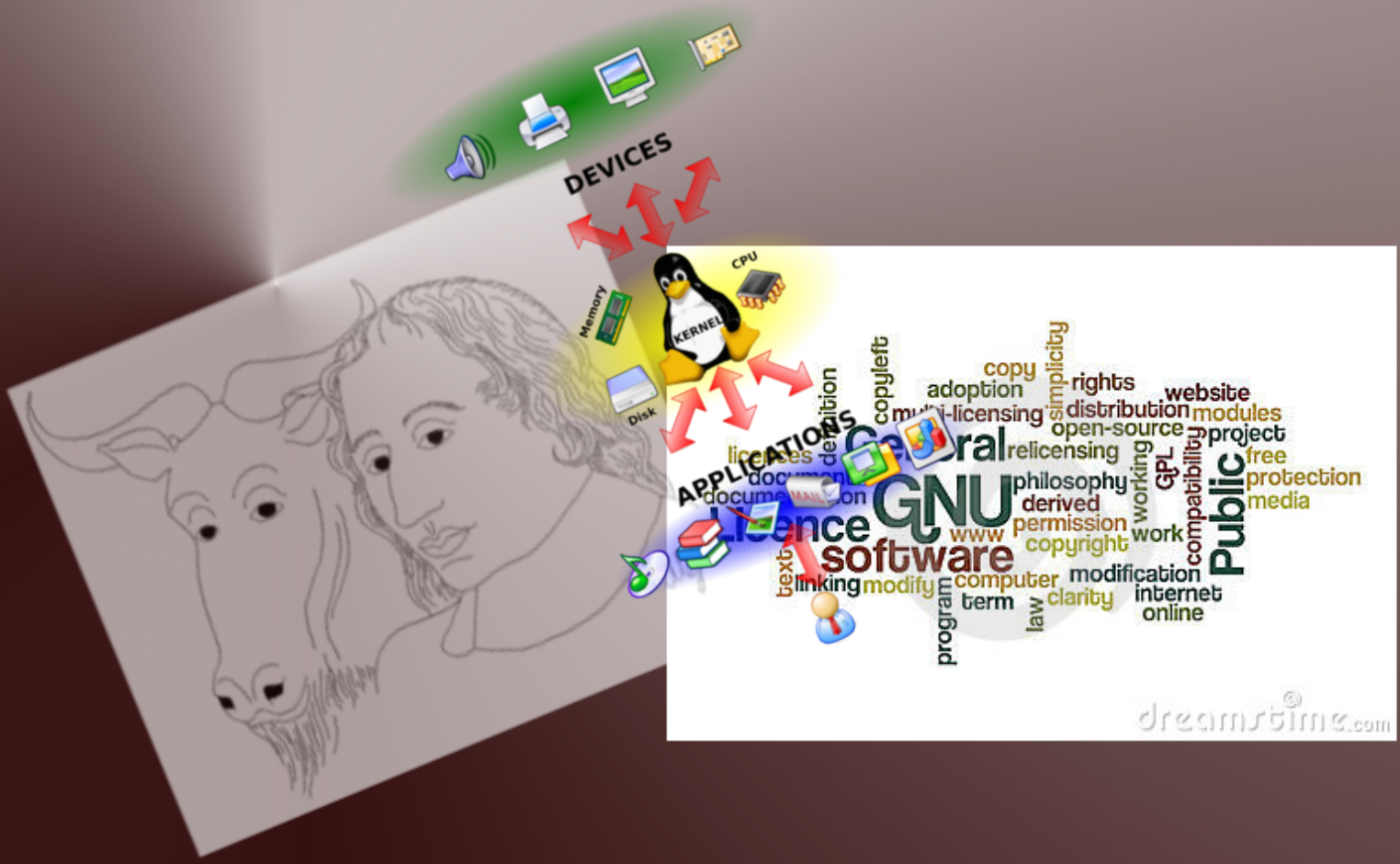
Informatikai Navigátor

Gondolatok a szoftverek használatáról és fejlesztéséről

Riportok, nyomtatványok és grafikonok készítése

CreedSoft

8. szám

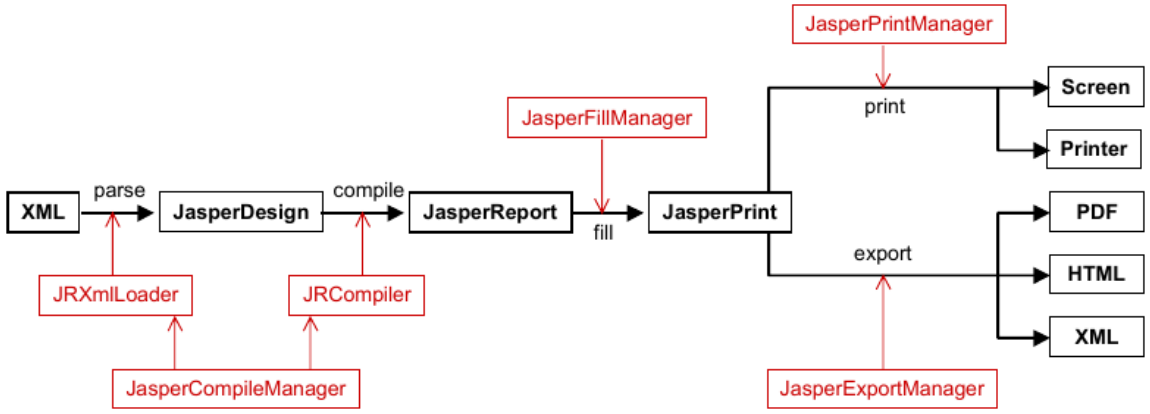


Tartalomjegyzék

Főszerkesztő: Nyiri Imre (imre.nyiri@gmail.com)

1. JasperReports - Bevezető ismeretek

A *JasperReports* egy open source (LGPL), sokoldalú és kiemelkedő minőségű riportkészítő eszköz, aminek kiemenete támogatja az ismertebb formátumokat: PDF, HTML, XLS, DOC, RTF, ODT, CSV, TXT és XML. Teljes egészében Java nyelven íródott és gyakorlatilag ennek a platformnak az első számú megoldása lett. A webhelye: `http://community.jaspersoft.com/`. Mindezt kiegészíti egy magas színvonalú tervező szoftver, az *iReport* (`http://community.jaspersoft.com/project/ireport-designer`). Aki komolyabb (Java) alkalmazásokat készít, annak mindenképpen célszerű ismernie ezt a programot, hiszen sokféle, nyomtatásra is alkalmas kimenetének szabványosított használatával kiváló eredményt érhetünk el és sok energiát takaríthatunk meg.



1.1. ábra: JasperReports - A működés folyamata (forrás: Ultimate Guide)

A JasperReports áttekintése

A képességek

Teodor Danciu 2001-ben kezdte meg ennek a kiemelkedő képességű szoftvernek a fejlesztését. A JasperReports legfontosabb jellemzői közül néhány:

- XML nyelven (JRXML[1]) leírt riport definíció, széleskörű külalak megadási lehetőségekkel.
- Szöveges és grafikus elemek használhatósága a riportokban.
- Többféle adatforrás (Datasource) megadási lehetőség: hagyományos adatbázis *SQL select*, *XML*, *POJO* (JavaBean), Java *MAP*.
- Vízjelet (*watermark*) helyezhetünk el.
- A riportok tartalmazhatnak alriportokat is, ezzel jelentősen szélesedik azon feladatok száma, amiket ezzel az eszközzel könnyebben meg tudunk oldani.
- Minden ismertebb dokumentumformátumot támogat.

A JasperReports több ismert Java library-t használ, így a feladatok jelentős részét ezekre delegálja tovább:

[1] JRXML=JasperReports XML

- *iText*: A PDF (és RTF, HTML, XML) dokumentumok előállítását és manipulálását támogató könyvtár. Többek között a digitális aláírási képességei is közkedveltek. Webhelye: `http://itextpdf.com/`.

- *JFreeChart*: Sokféle diagram és grafikon előállítási lehetősége elérhető a JasperReports riportokba ágyazva. Webhelye: `http://www.jfree.org/jfreechart/`.

- *Jakarta POI*: A különféle Microsoft dokumentumok (XLS, DOC, PPT) előállítását és manipulálását lehet vele megvalósítani. Az Apache Foundation keretében működő projekt webhelye: `http://poi.apache.org/`.

- *JAXP*: Szabványos Java könyvtár XML dokumentumok feldolgozására és előállítására.

- *Jakarta Commons*: Egy Java könyvtár gyűjtemény, ahol szinte minden újrahasznosításra alkalmas, általános célú rutin megtalálható, tudását itt nézhetjük meg: `http://commons.apache.org/`.

Hogyan működik?

Az 1.1. ábra mutatja azt a folyamatot, amit a riportok fejlesztése igényel. A fekete dobozok a logikai lépést, míg a pirosak az őket megvalósító Java osztályokat jelölik. Az első lépés (*XML doboz*) a riport definíció elkészítése, amire a *JRXML* nyelv szolgál. Itt a következő fontosabb dolgokat adhatjuk meg:

- A riport kinézete és tartalma.

- A dinamikus adatforrás, hiszen a riportok ilyet szinte mindig használnak. Itt jegyezzük meg, hogy az esetek nagy részében nem itt szokás ezt megadni, helyette a *JRDataSource* interfésszel rendelkező különféle objektumok használata lett közkedvelt.

- A riport szerkezeti tagolása, csoportosításai.

Az 1-1. Programlista egy nagyon egyszerű, Hello World típusú JRXML leírást mutat, ahol a riportban csak a „Helló Világ!” szöveg jelenik meg és nincs dinamikus adatforrás. A JRXML forrást a JasperReports eszközei ismerik (a gyakorlatban majd az iReport lesz legtöbbet használva), így azt módosítani, javítani lehet. Ezt nevezzük a riport tervezési (*JasperDesign doboz*) fázisának, aminek az eredménye a módosított *jrxml* fájl lesz. Miután elégedettek lettünk a riport szerkezetével és kinézetével, azt lefordíthatjuk Java bináris formára (a *JRCompiler* class segítségével). Ennek az eredménye egy *.jasper* kiterjesztésű fájl lesz, amit az ábra *JasperReport doboza* szemléltet. Ez a bináris forma az, amit adatokkal feltöltve a riport készítéséhez felhasználhatunk. A riport elkészítése a *fill* művelet, hatására egy *.jrprint* fájl generálódik le, ez maga az elkészített riport, natív JasperReports formátumban. Ezen fájlok képernyőn vagy nyomtatón való megjelenésére a környezet rendelkezik eszközökkel, de a gyakorlatban az *export* művelettel előállítható és hordozható formátumok előállítása vált népszerűvé.

A Hello World riport

Miután letöltöttük a *JasperReports* és *iReport* eszközöket, majd kicsomagoltuk őket egy számunkra megfelelő könyvtárba (a cikk írásakor ez mindkét csomag esetén az 5.0.0 verzió volt), készítsük el a Hello World riportot!

A riport megtervezése

A tervezés a számunkra megfelelő JRXML fájl elkészítését jelenti, amit az 1-1. Programlista

mutat. Minden riport a *jasperReport* gyökér elemmel kezdődik és érdemes betartani azt a névkonvenciót, hogy a *name* attribútum is a riport fájl neve legyen (esetünkben ez most HelloWorld). A riportok szekciókból állnak, mi ebből most csak a *detail* nevűt használtuk, amibe egy formázandó sávot (*band*) tettünk be, ezen is csak egyetlen *staticText* elemmel. Ennek leírása 2 részből áll:

- *reportElement* → Mindegyik elemi riport darabka (komponens) esetén annak síkbeli méreteit és elhelyezkedését adja meg.
- *text* → A *staticText* komponens szövege, amit mindig érdemes *CDATA*-val levédeni, nehogy a behelyezett szöveg valahol megtörje a jrxml XML struktúrát.

```
1  <!-- 1-1. Programlista: HelloWorld.jrxml -->
2
3  <?xml version="1.0"?>
4  <!DOCTYPE jasperReport
5    PUBLIC "-//JasperReports//DTD Report Design//EN"
6    "http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
7  <jasperReport name="HelloWorld">
8   <detail>
9      <band height="20">
10       <staticText>
11         <reportElement x="20" y="0" width="200" height="20"/>
12         <text><![CDATA[ Helló Világ! ]]></text>
13       </staticText>
14     </band>
15   </detail>
16 </jasperReport>
```

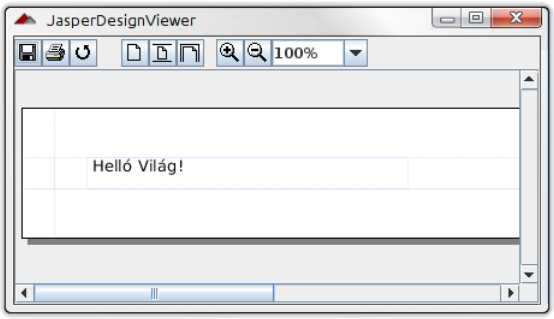
A JasperReports eszközök használata

A hatékony munka támogatására a JasperReports több támogató eszközt is felkínál. Ezek könnyű elérésére érdemes egy ANT *build.xml* fájlt készíteni (1-2. Programlista) és azon keresztül indítani őket. Ez alól az *iReport* kivétel, mert az egy komplex RAD eszköz, amit külön tölthetünk le és saját indító paranccsal rendelkezik.

```
1  <!-- 1-2. Programlista: ant build.xml -->
2
3  <?xml version="1.0"?>
4  <project name="Hello, World" default="viewDesignXML" basedir=".">
5    <description>Previews and compiles our First Report</description>
6    <property name="file.name" value="HelloWorld"/>
7    <!-- Ez a JasperReport könyvtár -->
8    <property name="jasper.dir" value="/home/java/JasperReports-5/iReport-5.0.0/➥
       ireport/modules"/>
9    <property name="my.dir" value="/home/tanulas/jasperreport/pelda-1"/>
10   <property name="lib.dir" value="${jasper.dir}/ext"/>
11   <property name="classes.dir" value="${my.dir}/classes"/>
12
13   <!-- A classpath helyei -->
```

```
14  <path id="classpath">
15    <pathelement location="./"/>
16    <pathelement location="${classes.dir}"/>
17    <fileset dir="${lib.dir}">
18      <include name="**/*.jar"/>
19    </fileset>
20  </path>
21
22  <target name="viewDesignXML" description="A Design Viewer használata megnézni a➥
      tervezett XML report designt">
23    <java classname="net.sf.jasperreports.view.JasperDesignViewer" fork="true">
24      <arg value="-XML"/>
25      <arg value="-F${file.name}.jrxml"/>
26      <classpath refid="classpath"/>
27    </java>
28  </target>
29
30  <target name="viewDesign" description="A Design Viewer hasznnálata megnézni a➥
      lefordított riportot">
31    <java classname="net.sf.jasperreports.view.JasperDesignViewer" fork="true">
32      <arg value="-F${file.name}.jasper"/>
33      <classpath refid="classpath"/>
34    </java>
35  </target>
36
37  <target name="compile" description="A jrxml fájl fordítása jasper bináris➥
      fájlra">
38    <taskdef name="jrc" classname="net.sf.jasperreports.ant.JRAntCompileTask">
39      <classpath refid="classpath"/>
40    </taskdef>
41    <jrc destdir=".">
42      <src>
43        <fileset dir=".">
44          <include name="**/*.jrxml"/>
45        </fileset>
46      </src>
47      <classpath refid="classpath"/>
48    </jrc>
49  </target>
50
51  <target name="view" description="Elindítja a Report Viewert megnézni a jrprint➥
      fájlt">
52    <java classname="net.sf.jasperreports.view.JasperViewer" fork="true">
53      <arg value="-F${file.name}.jrprint"/>
54      <classpath refid="classpath"/>
55    </java>
56  </target>
57
58  <target name="clean" description="Minden generált fájlt letöröl">
59    <delete>
60      <fileset dir="${my.dir}">
61        <include name="**/*.jasper"/>
```

```
62        </fileset>
63        <fileset dir="${my.dir}">
64          <include name="**/*.jrprint"/>
65        </fileset>
66      </delete>
67    </target>
68
69    <target name="writeApi">
70      <mkdir dir="${my.dir}/build/reports"/>
71      <taskdef name="jraw" classname="net.sf.jasperreports.ant.JRAntApiWriteTask">
72        <classpath refid="classpath"/>
73      </taskdef>
74      <jraw destdir="${my.dir}/build/reports">
75        <src>
76          <fileset dir="${my.dir}">
77            <include name="**/*.jasper"/>
78            <include name="**/*.jrxml"/>
79          </fileset>
80        </src>
81        <classpath refid="classpath"/>
82      </jraw>
83    </target>
84  </project>
```



1.2. ábra: A Jasper Design Viewer alkalmazás

A fenti *build.xml* a következő *target*-eket, azaz parancsokat tartalmazza, amit a *ant -projecthelp* paranccsal lehet kiíratni:

- *viewDesignXML*: A Design Viewer használata megnézni a tervezett XML report designt
- *viewDesign*: A Design Viewer hasznnálata megnézni a lefordított riportot
- *compile*: A *jrxml* fájl fordítása *jasper* bináris fájlra
- *view*: Elindítja a Report Viewert megnézni a *jrprint* fájlt
- *clean*: Minden generált fájlt letöröl
- *writeApi*

Az ant script első 20 sora a meghívható parancsok (target-ek) működési környezetének beállításáról szól, ami lényegében azt jelenti most, hogy a Java CLASSPATH helyesen legyen beállítva, azaz ismert legyen az összes jar fájl. A default target a *viewDesignXML*, ahogy azt a 4. sor *project* tag-jénél látjuk is. Ez azt jelenti, hogy az

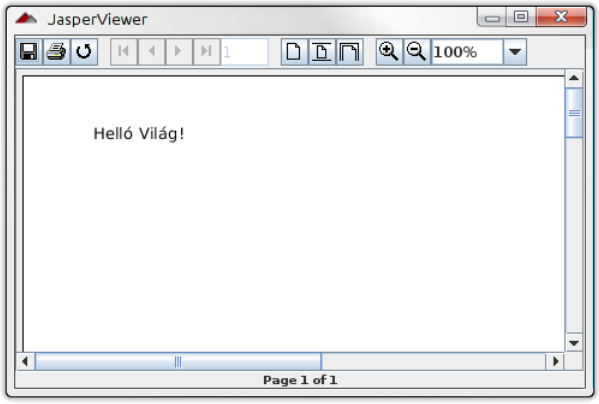
```
ant viewDesignXML
```

hajtódik végre egy egyszerű *ant* parancsra, aminek eredményeképpen az 1.2. ábra ablaka jelenik meg. Ez annyit tud, hogy a mindenkori design-t, azaz a JRXML vizuális kinézetet tekinthetjük meg vele. Esetünkben a *file.name* script változó értéke *HelloWorld*, ugyanis a megtekintendő fájl neve: *HelloWorld.jrxml*. Itt a -

XML argumentumnak pont az a szerepe, hogy tényleg a forrás tervet nézzük meg. Az *ant compile* parancs a már említett *jrxml* (XML szöveg) → *jasper* (lefordított, bináris forma) fordítás végzi el a *HelloWorld* riportra, megkapva ezzel a *HelloWorld.jasper* fájlt. Az

```
ant viewDesign
```

hívás már ezzel a bináris, natív formával képes megmutatni a leendő jelentés kinézetét, amihez szintén a *Jasper Design Viewer* alkalmazást használja.



1.3. ábra: A Jasper Viewer alkalmazás

Egy riport elkészítése után a *HelloWorld.jrprint* fájlt kapjuk, ami JasperReports formátumban tartalmazza a létrehozott jelentést. Az 1.3. ábra mutatja a Jasper Viewer alkalmazást, amit be lehet építeni a programunkba, mert képes a képernyőn megjeleníteni vagy akár kinyomtatni a keletkezett eredményt. Ez a *JasperViewer* osztály parancssorból is elindítható:

```
ant view
```

A *build.xml clean* opciója a generált fájlokat törli le. A *writeApi* a *JRAntApiWriteTask* ant taszk használatát mutatja meg. Amennyiben lefuttatjuk, úgy esetünkben most a *build/reports* könyvtárba nem a jasper fájlt generálja le, hanem helyette egy *jrxml→java* fordítást végez, azaz megtekinthetjük a riport definíció java forráskódját. Ez elsősorban hibakereséshez jelenthet egy hatékony eszközt, de a gyakorlatban erre nagyon ritkán kerül sor, amennyiben mi egy egyszerű riportkészítő programozók vagyunk. Érdekességként az 1-3. Programlista tartalmazza az így létrehozott java forráskódot, amire mindenképpen érdemes egy pillantást vetni.

```
// 1-3. Programlista: HelloWorld.java

/*
 * Generated by JasperReports - 2012.12.21. 10:59
 */
import java.awt.Color;

import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.xy.XYBubbleRenderer;

import net.sf.jasperreports.charts.*;
import net.sf.jasperreports.charts.design.*;
import net.sf.jasperreports.charts.util.*;
import net.sf.jasperreports.crosstabs.*;
import net.sf.jasperreports.crosstabs.design.*;
import net.sf.jasperreports.crosstabs.fill.calculation.BucketDefinition;
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.base.JRBaseChartPlot.JRBaseSeriesColor;
import net.sf.jasperreports.engine.base.JRBaseFont;
import net.sf.jasperreports.engine.design.*;
import net.sf.jasperreports.engine.type.*;
import net.sf.jasperreports.engine.util.ReportCreator;


public class HelloWorld implements ReportCreator
{
```

```
  public JasperDesign create() throws JRException
  {
    JasperDesign jasperDesign = new JasperDesign();
    jasperDesign.setName("FirstReport");
    jasperDesign.setLanguage("java");
    jasperDesign.setPageWidth( 595);
    jasperDesign.setPageHeight( 842);
    jasperDesign.setColumnWidth( 555);
    jasperDesign.setColumnSpacing( 0);
    jasperDesign.setLeftMargin( 20);
    jasperDesign.setRightMargin( 20);
    jasperDesign.setTopMargin( 30);
    jasperDesign.setBottomMargin( 30);

    JRDesignDataset reportMainDataset = new JRDesignDataset(true);
    reportMainDataset.setName("FirstReport");
    jasperDesign.setMainDataset(reportMainDataset);

    //detailBand
    //band name = detailBand0
    JRDesignBand detailBand0 = new JRDesignBand();
    detailBand0.setHeight( 20);
    detailBand0.setSplitType(net.sf.jasperreports.engine.type.SplitTypeEnum.STRETCH);
    JRDesignStaticText detailBand0_0 = new JRDesignStaticText(jasperDesign);
    detailBand0_0.setPositionType(net.sf.jasperreports.engine.type.PositionTypeEnum.➥
        FIX_RELATIVE_TO_TOP);
    detailBand0_0.setX( 20);
    detailBand0_0.setY( 0);
    detailBand0_0.setWidth( 200);
    detailBand0_0.setHeight( 20);
    JRParagraph detailBand0_0Paragraph = detailBand0_0.getParagraph();
    detailBand0_0.setText("␣Helló␣Világ!␣");
    detailBand0.addElement(detailBand0_0);

    ((JRDesignSection)jasperDesign.getDetailSection()).getBandsList().add(0, detailBand0);

    return jasperDesign;
  }
}
```

Általában persze nem parancssori eszközzel dolgozunk, hanem az *iReport* tervezővel, de mégis érdemes az ismertetett ant script lehetőségeivel tisztában lennünk, mert egyrészt jobb áttekintést kapunk az eszközrendszerről, másrészt bizonyos feladatokat (tipikusan a *jrxml→jasper* fordítást) hatékonyabban lehet vele elvégezni. Ugyanakkor a *Jasper Viewer* egy olyan kész panel, amit az alkalmazásainkba is beépíthetünk.

Mielőtt továbblépnénk, szeretnénk pár szóban kitérni a *build.xml*-ben használt alábbi minták jelentésére, ugyanis ezek az ismeretek fontosak és ugyanakkor sokszor nem eléggé ismertek:

- *single star (*)*: Illeszkedik 0 vagy több karakterre a PATH neven belül, de annak csak egy adott szintjét tekintve.
- *double star (**)*: Illeszkedik 0 vagy több karakterre a PATH neven belül és a teljes directory útvonalat nézi,
- *question mark (?)*: Pontosan 1 tetszőleges karakternyi illeszkedés a PATH neven belül

A jobb megértés érdekében tekintsük a következő könyvtár és fájl elrendezést:

1. *bar.txt* (a . könyvtárban)
2. *src/bar.c* (a ./src könyvtárban)
3. *src/baz.c* (a ./src könyvtárban)
4. *src/test/bartest.c* (a ./src/test könyvtárban)

Ekkor a **.c* kifejezés nem illeszkedik semmire, mert az aktuális könyvtárban nincs ilyen, hiszen ott csak egy *bar.txt* van. Az *src/*.c* illeszkedik a 2. és 3. fájlra is. A **/*.c* hasonlóképpen a 2. és 3. fájlokat választja ki, ugyanis a * csak 1 könyvtár szinten való illeszkedést jelent, a 4. sor pedig 2 könyvtárat is tartalmaz a *bartest.c* előtt. Amennyiben mégis az összes **.c* fájlt szeretnénk, úgy a ***/*.c* illeszkedik a 2, 3. és 4. fájlra is, mert a ** több szintű könyvtárat jelent. A *bar.** természetesen csak az 1. fájlra illeszkedik, de a ***/bar.** már az 1. és 2. fájlokra is. Nézzünk még 2 példát! A ***/bar*.** az 1, 2 és 4. állományokra, míg az *src/ba?.c* csak a 2. és 3. fájlokra illeszkedik.

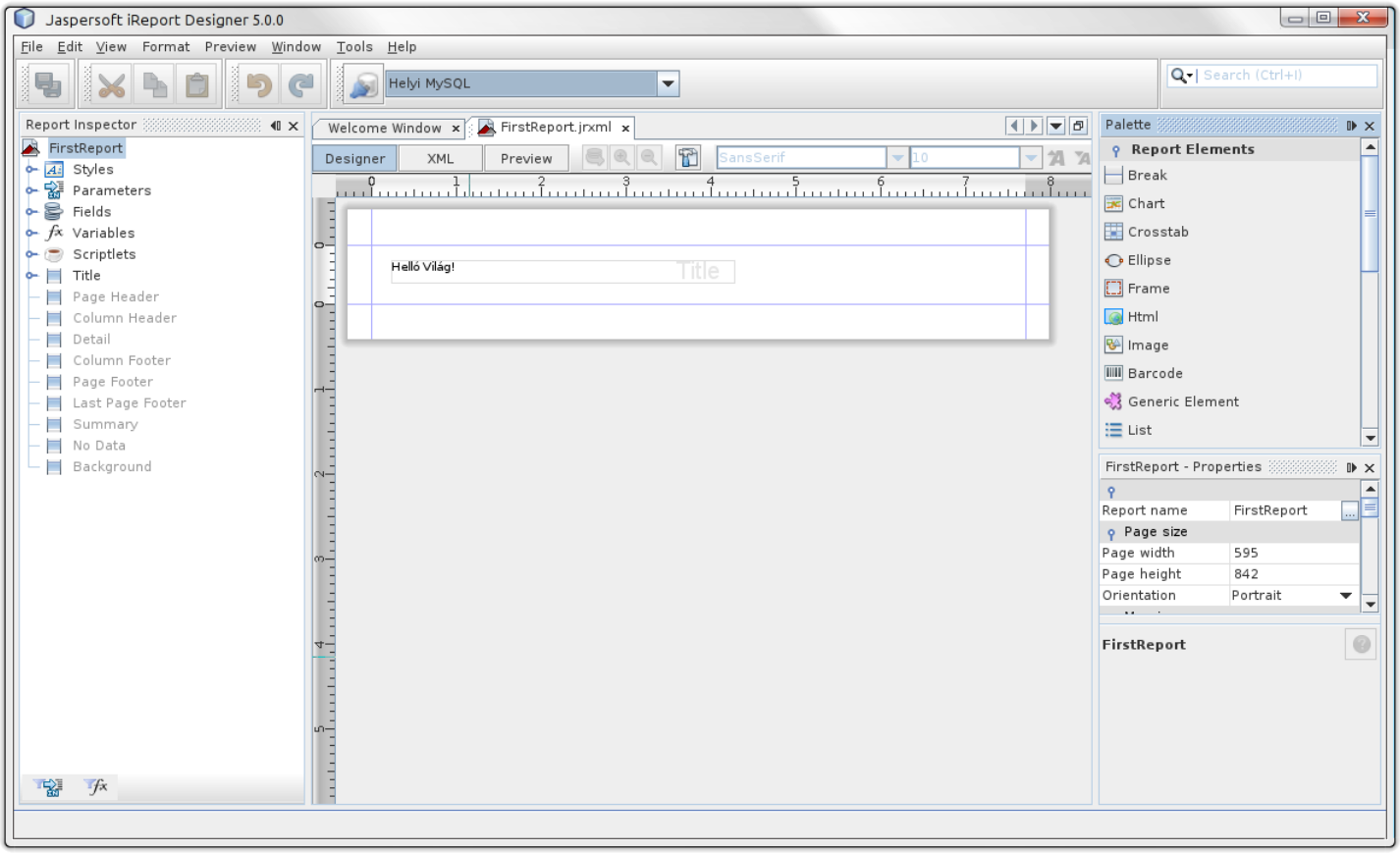
A riport előállítása és megtekintése

Az eddigiek során elkészítettük a *HelloWorld.jrxml* riport tervet és a megfelelő *ant* paranccsal lefordítottuk azt, így jelenleg rendelkezünk a *HelloWorld.jasper* fájllal is. Azt tudjuk, hogy a riportunk jelenleg semmilyen külső adatot nem igényel, egyedül csak egy statikus szöveget jelenít meg. Az 1-4. Programlista egy olyan Java programot mutat, amivel elkészíthetjük a jelentésünket, azaz előállíthatjuk a *HelloWorld.jrprint* állományt. Tekintsünk a forráskódra! A *JRHelper* class 23-26 sora közötti *makeReport()* metódus készíti el a riportot. Ez gyakorlatilag a *JasperFillManager* osztály segítségével történik. Az 1. paraméter a *.jasper* fájl neve, a 2. a külső paraméterek átvételét biztosító *java.util.Map* objektum, míg a 3. egy külső adatforrást reprezentáló *JRDataSource* objektum. Tekintettel arra, hogy most se paraméterünk nincs, se külső adatforrásunk, ezért üres *Map*-pel és a beépített *JREmptyDataSource* objektummal hívjuk meg a *main()* 40. sorában. A *JRHelper compile()* metódusát általában nem programozottan használjuk, de ugyanazt csinálja, mint az ismertetett *ant compile* parancs. Ennek az az oka, hogy egy jól letesztelt riportnak általában nem a külalakja változik, hanem a feldolgozott külső adatok, így azt hatékonysági okokból sem érdemes állandóan újrafordítani.

```
1  // 1-4. Programlista: JRHelper.java
2
3  package org.cs.test;
4
5  import java.util.HashMap;
6  import net.sf.jasperreports.engine.JRDataSource;
7  import net.sf.jasperreports.engine.JREmptyDataSource;
8  import net.sf.jasperreports.engine.JRException;
9  import net.sf.jasperreports.engine.JasperCompileManager;
10 import net.sf.jasperreports.engine.JasperFillManager;
11
12 /**
13  *
14  * @author inyiri
15  */
16 public class JRHelper
17 {
18     public void compile(String jrxmlFileName) throws JRException
19     {
20         JasperCompileManager.compileReportToFile(jrxmlFileName);
21     }
22
23     public void makeReport(String jasperFileName, HashMap params, JRDataSource jrDs) throws ➥
           JRException
24     {
25         JasperFillManager.fillReportToFile(jasperFileName, params, jrDs);
26     }
27
28     public static void main(String[] args) throws JRException
29     {
```

```
30          String jrxmlFile = "/home/tanulas/jasperreport/pelda-1/HelloWorld.jrxml";
31          String jasperFile = "/home/tanulas/jasperreport/pelda-1/HelloWorld.jasper";
32
33          JRHelper jh = new JRHelper();
34
35  //          System.out.println("Compiling report...");
36  //          jh.compile(jrxmlFile);
37
38          System.out.println("Make report...");
39          JRDataSource eds = new JREmptyDataSource();
40          jh.makeReport(jasperFile, new HashMap(), eds);
41
42          System.out.println("Done!");
43      }
44  } // end class
```

A *JRHelper* program lefuttatása után keletkezik a *HelloWorld.jrprint* fájl, amit a már ismert eszközzel (1.3. ábra) lehet megnézni az *ant view* parancs segítségével. Végezetül vessünk egy pillantást az *iReport* RAD eszközre (1.4. ábra), és haladjunk tovább a következő fejezeteken, amik áttekintik a JasperReports összes fontos lehetőségét!



1.4. ábra: iReport 5

2. Adatbázis alapú dinamikus riport készítése

Ebben a részben bemutatjuk az adatbázis lekérdezésekre épülő, dinamikus riportkészítés alapjait. Kétféle technikával fogunk megismerkedni. Az első az SQL query-t a jrxml fájlba ágyazva mutatja be, mely során bemutatjuk a riportok paraméterezési lehetőségét is. A második a *JRDataSource* objektumra épülő megoldás, ahol nincs már szükség a *queryString* tag-et beágyazni a jrxml állományba. A példákhoz az Informatikai Navigátor 7. számában megismert HSQLDB adatbáziskezelőt fogjuk használni.

A használt fejlesztői környezet

Adatbáziskezelő

Tekintettel arra, hogy szükségünk lesz egy adatbáziskezelőre, indítsuk el a HSQLDB-t szerver módba, amihez részletes segítséget kapunk az Informatikai Navigátor 7. számából. A megismert *orszagok* táblára épülő *select* paranccsal fogunk dinamikus adatokat szolgáltatni a jelentésünk részére.

JRXML fejlesztés – *iReport*

A továbbiakban a riport tervezési munkát általában az *iReport* eszközzel fogjuk végezni, így azt is indítsuk el. Első lépésként frissítjük a csomaggal érkezett HSQLDB jdbc drivert arra a verzióra, amit jelenleg használunk. Ehhez az iReport *Tools→Options→Classpath* fülére menjünk. Töröljük ki a régi HSQLDB drivert és vegyük fel az általunk használt frissebb *hsqldb.jar* fájlt. Ezután kattintsunk a *Report Datasources* ikonra és vegyünk fel egy új JDBC kapcsolatot a mi adatbázisunkra. Emlékeztetőül megadjuk szerverünk URL-jét: *jdbc:hsqldb:hsql://localhost/xdb* (user: *SA*).

Egy JRXML fájl felépítése

Itt most csak azokat a jrxml elemeket nézzük át, amik a cikkhez szükségesek, a továbbiakat majd a későbbi fejezetekben részletezzük, mindig ahhoz a témához kötődve, amit éppen ismertetünk.

Vegyük észre, hogy egy természeténél fogva hierarchikus felépítésű riport leírására az XML kifejezetten alkalmas eszköz, hiszen gyakorlatilag abban a szerkezetben tudjuk a jelentés templateet vele elkészíteni, ahogy annak majd ki kell néznie.

A riport szakaszai

A következő nagyobb riport szakaszokat részletesebben az *iReport* bemutatásánál fogjuk ismertetni, itt csak megemlítjük a szerepüket.

<title>. A riport címét tartalmazó szakasz, amennyiben használjuk, úgy a riport elejére fog renderelődni.

<pageHeader>. Az itt definiált layout minden új oldal elején fog látszódni.

<columnHeader>. Az oszlopok fejlécének kinézetét tartalmazó specifikációs szakasz.

<detail>. Ez a szakasz minden egyes új adatsor beérkezésekor fog megformálódni.

<columnFooter>. Az oszlopok láblécének kinézetét tartalmazó specifikációs szakasz.

<pageFooter>. Az itt definiált layout minden oldal végén fog látszódni.

<lastPageFooter>. Ugyanaz a szerepe, mint a *pageFooter* elemnek, de azt felváltja az utolsónak generált oldalon, amennyiben ilyen szakaszt definiálunk.

<summary>. A riport legvégére definiált szakasz, ami általában összefoglaló jellegű információkat és végösszegeket tartalmaz.

Nem vizuális elemek

<field>. A field (mező) szerepe az, hogy a kívülről (adatforrásokból) érkező adatokat összerendelje (mapping) a riport elemeihez tartozó adattartalommal. A mező megadására lássunk 2 példát:

```
<field name="ORSZAG" class="java.lang.String"/>
<field name="NEPESSEG" class="java.lang.Integer"/>
```

Egy mezőnek ezek szerint van neve és típusa, azaz egy befogadó változóként viselkedik. Mélyebben megnézve ezek mögött a *Groovy* nyelv áll, amit az Informatikai Navigátor 5. számában részletesebben is bemutattunk. Egy mezőre így lehet hivatkozni:

```
$F{ORSZAG} vagy $F{NEPESSEG}
```

A *$F* azt jelenti, hogy egy mező értékét kérjük. Itt emlékeztetünk arra, hogy a Groovy *${kifejezés}* formája is dinamikusan futtatja az ott megadott kódot, azaz makróhelyettesítés szerűen működik. Ez azt jelenti, hogy egy *String* típusú változót, mint egy kóddarabot futtat le. Példa, ami a 9-et jeleníti meg a képernyőn:

```
def X="9";
A="${X}"
print A;
```

Ez a lehetőség magyarázza meg a *field* működését, ami összeköti az érkező adatsorok oszlopneveit a riport elemek value (például *text*) jellemzőivel. Hogy működik mindez? Érkezik egy új adatsor *N* darab oszloppal. A JasperReports generátor próbálja a riport *$F{...}* részeit ebből kitölteni. Például veszi a *$F{ORSZAG}* mezőt és ez alapján próbálja lekérni az adatforrás *ORSZAG* nevű oszlopát (vagy adatát). A legtöbb esetben ez az egyszerű névegyezésen alapuló illesztés és értéklekérés működik a *field* és az *datasource* között.

<parameter>. A paraméter a mezőhöz hasonlóan a külvilág és a riportgenerátor közötti adatcserét valósítja meg, de a mechanizmus más. Itt az átadandó (kulcs, érték) párokat egy *java.util.Map* objektum segítségével adjuk át. A jrxml fájlban pedig így definiálunk paramétereket:

```
<parameter name="pTerulet" class="java.lang.Double"/>
<parameter name="pheaderTerulet" class="java.lang.➥
    String"/>
```

A különféle riport elemeken belül az egyes paraméterek értékeire *$P{...}* alakban hivatkozunk, ettől eltekintve a használata hasonló a mezőkével. Példák:

```
$P{pTerulet} vagy $P{pheaderTerulet}
```

Amennyiben a generátor a riport készítése során találkozik egy *$P{paraméter-név}* hivatkozással, úgy majd tudja, hogy ez egy paraméter lesz, amit a kapott map objektumból tud megszerezni úgy, hogy annak kulcsa a *paraméter-név* lesz.

<property>. A jrxml fájlba bármilyen tetszőleges (kulcs, érték) párt is elhelyezhetünk, azaz ezt nem kívülről kapjuk ebben az esetben. Példa:

```
<property name="ireport.zoom" value="1.0"/>
```

A property értékéhez a *JasperReport* osztály statikus *getProperty()* metódusával férhetünk hozzá.

<variable>. A változók segítségével egyszerűbbé tehetjük a riport tervét, illetve segítségével gyűjthetünk különféle értékeket, amiket a jelentés alkalmas helyein megjeleníthetünk.

```
<variable name="QuantitySum"
    class="java.lang.Double" calculation="Sum">
  <variableExpression>$F{Quantity}</variableExpression➥
      >
</variable>
```

A fenti példa változójának értékére a *$V{QuantitySum}* formával hivatkozhatunk, amikor azt egy olyan helyre tesszük a jrxml-ben, ahol arra szükség van. A változó kifejezés megadásánál hivatkozhatunk más, értékkel bíró riport elemekre, így akár más változókra is. Fontos fogalom a változók *reinicializálása,* azaz mikor kapnak újra kezdőértéket. Az alapértelmezett a *report* szint, ami azt jelenti, hogy egy alkalommal, a riportgenerálás kezdetekor kapnak csak kezdőértéket és kiszámításuk a jelentés végének elérésekor lesz komplett. Természetesen választhatunk alacsonyabb szintet is a reinicializáláshoz, nézzük meg ezeket:

- *page* → a változó minden lapkezdéskor nullázódik.
- *column* → minden új oszlopnál reset történik.
- *group* → a megadott csoport váltásakor nullázódik, így itt gyűjthető a csoportra jellemző aggregált mennyiség.
- *none* → soha nincs reset.

A következő példában a *page* szint megadását láthatjuk, ami egy új lapra ugráskor nullázódik és a végén lesz komplett a kiszámítása.

```
<variable name="QuantitySum" class="java.lang.Double"
          resetType="Page" calculation="Sum">
  <variableExpression>$F{Quantity}</variableExpression➥
      >
  <initialValueExpression>new Double(0) </➥
      initialValueExpression>
</variable>
```

A *calculation* attribútum azt határozza meg, hogy az aggregálás mi legyen:

- *Count* → Ennyiszer történt kalkuláció, azaz a nem null értékek száma.
- *Distinct Count* → Ugyanaz, mint az előző, de a csak a különböző értékeket veszi bele az összegbe.
- *Sum* → A *variableExpression* kifejezésekből kapott értékeket itt összeadja a változóba.
- *Average* → A számot adó kifejezés értékek számtani átlagát számolja.
- *Lowest* → A legkisebb érték.
- *Highest* → A legnagyobb érték.
- *Standard Deviation* → A standard szórás.
- *Variance* → A szórásnégyzet.
- *System* → Saját készítésű algoritmus.
- *First* → Az első kiszámított érték.

A változók működésének további lényeges jellemzője az *Increment Type*, azaz itt – ellentétben a reset-tel – azt határozzuk meg, hogy mihez van kötve a halmozás:

- *Report* → A változó csak 1 alkalommal halmozódik, amikor a riport végéhez ér a generátor.
- *Page* → Az inkrementálás minden lap befejezéskor történik.
- *Column* → Az inkrementálás minden oszlop befejezéskor történik.
- *Group* → A csoportok kezelését a külön cikkben ismertetjük.
- *None* → Minden egyes új rekorddal halmozódik a változó.

<queryString>. Amennyiben a jrxml-be szeretnénk beágyazni azt az SQL selectet, ami az adatsorokat hozza, úgy így tehetjük meg:

```
<queryString>
        <![CDATA[select * from orszagok]]>
</queryString>
```

Természetesen a *field* nevek ilyenkor az SQL select eredményhalmaz oszlopnevei lesznek.

<import>. A jrxml fájlban gyakran hivatkozunk java osztályokra, így ezeket be kell importálni. Például, ha egy *Date* objektumot szeretnénk használni egy *field* értékeként, akkor erre szükség lesz:

```
<import value="java.util.Date" />
```

Ekkor ennek használata egy *textField* vizuális jrxml komponensen belül:

```
<textFieldExpression class="java.util.Date"><![CDATA[
    new Date()]]></textFieldExpression>
```

<group>. Az adatsorok többszintű csoportosításának (összegfokozatos lista) definiálását teszi lehetővé. Használatának bemutatását a későbbiekben láthatjuk majd.

Vizuális JRXML címkék

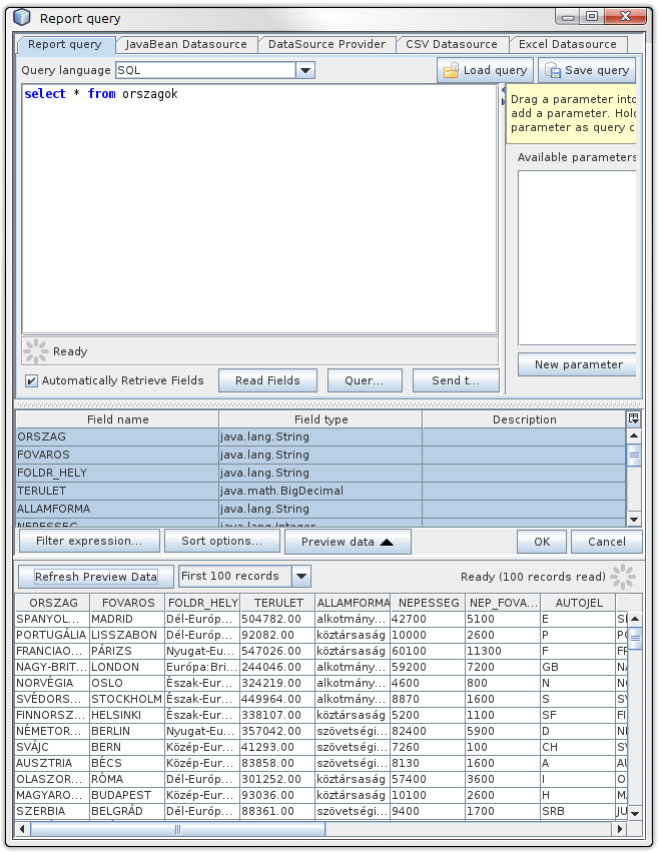
A következő fejezetekben még sok jrxml elemet fogunk bemutatni, ami a szövegeket, képeket, grafikonokat és egyéb lehetőségeket valósítják meg. Itt most ezekből csak az ebben a cikkben használt 2 komponenst mutatjuk be, de azt is csak érintőlegesen, hiszen a beállítási lehetőségeiket (stílusok, méretek, stb.) úgyis az *iReport* segítségével érdemes manipulálni. Szeretnénk kiemelni a *reportElement* címkét, ami a vizuális komponensek megadásánál mindig előfordul és annak geometriai kiterjedését és pozícióját adja meg.

<staticText>. Egy riportba épített szöveges elem, amit utólag már nem lehet változtatni.

<textField>. Egy riportba épített szöveges elem, amit utólag is lehet változtatni, így az előzőnél sokkal rugalmasabb a használata.

A példa riport megtervezése

A feladat az, hogy a HSQLDB adatbázisban található *orszagok* táblára építve készítsünk egy egyszerű jelentést. Itt tehát az adatainkat már egy valódi adatbázisból fogjuk nyerni.



2.1. ábra: A riport query vizuális megtervezése

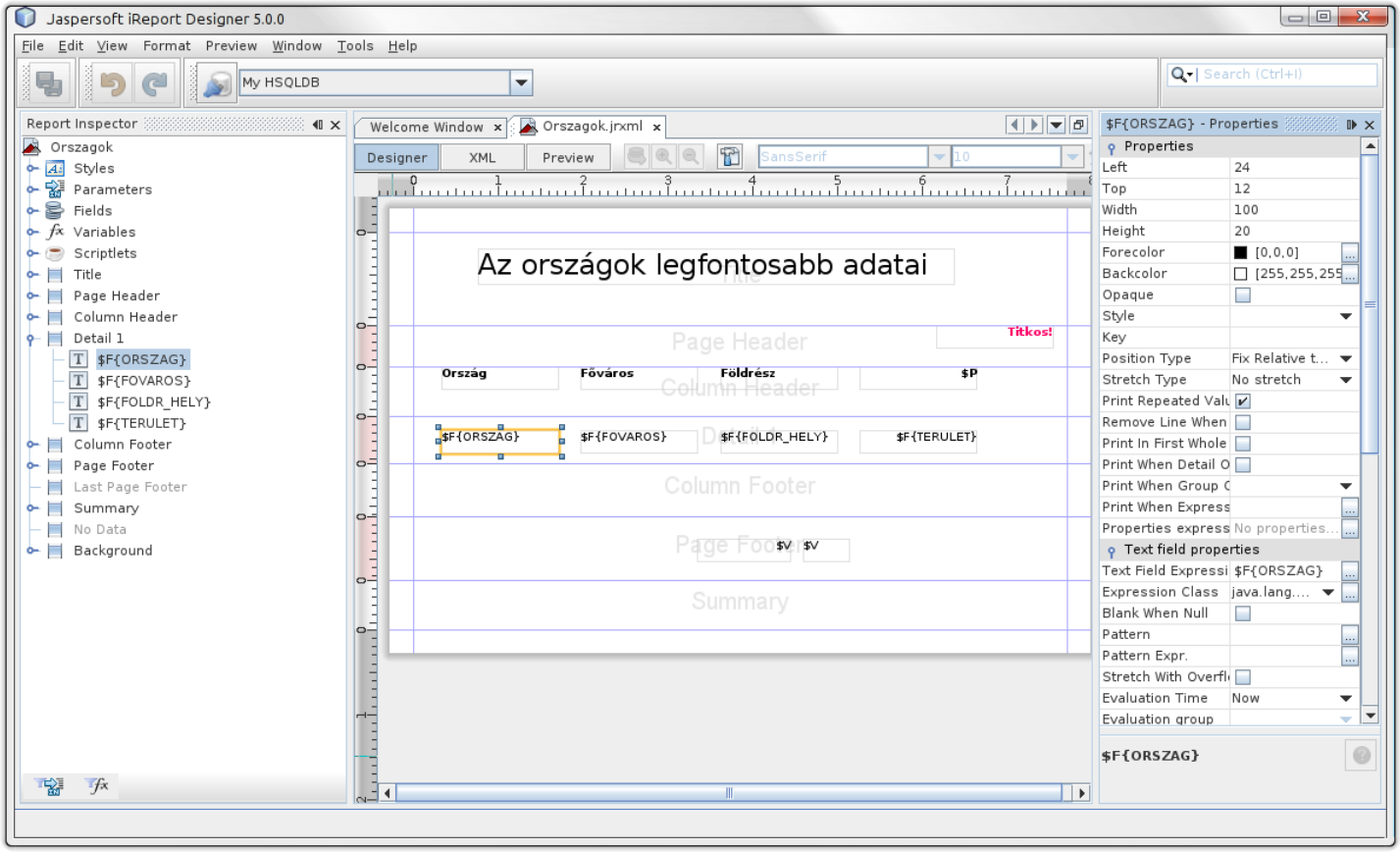
Hozzunk létre az iReport segítségével egy új riportot és legyen a neve: *Orszagok.jrxml*. A 2.1. ábra az iReport Report Query eszköze, ahol kiválasztottuk a korábban már létrehozott HSQLDB Connection-t és megadtuk a

```
select * from orszagok
```

SQL lekérdezést. A képen látható minden más részlet ezután már automatikusan állítódott össze. Alul láthatjuk a select várható eredményét, középen pedig az adatbázis oszlopait, amikből ezek a field sorok automatikusan bekerülnek a jrxml fájlba:

```
<field name="ORSZAG" class="java.lang.String"/>
```

Nyomjuk meg az *OK* gombot, amivel visszatérünk az tervező felületre, amit a 2.2. ábra mutat.

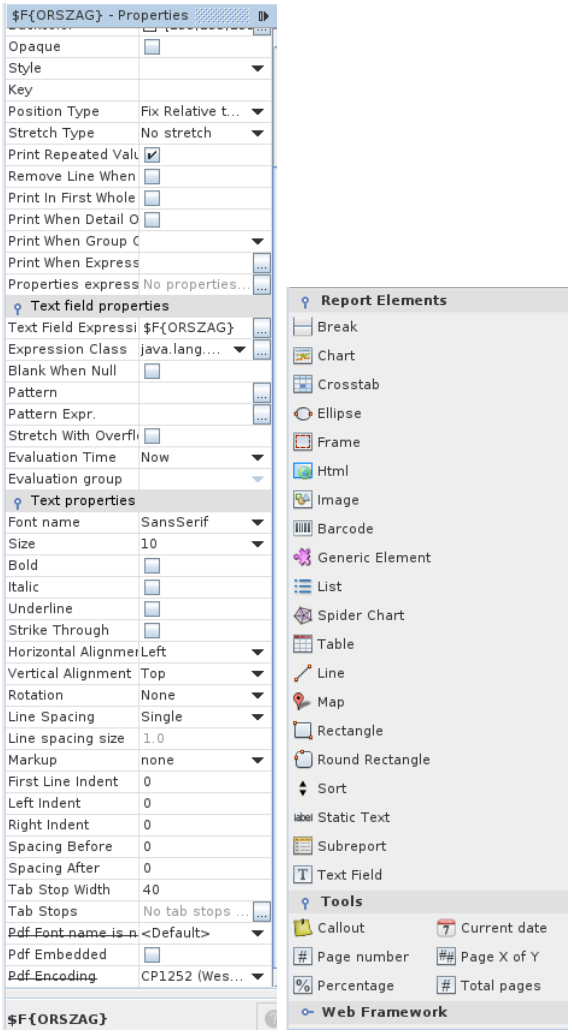


2.2. ábra: iReport - Orszagok.jrxml tervezése

Nézzük át az iReport azon ablakait, amikre ebben a példában szükségünk lesz! A baloldali *Report inspector* feladata, hogy a JRXML fájl szerkezetét, az abban lévő komponenseket mutassa, ott navigálhassunk. Látjuk a riportok főbb szakaszait. Amennyiben kinyitnánk a *Fields* ágat, látnánk azokat az automatikusan generált field elemeket, amik a select megadásakor jöttek létre. Mindezt kézzel is megírhatnánk, de az iReport igazán remek eszköz, hogy ebben a maximális támogatást adja nekünk. Az ábra közepe a tervező vásznat mutatja, ami esetünkben nem túl összetett kiépítésű, az ott látottakat már mi tettük fel oda. Nézzük meg sorjában őket!

- A *Title* részbe csak egy konstans *staticText* szöveget tettünk: *Az országok legfontosabb adatai*. Ez – ahogy már említettük – a riport elején, 1 alkalommal fog megjelenni. A működését sokféleképpen szabályozhatjuk, például mondhatnánk, hogy ez a szekció egy külön, első oldalon jelenjen meg.
- A *Page Header* is rendkívül egyszerű, hatására a piros *Titkos!* szöveget minden oldalon látni fogjuk a lap tetején.
- A lehetséges mezőkből most csak 4 darabot helyeztünk el a *Detail* sávba, a nekik megfelelő oszlopneveket pedig a *Column Header* részbe. A detail az a rész, ahol a beérkező adatsorok egymás alatt jelennek majd meg, persze lapváltás szükséges lesz, hiszen, ahogy látjuk majd a jelentésünk 14 oldalas lesz.
- Utoljára még a *Page Footer* területre tet-

tük az oldalszámozást, aktuális oldal/-összes oldal alakra formázva. Mindkét értéket az előre definiált *PAGE_NUMBER* változó adja, de az eltérő kiértékelési időpont miatt jól működik az értékének a használata.



2.3. ábra: Riport tervezési elemek

Vegyük észre, hogy a tervező nézet a következő 3 módon tudja megmutatni a JRXML fájlt:

1. *Designer*: Itt történik a vizuális tervezés, amihez a 2.3. ábra eszközeit is használhatjuk. Az ábrán éppen az *ORSZAG* mező van fókuszban, így annak beállítható jellemzőinek egy része látható. A jobb oldali ablakrészlet pedig a komponens palettát mutatja, azaz drag'n'drop módszerrel innen is feltehetünk elemeket a riportra.

2. *XML*: Ez maga a JRXML fájl, amit 2-útas módon (azaz utána látszik a vizuális felületen is) itt is szerkeszthetünk, tartalmát a 2-1. Programlistán tanulmányozhatjuk.

3. *Preview*: Itt tekinthetjük meg, hogy élőben milyen lesz a kinézete a jelentésünknek. A funkciója azonos a más ismert *ant viewDesignXML* paranccsal.

Ennyi előzmény után láthattuk, hogy a kézzel is elkészíthető *Orszagok.jrxml* fájlt milyen módon állítottuk elő pár perc alatt az iReport segítségével. Most nézzük meg egy kicsit alaposabban is a 2-1. Programlista tartalmát! Az első fontos rész a 8-10 sorok között látható, ahol megérthetjük a riportba ágyazott select, azaz a *queryString* tag megadási formátumát. A 11-26 sorok között mezőket találunk, amik a select alapján automatikusan jöttek létre, de mi ebből a riportban csak az első 4-et használjuk, azaz drag'n'droppal ezeket helyeztük el a vászonra. A *background* címkét (27-29 sorok) nem állítottuk be, de itt adhatnánk meg például vízjelet is. A 30-40 sorok közötti *title* tag már ismerős, az ő sávja (*band*) most éppen 79 px méretű, egyetlen eleme egy *staticText*, aminek 3 jellemzője van most explicite beállítva:

1. *reportElement:* A *staticText* méreteit és pozícióját határozza meg a sávhoz képest relatív módon.

2. *textElement*: Jelenleg csak a használt betű mérete tér el a default értéktől, hiszen ezt 24 px értékre állítottuk a tervezés során.

3. *text*: Az ismert cím szövege.

A 41-83 sorokban lévő *pageHeader* és *columnHeader* felépítése a *title*-hoz hasonló, szerepüket

ismerjük. A 84-107 sorok nagyon fontosak, hiszen a külső adatforrásból érkező sorok oszlopait itt határozzuk meg, ez valójában a 85-106 sorok közötti *band* tartalmát jelenti. A sávra 4 darab *textField* komponenst tettünk, amelyek közül például az *ORSZAG* így néz ki:

1. *reportElement:* A *textField* méreteit és pozícióját határozza meg a sávhoz képest relatív módon.

2. *textFieldExpression*: A 89. sorban végre megérthetjük, hogy egy *textField* hogyan tartalmazza azt a mezőt, ami az adatbázis oszlopával való névegyezésen alapszik és értékét *$F{ORSZAG}* alakban érjük el.

A 111-124 sorokban lévő *pageFooter*-t már ismerjük, jelenleg ez a sáv csak az oldalszámozás megjelenítésére szolgál. Itt most csak a 118. sorban lévő *textField evaluationTime="Report"* attribútumára hívnánk fel a figyelmet, ugyanis erre a mezőre megadtuk, hogy akkor fusson le rá a kiértékelés, ha elkészült a teljes riport. Ez logikus is, hiszen csak akkor tudható meg az is, hogy hány oldalas lett.

```
1  <!-- 2-1. Programlista: Orszagok.jrxml -->
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.➥
       w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/➥
       jasperreports http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok" ➥
       language="groovy" pageWidth="595" pageHeight="842" columnWidth="555" leftMargin="20" ➥
       rightMargin="20" topMargin="20" bottomMargin="20" uuid="9a1dbe76-1ae5-4418-9e43-e56f6ed98193➥
       ">
5          <property name="ireport.zoom" value="1.0"/>
6          <property name="ireport.x" value="0"/>
7          <property name="ireport.y" value="0"/>
8          <queryString>
9                  <![CDATA[select * from orszagok]]>
10         </queryString>
11         <field name="ORSZAG" class="java.lang.String"/>
12         <field name="FOVAROS" class="java.lang.String"/>
13         <field name="FOLDR_HELY" class="java.lang.String"/>
14         <field name="TERULET" class="java.math.BigDecimal"/>
15         <field name="ALLAMFORMA" class="java.lang.String"/>
16         <field name="NEPESSEG" class="java.lang.Integer"/>
17         <field name="NEP_FOVAROS" class="java.lang.Integer"/>
18         <field name="AUTOJEL" class="java.lang.String"/>
19         <field name="COUNTRY" class="java.lang.String"/>
20         <field name="CAPITAL" class="java.lang.String"/>
21         <field name="PENZNEM" class="java.lang.String"/>
22         <field name="PENZJEL" class="java.lang.String"/>
23         <field name="VALTOPENZ" class="java.lang.String"/>
24         <field name="TELEFON" class="java.lang.Integer"/>
25         <field name="GDP" class="java.lang.Integer"/>
26         <field name="KAT" class="java.lang.Integer"/>
27         <background>
28                 <band splitType="Stretch"/>
29         </background>
30         <title>
31                 <band height="79" splitType="Stretch">
32                         <staticText>
33                                 <reportElement uuid="b4880372-f0bd-40c2-98d9-9f0fe433dc21" x="55➥
                                        " y="14" width="405" height="31"/>
34                                 <textElement>
35                                         <font size="24"/>
36                                 </textElement>
37                                 <text><![CDATA[Az országok legfontosabb adatai]]></text>
38                         </staticText>
```

```
39                </band>
40        </title>
41        <pageHeader>
42                <band height="35" splitType="Stretch">
43                        <staticText>
44                                <reportElement uuid="deb81fed-e34f-4bd0-ae94-0508af09ff25" x="➥
                                      444" y="0" width="100" height="20" forecolor="#FF0066"/>
45                                <textElement textAlignment="Right">
46                                        <font isBold="true"/>
47                                </textElement>
48                                <text><![CDATA[Titkos!]]></text>
49                        </staticText>
50                </band>
51        </pageHeader>
52        <columnHeader>
53                <band height="42" splitType="Stretch">
54                        <staticText>
55                                <reportElement uuid="ce8a11bf-2f11-474c-bb67-6e8802752458" x="24➥
                                      " y="0" width="100" height="20"/>
56                                <textElement>
57                                        <font isBold="true"/>
58                                </textElement>
59                                <text><![CDATA[Ország]]></text>
60                        </staticText>
61                        <staticText>
62                                <reportElement uuid="491cf4f0-eeaa-4d9e-909a-8ee49518b24e" x="➥
                                      142" y="0" width="100" height="20"/>
63                                <textElement>
64                                        <font isBold="true"/>
65                                </textElement>
66                                <text><![CDATA[Főváros]]></text>
67                        </staticText>
68                        <staticText>
69                                <reportElement uuid="ff2efeea-6633-4b8a-8a04-9328b0ac56da" x="➥
                                      261" y="0" width="100" height="20"/>
70                                <textElement>
71                                        <font isBold="true"/>
72                                </textElement>
73                                <text><![CDATA[Földrész]]></text>
74                        </staticText>
75                        <staticText>
76                                <reportElement uuid="d4f1cdac-91ac-494f-a013-21e37d30e2d4" x="➥
                                      379" y="0" width="100" height="20"/>
77                                <textElement textAlignment="Right">
78                                        <font isBold="true"/>
79                                </textElement>
80                                <text><![CDATA[Terület (km2)]]></text>
81                        </staticText>
82                </band>
83        </columnHeader>
84        <detail>
85                <band height="40" splitType="Stretch">
86                        <textField>
87                                <reportElement uuid="a2856d02-9cef-48e3-9240-cf0a11b76488" x="24➥
                                      " y="12" width="100" height="20"/>
88                                <textElement/>
89                                <textFieldExpression><![CDATA[$F{ORSZAG}]]></textFieldExpression➥
                                      >
90                        </textField>
91                        <textField>
92                                <reportElement uuid="5e6ff35e-58db-418b-9bcb-e4f4044185e0" x="➥
                                      142" y="12" width="100" height="20"/>
93                                <textElement/>
94                                <textFieldExpression><![CDATA[$F{FOVAROS}]]></➥
                                      textFieldExpression>
```

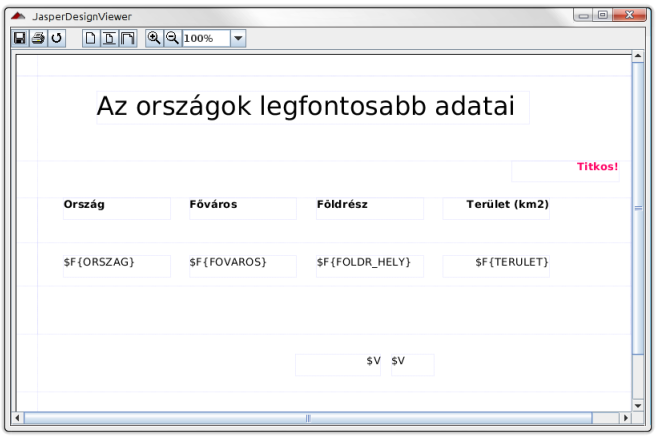
```
95                          </textField>
96                          <textField isStretchWithOverflow="true">
97                                  <reportElement uuid="be169129-86a9-47c3-829a-4196decf271c" x="➥
                                        261" y="12" width="100" height="20"/>
98                                  <textElement/>
99                                  <textFieldExpression><![CDATA[$F{FOLDR_HELY}]]></➥
                                        textFieldExpression>
100                         </textField>
101                         <textField>
102                                 <reportElement uuid="a583d337-91a5-4ee2-b1cf-3bcd45429f8a" x="➥
                                        379" y="12" width="100" height="20"/>
103                                 <textElement textAlignment="Right"/>
104                                 <textFieldExpression><![CDATA[$F{TERULET}]]></➥
                                        textFieldExpression>
105                         </textField>
106                 </band>
107         </detail>
108         <columnFooter>
109                 <band height="45" splitType="Stretch"/>
110         </columnFooter>
111         <pageFooter>
112                 <band height="54" splitType="Stretch">
113                         <textField>
114                                 <reportElement uuid="238f2dfb-d246-46b7-97b1-90fbdf7d869e" x="➥
                                        241" y="19" width="80" height="20"/>
115                                 <textElement textAlignment="Right"/>
116                                 <textFieldExpression><![CDATA[$V{PAGE_NUMBER}+" /"]]></➥
                                        textFieldExpression>
117                         </textField>
118                         <textField evaluationTime="Report">
119                                 <reportElement uuid="45046766-1c1b-4a3b-8380-9ec268d6f7df" x="➥
                                        331" y="19" width="40" height="20"/>
120                                 <textElement/>
121                                 <textFieldExpression><![CDATA[$V{PAGE_NUMBER}]]></➥
                                        textFieldExpression>
122                         </textField>
123                 </band>
124         </pageFooter>
125         <summary>
126                 <band height="42" splitType="Stretch"/>
127         </summary>
128 </jasperReport>
```

A riport elkészítése és megtekintése

Elkészült az első, adatbázis select-re épülő riport tervünk, azaz rendelkezünk egy jó *Orszagok.jrxml* fájllal. A következő teendőket az 1. cikkben már lényegében ismertettük, bár az adatbázis kapcsolat miatt újszerű elemek is lesznek. Az első az, hogy a korábban megismert *JRHelper* class *makeReport()* metódusának elkészítettük azt a változatát, ami a 3. paraméterben egy *java.sql.Connection* objektumot vár (2-2. Programlista). A megvalósításnál a *fillReportToFile()* metódusnak is ezt a változatát használtuk (13. sor).

```
1  // 2-2. Programlista: JRHelper.java
2
3  package org.cs.test;
4
5  import java.sql.Connection;
6  import java.util.HashMap;
7  ...
8  public class JRHelper
```

```
9  {
10 ...
11     public void makeReport(String jasperFileName, HashMap params, Connection conn) throws ➥
           JRException
12     {
13         JasperFillManager.fillReportToFile(jasperFileName, params, conn);
14     }
15 ...
16 }
```

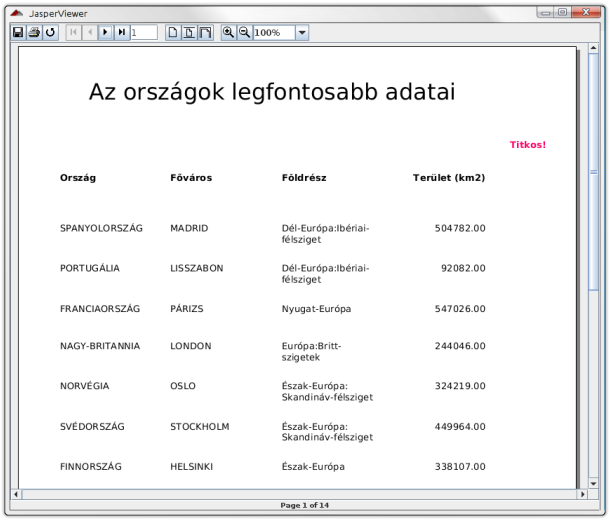


2.4. ábra: A Design megtekintése

Még nem fordítottuk le a jrxml fájlt jasper binárisra, így legyen ez a következő lépés. Az ant *build.xml*-ben a *file.name* értékét állítsuk be *Orszagok* értékre, majd adjuk ki az *ant compile* parancsot, ami létrehozza az *Orszagok.jasper* fájlt. A 2.4. ábra a riport tervet az *ant viewDesignXML* utasítással mutatja meg, azonban a továbbiakban ezt nem fogjuk használni, hiszen erre az iReport alapvetően alkalmas.

A *TestOrszagokRiport* osztály (2-3. Programlista) feladata, hogy segítségével elkészíthessük az *Orszagok.jrprint* jelentést. A 20. sorban szerzünk egy adatbázis kapcsolatot a HSQLDB szerver felé, majd a 22. sorban a *JRHelper* új *makeReport()* metódusával létrehozzuk a riportot. A 2. paraméter egy *Map*, amit most 0 darab elemmel adunk át, szerepét a következő pontban ismertetjük. A programot lefuttatva és az *ant view* parancsot használva a 2.5. ábrán mutatott riport készül el.

```
1  // 2-3. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.SQLException;
8  import java.util.HashMap;
9
10 public class TestOrszagokRiport
11 {
12     static String jasperFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jasper";
13
14     public static void main(String[] args) throws Exception
15     {
16         Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", ➥
               "");
17         JRHelper jh = new JRHelper();
18         jh.makeReport(jasperFile, new HashMap(), conn);
19         conn.close();
20     }
21 }
```



2.5. ábra: Az elkészült riport megtekintése

Paraméterek használata

Ebben a pontban bemutatjuk, hogy a *queryString* elem értékét hogyan tudjuk dinamikusan változtatni, amihez azt erre fogjuk cserélni:

```
select * from orszagok where terulet < $P{pTerulet}
```

A példában 2 riport paramétert fogunk használni, amiket az iReport *Report inspector Parameters* ágán tudunk könnyen a jrxml-hez adni, őket a 2-4. Programlista 5. és 6. sora mutatja. Az egyes paraméterek szerepe ez lesz:

- *pTerulet*: Ahogy látható, ez részévé vált a *queryString* select utasításnak (9. sor), így tudjuk majd azt manipulálni.
- *pheaderTerulet*: Ez csak egy próba arra, hogy a *columnHeader* utolsó oszlopának a nevét paraméterként adjuk át a jelentés generátorának (18. sor).

```
1  <!-- 2-4. Programlista: Orszagok.jrxml -->
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  ...
5          <parameter name="pTerulet" class="java.lang.Double"/>
6          <parameter name="pheaderTerulet" class="java.lang.String"/>
7
8          <queryString>
9                  <![CDATA[select * from orszagok where terulet < $P{pTerulet}]]>
10         </queryString>
11
12         <field name="ORSZAG" class="java.lang.String"/>
13         <field name="FOVAROS" class="java.lang.String"/>
14 ...
15                         <textField>
16                                 <reportElement uuid="8a198f6f-fbde-4b87-9604-a9cdff61dd97" x="➥
                                       411" y="0" width="100" height="20"/>
17                                 <textElement textAlignment="Right"/>
18                                 <textFieldExpression><![CDATA[$P{pheaderTerulet}]]></➥
                                       textFieldExpression>
19                         </textField>
20                 </band>
21         </columnHeader>
22 ...
23 </jasperReport>
```

Az új jrxml-hez is generáljunk jasper fájlt, majd tekintsük a *TestOrszagokRiport* új változatát (2-5. Programlista). A 20-21 sorokat már ismerjük, az újdonság most jön. A 23. sorban létrehozunk egy *params* nevű *HashMap* objektumot, majd a következő 2 sorban 2 elemet teszünk bele, amik a paraméterek nevei és értékei, majd a 27. sorban már ezzel felszerelve hívjuk a

korábbi *makeReport()*-ot. A jelentés generálója ahány alkalommal találkozik egy *$P{...}* hivatkozással, mindig innen próbálja annak az értékét feloldani. Ez igaz a select-re, de a 4. oszlopnévre egyaránt. Az új *where* feltétel miatt riportunk most 7 oldalasra csökkent. A 4. oszlop neve pedig a paraméterként átadott *Area* lett.

```
1  // 2-5. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.SQLException;
8  import java.util.HashMap;
9
10 /**
11  *
12  * @author inyiri
13  */
14 public class TestOrszagokRiport
15 {
16     static String jasperFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jasper";
17
18     public static void main(String[] args) throws Exception
19     {
20         Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", ➥
               "");
21         JRHelper jh = new JRHelper();
22
23         HashMap params = new HashMap();
24         params.put("pTerulet", 100000.00);
25         params.put("pheaderTerulet", "Area");
26
27         jh.makeReport(jasperFile, params, conn);
28         conn.close();
29     }
30 }
```

A queryString helyett adatforrás

A 2. fejezet befejezéseként vetünk egy pillantást a JasperReports adatforrások használatára, ugyanis nem feltétlenül szükséges a *queryString*-et a jrxml-be tennünk. A szemléltető példánk az előző marad, ugyanazt a feladatot fogjuk megoldani, de immár datasource használatával. Ehhez az első lépés az, hogy a teljes *queryString* részt, illetve az ahhoz kapcsolódó *pTerulet* paramétert kivesszük a jrxml fájlból, amit az egyértelműség érdekében ismét teljes terjedelmében tartalmaz a 2-6. Programlista.

```
1  <!-- 2-6. Programlista: Orszagok.jrxml -->
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.➥
       w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/➥
       jasperreports http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok" ➥
       language="groovy" pageWidth="595" pageHeight="842" columnWidth="555" leftMargin="20" ➥
       rightMargin="20" topMargin="20" bottomMargin="20" uuid="9a1dbe76-1ae5-4418-9e43-e56f6ed98193➥
       ">
5          <property name="ireport.zoom" value="1.0"/>
6          <property name="ireport.x" value="0"/>
7          <property name="ireport.y" value="0"/>
8
```

```
9         <parameter name="pheaderTerulet" class="java.lang.String"/>
10
11        <field name="ORSZAG" class="java.lang.String"/>
12        <field name="FOVAROS" class="java.lang.String"/>
13        <field name="FOLDR_HELY" class="java.lang.String"/>
14        <field name="TERULET" class="java.math.BigDecimal"/>
15        <field name="ALLAMFORMA" class="java.lang.String"/>
16        <field name="NEPESSEG" class="java.lang.Integer"/>
17        <field name="NEP_FOVAROS" class="java.lang.Integer"/>
18        <field name="AUTOJEL" class="java.lang.String"/>
19        <field name="COUNTRY" class="java.lang.String"/>
20        <field name="CAPITAL" class="java.lang.String"/>
21        <field name="PENZNEM" class="java.lang.String"/>
22        <field name="PENZJEL" class="java.lang.String"/>
23        <field name="VALTOPENZ" class="java.lang.String"/>
24        <field name="TELEFON" class="java.lang.Integer"/>
25        <field name="GDP" class="java.lang.Integer"/>
26        <field name="KAT" class="java.lang.Integer"/>
27        <background>
28                <band splitType="Stretch"/>
29        </background>
30        <title>
31                <band height="79" splitType="Stretch">
32                        <staticText>
33                                <reportElement uuid="b4880372-f0bd-40c2-98d9-9f0fe433dc21" x="55➥
                                    " y="14" width="405" height="31"/>
34                                <textElement>
35                                        <font size="24"/>
36                                </textElement>
37                                <text><![CDATA[Az országok legfontosabb adatai]]></text>
38                        </staticText>
39                </band>
40        </title>
41        <pageHeader>
42                <band height="35" splitType="Stretch">
43                        <staticText>
44                                <reportElement uuid="deb81fed-e34f-4bd0-ae94-0508af09ff25" x="➥
                                    444" y="0" width="100" height="20" forecolor="#FF0066"/>
45                                <textElement textAlignment="Right">
46                                        <font isBold="true"/>
47                                </textElement>
48                                <text><![CDATA[Titkos!]]></text>
49                        </staticText>
50                </band>
51        </pageHeader>
52        <columnHeader>
53                <band height="42" splitType="Stretch">
54                        <staticText>
55                                <reportElement uuid="ce8a11bf-2f11-474c-bb67-6e8802752458" x="24➥
                                    " y="0" width="100" height="20"/>
56                                <textElement>
57                                        <font isBold="true"/>
58                                </textElement>
59                                <text><![CDATA[Ország]]></text>
60                        </staticText>
61                        <staticText>
62                                <reportElement uuid="491cf4f0-eeaa-4d9e-909a-8ee49518b24e" x="➥
                                    142" y="0" width="100" height="20"/>
63                                <textElement>
64                                        <font isBold="true"/>
65                                </textElement>
66                                <text><![CDATA[Főváros]]></text>
67                        </staticText>
68                        <staticText>
```

```
69                              <reportElement uuid="ff2efeea-6633-4b8a-8a04-9328b0ac56da" x="➥
                                    261" y="0" width="100" height="20"/>
70                              <textElement>
71                                      <font isBold="true"/>
72                              </textElement>
73                              <text><![CDATA[Földrész]]></text>
74                      </staticText>
75                      <textField>
76                              <reportElement uuid="8a198f6f-fbde-4b87-9604-a9cdff61dd97" x="➥
                                    379" y="0" width="100" height="20"/>
77                              <textElement textAlignment="Right">
78                                      <font isBold="true"/>
79                              </textElement>
80                              <textFieldExpression><![CDATA[$P{pheaderTerulet}]]></➥
                                    textFieldExpression>
81                      </textField>
82              </band>
83      </columnHeader>
84      <detail>
85              <band height="40" splitType="Stretch">
86                      <textField>
87                              <reportElement uuid="a2856d02-9cef-48e3-9240-cf0a11b76488" x="24➥
                                    " y="12" width="100" height="20"/>
88                              <textElement/>
89                              <textFieldExpression><![CDATA[$F{ORSZAG}]]></textFieldExpression➥
                                    >
90                      </textField>
91                      <textField>
92                              <reportElement uuid="5e6ff35e-58db-418b-9bcb-e4f4044185e0" x="➥
                                    142" y="12" width="100" height="20"/>
93                              <textElement/>
94                              <textFieldExpression><![CDATA[$F{FOVAROS}]]></➥
                                    textFieldExpression>
95                      </textField>
96                      <textField isStretchWithOverflow="true">
97                              <reportElement uuid="be169129-86a9-47c3-829a-4196decf271c" x="➥
                                    261" y="12" width="100" height="20"/>
98                              <textElement/>
99                              <textFieldExpression><![CDATA[$F{FOLDR_HELY}]]></➥
                                    textFieldExpression>
100                     </textField>
101                     <textField>
102                             <reportElement uuid="a583d337-91a5-4ee2-b1cf-3bcd45429f8a" x="➥
                                    379" y="12" width="100" height="20"/>
103                             <textElement textAlignment="Right"/>
104                             <textFieldExpression><![CDATA[$F{TERULET}]]></➥
                                    textFieldExpression>
105                     </textField>
106             </band>
107     </detail>
108     <columnFooter>
109             <band height="45" splitType="Stretch"/>
110     </columnFooter>
111     <pageFooter>
112             <band height="54" splitType="Stretch">
113                     <textField>
114                             <reportElement uuid="238f2dfb-d246-46b7-97b1-90fbdf7d869e" x="➥
                                    241" y="19" width="80" height="20"/>
115                             <textElement textAlignment="Right"/>
116                             <textFieldExpression><![CDATA[$V{PAGE_NUMBER}+" /"]]></➥
                                    textFieldExpression>
117                     </textField>
118                     <textField evaluationTime="Report">
119                             <reportElement uuid="45046766-1c1b-4a3b-8380-9ec268d6f7df" x="➥
                                    331" y="19" width="40" height="20"/>
```

```
120                                 <textElement/>
121                                 <textFieldExpression><![CDATA[$V{PAGE_NUMBER}]]></➥
                                        textFieldExpression>
122                             </textField>
123                     </band>
124             </pageFooter>
125             <summary>
126                     <band height="42" splitType="Stretch"/>
127             </summary>
128 </jasperReport>
```

A jrxml más tekintetben nem változott, így annak rövid elemzésétől most eltekintünk, azonban a *jrxml→jasper* fordítást elvégezzük. A keletkezett új *Orszagok.jasper* binárisra elkészítettük a *TestOrszagokRiport* 3. változatát (2-7. Programlista). Ez már a *JRDataSource* interfésszel rendelkező adattermelő objektumra épül. A 20. sor mutatja – mint eddig is –, hogy milyen jasper fájlt fogunk használni. A 24. sor az ismerős select-ünk, ahol a '?' a kihagyott *pTerulet* paraméter helyett lesz használatos, hogy még hasonlóbb legyen a megoldás az előzőéhez. A 26. sorban az ismerős módon szerzünk egy adatbázis *Connection* objektumot, de itt nem ezt fogjuk átadni a riport generátorának, ami lényeges különbség. A 27-29 sorok között megadjuk a paramétert (értéke: 100000.00), majd lefuttatjuk a select-et, ami visszaad egy szabványos *rs* nevű *ResultSet* objektumot. És a 31. sorban érkezik az igazi újdonság! Az *rs*-t becsomagoljuk egy *JRDataSource* interfésszel bíró *jrds* nevű objektumba, ami most történetesen egy *JRResultSetDataSource* class példány. A 38. sorban megadjuk az egyetlen riport paraméterünket, aminek értéke most a pontosabb *Area (km2)* szöveg lett. A 40. sorban találkozunk a 2. jelentős változtatással, ugyanis itt újra azt a *makeReport()* metódust használtuk, amit az 1. cikkben megismertünk, hiszen az alkalmas egy *JRDataSource* fogadására.

```
1  // 2-7. Programlista: TestOrszagokRiport.java (JRDataSource)
2
3  package org.cs.test;
4
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.PreparedStatement;
8  import java.sql.ResultSet;
9  import java.sql.SQLException;
10 import java.util.HashMap;
11 import net.sf.jasperreports.engine.JRDataSource;
12 import net.sf.jasperreports.engine.JRResultSetDataSource;
13
14 /**
15  *
16  * @author inyiri
17  */
18 public class TestOrszagokRiport
19 {
20     static String jasperFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jasper";
21
22     public static void main(String[] args) throws Exception
23     {
24         String sql = "select * from orszagok where terulet < ?";
25
26         Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", ➥
               "");
27         PreparedStatement stmt = conn.prepareStatement( sql );
```

```
28          stmt.setDouble(1, 100000.00);
29          ResultSet rs = stmt.executeQuery();
30
31          JRDataSource jrds = new JRResultSetDataSource( rs );
32
33          JRHelper jh = new JRHelper();
34
35          HashMap params = new HashMap();
36
37  //          params.put("pTerulet", 100000.00);
38          params.put("pheaderTerulet", "Area (km2)");
39
40          jh.makeReport(jasperFile, params, jrds);
41
42          rs.close();
43          stmt.close();
44          conn.close();
45
46      }
47
48  }
```

A programot futtassuk le és nézzük meg a keletkezett *Orszagok.jrprint* tartalmát az *ant view* paranccsal! Leszámítva a 4. oszlop kissé megváltoztatott nevét, ugyanazt a 7 oldalt fogjuk látni, mint korábban. A *JRResultSetDataSource* pontosan úgy működik, ahogy vártuk. Ő is egy select-re épül, így az eredménytábla oszlopneveit hasonlóan illeszteni lehet a jrxml-be lévő field-ekre.

A JasperFillManager

Végezetül szeretnénk pár szót szólni *JasperFillManager fillReportToFile()* metódusának lehetőségeiről. Ez a metódus többszörösen túlterhelt, de mindegyik funkciója a *jrprint* fájl elkészítését szolgálja. A későbbiekben látni fogjuk, hogy a riportot nem szükséges fájlba mentenünk, az lehet például egy *OutputStream* is. A fájlba gyártó metódus lehetséges paraméterei ezek lehetnek:

- *JasperReport*: A lefordított jrprint tartalom memóriabeli reprezentációja. Bizonyos körülmények között használatos, de nem ez a tipikus.
- *Connection*: Egy *java.sql.Connection* objektum, amikor beágyazott query-re épül a riportkészítés.
- *JRDataSource*: Az utolsó példában látott adatforrás alapú riportkészítésnél használatos.
- *parameters*: Egy *java.util.HashMap*, ami lehetővé teszi a riport paramétereinek átadását.
- *sourceFileName*: A *JasperReport* objektum alternatívája, amikor jrprint fájl nevet adunk át a riport készítéséhez.
- *destFileName*: A legyártott jrprint fájl neve lesz. Amennyiben nem adjuk meg, úgy a jasper fájl nevével, de jrprint kiterjesztéssel generálódik majd a riport.

3. A JasperReports adatforrások használata

Az előző rész végén már megismerhettük az adatforrás használatát. Ez azonban egy sokkal általánosabb JasperReports lehetőség, hiszen minden olyan objektum lehet egy riport adatforrása, ami implementálja a *JRDataSource* interfészt. Ez a gyakorlatban szinte a végtelen lehetőségek tárházát jelenti, amikor jelentésünket adatokkal töltjük fel. Egy jól kitalált adatforrás objektum több helyről és meglehetősen összetett logikával képes táplálni az elkészítendő dokumentumunkat.

```
// 3-1. Programlista: JRDataSource.java

package net.sf.jasperreports.engine;

public interface JRDataSource
{
        /**
         * Tries to position the cursor on the next element in the data source.
         * @return true if there is a next record, false otherwise
         * @throws JRException if any error occurs while trying to move to the next element
         */

        public boolean next() throws JRException;

        /**
         * Gets the field value for the current position.
         * @return an object containing the field value. The object type must be the field ➥
             object type.
         */
        public Object getFieldValue(JRField jrField) throws JRException;
}
```

A JasperReports adatforrás

Egy JasperReports adatforrás bármilyen objektum lehet, ami implementálja a *JRDataSource* interfészt (3-1. Programlista), aminek 2 metódusa van: *next()* és *getFieldValue()*. Egyik sem okozhat különösebb meglepetést azzal, amit a működésüktől elvárunk, amikor egy osztály implementálja őket. A *next()* a következő sorra próbálja pozicionálni az adatforrást, ha ez sikerül akkor *true*, különben *false* értéket ad vissza. Ez lehetővé teszi, hogy a háttérben működő és a *next()* metódust visszahívó riportgenerátor működése megfelelően legyen vezérelve. A *getFieldValue()* metódust szintén a generátor hívja és 2 körülmény mindig igaz:

1. A hívás mindig az adatforrás aktuális sorára vonatkozik.
2. A *JRField* paramétert a generátor úgy állítja össze, hogy veszi az éppen aktuális *$F{field-name}* *jrxml*-beli hivatkozást és ebből egy ilyen típusú objektumot gyárt le, ezt átadva a *getFieldValue()* metódusnak, ami persze alkalmas módon van elkészítve. Ez azt jelenti, hogy a *jrField* objektum aktuális sorra való alkalmazása alapján kiválasztódik a visszaadandó érték.

Létezik egy származtatott adatforrás interface is, a *JRRewindableDataSource*, ahol a *moveFirst*() metódus implementálása is szükséges. Ezzel az adatforrás elejére tudunk „visszatekerni”.

Az üres adatforrás

Az 1-4. Programlista *JRHelper* teszt programjánál már megismertük *JREmptyDataSource* class

szerepét. Felmerül a kérdés, hogy erre miért is van szükség? A riportot készítő *JasperFillManager* osztály többféleképpen is lehetővé teszi a natív *jrprint* riport fájlok elkészítését. Ezek a metódusok az alábbi 3 típusba sorolhatóak:

- *fillReportToFile(...)* → Ekkor a *jrprint* dokumentum egy külső fájlba készül (eddig mindig ezt a lehetőséget használtuk példáinkban).
- *fillReport(...)* → Ez a metódus egy *JasperPrint* objektumot ad vissza, ami a *jrprint* objektum modelljét jelentő osztály.
- *fillToStream(...)* → A *jrprint* dokumentumot *OutputStream* objektumként adja vissza, ezzel a módszerrel például egy Java servlet választ tudunk generálni.

Korábban már említettük, hogy ezek a *fill...()* metódusok 3 input információt mindig kötelezően kapnak:

1. A riport template külső *jasper* fájlként vagy a memóriában már felépített *JasperReport* objektumként megadva.
2. A riport paraméterei, ami egy Java *Map* objektum.
3. A riport dinamikus adatai, ami egy SQL *Connection* vagy *JRDataSource* objektum közreműködésével szerezhető meg a *fill...()* számára.

Ennyi elmélet után képzeljünk el egy olyan riportot, ami valójában nem tartalmaz semmilyen adatforrásból érkező adatot sem. Ilyen lehet például egy nyomtatvány vagy számla. Ugyanakkor formálisan itt is szükséges egy adatforrás, így ezekben a helyzetekben mindig egy *JREmptyDataSource* objektumot használunk. A keletkezett riport természetesen ilyenkor sem statikus, hiszen ennek talán semmi értelme sem lenne. Az adatokat ekkor a riport paraméterek *Map* objektumán keresztül továbbítjuk a generátor részére.

A java Map alapú adatforrás

Amikor a 2. cikkben megismertük az SQL select alapú eredménytáblát, láthattuk hogy az oszlopok és a jrxml fájlban megadott mezők (field) nevének illeszkedése milyen természetes eleganciával működött. A *Map* típusú adatforrás is pont ehhez hasonló. Egy adatforrás sor nem más, mint egy *Map* objektum, ahol (kulcs, érték) párok vannak. A kulcs, mint név van abban a szerepben, mint a select eredményének oszlop neve. Az adatforrás azonban több sorból áll, amit az alábbi 2 módszerrel is utánozni tudunk:

- a *Map* objektumok tömbje
- a *Map* objektumok kollekciója (például láncolt lista)

Ebben a modellben már elkészíthető egy *JRDataSource* interfésszel rendelkező class, ahol az egyes metódusok így tudnak működni:

- *next()* → Értelmezhető a tömb következő indexű elemeként, mely ha még van, akkor a visszatérési érték *true*, különben *false*.
- *getFieldValue()* → A jrxml *field* aktuális sorra vonatkozó feltöltésekor a generátor megszerzi a mező nevét (legyen ez most *fieldName*) és ezt használja fel az érték előállításához egy *get(fieldName)* hívással. Azt is fontos észben tartani, hogy a mező típusos, hiszen a neve mellett az osztálya is meghatározásra került, így a generátor a megszerzett értéket esetleg még erre a class-ra át is konvertálja.

A 3-2. Programlista a *Map* adatforrás használatát mutatja be, miközben az adatbázis alapú példánál használt és ott lefordított *jasper* fájlt nem is változtattuk meg. Ezzel az is látható, hogy amennyiben egy *jrxml/jasper* pároshoz illeszkedő adatforrásról gondoskodunk, úgy azt valóban nem kell megváltoztatni, hiszen a generátor valójában nem is tudja, hogy az adatokat honnan kapja, ő csak egy *JRDataSource* interfésszel rendelkező objektummal kommunikál.

```
1  // 3-2. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4  ...
5  public class TestOrszagokRiport
6  {
7      static String jasperFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jasper";
8      static String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
9  ...
10     public static void testMapArray() throws Exception
11     {
12         Map[] aMap = new HashMap[1000];
13
14         for (int i = 0; i < 1000; i++)
15         {
16             Map map = new HashMap();
17             map.put("ORSZAG", "ORSZAG-" + i);
18             map.put("FOVAROS", "FOVAROS-" + i);
19             map.put("FOLDR_HELY", "FOLDR_HELY-" + i);
20             map.put("TERULET", new BigDecimal((i + 1) * 1000));
21             aMap[ i] = map;
22         }
23
24         JRDataSource jrds = new JRMapArrayDataSource(aMap);
25
26         HashMap params = new HashMap();
27         params.put("pheaderTerulet", "Area (km2)");
28
29         JasperFillManager.fillReportToFile(jasperFile, jrprintFile, params, jrds);
30     }
31
32     public static void testMapCollection() throws Exception
33     {
34         List list = new ArrayList();
35
36         for (int i = 0; i < 1000; i++)
37         {
38             Map map = new HashMap();
39             map.put("ORSZAG", "ORSZAG-" + i);
40             map.put("FOVAROS", "FOVAROS-" + i);
41             map.put("FOLDR_HELY", "FOLDR_HELY-" + i);
42             map.put("TERULET", new BigDecimal((i + 1) * 1000));
43
44             list.add(map);
45         }
46
47         JRDataSource jrds = new JRMapCollectionDataSource(list);
48
49         HashMap params = new HashMap();
50         params.put("pheaderTerulet", "Area (km2)");
51
52         JasperFillManager.fillReportToFile(jasperFile, jrprintFile, params, jrds);
53     }
54 ...
55     public static void main(String[] args) throws Exception
56     {
57 //         testMapArray();
58         testMapCollection();
59     }
60 }
```

3.1. ábra: Az országok riport - Map adatforrással

A *testMapArray()* és *testMapCollection()* tesztmetódusok a tömb és *List* modellben megvalósított *Map* adatforrás használatot mutatják be, remélhetőleg ezek működését mindenki egyből érthetőnek találja. Nézzük a tömb esetét, a lista ezzel teljesen analóg! Felveszünk egy 1000 elemet befogadni képes *aMap*, tömböt, majd egy *for* ciklusban feltöltjük generált értékekkel. Arra vigyázunk, hogy az egyes tömbelemek map objektumaiban használt kulcsnevek egyezzenek meg azzal, ami a jrxml mező nevek, hiszen ez a titka az egész működésnek. A *jrds* nevű *JRMapArrayDataSource* objektum az *aMap* objektummal lesz megkonstruálva. Ezután az eddigiek során is használt 1 db riport paraméterről is gondoskodunk, majd *fillReportToFile()* 4. paramétereként ezt a *jrds* adatforrást használjuk. Az eredményt, azaz a létrejött riport egy részletét a 3.1. ábrán tekinthetjük meg. A lista alapú példa is hasonlóan működik és természetesen pontosan ugyanezt a *jrprint* fájlt hozza létre.

[2] *POJO=Plain Old Java Object*

A java objektum alapú adatforrás

A következő és a Java eszköztárához közel álló adatforrás típus a *JavaBean* (más néven: *POJO*[2] vagy *Value Object*) alapú megoldás. Itt is létezik egy névkonvenciós szabvány, ami jó alapja lehet a jrxml field nevek és a bean property-k közötti összekapcsolásnak. Egy property neve legyen *property*, ekkor annak publikus *getter* és *setter* metódusának a neve ebből úgy lesz képezve, hogy az *set* vagy *get* karaktersorozattal kezdődik, majd a property neve úgy, hogy az első karaktere nagybetű lesz. Példa: *getProperty()*. Ennek megfelelően csináltunk egy másolatot *Orszagok-Bean.jrxml* (egy részletét mutatja a 3-3. Programlista) néven, ahol csak a következő 3 változtatást eszközöltük:

- A mezők nevét csupa kisbetűre alakítottuk
- A *textField* komponensekben lévő *$F{}* hivatkozásokat is javítottuk, azaz például a *$F{ORSZAG}*→*$F{orszag}* változtatást megtettük.
- A fölösleges, a riportban és emiatt a beanből hiányzó field-ek törlése.

```
<!-- 3-3. Programlista: Orszagok-Bean.jrxml -->
...
  <field name="orszag" class="java.lang.String"/>
  <field name="fovaros" class="java.lang.String"/>
  <field name="foldrajzi_hely" class="java.lang.String➥
      "/>
  <field name="terulet" class="java.math.BigDecimal"/>
...
```

Az *OrszagBean* class (3-4. Programlista) úgy készült, hogy ezeket a mezőket tartalmazza és illeszkedjen ezekre a property nevekre. Itt említjük meg a riportgenerátorral kapcsolatos egyik gyakorlati tapasztalatunkat, ami az a mechanizmus, ahogy az aktuális sorból a generátor kiveszi az értékeket. Az első kísérletnél a jrxml fájlban minden mezőt otthagytunk, ami egy program exception-höz vezetett, ugyanis a következő adatforrás sorra lépéskor a generátor előre fel akarja tölteni a field-ek értékét, de a felvett 4 property kivételével a többit természetesen nem tudta. Ez azt is jelenti, hogy a JasperReports nem csak a riportban ténylegesen meghivatkozott mezőket hívja fel, hanem az összes deklaráltat, emiatt volt szükséges a használt 4 bean jellemzőn kívüliek megszüntetése. Az alternatíva az *OrszagBean* class kibővítése lett volna.

```
// 3-4. Programlista: OrszagBean.java

package org.cs.test;

import java.math.BigDecimal;

public class OrszagBean
{
    private String orszag;
    private String fovaros;
    private String foldrajzi_hely;
    private BigDecimal terulet;

    public String getOrszag()
    {
        return orszag;
    }

    public void setOrszag(String orszag)
    {
        this.orszag = orszag;
    }

    public String getFovaros()
    {
        return fovaros;
    }

    public void setFovaros(String fovaros)
    {
        this.fovaros = fovaros;
    }

    public String getFoldrajzi_hely()
    {
        return foldrajzi_hely;
    }

    public void setFoldrajzi_hely(String ➥
        foldrajzi_hely)
    {
        this.foldrajzi_hely = foldrajzi_hely;
    }

    public BigDecimal getTerulet()
    {
        return terulet;
    }

    public void setTerulet(BigDecimal terulet)
    {
        this.terulet = terulet;
    }
}
```

Végül nézzük meg a 3-5. Programlista segítségével a JavaBean adatforrás használatát!

```
1  // 3-5. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4  ...
5  public class TestOrszagokRiport
6  {
7     static String jasperFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jasper";
8     static String jasperBeanFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-Bean.jasper";
9     static String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
10 ...
11   public static void testBeanArray() throws Exception
12     {
13         OrszagBean[] beans = new OrszagBean[1000];
14
15         for (int i = 0; i < 1000; i++)
16         {
17             OrszagBean bean = new OrszagBean();
18             bean.setOrszag("BORSZAG-" + i);
19             bean.setFovaros( "BFOVAROS-" + i);
20             bean.setFoldrajzi_hely("BFOLDR_HELY-" + i);
21             bean.setTerulet(new BigDecimal((i + 1) * 1000));
22             beans[ i] = bean;
23         }
24
25         JRDataSource jrds = new JRBeanArrayDataSource( beans );
26
27         HashMap params = new HashMap();
28         params.put("pheaderTerulet", "Area (km2)");
29
30         JasperFillManager.fillReportToFile(jasperBeanFile, jrprintFile, params, jrds);
31     }
32
33     public static void testBeanCollection() throws Exception
34     {
35         List<OrszagBean> list = new ArrayList<OrszagBean>();
36
37         for (int i = 0; i < 1000; i++)
38         {
39             OrszagBean bean = new OrszagBean();
40             bean.setOrszag("BLORSZAG-" + i);
41             bean.setFovaros( "BLFOVAROS-" + i);
42             bean.setFoldrajzi_hely("BLFOLDR_HELY-" + i);
43             bean.setTerulet(new BigDecimal((i + 1) * 1000));
44             list.add(bean);
45         }
46
47         JRDataSource jrds = new JRBeanCollectionDataSource(list);
48
49         HashMap params = new HashMap();
50         params.put("pheaderTerulet", "Area (km2)");
51
52         JasperFillManager.fillReportToFile(jasperBeanFile, jrprintFile, params, jrds);
53
54     }
55 ...
56     public static void main(String[] args) throws Exception
57     {
58 //         testBeanArray();
59         testBeanCollection();
60     }
61 }
```

A 11-31 sorok közötti *testBeanArray()* a tömbbe szervezett JavaBean-ek esete. A 13. sorban egy 1000 elemű *beans* tömb kerül megadásra, aminek a szerepe megegyezik a *Map*-es példáéval, azaz az egyes tömbelemek az adatforrás „sorai". A 15-23 sorok a tesztadatok generálását valósítják meg. Témánk szempontjából a 25. sor szerepe fontos, ugyanis itt hozzuk létre a *JRBeanArrayDataSource jrds* objektumot, azaz a JavaBean alapú adatforrást. A *jrds* belül természetesen a *beans* tömbre épülve fog működni. A 27-28 sorok a szokásos 1 db riport paramétert adják meg, a példa szempontjából lényegtelen a szerepe, akár törölhetnénk is. A 30. sor gyártja le a riportot, immár a *beans* adataira épülve. A *fillReportToFile()* 1. paramétere a jasper fájl, azaz a riport template. A 2. paramétere az eredményként létrejövő *jrprint* fájl neve. A 3. a paraméterek Map-ja. A 4. pedig a bean alapú adatforrásunk. A 33-54 sorok közötti *testBeanCollection()* teszt metódus működése teljesen hasonló, de itt a tömb helyett egy lista tartalmazza az *OrszagBean* objektumokat, ennek megfelelően a 47. sor adatforrás objektuma is ilyen működésű implementációt példányosít. Végezetül szeretnénk megjegyezni, hogy a JavaBean alapú adatforrás implementációja az Apache Commons Bean Utils megoldásra épül: `http://commons.apache.org/beanutils/`, így annak *jar* könyvtára szükséges.

XML adatforrás

Az XML egy kiemelt jelentőséggel bíró adatformátum, mert több jó tulajdonsága is van:

- könnyen hordozható a platformok között és önleíró
- hierarchikus szerkezetével az adatok közötti belső szerkezetet is jól lehet ábrázolni

Emiatt nagy előny, hogy a JasperReports támogatja azt a lehetőséget, hogy a jelentés adatforrása egy XML dokumentum is lehet, aminek kezelési formátuma is tetszőleges: fájl, string, stream (byte folyam). Maradjunk az eddigi „országos" példánknál, így elkészítettünk egy új, *orszagok.xml* fájlt (3-6. Programlista), amiben az */orszagok/item* útvonal alatt a már ismerős adataink foglalnak helyet.

```
// 3-6. Programlista: orszagok.xml

<orszagok>
    <item>
        <orszag>Magyar1</orszag>
        <fovaros>Budapest</fovaros>
        <foldrajzi_hely>CE</foldrajzi_hely>
        <terulet>93000</terulet>
    </item>
    <item>
        <orszag>Orszag 2</orszag>
        <fovaros>Fováros 2</fovaros>
        <foldrajzi_hely>CE</foldrajzi_hely>
        <terulet>3213123</terulet>
    </item>
    <item>
        <orszag>Orszag 3</orszag>
        <fovaros>Fovaros 3</fovaros>
        <foldrajzi_hely>CE</foldrajzi_hely>
        <terulet>9534500</terulet>
    </item>
    <item>
        <orszag>Orszag 4</orszag>
        <fovaros>Fovaros 4</fovaros>
        <foldrajzi_hely>CE</foldrajzi_hely>
        <terulet>93000</terulet>
    </item>
</orszagok>
```

Az *Orszagok.jrxm* template alapján elkészítettünk egy *Orszagok-XML.jrxml* (3-7. Programlista) változatot, ami a field mezők kivételével mindenben megegyezik az előzőekkel, így szemléltetésként csak azt a jrxml részletet tettük ide be. A *fieldDescription* most egy új elem, eddig erre nem volt szükség, amikor a *field*-eket megadtuk. XML esetén viszont kötelező, azt mondjuk meg vele, hogy például a *fovaros* mező az XML *fovaros* tag-jéből veszi ki az adatokat. Itt szeretnénk kiemelni, hogy a *field* neve és az *XML tag* neve eltérhet egymástól, ezek egyezése nem követelmény, hiszen itt úgyis összerendelődnek. A *<fovaros>* tag neve lehetett volna akár *<capital>* is. Ez a mechanizmus rugalmasabbá teszi a különféle helyekről érkező XML adatforrások használatának lehetőségét.

```
1  // 3-7. Programlista: Orszagok-XML.jrxml
2
3  <jasperReport...
4  ...
5          <parameter name="pheaderTerulet" class="java.lang.String"/>
6
7          <field name="fovaros" class="java.lang.String">
8                  <fieldDescription><![CDATA[fovaros]]></fieldDescription>
9          </field>
10         <field name="orszag" class="java.lang.String">
11                 <fieldDescription><![CDATA[orszag]]></fieldDescription>
12         </field>
13         <field name="foldrajzi_hely" class="java.lang.String">
14                 <fieldDescription><![CDATA[foldrajzi_hely]]></fieldDescription>
15         </field>
16         <field name="terulet" class="java.math.BigDecimal">
17                 <fieldDescription><![CDATA[terulet]]></fieldDescription>
18         </field>
19         <background>
20                 <band splitType="Stretch"/>
21         </background>
22 ...
23 </jasperReport>
```

Van *jrxml* template-ünk, le is fordítottuk, illetve rendelkezünk egy XML adatforrással. Még az van hátra, hogy megtanuljuk a *JRXmlDataSource* használatát, amit az eddigi gyakorlatot folytatva a *TestOrszagokRiport* class egy módosított verziója (3-8. Programlista) segítségével fogunk bemutatni. Példánkban az is újdonság, hogy most kétféleképpen is megmutatjuk a használatot:

- Az XML-t közvetlenül a fájlra hivatkozva szerezzük meg: *testXMLFromFile()*
- Az XML egy *InputStream* objektumból érkezik: *testXML()*

Nézzük a példát! A 30. sorban egy új *jasper* fájlra való hivatkozást látunk, ami a 3-7. Programlistán mutatott kis *jrxml* változtatás miatt vált szükségessé. A *testXML()* 35. sorában a *JRLoader* beépített class segítségével egy *InputStream* formájában állítjuk elő az XML fájlunk tartalmát. Ez a byte-sorozat előállítás persze több más módon is történik a gyakorlatban, de ha már itt az XML fájl, akkor miért ne abból vegyük ki. A *testXMLFromFile()* 46. sorában ezzel szemben egy jól ismert java *File* objektumot hozunk létre, persze szintén az *orszagok.xml* alapján. A 37., illetve 47. sorokban lévő *JRXmlDataSource* objektum előállítása innen már vagy az *InputStream*, vagy a *File* objektumra épülve megy. Vegyük észre azonban, hogy a konstruktor mindkét esetben tartalmaz egy *XPath* hivatkozást az XML fa egy útjára: */orszagok/item*. Ez azt jelenti, hogy az XML-ben eljutva az *item* elem szintjéig, azon kell végiglépdelnie az adatforrásnak. A példa XML például 4 soros, mert most éppen 4 db *item* eleme van. A *jrds* adatforrás objektum legyártása után már az ismert paraméterezés és riportkészítés történik. Ismétlésként a *fillReportToFile()* metódus paramétereinek jelentése:

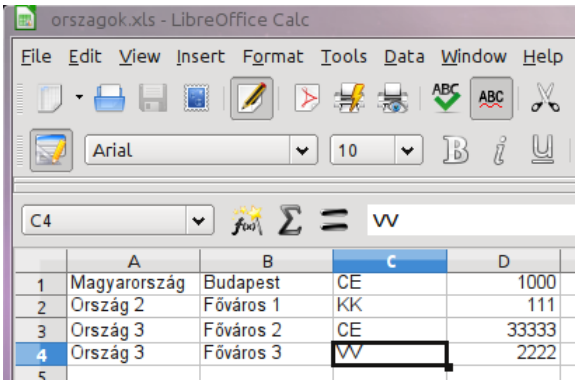
- *jasperFile*: A jasper fájl, ami a jrxml fájl fordításaként keletkezik (*String* típus).
- *jrprintFile*: A *jrprint* fájl neve, ahova elkészül a riport (*String* típus).
- *params*: A paraméterek objektuma *(Map* típus).
- *jrds*: A riport elkészítéséhez használt adatforrás (*JRXmlDataSource* típus).

```
1  // 3-8. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4
5  import java.io.File;
6  import java.io.InputStream;
7  import java.math.BigDecimal;
8  import java.util.ArrayList;
9  import java.util.HashMap;
10 import java.util.List;
11 import java.util.Locale;
12 import java.util.Map;
13 import net.sf.jasperreports.engine.JRDataSource;
14 import net.sf.jasperreports.engine.JRException;
15 import net.sf.jasperreports.engine.JRResultSetDataSource;
16 import net.sf.jasperreports.engine.JasperFillManager;
17 import net.sf.jasperreports.engine.JasperReport;
18 import net.sf.jasperreports.engine.data.JRBeanArrayDataSource;
19 import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;
20 import net.sf.jasperreports.engine.data.JRCsvDataSource;
21 import net.sf.jasperreports.engine.data.JRMapArrayDataSource;
22 import net.sf.jasperreports.engine.data.JRMapCollectionDataSource;
23 import net.sf.jasperreports.engine.data.JRXlsDataSource;
24 import net.sf.jasperreports.engine.data.JRXlsxDataSource;
25 import net.sf.jasperreports.engine.data.JRXmlDataSource;
26 import net.sf.jasperreports.engine.util.JRLoader;
27
28 public class TestOrszagokRiport
29 {
30    static String jasperFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-XML.jasper";
31
32 ...
33     public static void testXML() throws Exception
34     {
35         InputStream is = JRLoader.getLocationInputStream("/home/tanulas/jasperreport/pelda-1/➥
               orszagok.xml");
36
37         JRDataSource jrds = new JRXmlDataSource(is, "/orszagok/item");
38         HashMap params = new HashMap();
39         params.put("pheaderTerulet", "Area (km2)");
40
41         JasperFillManager.fillReportToFile(jasperFile, jrprintFile, params, jrds);
42     }
43
44     public static void testXMLFromFile() throws Exception
45     {
46         File file = new File("/home/tanulas/jasperreport/pelda-1/orszagok.xml");
47         JRDataSource jrds = new JRXmlDataSource(file, "/orszagok/item");
48         HashMap params = new HashMap();
49         params.put("pheaderTerulet", "Area (km2)");
50
51         JasperFillManager.fillReportToFile(jasperFile, jrprintFile, params, jrds);
52     }
53 ...
54         // Only test
55     public static void main(String[] args) throws Exception
56     {
57 //         testXMLFromFile();
58         testXML()
59     }
60 }
```

Az elkészült riport kinézetét a 3.1. ábrán már láthattuk. Itt is megjegyezzük, hogy az XML adatforrások használatához a *xalan* jar fájlt (Java XPath kezelő könyvtár) szükséges használni, ami természetesen része a JasperReports csomagnak, de letölthető innen is: `http://xml.apache.org/xalan-j/`.

Excel adatforrás

Az excel formátum vitathatatlanul az egyik legelterjedtebb, táblázatos adattárolási forma. Bár a pivot tábla kiváló riportkészítő eszköz, azért elképzelhető, hogy a feladatot valamilyen szempontból hatékonyabb, kifejezetten riportkészítő környezetben célszerű elvégezni. Esetleg így jobban automatizálható vagy talán az excel tábla csak egy összetettebb adatforrás része. A 3.2. ábra az ismert példánkhoz illeszkedő adatok *xls* táblába való elhelyezését mutatja (*orszagok.xls*).



3.2. ábra: Excel adatok

```
1  // 3-9. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4  ...
5  import jxl.Workbook;
6  import jxl.WorkbookSettings;
7  import jxl.write.WritableWorkbook;
8  ...
9  public class TestOrszagokRiport
10 {
11     static String jasperBeanFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-Bean.jasper";
12     static String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
13 ...
14     public static void testXls() throws Exception
15     {
16         String fileName = "/home/tanulas/jasperreport/pelda-1/orszagok.xls";
17
18         String[] columnNames = new String[]
19         {
20             "orszag", "fovaros", "foldrajzi_hely", "terulet"
21         };
22         int[] columnIndexes = new int[]
23         {
24             0, 1, 2, 3
25         };
26
27 //          InputStream is = JRLoader.getLocationInputStream(fileName);
28 //          JRDataSource jrds = new JRXlsDataSource(is);
29 //          JRDataSource jrds = new JRXlsDataSource(new File(fileName));
30
31         WorkbookSettings ws = new WorkbookSettings();
32         ws.setEncoding("Cp1252");
33         Workbook wb = Workbook.getWorkbook(new File(fileName), ws);
34
35         JRDataSource jrds = new JRXlsDataSource( wb );
36
37         ((JRXlsDataSource) jrds).setColumnNames(columnNames, columnIndexes);
38         ((JRXlsDataSource) jrds).setLocale(new  Locale("hu_HU"));
```

```
39
40          HashMap params = new HashMap();
41          params.put("pheaderTerulet", "Area (km2)");
42
43          JasperFillManager.fillReportToFile(jasperBeanFile, jrprintFile, params, jrds);
44      }
45  ...
46      public static void main(String[] args) throws Exception
47      {
48          testXls();
49      }
50  }
```

Az xls tábla, mint adatforrás használatát a 3-9. Programlista mutatja. Készítsünk a 16. sorban megadott szerény, 4 soros táblánkhoz egy eddigiekhez tökéletesen hasonló jelentést. Ezt úgy biztosítjuk, hogy a 11. sorban is látottak szerint pont ugyanazt a riport template-et használjuk, amit a Java Bean-es példában is. A 18-25 sorok között *columnNames* és *columnIndexes* tömböket azért hoztuk létre, hogy

1. az egyes oszlopoknak nevet adjunk, hiszen a riport field-ek ezt igénylik,

2. megmondjuk, hogy az egyes nevek melyik sorszámú (nullával kezdve) oszlophoz tartoznak.

A 27-29 sorok szerint is tudnánk *JRXlsDataSource* adatforrást készíteni, de a példában most 31-33 sorok között létrejött *Workbook* objektumot (a neve: *wb*) használjuk erre. Ez az osztály a *JExcelAPI* (webhelye: `http://jexcelapi.sourceforge.net/`) jar könyvtárból származik, ami természetesen szintén része a JasperReports csomagnak, de külön is letölthető. A 35. sorban legyártjuk az excel adatforrást, majd a 37-38 sorokban beállítjuk az oszlopneveket és a nyelvet. Itt a *cast* azért szükséges, mert ezek a metódusok már implementáció függőek és mint ilyet nem lehet a *JRDataSource* felületen elérni őket. A 43. sorban ismert módon elkészítjük a *jrprint* fájlt.

CSV adatforrás

Tekintsük a 3-10. Programlista által tartalmazott *csv* fájlt! A most mutatott példa akkor is működik, ha az egyes értékeket idézőjel közé tesszük, például „Budapest”.

```
1  // 3-10. Programlista: orszagok.csv
2
3  Magyarország,Budapest,CE,1000
4  Ország 2,Főváros 1,CE,111
5  Ország 3,Berlin,CE,33333
6  Ország 4,Főváros 3,CE,2222
```

```
1   // 3-11. Programlista: TestOrszagokRiport.java
2
3   package org.cs.test;
4   ...
5   public class TestOrszagokRiport
6   {
7       static String jasperBeanFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-Bean.jasper";
8       static String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
9
10      public static void testCsv() throws Exception
11      {
12          String fileName = "/home/tanulas/jasperreport/pelda-1/orszagok.csv";
13
14          String[] columnNames = new String[]
15          {
16              "orszag", "fovaros", "foldrajzi_hely", "terulet"
```

```
17          };
18          int[] columnIndexes = new int[]
19          {
20              0, 1, 2, 3
21          };
22
23  //          InputStream is = JRLoader.getLocationInputStream(fileName);
24          JRDataSource jrds = new JRCsvDataSource(fileName);
25          ((JRCsvDataSource) jrds).setRecordDelimiter("\n");
26
27          ((JRCsvDataSource) jrds).setColumnNames(columnNames);
28  //          ((JRXlsDataSource)jrds).setLocale(new Locale("hu_HU"));
29
30          HashMap params = new HashMap();
31          params.put("pheaderTerulet", "Area (km2)");
32
33          JasperFillManager.fillReportToFile(jasperBeanFile, jrprintFile, params, jrds);
34
35      }
36
37      public static void main(String[] args) throws Exception
38      {
39          testCsv();
40      }
41  }
```

A 3-11. Programlista programja az exceles testvéréhez hasonlóan működik. Használhatnánk az *is InputStream*-et (23. sor) is, de most inkább a 24. sorban bemutatott módszerrel gyártottuk le a *JRCsvDataSource* adatforrást. Esetünkben a sorok (a record) és oszlopok elhatárolóinak megadása is lényeges. A 25. sorban azt definiáltuk, hogy 2 sort a *newline* karakter választ el egymástól. Az oszlopok elválasztó jelét a *setFieldDelimiter()* metódussal adhattuk volna meg, de most hagytuk, hogy az alapértelmezett vessző maradjon. A 33. sorban pedig elkészül a riportfájl.

JSON adatforrás

A *JSON*[3] az XML mellett manapság a másik elterjedt, hordozható adatformátum. Jelentősége akkor növekedett meg, amikor a WEB 2 rohamosan terjedni kezdett, hiszen leginkább a JavaScript biztosítja a böngészőben lévő működés dinamizmusát. A JasperReports a *Jackson* (webhelye: `http://jackson.codehaus.org/`) *JSON* kezelő könyvtárat használja. A JSON egy JavaScript objektum szöveges leírással való megadása (*Marshalling*), amit

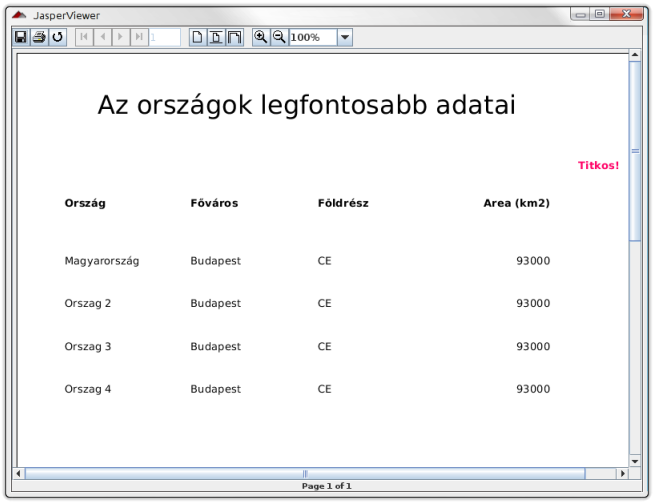
- a JavaScript elő tud állítani és
- belőle fel tudja építeni újra az objektumot.

A Jackson ugyanerre képes, de az objektum ekkor egy Java osztálybeli példány. Itt egy izgalmas lehetőség mutatkozik a *JavaScript↔Java* kommunikációra, amint ezt az olvasó bizonyára tudja vagy sejti. Javasoljuk elolvasni a wiki JSON leírását: `http://hu.wikipedia.org/wiki/JSON`. A Jackson használatával egy JSON szöveget könnyen át tudunk alakítani JavaBean, Map vagy XML formára, majd riportot készíthetünk vele, azonban a JasperReports ezt közvetlenül is el tudja végezni a *JsonDataSource* osztály használatával. Továbbfolytatva a hagyományt, most az országok adatait JSON formában is leírtuk, amit a 3-12. Programlista tartalmaz. Látható, hogy az *Orszagok* objektum *Items* mezője egy tömb, ami a mi ismerős sorainkat (rekordjainkat) tartalmazza.

[3] JSON=JavaScript Object Notation

```
// 3-12. Programlista: orszagok.json

{"Orszagok": {
  "Items": [
    {
      "orszag": "Magyarország",
      "fovaros": "Budapest",
      "foldrajzi_hely": "CE",
      "terulet": "93000"
    },
    {
      "orszag": "Orszag 2",
      "fovaros": "Budapest",
      "foldrajzi_hely": "CE",
      "terulet": "93000"
    },
    {
      "orszag": "Orszag 3",
      "fovaros": "Budapest",
      "foldrajzi_hely": "CE",
      "terulet": "93000"
    },
    {
      "orszag": "Orszag 4",
      "fovaros": "Budapest",
      "foldrajzi_hely": "CE",
      "terulet": "93000"
    }
  ]
}}
```



3.3. ábra: A JsonDataSource

A 3-13. Programlista ennek a feldolgozását, azaz a jelentés elkészítését mutatja, aminek az eredményét a 3.3. ábrán láthatjuk.

```
1  // 3-13. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4  ...
5  import net.sf.jasperreports.engine.util.JRLoader;
6  import net.sf.jasperreports.engine.data.JsonDataSource;
7  ...
8  public class TestOrszagokRiport
9  {
10     static String jasperXMLFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-XML.jasper";
11     static String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
12
13     public static void testJSON() throws Exception
14     {
15         InputStream is = JRLoader.getLocationInputStream("/home/tanulas/jasperreport/pelda-1/➥
               orszagok.json");
16
17         JRDataSource jrds = new JsonDataSource(is, "Orszagok.Items");
18         HashMap params = new HashMap();
19         params.put("pheaderTerulet", "Area (km2)");
20
21         JasperFillManager.fillReportToFile(jasperXMLFile, jrprintFile, params, jrds);
22     }
23
24     public static void main(String[] args) throws Exception
25     {
26         testJSON();
27     }
28 }
```

A JSON jrxml-ben lévő *field* – az XML adatforráshoz hasonlóan – is igényli a *fieldDescription* leírást, emiatt ugyanazt a jasper lefordított fájlt használjuk a példában. A 15. sorban egy *is InputStream*-et készítünk az *orszagok.json* tartalma alapján. A 17. sorban gyártjuk le a *jrds* adatforrás objektumot, ahol a konstruktor 2. paraméterében szintén megadtuk azt az útvonalat, ahogy a mezőinkhez el lehet jutni. Itt persze most az objektumoknál megszokott „.”-tal választottuk el a path elemeit: *Orszagok.Items*. Erre úgy is gondolhatunk, mint egy *select*-re, hiszen annyi sort ad vissza, amennyi eleme van az *Items* tömbnek, az oszlopok pedig a rekord mezői. A 21. sorban a már többször látott riportkészítési lépés történik.

Saját készítésű adatforrás

Mostanra sokféle adatforrást átnéztünk, talán elég benyomást és tapasztalatot szereztünk abban, hogy próbaként készítsünk egy sajátot is, ami a *JRDataSource* Java interface egy implementációjának megadását jelenti. Itt bármi olyan struktúra szóba jöhet, amit sorként és oszlopként lehet értelmezni. Azonban a lehetőségek még ennél is rugalmasabbak, mert ezeknek csak logikai kapcsolat szempontjából kell létezniük és csak arról kell gondoskodnunk, hogy a *next()* és *getFieldValue()* metódusok kiszámíthatóak legyenek. Például, ha létezik 1 adatbázis, ami az országok adatait tartalmazza, de nincs benne ez a 2 adat:

- Az utolsó év GDP értéke (*gdp*)
- A munkaképes lakosság száma (*workers*)

A zárójelbe a képzeletbeli jrxml field neveket tettük. Találtunk az Interneten 2 webservice-t, mindkettőnek az ország ISO kódját lehet átadni és a kívánt gdp vagy workers értéket visszaadja. Ekkor erre építve már készíthetnénk egy *JROkonomiaDataSource* class-t, amivel a szükséges riportot könnyedén elkészíthetnénk.

```
1  // 3-14. Programlista: JRMyExcelDatasource.java
2
3  package org.cs.test;
4
5  import java.io.File;
6  import java.io.IOException;
7  import java.math.BigDecimal;
8  import java.math.BigInteger;
9  import jxl.Cell;
10 import jxl.CellType;
11 import jxl.Sheet;
12 import jxl.Workbook;
13 import jxl.WorkbookSettings;
14 import net.sf.jasperreports.engine.JRDataSource;
15 import net.sf.jasperreports.engine.JRException;
16 import net.sf.jasperreports.engine.JRField;
17 import net.sf.jasperreports.engine.base.JRBaseField;
18
19 public class JRMyExcelDatasource implements JRDataSource
20 {
21     Workbook wb = null;
22     Sheet sheet = null;
23     int actualRow = 0;
24
25     public JRMyExcelDatasource(String fileName, int sheetNumber) throws Exception
26     {
27         actualRow = -1;
28         WorkbookSettings ws = new WorkbookSettings();
29         ws.setEncoding("Cp1252");
30         wb = Workbook.getWorkbook(new File(fileName), ws);
```

```
31          sheet = wb.getSheet(sheetNumber);
32      }
33
34      @Override
35      public boolean next() throws JRException
36      {
37          boolean existNext = false;
38          actualRow++;
39          Cell c = null;
40          try
41          {
42              c = sheet.getCell(actualRow, 0);
43          }
44          catch (Exception e)
45          {
46               existNext = false;
47               return existNext;
48          }
49
50          CellType ct = c.getType();
51
52          if (ct.equals(CellType.EMPTY))
53          {
54              existNext = false;
55              wb.close();
56          }
57          else
58          {
59              existNext = true;
60          }
61
62          return existNext;
63      }
64
65      @Override
66      public Object getFieldValue(JRField jrf) throws JRException
67      {
68          Object o = null;
69          String numberStr = jrf.getName().substring(1);
70          int col = Integer.parseInt(numberStr);
71
72          Cell c = sheet.getCell(col, actualRow);
73          String str = c.getContents();
74
75          // java.math.BigDecimal
76          if ( jrf.getValueClassName().equals( BigDecimal.class.getName() ) )
77          {
78              BigDecimal bd = new BigDecimal( str );
79              o = bd;
80          }
81          else
82          {
83              o = str;
84          }
85          return o;
86      }
87  }
```

A 3-14. Programlista célja csak az adatforrás implementáció megtanítása, ezért egy Excel adatforrás (*JRMyExcelDatasource*) megvalósítását mutatja be, ugyanazzal a java library-vel, amit a beépített adatforrás is használ. A 25-32 sorok közötti konstruktor az *xls* fájl nevét és a használni kívánt excel munkalap sorszámát kapja meg, ami alapján létrehoz egy *Workbook*

(neve: *wb*) és egy *Sheet* (*sheet*) példányt. A 35. sornál kezdődő *next()* pont úgy működik, ahogy elvárjuk tőle: a tábla következő sorára áll, amíg azok el nem fogynak. Most nézzük az értékek előállítását biztosító *getFieldValue()* metódust! Itt említenénk meg egy tervezési döntést: a jrxm-ben az egyes mezők szerkezete ilyen lehet, amikor az egyes oszlopokra hivatkozunk: *<betű><oszlop sorszám>*. Ez azt jelenti, hogy például a egy *$F{O2}* hivatkozás esetén, a *JRField* paraméter – más információk kíséretében – ezt a nevet fogja leszállítani a metódus számára. A 68-70 sorokban ez már elég lesz annak az adatnak a megszerzéséhez, hogy melyik sorszámú oszlop értéke kell, ezt a *col* változóba mentjük. Ezután a 72-73 sorban már Stringként tudjuk is a keresett értéket. A *JRField* az érték típusát is átadja, így például *BigDecimal* esetén még erre át kell konvertálnunk a Stringet. Szeretnénk kiemelni, hogy ez az adatforrás implementáció nem produktív céllal készült, annak nem minden ága van rendesen lekezelve, ugyanakkor a lehetséges érték típusok közül csak *String* és *BigDecimal* visszaadására képes, ami hiányos, hiszen a *JRField* a típus információt is átadja. A field pedig számít arra, hogy megfelelő típusú *Object* jött neki vissza. Persze, ha vigyázunk, hogy csak ezen 2 típust használjuk a jrxml-ben, akkor már egészen jól működik az adatforrás. Ezt látjuk a 3-15. Programlistán, ezeket a mezőket használtuk tesztelés során.

```
// 3-15. Programlista: Orszagok-XLS.jrxml

<jasperReport...
...
        <field name="C0" class="java.lang.➥
            String"/>
        <field name="C1" class="java.lang.➥
            String"/>
        <field name="C2" class="java.lang.➥
            String"/>
        <field name="C3" class="java.math.➥
            BigDecimal"/>
        <background>
...
</jasperReport>
```

```
1  // 3-16. Programlista: TestOrszagokRiport.java
2  ...
3  public class TestOrszagokRiport
4  {
5      static String jasperMyFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-MyXls.jasper";
6      static String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
7
8      public static void testMyJRDS() throws Exception
9      {
10         String fileName = "/home/tanulas/jasperreport/pelda-1/orszagok.xls";
11         JRDataSource jrds = new JRMyExcelDatasource(fileName, 0);
12
13         HashMap params = new HashMap();
14         params.put("pheaderTerulet", "Area (km2)");
15
16         JasperReport jr = (JasperReport)JRLoader.loadObject(new File(jasperMyFile));
17
18 //         JasperFillManager.fillReportToFile(jasperMyFile, jrprintFile, params, jrds);
19         JasperFillManager.fillReportToFile(jr, jrprintFile, params, jrds);
20     }
21
22     public static void main(String[] args) throws Exception
23     {
24         testMyJRDS();
25     }
26 }
```

A 3-16. Programlista a saját készítésű adatforrásunk használatát mutatja be, abban semmi újdonság nem található.

4. Hordozható dokumentum formátumok készítése

Az eddigiek során mindig a JasperReports *jrprint* natív dokumentum formátumot állítottuk elő. Ezzel nincsen semmi baj, kivéve azt, hogy használatához a jasper környezet szükséges. A végtermméknek nyilvánvalóan egy ismert dokumentum formátumnak kell lennie, amiket minden operációs rendszer alatt kezelni tudunk. Ebben a cikkben áttekintjük a szabványos, hordozható formátum előállításának lehetőségeit.

Rövid ismétlő áttekintés

Mostanra sok mindent megtanultunk, a továbblépés előtt érdemes egy kis összegző áttekintést készíteni. A riportok kinézeti tervét a *jrxml* fájl tartalmazza, aminek a memóriában felépített *JasperDesign* class felel meg. Az 1-3. Programlista még azt is megmutatta, hogy a jrxml-ből hogyan tudjuk ennek és a beállításainak a forráskódját előállítani. A *JasperReport* osztály a lefordított jrxml (azaz *JasperDesign*) objektumok osztálya, amiket a fájlrendszerbe *jasper* fájlokba tudunk perzisztálni. Az elkészült riportot a *JasperPrint* class reprezentálja, amit *jrprint* fájlba tudunk menteni. Az 1-4. Programlista *compile()* metódusa megmutatta, hogy egy *jrxml fájl→jasper fájl* fordítást milyen módon lehet programozottan elvégezni. A 4-1. Programlista tartalmaz egy új *compileToJasperReport()* nevű metódust, ahol *JasperCompileManager* class-tól azt kérjük, hogy jasper fájl helyett egy *JasperReport* objektumot adjon vissza. Megjegyezzük, hogy a *JRXmlLoader* osztály *load()* metódusa túlterhelt, a megjegyzésbe tett rész egy alternatíva, ha nem *InputStream*-ből, hanem közvetlenül a jrxml fájlból akarjuk a *jasperDesign* objektumot elkészíteni.

```
1  // 4-1. Programlista: JRHelper.java
2
3  package org.cs.test;
4  ...
5  public class JRHelper
6  {
7  ...
8     public JasperReport compileToJasperReport(String jrxmlFileName) throws Exception
9      {
10         JasperReport jr = null;
11         InputStream is = new FileInputStream( new File(jrxmlFileName) );
12 //         JasperDesign jasperDesign = JRXmlLoader.load( jrxmlFileName );
13         JasperDesign jasperDesign = JRXmlLoader.load( is );
14         JasperReport jasperReport = JasperCompileManager.compileReport(jasperDesign);
15
16         return jr;
17     }
18 ...
19 }
```

Az „üres” adatforrások pontnál már említettük a *JasperFillManager* azon képességeit, hogy milyen formátumokba képes a riportot elkészíteni. Amennyiben már van elkészített *jrprint* fájlunk, úgy azt a 4-2. Programlistán látott *getJasperPrint()* metódussal lehet *JasperPrint* osztálybeli objektumba betölteni.

```
1  // 4-2. Programlista: JRHelper.java
2
3  package org.cs.test;
4  ...
5  public class JRHelper
6  {
7  ...
8      public JasperPrint getJasperPrint(String jrprintFileName) throws JRException
9      {
10         File file = new File(jrprintFileName);
11         JasperPrint jp = (JasperPrint) JRLoader.loadObject(file);
12
13         return jp;
14     }
15 ...
16 }
```

A 4-3. Programlista *makeReportToJasperPrint()* metódusa a jasper fájl, paraméter map és egy adatforrás alapján közvetlenül a *JasperPrint* objektum legyártására képes. Eddig mindig a jrprint fájl készítést alkalmaztuk.

```
1  // 4-3. Programlista: JRHelper.java
2
3  package org.cs.test;
4  ...
5  public class JRHelper
6  {
7  ...
8      public JasperPrint makeReportToJasperPrint(String jasperFileName, HashMap params, ➥
           JRDataSource jrDs) throws JRException
9      {
10         JasperPrint jrp = null;
11         jrp = JasperFillManager.fillReport(jasperFileName, params, jrDs);
12
13         return jrp;
14     }
15 ...
16 }
```

Most nem lesz rá szükség, de a teljesség érdekében a 4-4. Programlista *makeReportToOutputStream()* metódusa azt is bemutatja, hogy a reportot miként tudnánk *OutputStream* objektumként is elkészíteni. A 3-16. Programlista 16. sora azt tanította meg, hogy egy már lefordított és elmentett jasper fájlból hogyan lehet a memóriába visszaállítani a *JasperReport* objektumot. Ezt a technikát most itt is alkalmaztuk.

```
1  // 4-4. Programlista: JRHelper.java
2
3  package org.cs.test;
4  ...
5  public class JRHelper
6  {
7  ...
8      public OutputStream makeReportToOutputStream(String jasperFileName, HashMap params, ➥
           JRDataSource jrDs) throws Exception
9      {
10         OutputStream os = null;
```

```
11
12 //         JasperReport jr = compileToJasperReport(jasperFileName);
13          JasperReport jr = (JasperReport)JRLoader.loadObject(new File(jasperFileName));
14
15          JasperFillManager.fillReportToStream(jr, os, params, jrDs);
16
17          return os;
18      }
19 ...
20 }
```

Ennyi előkészítés után vágjunk bele a különféle dokumentum formátumokba való exportálás lehetőségeinek megismerésébe!

A JRExporter interface

Amikor egy *JasperPrint* (vagy *jrprint*) → hordozható dokumentum exportálást végzünk, ezt mindig egy *JRExporter* interface-t biztosító objektummal tesszük meg, aminek a forráskódját a 4-5. Programlista mutatja. Az *exportReport()* metódus végzi el magát az exportálást, a többi csak annak működését befolyásolja. A *setParameter()* segítségével tetszőleges működési paramétereket adhatunk át, amire példákat majd a konkrét exportereknél fogunk látni.

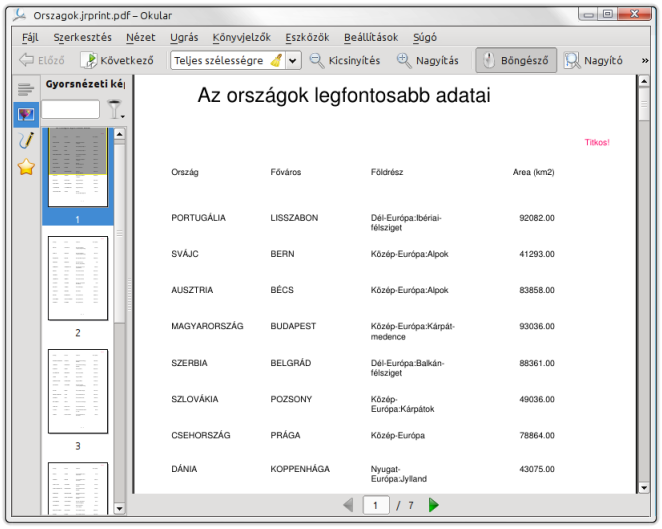
```
1  // 4-5. Programlista: JRExporter.java
2
3  package net.sf.jasperreports.engine;
4
5  import java.util.Map;
6  public interface JRExporter
7  {
8          /**
9           * Sets an export parameter for advanced customization of the export process. Parameters➥
                can be either
10          * common parameters or specialized ones, depending on the exporter type.
11          * @param parameter the parameter, selected from the static parameters defined by ➥
                JasperReports
12          * @param value the parameter value
13          * @see JRExporterParameter
14          */
15         public void setParameter(JRExporterParameter parameter, Object value);
16
17
18         /**
19          * Gets an export parameter.
20          */
21         public Object getParameter(JRExporterParameter parameter);
22
23
24         /**
25          * Sets export parameters from a specified map.
26          * @see JRExporter#setParameter(JRExporterParameter, Object)
27          */
28         public void setParameters(Map parameters);
29
30
31         /**
32          * Gets a map containing all export parameters.
33          */
34         public Map getParameters();
35
36
37         /**
```

```
38          * Actually starts the export process.
39          */
40         public void exportReport() throws JRException;
41
42 }
```

PDF dokumentumok

Első feladatunk legyen a PDF exporter megismerése, hiszen talán ez a formátum a leggyakoribb. Példáinkat a *JRExportingHelper* class tartalmazza, így nézzük meg a 4-6. Programlistát! A 17-28 sorok közötti *pdfExport()* metódus egy *jrprint* → *pdf* exportálást képes elvégezni. A példaként szereplő jrprint fájl a 2. cikk adatbázisos példájából származik. A 20. sorban a *JRHelper* class korábban ismertetett metódusával az *Orszagok.jrprint* fájl tartalmát visszaadjuk egy *JasperPrint* objektumként (a neve: *jp*). A 22. sorban létrehozzuk az exporter objektumot, amit a rákövetkező 3 sorban rendre ezekkel a paraméterekkel látjuk el:

- A *jp JasperPrint* objektum
- A legyártandó PDF fájl neve
- A karakterkódolás



4.1. ábra: Exportált PDF dokumentum

Végül a 27. sorban meghívjuk az *exportReport()* metódust, ami létrehozza a riport PDF formátumát. A 30-36 sorok között a most elkészített metódust le is teszteljük, aminek az eredményét a 4.1. ábrán láthatjuk.

```
1  // 4-6. Programlista: JRExportingHelper.java
2
3  package org.cs.test;
4
5  import java.io.File;
6  import net.sf.jasperreports.engine.JRException;
7  import net.sf.jasperreports.engine.JRExporter;
8  import net.sf.jasperreports.engine.JRExporterParameter;
9  import net.sf.jasperreports.engine.JasperExportManager;
10 import net.sf.jasperreports.engine.JasperPrint;
11 import net.sf.jasperreports.engine.util.JRLoader;
12 ...
13 import net.sf.jasperreports.engine.export.JRPdfExporter;
14
15 public class JRExportingHelper
16 {
17    public void pdfExport(String jrprintFileName) throws JRException
18     {
19         JRHelper jh = new JRHelper();
20         JasperPrint jp = jh.getJasperPrint(jrprintFileName);
21
22         JRExporter exporter = new JRPdfExporter();
23         exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
```

```
24          exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".pdf");
25          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
26
27          exporter.exportReport();
28      }
29
30      public static void main(String[] args) throws JRException
31      {
32          String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
33
34          JRExportingHelper jreh = new JRExportingHelper();
35          jreh.pdfExport(jrprintFile);
36      }
37  }
```

Egyszerű text dokumentumok

A *textExport()* metódust és annak a használatát a 4-7. Programlista foglalja magába. Az elv természetesen hasonló a PDF-éhez, bár ahogy láthatjuk, most több vezérlő paramétert is megadtunk.

```
1   // 4-7. Programlista: JRExportingHelper.java
2
3   package org.cs.test;
4
5   import java.io.File;
6   import net.sf.jasperreports.engine.JRException;
7   import net.sf.jasperreports.engine.JRExporter;
8   import net.sf.jasperreports.engine.JRExporterParameter;
9   import net.sf.jasperreports.engine.JasperExportManager;
10  import net.sf.jasperreports.engine.JasperPrint;
11  import net.sf.jasperreports.engine.util.JRLoader;
12  ...
13  import net.sf.jasperreports.engine.export.JRTextExporter;
14  import net.sf.jasperreports.engine.export.JRTextExporterParameter;
15
16  public class JRExportingHelper
17  {
18      public void textExport(String jrprintFileName) throws JRException
19      {
20          JasperPrint jp = getJasperPrint(jrprintFileName);
21          JRTextExporter exporter = new JRTextExporter();
22
23          exporter.setParameter(JRTextExporterParameter.BETWEEN_PAGES_TEXT ,"\f");
24          exporter.setParameter(JRTextExporterParameter.PAGE_HEIGHT       , new Float(798));
25          exporter.setParameter(JRTextExporterParameter.PAGE_WIDTH        , new Float(581));
26          exporter.setParameter(JRTextExporterParameter.CHARACTER_WIDTH   , new Float(  7));
27          exporter.setParameter(JRTextExporterParameter.CHARACTER_HEIGHT  , new Float( 14));
28          exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
29          exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".txt");
30  //          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
31          exporter.exportReport();
32      }
33
34      public static void main(String[] args) throws JRException
35      {
36          String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
37
38          JRExportingHelper jreh = new JRExportingHelper();
39          jreh.textExport(jrprintFile);
40      }
41  }
```

Az eredmény egy részletét a *Orszagok.jrprint.txt* nevű szöveg mutatja.

```
Orszagok.jrprint.txt


            Az országok legfontosabb adatai




                                                                    Titkos!


       Ország              Főváros             Földrész                Area (km2)

       PORTUGÁLIA          LISSZABON           Dél-Európa:Ibé          92082.00
                                               riai-félsziget

       SVÁJC               BERN                Közép-Európa:A          41293.00
                                               lpok

       AUSZTRIA            BÉCS                Közép-Európa:A          83858.00
                                               lpok

       MAGYARORSZÁG        BUDAPEST            Közép-Európa:K          93036.00
                                               árpát-medence

       SZERBIA             BELGRÁD             Dél-Európa:Bal          88361.00
                                               kán-félsziget

       SZLOVÁKIA           POZSONY             Közép-Európa:K          49036.00
                                               árpátok
...
       MOLDOVA             CHISINAU            Kelet-Európa            33700.00

       TAJVAN              TAJPEJ              Ázsia:Távol-Ke          36000.00
                                               let




                                      6 / 7

...
```

MS Word dokumentumok

Az MS Word dokumentum előállítását a 4-8. Programlista kódja valósítja meg.

```
// 4-8. Programlista: JRExportingHelper.java

package org.cs.test;

import java.io.File;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JRExporter;
import net.sf.jasperreports.engine.JRExporterParameter;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperPrint;
```

```
11 import net.sf.jasperreports.engine.util.JRLoader;
12 ...
13 import net.sf.jasperreports.engine.export.ooxml.JRDocxExporter;
14
15 public class JRExportingHelper
16 {
17     public void docExport(String jrprintFileName) throws JRException
18     {
19         JasperPrint jp = getJasperPrint(jrprintFileName);
20
21         JRExporter exporter = new JRDocxExporter();
22         exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
23         exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".doc");
24 //          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
25
26         exporter.exportReport();
27     }
28
29     public static void main(String[] args) throws JRException
30     {
31         String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
32
33         JRExportingHelper jreh = new JRExportingHelper();
34         jreh.docExport(jrprintFile);
35     }
36 }
```

MS Excel dokumentumok

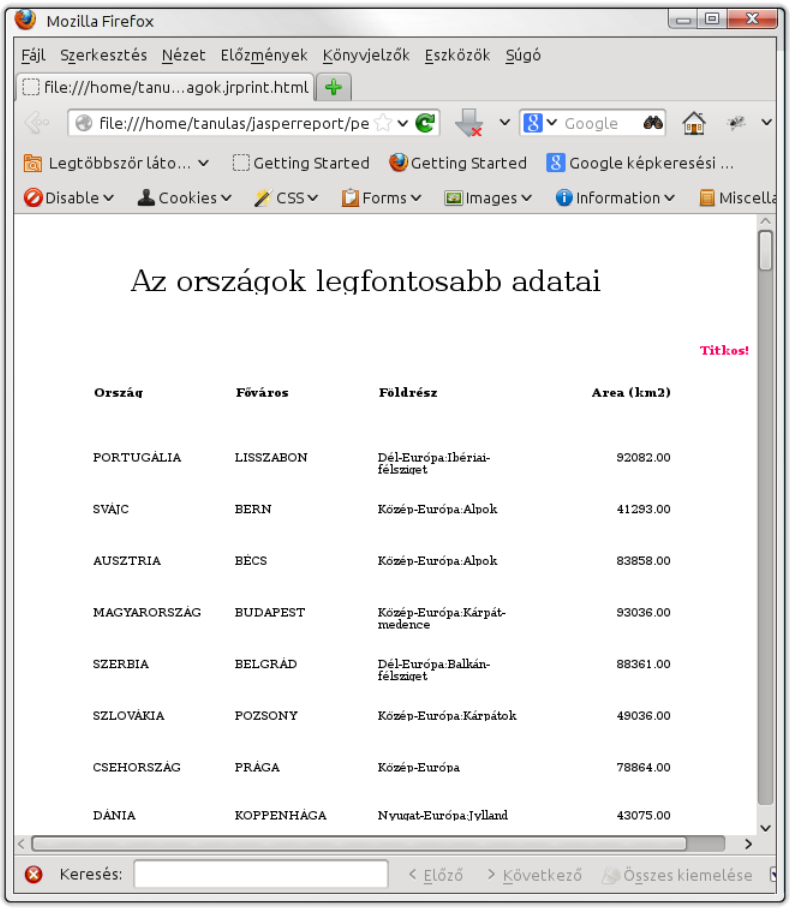
Az MS Word dokumentum előállítását a 4-9. Programlista kódja valósítja meg.

```
1  // 4-9. Programlista: JRExportingHelper.java
2
3  package org.cs.test;
4
5  import java.io.File;
6  import net.sf.jasperreports.engine.JRException;
7  import net.sf.jasperreports.engine.JRExporter;
8  import net.sf.jasperreports.engine.JRExporterParameter;
9  import net.sf.jasperreports.engine.JasperExportManager;
10 import net.sf.jasperreports.engine.JasperPrint;
11 import net.sf.jasperreports.engine.util.JRLoader;
12 ...
13 import net.sf.jasperreports.engine.export.JRXlsExporter;
14
15 public class JRExportingHelper
16 {
17     public void xlsExport(String jrprintFileName) throws JRException
18     {
19         JasperPrint jp = getJasperPrint(jrprintFileName);
20
21         JRExporter exporter = new JRXlsExporter();
22         exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
23         exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".xls");
24
25 //          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
26
27         exporter.exportReport();
28     }
29
30     public static void main(String[] args) throws JRException
31     {
32         String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
```

```
33
34          JRExportingHelper jreh = new JRExportingHelper();
35          jreh.xlsExport(jrprintFile);
36      }
37  }
```

HTML dokumentumok

HTML exporter használata is hasonló, a *htmlExport()* metódust és annak tesztjét a 4-10. Programlista tartalmazza, aminek eredményét a 4.2. ábrán meg is mutattuk.



4.2. ábra: Exportált HTML dokumentum

```
1  // 4-10. Programlista: JRExportingHelper.java
2
3  package org.cs.test;
4
5  import java.io.File;
6  import net.sf.jasperreports.engine.JRException;
7  import net.sf.jasperreports.engine.JRExporter;
8  import net.sf.jasperreports.engine.JRExporterParameter;
9  import net.sf.jasperreports.engine.JasperExportManager;
```

```
10 import net.sf.jasperreports.engine.JasperPrint;
11 import net.sf.jasperreports.engine.util.JRLoader;
12 ...
13 import net.sf.jasperreports.engine.export.JRHtmlExporter;
14
15 public class JRExportingHelper
16 {
17     public void htmlExport(String jrprintFileName) throws JRException
18     {
19         JasperPrint jp = getJasperPrint(jrprintFileName);
20
21         JRExporter exporter = new JRHtmlExporter();
22         exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
23         exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".html");
24 //          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
25
26         exporter.exportReport();
27     }
28
29     public static void main(String[] args) throws JRException
30     {
31         String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
32
33         JRExportingHelper jreh = new JRExportingHelper();
34         jreh.htmlExport(jrprintFile);
35     }
36 }
```

OpenDocument táblázat

Az OpenDocument táblázat dokumentum előállítását a 4-11. Programlista kódja valósítja meg.

```
1  // 4-11. Programlista: JRExportingHelper.java
2
3  package org.cs.test;
4
5  import java.io.File;
6  import net.sf.jasperreports.engine.JRException;
7  import net.sf.jasperreports.engine.JRExporter;
8  import net.sf.jasperreports.engine.JRExporterParameter;
9  import net.sf.jasperreports.engine.JasperExportManager;
10 import net.sf.jasperreports.engine.JasperPrint;
11 import net.sf.jasperreports.engine.util.JRLoader;
12 ...
13 import net.sf.jasperreports.engine.export.oasis.JROdsExporter;
14
15 public class JRExportingHelper
16 {
17     public void odsExport(String jrprintFileName) throws JRException
18     {
19         JasperPrint jp = getJasperPrint(jrprintFileName);
20
21         JRExporter exporter = new JROdsExporter();
22         exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
23         exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".ods");
24 //          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
25
26         exporter.exportReport();
27     }
28
29     public static void main(String[] args) throws JRException
30     {
31         String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
```

```
32
33          JRExportingHelper jreh = new JRExportingHelper();
34          jreh.odsExport(jrprintFile);
35      }
36  }
```

OpenDocument szöveg

Az OpenDocument szöveges dokumentum előállítását a 4-12. Programlista kódja valósítja meg.

```
1   // 4-12. Programlista: JRExportingHelper.java
2
3   package org.cs.test;
4
5   import java.io.File;
6   import net.sf.jasperreports.engine.JRException;
7   import net.sf.jasperreports.engine.JRExporter;
8   import net.sf.jasperreports.engine.JRExporterParameter;
9   import net.sf.jasperreports.engine.JasperExportManager;
10  import net.sf.jasperreports.engine.JasperPrint;
11  import net.sf.jasperreports.engine.util.JRLoader;
12  ...
13  import net.sf.jasperreports.engine.export.oasis.JROdtExporter;
14
15  public class JRExportingHelper
16  {
17      public void odtExport(String jrprintFileName) throws JRException
18      {
19          JasperPrint jp = getJasperPrint(jrprintFileName);
20
21          JRExporter exporter = new JROdtExporter();
22          exporter.setParameter(JRExporterParameter.JASPER_PRINT, jp);
23          exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME, jrprintFileName+".odt");
24  //          exporter.setParameter(JRExporterParameter.CHARACTER_ENCODING, "UTF8");
25
26          exporter.exportReport();
27      }
28
29      public static void main(String[] args) throws JRException
30      {
31          String jrprintFile = "/home/tanulas/jasperreport/pelda-1/Orszagok.jrprint";
32
33          JRExportingHelper jreh = new JRExportingHelper();
34          jreh.odtExport(jrprintFile);
35      }
36  }
```

A JasperExportManager class

A fenti dokumentum exportálási lehetőségekhez a JasperReports csomag tartalmaz egy homlokzat osztályt, ez a *JasperExportManager*. Ebben statikus metódusok biztosítják az ismert formátumú dokumentumok előállítását. Például a PDF előállítást így is kérhettük volna:

```
JasperExportManager.exportReportToPdfFile(jp,
    jrprintFileName+".pdf");
```

5. Többnyelvű riportok készítése

Sok esetben elengedhetetlen, hogy jelentéseink többnyelvűek legyenek. Az alkalmazásoknál már megszoktuk, hogy egyetlen kattintással váltunk a nyelvek között, ami azt is jelenti, hogy ezt riportjainknak is követniük kell. Nézzük meg ennek a megvalósítását JasperReports-ban!

A Java nyelv már a kezdeteknél támogatja a többnyelvűséget, ami alapvetően 3 komponensből áll:

- A *ResourceBundle* class és a szövegeket leíró property fájlok, ahol kulcs/érték párokat használunk.
- A *Locale* class, ami egy ország és nyelv együttesének értékét tárolja.
- A különféle szöveges formátumok kultúrafüggő megjelentése (számok, pénznem, dátum).

Nyelvfüggő szövegek

Az országok riportnak most elkészítjük a nyelvfüggő változatát, ez most annyit fog jelenteni, hogy az adott nyelvnek megfelelő szövegek jelennek majd meg a keletkezett dokumentumban. Három nyelvre készítjük fel a riportunkat: német, angol és magyar. Ennek megfelelően az egyes szövegelemeket egy-egy szöveges fájlban tároljuk el, amik rendre ezeket a fájl neveket kapják most:

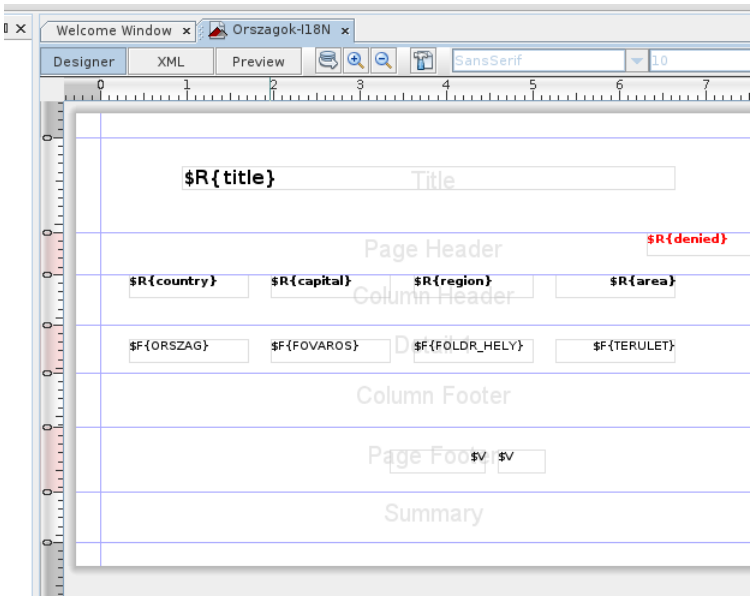
- orszagokText_de.properties
- orszagokText_de.properties
- orszagokText_hu.properties

Figyeljük meg, hogy az egyes nevek csak a *_hu*, *_de* és *_en* betűkben különböznek egymástól, az „_” jel után az ország 2 karakteres ISO kódja következik. Az egyes fájlok tartalma pedig az alábbi:

```
// orszagokText_hu.properties
title=Az orsz\u00e1gok legfontosabb adatai
denied=Tilos
country=Orsz\u00e1g
capital=F\u0151v\u00e1ros
region=F\u00f6ldrajzi hely
area=Ter\u00fclet (km2)

// orszagokText_en.properties
title=The most important data of countries
denied=Top Secret
country=Country
capital=Capital
region=Region
area=Area (km2)

// orszagokText_de.properties
title=In den meisten L\u00e4ndern, Daten
denied=Nicht
country=Land
capital=Hauptstadt
region=Region
area=Bereich (km2)
```



5.1. ábra: Többnyelvű riport terv (I18N)

Az 5.1. ábra az *iReport*-ban áttervezett riportot mutatja, ami nagyrészt megegyezik az *Orszagok.jrxml* template-tel, az eddigiekhez képest eltérés csak a fájl első részében van, amely részletet teljes egészében tartalmazza az 5-1. Programlista (*Orszagok-I18N.jrxml*).

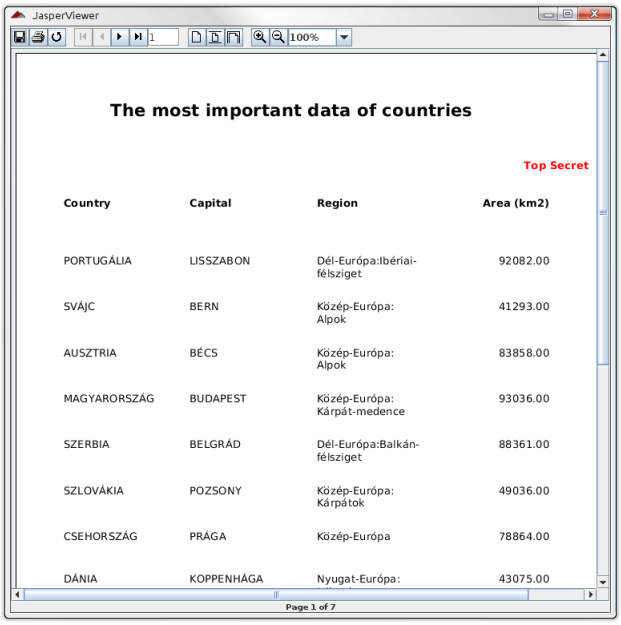
```
5-1. Programlista: Orszagok-I18N.jrxml

<?xml version="1.0" encoding="UTF-8"?>
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-I18N" language="groovy" pageWidth="595" pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20" resourceBundle="orszagokText" uuid="9a1dbe76-1ae5-4418-9e43-e56f6ed98193">
    <property name="ireport.zoom" value="1.0"/>
    <property name="ireport.x" value="0"/>
    <property name="ireport.y" value="0"/>
    <parameter name="pheaderTerulet" class="java.lang.String"/>
    <queryString language="SQL">
        <![CDATA[select * from orszagok]]>
    </queryString>
    <field name="ORSZAG" class="java.lang.String"/>
    <field name="FOVAROS" class="java.lang.String"/>
    <field name="FOLDR_HELY" class="java.lang.String"/>
    <field name="TERULET" class="java.math.BigDecimal"/>
    <field name="ALLAMFORMA" class="java.lang.String"/>
    <field name="NEPESSEG" class="java.lang.Integer"/>
    <field name="NEP_FOVAROS" class="java.lang.Integer"/>
    <field name="AUTOJEL" class="java.lang.String"/>
    <field name="COUNTRY" class="java.lang.String"/>
    <field name="CAPITAL" class="java.lang.String"/>
    <field name="PENZNEM" class="java.lang.String"/>
    <field name="PENZJEL" class="java.lang.String"/>
    <field name="VALTOPENZ" class="java.lang.String"/>
    <field name="TELEFON" class="java.lang.Integer"/>
    <field name="GDP" class="java.lang.Integer"/>
    <field name="KAT" class="java.lang.Integer"/>
    <background>
        <band splitType="Stretch"/>
    </background>
    <title>
        <band height="79" splitType="Stretch">
            <textField>
                <reportElement uuid="69af5ddd-3722-48b7-8620-5680547195ea" x="68" y="24" width="411" height="20"/>
                <textElement>
                    <font size="16" isBold="true"/>
                </textElement>
                <textFieldExpression><![CDATA[$R{title}]]></textFieldExpression>
            </textField>
        </band>
    </title>
    <pageHeader>
        <band height="35" splitType="Stretch">
            <textField>
                <reportElement uuid="e330b4e8-99b2-4281-8260-3c473d366492" x="455" y="0" width="100" height="20" forecolor="#FF0000"/>
                <textElement>
                    <font isBold="true"/>
                </textElement>
                <textFieldExpression><![CDATA[$R{denied}]]></textFieldExpression>
            </textField>
        </band>
    </pageHeader>
    <columnHeader>
        <band height="42" splitType="Stretch">
            <textField>
                <reportElement uuid="77010e10-4cfa-4847-9047-e966ee16baec" x="24" y="0" width="100" height="20"/>
                <textElement>
                    <font isBold="true"/>
                </textElement>
                <textFieldExpression><![CDATA[$R{country}]]></textFieldExpression>
            </textField>
            <textField>
                <reportElement uuid="e68fcc3a-b8f5-411e-959c-83c8459060bf" x="142" y="0" width="100" height="20"/>
                <textElement>
                    <font isBold="true"/>
                </textElement>
                <textFieldExpression><![CDATA[$R{capital}]]></textFieldExpression>
            </textField>
            <textField>
                <reportElement uuid="33381688-a5c5-4256-a309-3576b7bed762" x="261" y="0" width="100" height="20"/>
                <textElement>
```

```
72                                      <font isBold="true"/>
73                                  </textElement>
74                                  <textFieldExpression><![CDATA[$R{region}]]></textFieldExpression>
75                              </textField>
76                              <textField>
77                                  <reportElement uuid="7cbb7ca1-8dca-4eb5-8bbc-c20d9013f4d2" x="379" y="0" width="➥
                                        100" height="20"/>
78                                  <textElement textAlignment="Right">
79                                      <font isBold="true"/>
80                                  </textElement>
81                                  <textFieldExpression><![CDATA[$R{area}]]></textFieldExpression>
82                              </textField>
83                      </band>
84              </columnHeader>
85  ...
```

Az első fontos különbség a *jasperReport* tagben megadott *resourceBundle="orszagokText"* attribútum. Ez arra utasít, hogy a nyelvfüggő szövegelemeket az ilyen perfixű property fájlokból vegye ki, aminek egyébként a *CLASSPATH*-on kell lennie. A másik változtatás a *title*, *pageHeader* és *columnHeader* címkéknél található *textElement*-eknél van (a *staticText*-eket visszamozgattuk). Vegyük azt is észre, hogy a *textFieldExpression* tag *$R{...}* hivatkozásokat tartalmaz, ahol az *R* a resource bundle-ra való utalás. Például a *$R{capital}* azt jelenti, hogy az a szöveg kell most ide, ami a megfelelő property fájl *capital* kulcsához van írva. Az éppen aktuálisan használt *Locale* objektum választja majd ki, hogy a most lehetséges 3 nyelvfüggő fájl melyikéből emeljük ki. Az 5-2. Programlista pont azt mutatja be, hogy mindezt hogyan szabályozzuk, így nézzük meg!

```
1   // 5-2. Programlista: TestOrszagokRiport.java
2
3   package org.cs.test;
4   ...
5   public class TestOrszagokRiport
6   {
7      static String jasperI18NFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-I18N.jasper";
8
9       public static void testI18NRDBMS() throws Exception
10      {
11          String sql = "select * from orszagok where terulet < ?";
12
13          Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", ➥
                "");
14          PreparedStatement stmt = conn.prepareStatement(sql);
15          stmt.setDouble(1, 100000.00);
16          ResultSet rs = stmt.executeQuery();
17          JRDataSource jrds = new JRResultSetDataSource(rs);
18          JRHelper jh = new JRHelper();
19          HashMap params = new HashMap();
20          params.put("pheaderTerulet", "Area (km2)");
21
22          params.put(JRParameter.REPORT_LOCALE, Locale.ENGLISH);
23          jh.makeReport(jasperI18NFile, params, jrds);
24
25          rs.close();
26          stmt.close();
27          conn.close();
28      }
29
30      public static void main(String[] args) throws Exception
31      {
32          testI18NRDBMS();
33      }
34  }
```



5.2. ábra: Többnyelvű riport (I18N)

A 7. sor az a jasper fájl, amit frissen fordítottunk a megváltoztatott új jrxml miatt. A riportunk most ismét a HSQLDB adatbázisból fogja venni az adatokat az ismert 11. sorban megadott SQL select szerint. Ezután a program egészen a 20. sorig ismerős dolgokat tartalmaz. A 22. sor az, ahol átadjuk a riportnak, hogy most éppen melyik lokalitás nyelvén szeretnénk a riportot elkészíteni. Ehhez a *JRParameter.REPORT_LOCALE* riport paramétert állítjuk *Locale.ENGLISH* értékre, azaz jelentésünk angol lesz. A 23. sorban elkészült riportot láthatjuk az 5.2. ábrán.

Lokális adatformátumok

A ResourceBundle class

Már említettük, hogy a Java osztálykönyvtár része a *ResourceBundle* class. Valójában az előző pontban megismert *$R{...}* hivatkozás is ezt alkalmazza a háttérben. Használatát az 5-3. Programlistán mutatott kódrészlet mutatja be. A jól ismert *Locale* class azt az információt tárolja, hogy melyik ország, melyik nyelvére gondolunk. Most a német értéket tettük a *locale* változóba. A 4. sorban a *myResources* nevű változó hivatkozást kap egy olyan *ResourceBundle* objektumra, ami a már ismert *orszagokText* prefixű *properties* fájlokban tárolja a szöveg elemeket és a német változaton (2. paraméter) dolgozik. A következő sorokban a használatát és annak képernyőn megjelenő eredményét láthatjuk.

```
1 // 5-3. Programlista: ResourceBundle
2
3 Locale locale = Locale.GERMANY;
4 ResourceBundle myResources = ResourceBundle.getBundle("orszagokText", locale);
5 System.out.println( myResources.getString("capital") );
6
7 Futási eredmény:
8 Hauptstadt
```

A nyelvfüggő dátumformátum

Első lépésként a szövegrészleteket tartalmazó *orszagokText_xxx.properties* fájlokat kiegészítettük a megfelelő területre jellemző *dateFormat* kulccsal, ami a nyelvfüggő dátumformátum maszk (yyyy=év, MM=hónap, dd=nap). Az 5-3. Programlistán mutatott módon ehhez hozzáférhetünk, így az értékét paraméterként átadhatjuk a riport számára, aminek a neve a példánkban *patternOfDate* lesz majd.

```
Német:
dateFormat=MM/dd/yyyy

Angol:
dateFormat=dd/MM/yyyy

Magyar:
dateFormat=yyyy/MM/dd
```

Előtte azonban tekintsük át a *Orszagok-I18N.jrxml* terven tett változtatásokat, hiszen most ki kell tennünk egy dátumot tartalmazó *textField* komponenst, aminek *textFieldExpression* részéhez definiálnunk kell a *patternOfDate* paramétert is. Az 5-4. Programlistán csak a változtatásokat láthatjuk. A 6. sorban találjuk az új paraméterünket, ez fogja majd átvenni a megfelelő dátumformázó string értékét. A dátumot a riport *title* részébe helyeztük el (23-27 sorok). A 26. sor *textFieldExpression* kifejezése mutatja a megadható kifejezést, ami a pillanatnyi dátumot fogja megjeleníteni a *$P{patternOfDate}* formátum szerint.

```
1  5-4. Programlista: Orszagok-I18N.jrxml
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport...
5  ...
6      <parameter name="patternOfDate" class="java.lang.String"/>
7
8          <queryString language="SQL">
9                  <![CDATA[select * from orszagok]]>
10         </queryString>
11
12         <field name="ORSZAG" class="java.lang.String"/>
13 ...
14         <title>
15                 <band height="93" splitType="Stretch">
16                         <textField>
17                                 <reportElement uuid="69af5ddd-3722-48b7-8620-5680547195ea" x="68" y="24" width="➥
                                       411" height="20"/>
18                                 <textElement>
19                                         <font size="16" isBold="true"/>
20                                 </textElement>
21                                 <textFieldExpression><![CDATA[$R{title}]]></textFieldExpression>
22                         </textField>
23                         <textField>
24                                 <reportElement uuid="e740452a-0179-45f5-87c5-c1533b560fd3" x="432" y="60" width=➥
                                       "100" height="20"/>
25                                 <textElement/>
26                                 <textFieldExpression><![CDATA[(new SimpleDateFormat($P{patternOfDate})).format(➥
                                       Calendar.getInstance().getTime())]]></textFieldExpression>
27                         </textField>
28                 </band>
29         </title>
30 ...
```

A példa befejezéséhez még azt kell megadnunk, hogy a jelentéskészítő program milyen szerkezettel épül fel, ezt tartalmazza az 5-5. Programlista. A 18. sorig ismert a program. A 22-24 sorokban a már bemutatott módon megszerezzük a magyar formátum stringet, amit a 27. sorban adunk át paraméterként. A 28. sor közli a jelentéskészítő generátorral, hogy most a *locale* változó tartalma szerint (azaz magyar) vegye a *$R{...}* értékeket. A program többi része ismét ismerős az előző példákból, így nem részletezzük. Csak érdekességként tekintsünk rá az 5-6. Programlista programtöredékére is, ahol az adatforrásból érkező *HireDate* mezőre alkalmazott formázási lehetőségre adtunk példát.

```
1  // 5-5. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4  ...
5  public class TestOrszagokRiport
6  {
7     static String jasperI18NFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-I18N.jasper";
8
9      public static void testI18NRDBMS() throws Exception
10     {
11        String sql = "select * from orszagok where terulet < ?";
12
```

```
13          Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", "➥
                ");
14          PreparedStatement stmt = conn.prepareStatement(sql);
15          stmt.setDouble(1, 100000.00);
16          ResultSet rs = stmt.executeQuery();
17          JRDataSource jrds = new JRResultSetDataSource(rs);
18          JRHelper jh = new JRHelper();
19
20          //Locale locale = Locale.ENGLISH;
21          //Locale locale = Locale.GERMAN;
22          Locale locale = new Locale("hu", "HU");
23          ResourceBundle myResources = ResourceBundle.getBundle("orszagokText", locale);
24          String dateFormat = myResources.getString("dateFormat");
25
26          HashMap params = new HashMap();
27          params.put("patternOfDate", dateFormat);
28          params.put(JRParameter.REPORT_LOCALE, locale);
29
30          jh.makeReport(jasperI18NFile, params, jrds);
31
32           rs.close();
33           stmt.close();
34           conn.close();
35      }
36
37      public static void main(String[] args) throws Exception
38      {
39          testI18NRDBMS();
40      }
41  }
```

```
1  // 5-6. Programlista
2
3  $F{FirstName} + "␣" + $F{LastName} + "␣was␣hired␣on␣" +
4  (new SimpleDateFormat($P{patternOfDate})).format($F{HireDate}) + "."
5  </textFieldExpression>
```

A nyelvfüggő adatformátumok

Érdemes megismerni a Java egyszerű, lokalitás függő szövegformázó lehetőségeit, mert nagy hasznát vehetjük akkor, amikor ezeket felhasználjuk riportjaink készítése során. Az 5-7. Programlista ismét a dátumok formázására mutat példát, de a kinézet mintáját nem mi adjuk meg, hanem a *DateFormat* class statikus *getDateInstance()* gyártómetódus 2. paraméterében lévő *locale* változó segítségével biztosítjuk a helyes formátum szerinti működést. A példában csak a *MEDIUM* és *FULL* szerinti formátumokat teszteltük, de jó pár további, előre programozott eset érhető még el. A példa végén a program futási eredményét is láthatjuk.

```
1  // 5-7. Programlista: MessageFormatTest.java
2
3  package org.cs.test;
4
5  import java.text.DateFormat;
6  ...
7  import java.util.Locale;
8
9  public class MessageFormatTest
10 {
11    public static void testDate()
```

```
12      {
13          Locale locale = Locale.GERMAN;
14
15          DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM, locale);
16          System.out.println( df.format(new Date() ));
17
18          df = DateFormat.getDateInstance(DateFormat.FULL, locale);
19          System.out.println( df.format(new Date() ));
20      }
21
22      public static void main(String[] args)
23      {
24          testDate();
25      }
26  }
27
28  Futási kép:
29  19.01.2013
30  Samstag, 19. Januar 2013
```

A következő példánkban (5-8. Programlista) a *NumberFormat* osztály segítségével a számok országfüggő formázását tudjuk úgy automatizálni, hogy *getIntegerInstance()* segítségével lekérjük az adott lokalitás szokásos formázási szokását. A *testInteger()* metódusban ezt megtesszük a német, angol és magyar esetre egyaránt, majd a futási eredményt is láthatjuk.

```
1   // 5-8. Programlista: MessageFormatTest.java
2
3   package org.cs.test;
4
5   import java.text.DateFormat;
6   ...
7   import java.util.Locale;
8
9   public class MessageFormatTest
10  {
11      public static void testInteger()
12      {
13          Locale locale = Locale.GERMAN;
14          NumberFormat nf = NumberFormat.getIntegerInstance(locale);
15          System.out.println( nf.format( 1234567 ) );
16
17          locale = Locale.ENGLISH;
18          nf = NumberFormat.getIntegerInstance(locale);
19          System.out.println( nf.format( 1234567 ) );
20
21          locale = new Locale("hu", "HU");
22          nf = NumberFormat.getIntegerInstance(locale);
23          System.out.println( nf.format( 1234567 ) );
24      }
25
26      public static void main(String[] args)
27      {
28          testInteger();
29      }
30  }
31
32  Futási kép:
33  1.234.567
34  1,234,567
35  1 234 567
```

A *testDecimal()* tesztmetódus (5-9. Programlista) abban különbözik az előző példától, hogy itt a szám formázására mi adhatjuk meg a formátum maszkot, amihez a *DecimalFormat* osztályt tudjuk segítségül hívni. A *main()* metódusban 3 formázási esetet is kipróbáltunk a *1234567* számra, aminek eredményét is közöltük.

```
1  // 5-9. Programlista: MessageFormatTest.java
2
3  package org.cs.test;
4
5  import java.text.DateFormat;
6  ...
7  import java.util.Locale;
8
9  public class MessageFormatTest
10 {
11     public static void testDecimal(String pattern)
12     {
13         NumberFormat nf = new DecimalFormat(pattern);
14         System.out.println( nf.format( 1234567 ) );
15     }
16
17     public static void main(String[] args)
18     {
19         testDecimal("#,##0");
20         testDecimal("#,##0.00");
21         testDecimal("\u00A4#,##0.00");
22     }
23 }
24
25 Futási kép:
26 1 234 567
27 1 234 567,00
28 Ft1 234 567,00
```

Amennyiben pénzügyi számokat szeretnénk formázni, úgy abban a *testCurrency()* (5-10. Programlista) megértése siet a segítségünkre.

```
1  // 5-10. Programlista: MessageFormatTest.java
2
3  package org.cs.test;
4
5  import java.text.DateFormat;
6  ...
7  import java.util.Locale;
8
9  public class MessageFormatTest
10 {
11     public static void testCurrency()
12     {
13         Locale locale = Locale.GERMAN;
14
15         NumberFormat nf = NumberFormat.getCurrencyInstance(locale);
16         System.out.println( nf.format( 1234567 ) );
17
18         locale = Locale.ENGLISH;
19
20         nf = NumberFormat.getCurrencyInstance(locale);
21         System.out.println( nf.format( 1234567 ) );
22
23         locale = new Locale("hu", "HU");
24         nf = NumberFormat.getCurrencyInstance(locale);
25         System.out.println( nf.format( 1234567 ) );
```

```
26      }
27
28      public static void main(String[] args)
29      {
30          testCurrency();
31      }
32  }
33
34  Futási kép:
35  1.234.567,00
36  1,234,567.00
37  1 234 567 Ft
```

Az 5-11. Programlista *testTime()* metódusa az óra/perc/másodperc információt tudja megjeleníteni nyelvfüggő módon, a már megismert *DateFormat* class használatával megszerezve a formázást végző *df* objektumot.

```
1   // 5-11. Programlista: MessageFormatTest.java
2
3   package org.cs.test;
4
5   import java.text.DateFormat;
6   ...
7   import java.util.Locale;
8
9   public class MessageFormatTest
10  {
11     public static void testTime()
12      {
13          Locale locale = Locale.GERMAN;
14
15          DateFormat df = DateFormat.getTimeInstance(DateFormat.SHORT, locale);
16          System.out.println( df.format(new Date() ));
17
18          df = DateFormat.getTimeInstance(DateFormat.LONG, locale);
19          System.out.println( df.format(new Date() ));
20
21          locale = new Locale("hu", "HU");
22          df = DateFormat.getTimeInstance(DateFormat.MEDIUM, locale);
23          System.out.println( df.format(new Date() ));
24      }
25
26      public static void main(String[] args)
27      {
28          testTime();
29      }
30  }
31
32  Futási kép:
33  16:26
34  16:26:34 GMT
35  16:26:34
```

A *testPercent()* (5-12. Programlista) a százalékok kezelésére ad egy példát.

```
1   // 5-12. Programlista: MessageFormatTest.java
2
3   package org.cs.test;
4
5   import java.text.DateFormat;
6   ...
```

```
7  import java.util.Locale;
8
9  public class MessageFormatTest
10 {
11     public static void testPercent()
12     {
13         Locale locale = Locale.GERMAN;
14
15         NumberFormat nf = NumberFormat.getPercentInstance(locale);
16         System.out.println( nf.format( 0.25 ) );
17
18         locale = Locale.ENGLISH;
19
20         nf = NumberFormat.getPercentInstance(locale);
21         System.out.println( nf.format( 0.25 ) );
22
23         locale = new Locale("hu", "HU");
24         nf = NumberFormat.getPercentInstance(locale);
25         System.out.println( nf.format( 0.25 ) );
26     }
27
28     public static void main(String[] args)
29     {
30         testPercent();
31     }
32 }
33
34 Futási kép:
35 25%
36 25%
37 25%
```

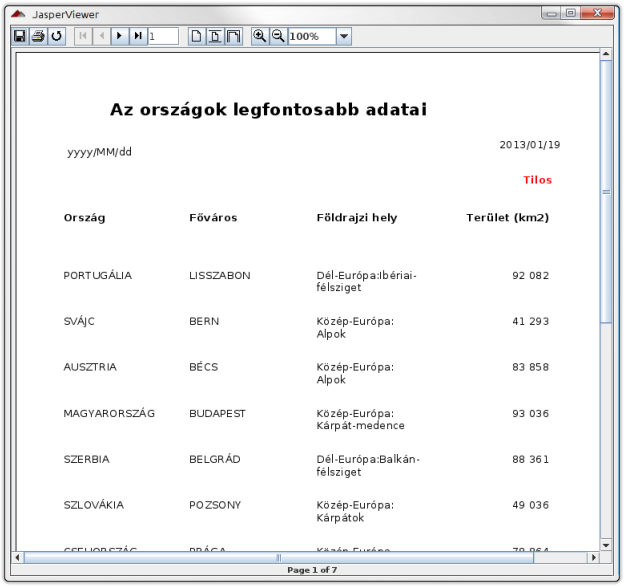
A szövegelemek előállításának másik fontos fajtája a szövegrészletek helyettesítése, amit *testMessageFormat()* teszteset (5-13. Programlista) ismertet meg velünk. A megoldás alapja a Java *MessageFormat* osztály, ami egy szövegmintát tartalmaz, benne tetszőleges számú feloldatlan, *{<number>}* jellel jelölt pozícióval. Amikor meghívjuk a *format()* metódust, akkor átadjuk a pozíciókba helyettesítendő értékeket is, így előáll a végleges szöveg, ahogy a futási kép is mutatja. Ezzel a módszerrel tehát úgy tehetünk össze szövegeket, hogy nem kell sok string összefűző műveletet végezni, ami jelentősen rontani szokta a program olvashatóságát. Érdekes felhasználása lehet például az SQL parancsok összeállítása is, ugyanis sokszor találkozunk azzal a gyakorlattal, hogy azt több soron rakja össze a programozó. Esetünkben egy jó jelölt lehet a query string is.

```
1  // 5-13. Programlista: MessageFormatTest.java
2
3  package org.cs.test;
4
5  import java.text.MessageFormat;
6  ...
7  import java.util.Locale;
8
9  public class MessageFormatTest
10 {
11     public static void testMessageFormat()
12     {
13         String result = MessageFormat.format("Az␣{0}␣ára:␣{1}.␣Ma␣van:␣{2,date}", "alma", 32, ➥
               new Date() );
14         System.out.println(result);
```

```
15      }
16
17      public static void main(String[] args)
18      {
19          testMessageFormat();
20      }
21  }
22
23  Futási kép:
24  Az alma ára: 32. Ma van: 2013.01.20.
```

Lokális adatformátumú riport

Eljutva idáig megtanultuk adatainkat országfüggő módon átalakítani szöveges formátumra. Befejezésül az *Orszagok-I18N.jrxml* riport tervbe (5-14. Programlista) betesszük a *terulet* oszlop nyelvfüggően formázott változatát. Nézzük meg az új jrxm fájlt, aminek csak a változást tartalmazó részét szúrtuk be! Az 1. újdonság a 7. sorba felvett *formatInteger* paraméter, ami *NumberFormat* típusú. A 2. új elem a 49-53 sorok között látható, megváltoztatott *textField* komponens. A *textFieldExpression* résznél (52. sor) jól látható, ahogy a paraméterül kapott *formatInteger* objektumnak meghívtuk a *format()* metódusát a *$F{TERULET}* értékre. Az így megváltoztatott riport elkészült változatát az 5.3. ábra mutatja. Láthatjuk, ahogy a terület oszlop számai szépen formázva jelennek meg. Az előző példából maradt címsorban megjelenített dátumra is érdemes egy pillantást vetni. A bal oldalon látható dátum formázó string csak az érdekesség kedvéért maradt a riportban.



5.3. ábra: Országfüggő riport

```
1   5-14. Programlista: Orszagok-I18N.jrxml
2
3   <?xml version="1.0" encoding="UTF-8"?>
4   <jasperReport...
5   ...
6       <parameter name="patternOfDate" class="java.lang.String"/>
7       <parameter name="formatInteger" class="java.text.NumberFormat"/>
8
9           <queryString language="SQL">
10              <![CDATA[select * from orszagok]]>
11          </queryString>
12
13          <field name="ORSZAG" class="java.lang.String"/>
14  ...
15          <title>
16              <band height="93" splitType="Stretch">
17                  <textField>
18                      <reportElement uuid="69af5ddd-3722-48b7-8620-5680547195ea" x="68" y="24" width="➥
                            411" height="20"/>
19                      <textElement>
20                          <font size="16" isBold="true"/>
21                      </textElement>
22                      <textFieldExpression><![CDATA[$R{title}]]></textFieldExpression>
23                  </textField>
24                  <textField>
```

```
25                                  <reportElement uuid="e740452a-0179-45f5-87c5-c1533b560fd3" x="432" y="60" width=➥
                                        "100" height="20"/>
26                                  <textElement/>
27                                  <textFieldExpression><![CDATA[(new SimpleDateFormat($P{patternOfDate})).format(➥
                                        Calendar.getInstance().getTime())]]></textFieldExpression>
28                          </textField>
29                  </band>
30          </title>
31  ...
32          <detail>
33                  <band height="40" splitType="Stretch">
34                          <textField>
35                                  <reportElement uuid="a2856d02-9cef-48e3-9240-cf0a11b76488" x="24" y="12" width="➥
                                        100" height="20"/>
36                                  <textElement/>
37                                  <textFieldExpression><![CDATA[$F{ORSZAG}]]></textFieldExpression>
38                          </textField>
39                          <textField>
40                                  <reportElement uuid="5e6ff35e-58db-418b-9bcb-e4f4044185e0" x="142" y="12" width=➥
                                        "100" height="20"/>
41                                  <textElement/>
42                                  <textFieldExpression><![CDATA[$F{FOVAROS}]]></textFieldExpression>
43                          </textField>
44                          <textField isStretchWithOverflow="true">
45                                  <reportElement uuid="be169129-86a9-47c3-829a-4196decf271c" x="261" y="12" width=➥
                                        "100" height="20"/>
46                                  <textElement/>
47                                  <textFieldExpression><![CDATA[$F{FOLDR_HELY}]]></textFieldExpression>
48                          </textField>
49                          <textField>
50                                  <reportElement uuid="a583d337-91a5-4ee2-b1cf-3bcd45429f8a" x="379" y="12" width=➥
                                        "100" height="20"/>
51                                  <textElement textAlignment="Right"/>
52                                  <textFieldExpression><![CDATA[$P{formatInteger}.format( $F{TERULET} )]]></➥
                                        textFieldExpression>
53                          </textField>
54                  </band>
55          </detail>
56  ...
57  </jasperReport>
```

A jelentés elkészítését végző programot az 5-15. Programlista tartalmazza. A *testI18NRDBMS()* metódust már ismerjük, a változás a 30-31 sorokban van, ahol létrehozunk egy *NumberFormat* típusú *nf* objektumot, majd *formatInteger* néven betesszük azt a riport paraméterei közé. Emlékeztetőül itt is kiemeljük, hogy ezt a paramétert a jrxml-ben ezzel a hívással használjuk:

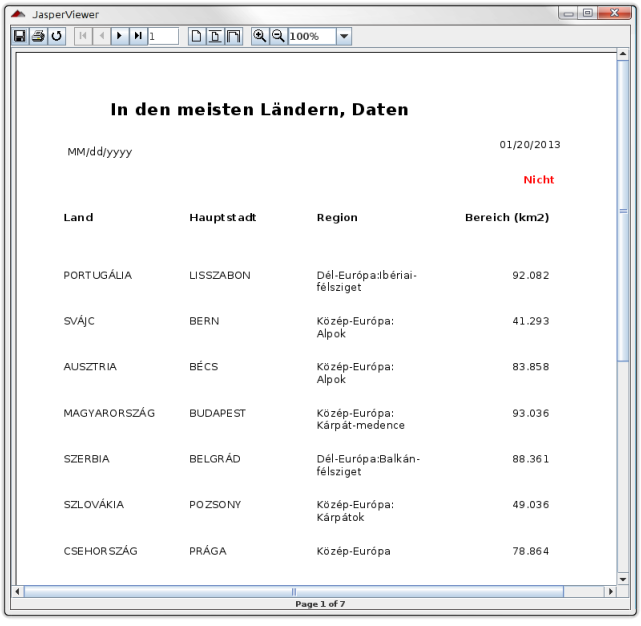
```
$P{formatInteger}.format( $F{TERULET} )
```

```
1   // 5-15. Programlista: TestOrszagokRiport.java
2
3   package org.cs.test;
4   ...
5   public class TestOrszagokRiport
6   {
7      static String jasperI18NFile = "/home/tanulas/jasperreport/pelda-1/Orszagok-I18N.jasper";
8
9       public static void testI18NRDBMS() throws Exception
10      {
11         //Locale locale = Locale.ENGLISH;
12         //Locale locale = Locale.GERMAN;
13         Locale locale = new Locale("hu", "HU");
14         ResourceBundle myResources = ResourceBundle.getBundle("orszagokText", locale);
15
16         String dateFormat = myResources.getString("dateFormat");
17
18         String sql = "select * from orszagok where terulet < ?";
19
20         Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", "➥
               ");
21         PreparedStatement stmt = conn.prepareStatement(sql);
22         stmt.setDouble(1, 100000.00);
```

```
23          ResultSet rs = stmt.executeQuery();
24          JRDataSource jrds = new JRResultSetDataSource(rs);
25          JRHelper jh = new JRHelper();
26
27          HashMap params = new HashMap();
28          params.put("patternOfDate", dateFormat);
29          params.put(JRParameter.REPORT_LOCALE, locale);
30          NumberFormat nf = NumberFormat.getIntegerInstance(locale);
31          params.put("formatInteger", nf);
32
33          jh.makeReport(jasperI18NFile, params, jrds);
34
35          rs.close();
36          stmt.close();
37          conn.close();
38
39      }
40
41      public static void main(String[] args) throws Exception
42      {
43          testI18NRDBMS();
44      }
45  }
```

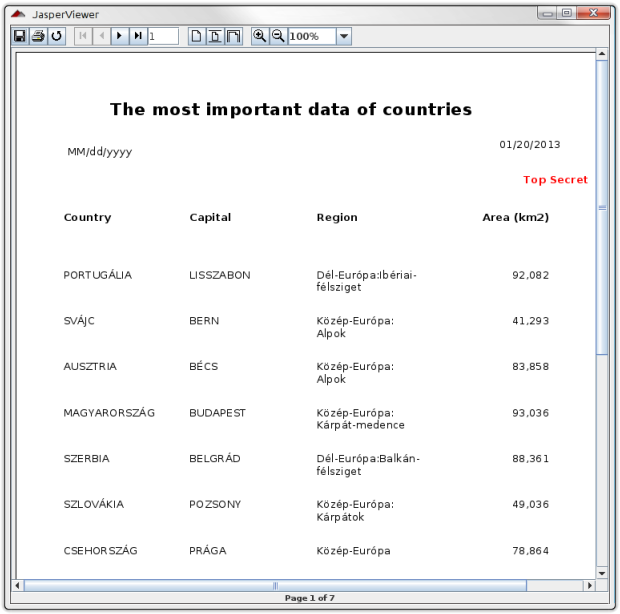
Most állítsuk be a *locale* változót erre az értékre és futassuk le a riportot!

```
Locale locale = Locale.GERMAN;
```



5.4. ábra: Német riport

Az eredményt az 5.4. ábra mutatja. Láthatjuk, hogy a dátum és a számok formátuma is változott. A számok tagolása hasonló a magyarhoz, azonban szóköz helyet „.’ karakter található. Az 5.5. ábra az angol változat, ahol a számok „,” karakterrel választódnak el egymástól.



5.5. ábra: Angol riport

6. Változók és kifejezések használata

Ebben a részben a JasperReports 2 hatékony eszközét vizsgáljuk meg. Az egyik a változók használata, ami nélkül sok feladat megoldása sokkal nehezebb vagy esetleg lehetetlen lenne. A másik a kifejezések alkalmazása, amiket már eddig is használtunk, azonban megtanuljuk, hogy sokkal többre is képesek.

Van néhány hely a *jrxml* fájlon belül, ahol kifejezéseket adhatunk meg:

- <textFieldExpression>
- <variableExpression>
- <initialValueExpression>
- <groupExpression>
- <printWhenExpression>
- <imageExpression>

Egy kifejezés változókból, paraméterekből és természetesen adatforrás mezőkből álló műveletek és metódushívások kombinációja lehet. Első lépésként tekintsük át a változókat egy kicsit részletesebben is, utána pedig elmélyedünk a kifejezések világában.

A változók

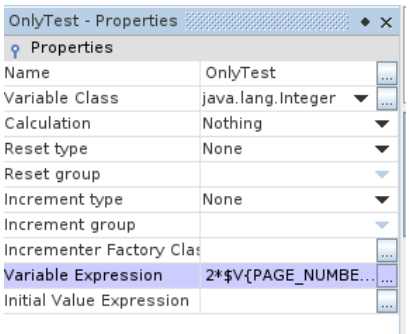
Alapismeretek

Egy riportban tetszőleges számú változót deklarálhatunk, aminek megadási módját a 6-1. Programlista *jrxml* töredéke mutatja be. A példában a *Nepsuruseg, var_1* és *var_2* egy-egy változó, amelyek alapvetően mindig valamilyen kifejezésnek a névvel hivatkozott tárolói. A nevet a *name* attribútumnál adjuk meg, amit a *class* megadása követ. Minden változó valamilyen Java osztályhoz tartozik, de természetesen a kalkulációs műveletek csak a szám típusokra fognak működni. A *variableExpression* az a tag, ahol egy Java kifejezés adható meg. A változó értékének kalkulációja az itt megadott kifejezés szerint fog történni.

```
1  // 6-1. Programlista: 2 változó deklaráslása
2
3  ...
4  <variable name="Nepsuruseg" class="java.lang.Double">
5                  <variableExpression><![CDATA[$F{NEPESSEG}.doubleValue() / $F{TERULET}.➥
                       doubleValue()]]></variableExpression>
6  </variable>
7
8  <variable name="var_1" class="java.lang.Integer" resetType="None">
9          <variableExpression><![CDATA[2*$V{PAGE_NUMBER}]]></variableExpression>
10         <initialValueExpression><![CDATA[ ]]></initialValueExpression>
11 </variable>
12
13 <variable name="var_2" class="java.lang.Integer" resetType="None">
14         <variableExpression><![CDATA[2 ]]></variableExpression>
15         <initialValueExpression><![CDATA[5 ]]></initialValueExpression>
16 </variable>
17 ...
```

A *Nepsuruseg* változó a népesség és a terület adatforrás mezők hányadosát tárolja (5. sor). A *var_1* az aktuális oldal számának a kétszeresét adja (9. sor), míg a *var_2* kifejezése a 2 értékű konstans (14. sor). Az *initialValueExpression* címke a változó induló értékének megadását biztosítja a riportkészítés megkezdésekor. A 6.1. ábra mutatja a változókra beállítható property lehetőségeket, amit mindenképpen meg kell értenünk. A *Reset type* egyes lehetőségei a következőek:

- *None*: Ekkor a változót a riport készítése során sosem állítjuk alapértékre (*No reset*).
- *Report*: Ekkor a változó egy alkalommal, a riport elkészítés elején inicializálódik. Ez az alapértelmezett működési mód.
- *Page*: A változó minden oldal megkezdésekor újra inicializálódik.
- *Column*: A változó minden oszlop megkezdésekor újra inicializálódik.
- *Group*: Egy új csoporthoz érve, annak megkezdésekor történik a reset. Ilyenkor a *Reset group* megadása is kötelező, hiszen egy riport adatai több szempont szerint csoportosulhatnak.



6.1. ábra: A változó tulajdonságai

A következő pontokban gyakori használati eseteket mutatunk be, azonban megjegyezzük, hogy a *Group*-okra alkalmazott beállításokat – noha talán ez a legfontosabb – csak a következő cikkben tárgyaljuk.

Az *Increment type* segítségével nagyon pontosan meg tudjuk adni azt a pillanatot, amikor a változó értékét változtatni, halmozni szükséges. Ugyanazokra az értékekre állíthatjuk, mint a *Reset type*-ot.

- *None*: Minden sornál megtörténik az inkrementálás.
- *Report*: A riport készítése során soha nem lesz aggregálva a változó.
- *Page*: A változó minden oldal megkezdésekor lesz növelve.
- *Column*: A változó minden új oszlop megkezdésekor halmozódik.
- *Group*: A következő – group-okkal foglalkozó – fejezetben részletesen bemutatjuk.

A *Calculation* lehetséges értékeit (*Count*=számlálás, *Sum*=összegzés, ...) a 2. cikkben már bemutattuk.

Beépített riport változók

A JasperReports tartalmaz néhány előre létrehozott riport változót, amit érdemes röviden áttekintenünk.

- *PAGE_NUMBER*: Ez a változó mindig a pillanatnyi oldalszámot tartalmazza, így értéke az 1. oldalon 1, a 12. oldalon 12. Van egy érdekes használati lehetősége, amikor az aktuális oldal / összes oldal értéket szeretnénk megjeleníteni a lapon (például annak footer részében). A *textField* vizuális riport komponens rendelkezik egy *evalutionTime* jellemzővel. Amennyiben a *PAGE_NUMBER* értékét 2 mezőbe is betesszük (legyen *Field_Act* és *Field_Pages*), akkor az 1. mező értékét

now, a 2. mezőét *Report* értékre állítva a kívánt hatást érjük el.

- *COLUMN_NUMBER*: Az éppen feldolgozás alatt lévő oszlop száma. Példa: Amennyiben a jelentés 3 oszlopos, úgy értékei: 1, 2, 3.
- *REPORT_COUNT*: Az adatforrásból jött, feldolgozott sorok száma.
- *PAGE_COUNT*: Az aktuális lapon eddig feldolgozott rekordok száma.
- *COLUMN_COUNT*: Az aktuális oszlopnál eddig feldolgozott rekordok száma.
- *GroupName_COUNT*: Az adott groupra vonatkozó eddig feldolgozott rekordok száma.

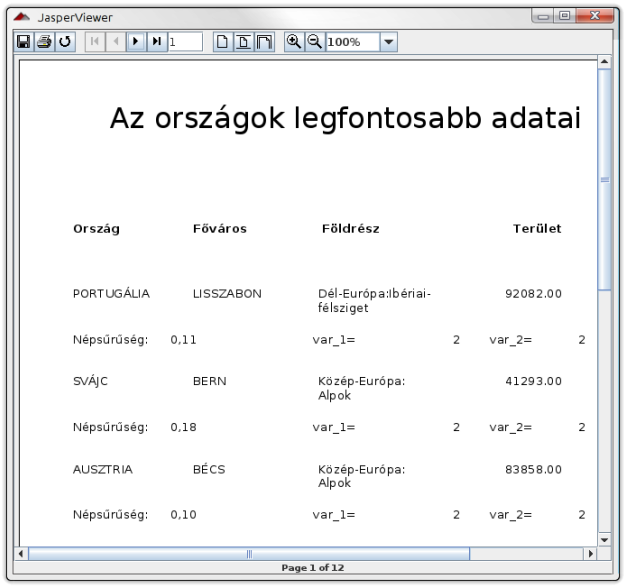
Egyszerű soronkénti kifejezés

A 6-1. Programlistán mutatott 3 riport változót kitettük a *detail* sávba a 6-2. Programlista szerint. Ezzel most minden adatforrás rekordhoz 2 sor tartozik a riportban (a 2. sor tartalmazza a 3 db változónkat és azok címkéjét), ahogy azt a 6.2. ábra mutatja. A *jrxml* részletben az egyedüli újdonság a 4. sorban található *pattern*, ugyanis a 2 *double* hányadosát 2 tizedesjegy hosszig jelenítjük meg. A változók jelenlegi beállításai (6.1. ábra) azt teszik lehetővé, hogy minden egyes új adatsor esetén a változók kiértékelődnek és ezt az értéket meg is kapják, ez fog megjelenni. Itt most semmilyen halmozásra nincs szükségünk, hiszen csak a sorokra vonatkoztatott számított adatokat szeretnénk kiírni.

```
1  // 6-2. Programlista: 3 riport mező
2
3  ...
4                          <textField pattern="###0.00;(###0.00)">
5                                  <reportElement uuid="812ff2c8-d729-49b0-b97d-825e4268cd22" x="105" y="50" width=➥
                                       "42" height="20"/>
6                                  <textElement/>
7                                  <textFieldExpression><![CDATA[$V{Nepsuruseg}]]></textFieldExpression>
8                          </textField>
9                          <textField>
10                                 <reportElement uuid="5af878b2-4702-45d7-9beb-2b7e947ed463" x="273" y="50" width=➥
                                       "73" height="20"/>
11                                 <textElement textAlignment="Right"/>
12                                 <textFieldExpression><![CDATA[$V{var_1}]]></textFieldExpression>
13                         </textField>
14                         <textField>
15                                 <reportElement uuid="0b9a19be-fdb1-4b4d-8e99-0fd3a81a9a1b" x="444" y="50" width=➥
                                       "100" height="20"/>
16                                 <textElement/>
17                                 <textFieldExpression><![CDATA[$V{var_2}]]></textFieldExpression>
18                         </textField>
19                         <staticText>
20                                 <reportElement uuid="6d87e249-1e09-4f25-8702-6b2c297a61e0" x="24" y="50" width="➥
                                       66" height="20"/>
21                                 <textElement/>
22                                 <text><![CDATA[Népsűrűség:]]></text>
23                         </staticText>
24                         <staticText>
25                                 <reportElement uuid="02007466-be21-4182-bafa-9bda944feb82" x="160" y="50" width=➥
                                       "100" height="20"/>
26                                 <textElement textAlignment="Right"/>
27                                 <text><![CDATA[var_1=]]></text>
28                         </staticText>
29                         <staticText>
30                                 <reportElement uuid="f725cc06-9408-468c-b41b-f41c82df2d68" x="371" y="50" width=➥
                                       "100" height="20"/>
31                                 <textElement/>
32                                 <text><![CDATA[var_2=]]></text>
33                         </staticText>
34 ...
```

A *Nepsuruseg* ezzel mindig az adott sorra, azaz adott országra számított érték lesz. A *var_1* a *2*$V{PAGE_NUMBER}* kifejezés szerint működik, azaz az egyes oldalakon végig a *2, 4, ..., 24* lesz az értéke (egy lapon belül persze ugyanaz az érték), ugyanis az egész jelentés 12

oldalas. A *var_2* értéke csak egy konstans, ami most a riport minden sorában a 2 értéket veszi fel. Aki az eddigieket értette, annak bizonyára egyből érthető, hogy minden más változatlanul hagyása mellett a *Reset type=Report* esetén is ugyanez lesz a végeredmény.



6.2. ábra: Egyszerű kifejezés - Riport részlet

Számlálás

Most állítsuk be a *var_1 és var_2* változók *Calculation* jellemzőjét *Count* értékre (a *Reset type=Riport* maradt). Az így elkészített riport mindkét változóra soronként 1-gyel nagyobb értéket ír ki. Ez azt jelenti, hogy minden egyes sorra mindkét változó értéke ez a sorozat lesz: 1, 2, 3, ..., 89. A következő kísérlet legyen csak annyi változtatás, hogy a *Reset type=Page* beállítást csináljuk. Ekkor valóban azt kapjuk, amit vártunk, azaz minden oldalon elölről kezdődik az 1, 2, ... sorozat.

Főösszegek

Most vizsgáljuk meg a másik tipikus aggregálási műveletet, az összegzést! Ehhez a *Calculation=Sum* és *Reset type=Report* beállítást végezzük el a *var_1 és var_2* változókon. A következő eredményt fogjuk kapni az egyes változókra:

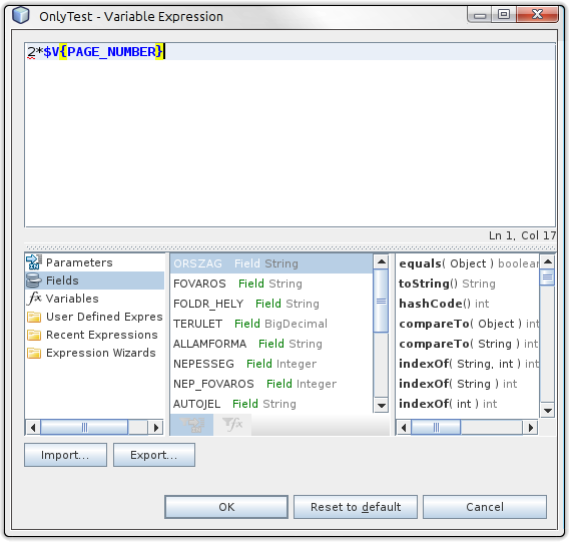
1. *var_1*:
   (a) 1. oldal: 2, 4, 6, ..., 14 (halmozás 2x1=2 hozzáadásával)
   (b) 2. oldal: 18, 22, 26, ..., 46 (halmozás 2x2=4 hozzáadásával)
   (c) 3. oldal: 52, 58, 64, ... (halmozás 2x3=6 hozzáadásával)
2. *var_2*:
   (a) 1. oldal: 2, 4, 6, ..., 14
   (b) 2. oldal: 16, 18, 20, ...

Befejezésül nézzük meg a *Calculation=Sum* és *Reset type=Page* beállítást! Ilyenkor minden lapkezdéskor reset történik mindkét változóra, a jelentés soronkénti értékei pedig így alakulnak:

1. *var_1*:
   (a) 1. oldal: 2, 4, 6, ..., 14 (halmozás 2x1=2 hozzáadásával)
   (b) 2. oldal: 4, 8, 12, ..., 32 (halmozás 2x2=4 hozzáadásával)
   (c) 3. oldal: 6, 12, 18, ... (halmozás 2x3=6 hozzáadásával)
2. *var_2*:
   (a) 1. oldal: 2, 4, 6, ..., 14
   (b) 2. oldal: 2, 4, 6, ...

A kifejezések összeépítése

Egy JasperReports kifejezés operandusai azok az objektumok lehetnek, amik a 6.3. ábra alsó részén láthatóak. Amit látunk, az egy *iReport* segédeszköz a kifejezések gyors összekattintására, de természetesen ezt kézzel írva is megadhatjuk. Az ablak például akkor bukkan fel, amikor a 6.1. ábra *Variable Expression* melletti „...” nyomógombra kattintunk. Ugyanakkor ilyen lehetőség több is van, hiszen a vizuális felületen sok olyan pont van, ahol éppen egy kifejezést lehet megadni. Látható, hogy a mezők, változók és paraméterek, illetve ezek metódushívásaival visszakapott értékek alkotják a kifejezésben résztvevő tagokat. A tagok között a Java mindegyik operátorát használhatjuk. A kifejezések azért nagyon fontosak, mert ezzel is dinamikusan lehet adatokat előállítani a jelentés számára. Vegyük példaként az *<imageExpression>* címkét! Itt egy képfájl útvonalát adhatjuk meg, azonban azt nem kell statikusan beégetni a *jrxml* fájlba, esettől függően akár soronként más-más kép jelenhet meg, attól függően, hogy az ide definiált sztring kifejezés éppen melyik képfájl nevét adta vissza.



6.3. ábra: Kifejezések összeállítása

Az if-then-else megvalósítása

Ebben a pontban szeretnénk elmélyíteni a kifejezések használatának megértését, ezért

1. először készítünk egy hasznos osztályt (*JasperIf*),
2. majd bemutatjuk annak használatát egy összetett kifejezés részeként.

A *JasperIf* osztály létrehozása

A *JasperIf* osztály (6-3. Programlista) céljának és működésének megértéséhez először nézzük meg a 89-100 sorok közötti *main()* metódust, ami példát mutat a használatára. A cél az, hogy egy láncolt kifejezéssel le tudjunk írni egy több elágazásos *if* utasítást. Itt jegyezzük meg, hogy ehhez a builder tervezési minta egy implementációs technikáját használjuk, ahol az egyes láncszemek azért fűzhetőek fel egymás utáni metódushívásokkal, mert mindegyik a *this* objektumot adja vissza. Ez alól csak a lánc végén meghívott *evaluate()* a kivétel, ami – gondolva a riportban való használat céljára – most *String*et ad vissza. A 93. sor *a* és *b* változója csak a példa kedvéért lett felvéve, velük fogalmazzuk meg a 95. és 98. sorokban lévő használati esetek feltételeit. A használat jobb megértéséhez nézzük most a 98. sorban megfogalmazott kifejezést! A *j* változó egy *JasperIf* objektumra mutat, amit a 91. sorban hoztunk létre. Balról nézve az 1. metódushívás a *jif(a < b)*, ami egy *JasperIf* objektumot ad vissza, ez pontosabban maga a *j* által mutatott, már említett objektum (más szavakkal ez a *this*). A 2. hívás a *jthen("a < b")*, ami az *if* igaz értéke esetén visszaadott *String* értékét állítja be, miközben a visszatérési érték megint a *j* referencia. A 3. hívás a *j* változónkra vonatkozó *jelseif(a == b, "a == b")*, ahol ennek igazsága mellett (1. paraméter) visszaadandó String értéket (2. paraméter) definiáljuk. A 4. metódus ugyanígy működik. A

lánc végén lévő *evaluate()* végül elvégzi a számítást és a megfelelő ághoz tartozó String értéket fogja az *s*-hez rendelni. A program futtatása után ez a 2 sor jelenik meg a képernyőn:

```
a >= b
a > b
```

A *JasperIf* class egy szép példa az építő minta használatára. A konstruktor (19-22 sorok) előállít egy induló objektumot, amit utána az építő metódusokkal alakítjuk a véglegesre:

- *jif*: eltárolja a paraméterként megadott logikai kifejezés értékét (az *ifCondition* változóba)
- *jthen*: eltárolja az *ifCondition==true* esetre visszaadandó Stringet.
- *jelse*: eltárolja az *else* esetre visszaadandó Stringet.
- *jelseif*: egy *ElseIf* segédosztályokból álló (boolean, String párok) listát épít.

Az 53-69 sorok között található *evaluate()* a builder metódusok által kialakított *JasperIf* objektumot értékeli ki, azaz végül azt a Stringet adja vissza, ami az igaz ágnak felel meg.

```
1  // 6-3. Programlista: JasperIf.java
2
3  package org.cs.test;
4
5  import java.util.ArrayList;
6  import java.util.List;
7
8  /**
9   * Builder minta
10  */
11 public class JasperIf
12 {
13
14     private Boolean ifCondition;
15     private List<ElseIf> elseifConditions;
16     private String thenStatement;
17     private String elseStatement;
18
19     public JasperIf()
20     {
21         this.elseifConditions = new ArrayList<ElseIf>();
22     }
23
24     public JasperIf jif(Boolean ifCond)
25     {
26         this.ifCondition = ifCond;
27         return this;
28     }
29
30     public JasperIf jthen(String then)
31     {
32         this.thenStatement = then;
33         return this;
34     }
35
36     public JasperIf jelse(String elseStatement)
37     {
38         this.elseStatement = elseStatement;
39         return this;
40     }
41
42     public JasperIf jelseif(Boolean elseif, String elseifStatement)
43     {
44         this.elseifConditions.add(new ElseIf(elseif, elseifStatement));
```

```
45          return this;
46      }
47
48      public String toString()
49      {
50          return evaluate();
51      }
52
53      private String evaluate()
54      {
55          if (ifCondition)
56          {
57              return thenStatement;
58          } else
59          {
60              for (ElseIf elseif : elseifConditions)
61              {
62                  if (elseif.cond)
63                  {
64                      return elseif.statement;
65                  }
66              }
67          }
68          return elseStatement;
69      }
70
71      private class ElseIf
72      {
73
74          private Boolean cond;
75          private String statement;
76
77          ElseIf(Boolean cond, String statement)
78          {
79              this.cond = cond;
80              this.statement = statement;
81          }
82      }
83
84      /**
85       * Only test
86       *
87       * @param args
88       */
89      public static void main(String[] args)
90      {
91          JasperIf j = new JasperIf();
92
93          int a = 5, b = 3;
94
95          String s = j.jif(a < b).jthen("a < b").jelse("a >= b").evaluate();
96          System.out.println(s);
97
98          s = j.jif(a < b).jthen("a < b").jelseif(a == b, "a == b").jelseif(a > b, "a > b").➥
                evaluate();
99          System.out.println(s);
100     }
101 }
```

A *JasperIf* osztály használata

Most, hogy rendelkezünk egy egészen csinos *if-then-else* lehetőséggel, próbáljuk ki ennek a JasperReportsban történő használatát! Az *Orszagok-VAR.jrxml* (6-4. Programlista) fontos változtatásokat tartalmaz, nézzük meg őket

egyesével:

1. Egy import utasítás (6. sor): A *JasperIf* osztályt használjuk, ezért importálni kell. Fontos, hogy az *iReport Tools* → *Options* → *Classpath* fülnél felvettük a classpathhoz az elérhetőségét is.

2. Egy új *sizeSelector* nevű paraméter (8. sor): Ez a paraméter azt vezérli majd, hogy a kifejezésben mely terület méretha-tárnál kezdődjön a „Nagy” ország.

3. A *TERULET textField* (18. sor): Az ország mérete helyett most ezzel a kifejezéssel csak annyit írunk ki, hogy „Nagy” vagy „Kicsi”:

```
new JasperIf()).jif($F{TERULET}>$P{sizeSelector}).jthen("Nagy").jelse("Kicsi").evaluate()
```

```
1  // 6-4. Programlista: Orszagok-VAR.jrxml
2
3  ?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport...
5  ...
6  <import value="org.cs.test.*"/>
7  ...
8  <parameter name="sizeSelector" class="java.lang.Integer"/>
9
10 <queryString language="SQL">
11         <![CDATA[select * from orszagok]]>
12 </queryString>
13 ...
14 <detail>
15                         <textField>
16                                 <reportElement uuid="a583d337-91a5-4ee2-b1cf-3bcd45429f8a" x="331" y="12" width=
                                       "100" height="20"/>
17                                 <textElement textAlignment="Right"/>
18                                 <textFieldExpression><![CDATA[(new JasperIf()).jif($F{TERULET}>$P{sizeSelector})
                                       .jthen("Nagy").jelse("Kicsi").evaluate()]]></textFieldExpression>
19                         </textField>
20 </detail>
21 ...
22 <jasperReport>
```

A riportot a 6-5. Programlista kódjával készíthetjük el, ahol látható a *sizeSelector* paraméter értékének az átadása is. Az eredményt a 6.4. ábra mutatja, ami a teljes riportból kivágott kis részlet. Nézzük meg, hogy a terület oszlop alatt az eddigi számértékek helyett most a kíván 2 szó valamelyike jelenik meg!

```
1  // 6-5. Programlista: TestOrszagokRiport.java
2
3  package org.cs.test;
4
5  public class TestOrszagokRiport
6  {
7  ...
8      public static void testRDBMS() throws Exception
9      {
10         String sql = "select * from orszagok where terulet < ?";
11
12         Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA",
               "");
13         PreparedStatement stmt = conn.prepareStatement(sql);
14         stmt.setDouble(1, 1000000.00);
15         ResultSet rs = stmt.executeQuery();
16         JRDataSource jrds = new JRResultSetDataSource(rs);
17
18         HashMap params = new HashMap();
19         params.put("sizeSelector", 100000);
20         JasperFillManager.fillReportToFile(jasperVarFile, jrprintFile, params, jrds);
21
```

```
22          rs.close();
23          stmt.close();
24          conn.close();
25      }
26  ...
27      public static void main(String[] args) throws Exception
28      {
29          testRDBMS();
30      }
31  }
```

Ország	Főváros	Földrész	Terület	Népesség
SPANYOLORSZÁG	MADRID	Dél-Európa:Ibériai-félsziget	Nagy	42700
Népsűrűség: 0,08		var_1= 2	var_2= 2	
PORTUGÁLIA	LISSZABON	Dél-Európa:Ibériai-félsziget	Kicsi	10000
Népsűrűség: 0,11		var_1= 4	var_2= 4	
FRANCIAORSZÁG	PÁRIZS	Nyugat-Európa	Nagy	60100
Népsűrűség: 0,11		var_1= 6	var_2= 6	

6.4. ábra: Kifejezés - Kicsi és Nagy ország

Az ant script használatához is ki kell egészítenünk a build.xml fájlt, hiszen az ant compile igényelni fogja a *JasperIf* osztályt.

```
<path id="classpath">
    <pathelement location="./"/>
    <pathelement location="${classes.dir}"/>
    <pathelement location="/opt/java/classes"/>
    <fileset dir="${lib.dir}">
      <include name="**/*.jar"/>
    </fileset>
</path>
```

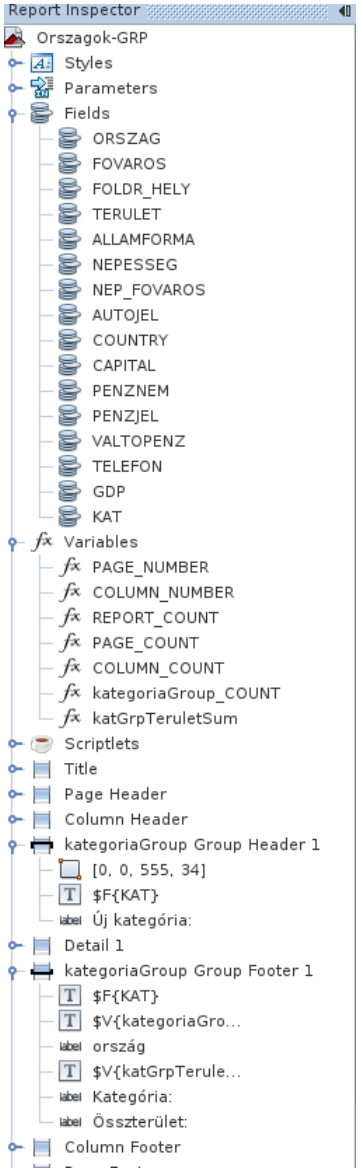
Záró gondolatok

Ebben a részben ismét egy hatékony eszközt ismertünk meg. A riport generálása során – ahogy láttuk is – bármelyik adatelem egy kifejezésen keresztül is bekerülhet a riportba. Az adatforrás persze alapvető input adat, azonban a kifejezések használatával szinte minden adatot származtatni vagy transzformálni tudunk. Kevés olyan riportkészítő eszköz létezik amelyik ilyen eleganciával teszi lehetővé a jelentés tartalmának az összeállítását. A későbbiekben majd még arra is látunk példát, hogy a kifejezésekkel milyen módon lehet az elkészült dokumentumainkat formázni, esetfüggő külalakkal ellátni (például a kritikus számok piros háttérrel jelenjenek meg). A kifejezések természetesen bármilyen távoli hívást is tartalmazhatnak, például egy tőzsdei jegyzéskód helyett annak nevét írjuk ki, amit egy webservice segítségével szereztünk meg.

7. Az adatok csoportokba foglalása

Amikor réges régen elkezdtem foglalkozni az informatikával, egy nagy számítóközpontban dolgoztam. Rengeteg nyomtatott tabló készült és ezeket a programozók „kontroll” vagy „összegfokozatos” listáknak nevezték. A lényege mindegyiknek az volt, hogy az adatsorokat logikailag csoportokba foglalta, miközben annak vég és részösszegeit is feltüntette. Most azt nézzük meg, hogy mindezt milyen módon lehet megvalósítani JasperReportsban.

Egy egyszerű csoportosítás



7.1. ábra: Egy új riport

Az *orszagok* adatbázis táblára és az első *Orszagok.jrxml* riportunkra alapozva most készítünk egy olyan jelentést (a design template neve *Orszagok-GRP.jrxml* lesz), ami kiegészül a *GDP* és *ország kategória* oszlopokkal. A feladat az lesz, hogy a lista 1 összegfokozatot tartalmazzon, amit jelen esetben a kategória szerinti csoportosítás fog képezni.

A *kategoriaGroup* csoport létrehozása

Első lépésként meg kell terveznünk a 7.3. ábra által mutatott jelentés template-et. A sima 6 oszlopos riportból indulunk ki. Menjünk az egérrel a *Report Inspector* (7.1. ábra) gyökér eleméhez (*Orszagok-GRP*) és a jobb egérgombot megnyomva válasszuk ki az *Add Report Group* funkciót. Egy varázsló bekéri tőlünk a group nevét (ez lett a *kategoriaGroup*) és a csoportosító kifejezést, amire most *$F{KAT}* értéket adtuk meg. Egy-egy checkbox segítségével azt is meg kell adnunk, hogy szeretnénk-e a csoport számára header és/vagy footer sávokat létrehozni (mi most mindkettőt kiválasztottuk). A sávok – ahogy a nevük alapján bizonyára kitalálta mindenki – a csoportok előtt és után jelenítődnek meg, általában valami összegző, a csoportra jellemző információval. Szeretnénk kiemelni, hogy a *kategoriaGroup_COUNT* változót a rendszer automatikusan hozta létre, a csoport pillanatnyi elemszámát tartalmazza, így amikor a footerben megjelenítjük, akkor a teljes csoport sorainak számát fogja hordozni. A létrejött 2 sáv a riport tervezőben is megjelent (7.3. ábra), egy-

előre persze üresen, a későbbiekben mi teszünk ki rá tartalmat.

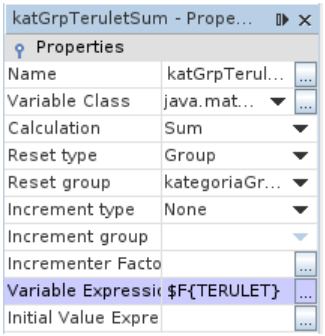
A *katGrpTeruletSum* változó létrehozása

Szeretnénk a csoport footer sávjában megjeleníteni az adott kategóriába tartozó összes országot és azok összterületét. Az 1. értékét a már említett automatikusan létrejött változó tartalmazza, azonban a 2. tartalmi elem értékének gyűjtéséhez hozzunk létre egy *katGrpTeruletSum* nevű változót, aminek beállításait a 7.2. ábra mutatja. Itt több újdonságra felhívjuk a figyelmet:

1. A *Reset type* értéke most *Group* lett, azaz egy csoport kezdetekor kell a változót inicializálni.

2. A *Reset group* pedig a mi *kategoriaGroup* csoportunk lett.

A *Calculation* értéke *Sum*, hiszen most az oszlop értékeinek összege kell nekünk. Emiatt persze a *Variable Class* is számtípus lesz, esetünkben *BigDecimal*.



7.2. ábra: A *katGrpTeruletSum* változó

A tartalom befejezése és egy kis formázás

A 7-1. Programlista mutatja a jrxml fájl esetünk szempontjából lényeges részeit, ennek design nézetét mutatja az *iReport* 7.3. ábrán látható részlete. Vegyük sorra a lényeges változtatásokat!

```
1  // 7-1. Programlista: Orszagok-GRP.jrxml
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport...
5  ...
6          <queryString language="SQL">
7                  <![CDATA[select * from orszagok order by kat]]>
8          </queryString>
9  ...
10         <field name="GDP" class="java.lang.Integer"/>
11         <field name="KAT" class="java.lang.Integer"/>
12 ...
13         <variable name="katGrpTeruletSum" class="java.math.BigDecimal" resetType="Group" resetGroup="➥
               kategoriaGroup" calculation="Sum">
14                 <variableExpression><![CDATA[$F{TERULET}]]></variableExpression>
15         </variable>
16         <group name="kategoriaGroup">
17                 <groupExpression><![CDATA[$F{KAT}]]></groupExpression>
18                 <groupHeader>
19                         <band height="34">
20                                 <rectangle>
21                                         <reportElement uuid="1fd2eb7a-fd09-4849-9f56-50a078657bec" mode="Opaque"➥
                                               x="0" y="0" width="555" height="34" backcolor="#FF6633"/>
22                                 </rectangle>
23                                 <textField>
24                                         <reportElement uuid="fc5f354c-3535-4035-a396-84619bc058d3" x="85" y="10"➥
                                               width="100" height="20"/>
25                                         <textElement>
26                                                 <font isBold="true"/>
27                                         </textElement>
28                                         <textFieldExpression><![CDATA[$F{KAT}]]></textFieldExpression>
29                                 </textField>
30                                 <staticText>
31                                         <reportElement uuid="89aa5dcd-568a-4c24-b2e7-cc6c13eda27c" x="5" y="10" ➥
                                               width="77" height="20"/>
32                                         <textElement>
33                                                 <font isBold="true"/>
34                                         </textElement>
35                                         <text><![CDATA[Új kategória:]]></text>
36                                 </staticText>
37                         </band>
38                 </groupHeader>
```

```
39                  <groupFooter>
40                          <band height="37">
41                                  <textField>
42                                          <reportElement uuid="0192bb82-605b-42a6-ad30-0de909bd1bc8" x="76" y="7" ➥
                                                width="100" height="20"/>
43                                          <textElement>
44                                                  <font isBold="true"/>
45                                          </textElement>
46                                          <textFieldExpression><![CDATA[$F{KAT}]]></textFieldExpression>
47                                  </textField>
48                                  <textField>
49                                          <reportElement uuid="bc399f7c-38d0-4586-8b17-c16d1654e057" x="160" y="7"➥
                                                 width="40" height="20"/>
50                                          <textElement textAlignment="Right">
51                                                  <font isBold="true"/>
52                                          </textElement>
53                                          <textFieldExpression><![CDATA[$V{kategoriaGroup_COUNT}]]></➥
                                                textFieldExpression>
54                                  </textField>
55                                  <staticText>
56                                          <reportElement uuid="2922ed41-e2d2-4d63-8691-7e48226b3507" x="202" y="7"➥
                                                 width="100" height="20"/>
57                                          <textElement>
58                                                  <font isBold="true"/>
59                                          </textElement>
60                                          <text><![CDATA[ország]]></text>
61                                  </staticText>
62                                  <textField pattern="#,##0.00;-#,##0.00">
63                                          <reportElement uuid="124d15ae-9111-43c6-aad3-69eb40edf4ea" x="401" y="7"➥
                                                width="100" height="20"/>
64                                          <textElement textAlignment="Right">
65                                                  <font isBold="true"/>
66                                          </textElement>
67                                          <textFieldExpression><![CDATA[$V{katGrpTeruletSum}]]></➥
                                                textFieldExpression>
68                                  </textField>
69                                  <staticText>
70                                          <reportElement uuid="89aa5dcd-568a-4c24-b2e7-cc6c13eda27c" x="4" y="7" ➥
                                                width="57" height="20"/>
71                                          <textElement>
72                                                  <font isBold="true"/>
73                                          </textElement>
74                                          <text><![CDATA[Kategória:]]></text>
75                                  </staticText>
76                                  <staticText>
77                                          <reportElement uuid="91eec85b-5f4f-4c52-9478-fd8d364dc218" x="321" y="7"➥
                                                 width="70" height="20"/>
78                                          <textElement>
79                                                  <font isBold="true"/>
80                                          </textElement>
81                                          <text><![CDATA[Összterület:]]></text>
82                                  </staticText>
83                          </band>
84                  </groupFooter>
85          </group>
86  ...
87  </jasperReport>
```

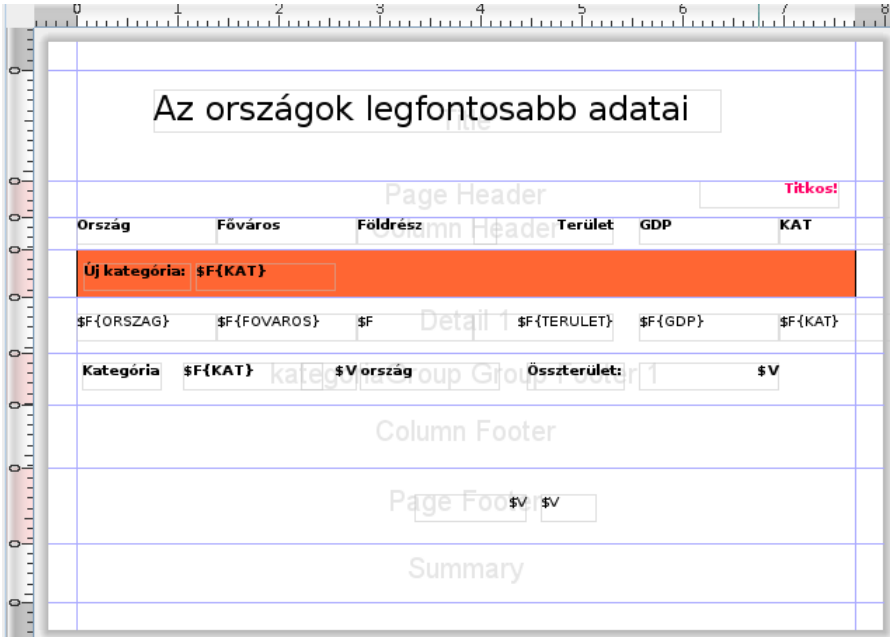
Az első figyelemre méltó dolog a 7. sorban lévő *queryString*, ahol az *order by* résznél a *KAT* (kategória) szerinti rendezettséget kértük, ugyanis a riport ezen szempont szerint lesz csoportosítva. Amennyiben ez elmarad, úgy a JasperReports csinál csoportokat, de a végeredmény nem lesz helyes, hiszen ő csak annyit néz, hogy a következő rekord kulcsa egyezik-e a jelenlegivel. A 10-11 sorok csak a 2 új mezőt mutatják, az eddigiekben ezek nem voltak használva. A 13-15 sorok a már ismert *katGrpTeruletSum* változót vezetik be. A 16-85 sorok a teljes *kategoriaGroup* csoportot leírják. A 17. sor tartalmazza a *groupExpression* címkét, ahol a csoportképző kifejezésünk most ez: *$F{KAT}*. A *groupHeader* sáv a következő tartalomból áll:

- Egy téglalap (narancssárga színnel).
- Az „Új kategória:” címke.
- Egy *textField* a *$F{KAT}* értékkel, azaz mutatjuk, hogy éppen melyik csoportba léptünk be.

A 39. sorban kezdődő *groupFooter* sáv tartalma:

- Mutatja, hogy éppen melyik kategóriát zárjuk le.

- Mutatva a csoport számosságát, megjeleníti a *$V{kategoriaGroup_COUNT}* értéket.
- Feltünteti a *$V{katGrpTeruletSum}* értéket is, ami a csoportba tartozó összterület.



7.3. ábra: A riport template

MOLDOVA	CHISINAU	Kelet-Európa	33700.00	380	2
VIETNAM	HANOI	Ázsia:Indokinai-félsziget	329556.00	460	2
BELARUSZ	MINSZK	Kelet-Európa	207600.00	1790	2
KAMBODZSA	PHNOM PENH	Ázsia:Indokinai-félsziget	181035.00	280	2
Kategória 2		**50 ország**	**Összterület:**	**28 612 378,00**	
Új kategória: 3					
BURUNDI	BUJUMBURA	Közép-Kelet-Afrika	27834.00	100	3
BISSAU-GUINEA	BISSAU	Nyugat-Afrika	36125.00	460	3

7.4. ábra: A kategóriánként való összegfokozatos lista egy részlete

A 7.3. ábrán látható riportot az iReport *Preview* fülével is lefuttathatjuk, eredményül a 7.4. ábrán látható eredményt kapjuk. A formázás egyszerű, de talán már ez is hatásos. A csoportok nyitását a narancssárga, átlátszó téglalap emeli ki, azok zárását pedig csak a vastagon szedett footer sor.

Egy új csoportosítási szempont

Az előző riportot kiegészítjük egy új csoportosítási szemponttal, azaz a kategóriákon belül az országok neveit is csoportba szedjük. A helyes működés érdekében a *queryString order by* részébe elhelyeztük az *orszag* oszlopot is:

```
select * from orszagok order by kat, orszag
```

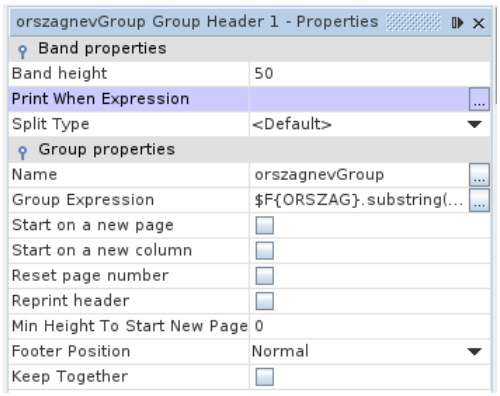
A kezdőbetűk szerinti csoport

Vegyünk fel egy új *orszagnevGroup* nevű csoportot, aminek a jellemzőit a 7.5. ábra mutatja. A csoportképző kifejezés a következő:

```
$F{ORSZAG}.substring( 0, 1 )
```

Ez azt jelenti, hogy a generátor akkor teszi a következő sort új csoportba, ha az ország kezdőbetűje változik. Itt ragadjuk meg a lehetőséget, hogy a csoportok néhány további jellemzőjét ismertessük:

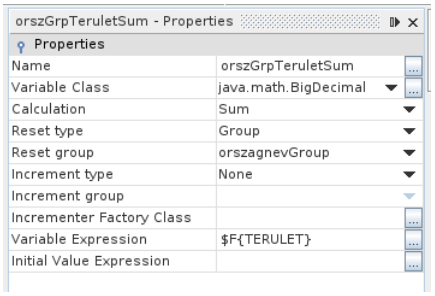
- *Band height*: a csoport header vagy footer sávjának magassága.
- *Start on new page*: Amennyiben igaz, úgy a csoport renderelése új lapra lesz elkezdve.
- *Reset page number*: Igaz esetén a csoportváltáskor a lapszámozás is elölről kezdődik.
- *Reprint header*: Amennyiben kérjük, úgy a csoporthoz a fejlécszövegek is kinyomtatódnak.
- *Footer Position*: Hol legyen a csoport footer sáv.



7.5. ábra: Az *orszagnevGroup* csoport

A csoport létrehozása után ismét készítsünk egy összegző változót (neve: *orszGrpTeruletSum*), amit 7.6. ábra ismertet. A változó 3 legfontosabb tulajdonsága a következő:

1. A *Reset type=Group*, ami most az *orszagnevGroup* csoportra van állítva.
2. Az változó kifejezése *$F{TERULET}*, hiszen a területet akarjuk összegezni.
3. A *Calculation=Sum*



7.6. ábra: Az *orszGrpTeruletSum* változó

A megváltoztatott riport terv

A 7.7. ábra az új *jrxml* fájl vizuális képét tartalmazza az iReport eszközzel való tervezés közben. A kategória csoportot narancssárga, míg az ország kezdőbetű csoportot sárga színnel emeltük ki, hogy a jelentés olvasóinak könnyebb legyen eligazodnia a 2 összegfokozat váltás között. Ezt a hatást úgy tudtuk elérni, hogy kitettünk

néhány téglalap (*rectangle*) elemet, átlátszó beállítással és a megfelelő háttérszínnel. A sárga sávokra is felhelyeztük a megfelelő 2 változót, a konstans szövegeket és a csoportosítást adó mezőt és annak megnevezését.

7.7. ábra: A riport template

7.8. ábra: Riport eleje

CSEHORSZÁG	PRÁGA	Közép-Európa	78864.00	7990
C betű: 2 ország Összterület: 88 115,00				
Kezdőbetű: D				
DÁNIA	KOPPENHÁGA	Nyugat-Európa:	43075.00	38120
DÉL-KOREA	SZÖUL	Ázsia:Koreai-	98484.00	10480
D betű: 2 ország Összterület: 141 559,00				
Kezdőbetű: E				
EGYIPTOM	KAIRÓ	Észak-Afrika	1001449.00	1300
E betű: 1 ország Összterület: 1 001 449,00				
Kezdőbetű: F				
FINNORSZÁG	HELSINKI	Észak-Európa	338107.00	30360
FRANCIAORSZÁG	PÁRIZS	Nyugat-Európa	547026.00	27520
F betű: 2 ország Összterület: 885 133,00				
Kezdőbetű: G				
GÖRÖGORSZÁG	ATHÉN	Dél-Európa:	131944.00	15060
G betű: 1 ország Összterület: 131 944,00				
Kezdőbetű: H				

7.9. ábra: Riport közepe

É betű: 2 ország Összterület: 165 765,00				
Kezdőbetű: Í				
ÍRORSZÁG	DUBLIN	Európa:Britt-	70283.00	38430
Í betű: 1 ország Összterület: 70 283,00				
Kezdőbetű: Ú				
ÚJ-ZÉLAND	WELLINGTON	Csendes-óceán	269112.00	18080
Ú betű: 1 ország Összterület: 269 112,00				
Kategória 1 69 ország Összterület: 84 050 556,36				
Új kategória: 2				
Kezdőbetű: A				
AFGANISZTÁN	KABUL	Közép-Ázsia	652225.00	700
ALGÉRIA	ALGÍR	Észak-Afrika	2381741.00	2080
AZERBAJDZSÁN	BAKU	Elő-Ázsia:	86600.00	1770
A betű: 3 ország Összterület: 3 120 566,00				
Kezdőbetű: B				
BAHAMA-SZIGETEK	NASSAU	Közép-Amerika:	13939.00	18690

7.10. ábra: A riport közepe

A 7.8, 7.9 és 7.10. ábrák az elkészült jelentés szerkezeti szempontból lényegesebb részeit mutatják be. Tekintettel arra, hogy most sok változtatást tettünk a *jrxml* terven és annak tanulmányozásából is sokat lehet tanulni, úgy most a 7-2. Programlista a teljes fájlt tartalmazza.

```
1  // 7-2. Programlista: Orszagok-GRP.jrxml
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/➥
       XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://➥
       jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-GRP" language="groovy" pageWidth="595" ➥
       pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20" uuid➥
       ="9a1dbe76-1ae5-4418-9e43-e56f6ed98193">
5          <property name="ireport.zoom" value="1.0"/>
6          <property name="ireport.x" value="0"/>
7          <property name="ireport.y" value="0"/>
8          <queryString language="SQL">
9                  <![CDATA[select * from orszagok order by kat, orszag]]>
10         </queryString>
11         <field name="ORSZAG" class="java.lang.String"/>
12         <field name="FOVAROS" class="java.lang.String"/>
13         <field name="FOLDR_HELY" class="java.lang.String"/>
14         <field name="TERULET" class="java.math.BigDecimal"/>
15         <field name="ALLAMFORMA" class="java.lang.String"/>
16         <field name="NEPESSEG" class="java.lang.Integer"/>
17         <field name="NEP_FOVAROS" class="java.lang.Integer"/>
18         <field name="AUTOJEL" class="java.lang.String"/>
19         <field name="COUNTRY" class="java.lang.String"/>
20         <field name="CAPITAL" class="java.lang.String"/>
21         <field name="PENZNEM" class="java.lang.String"/>
22         <field name="PENZJEL" class="java.lang.String"/>
23         <field name="VALTOPENZ" class="java.lang.String"/>
24         <field name="TELEFON" class="java.lang.Integer"/>
25         <field name="GDP" class="java.lang.Integer"/>
26         <field name="KAT" class="java.lang.Integer"/>
27         <variable name="katGrpTeruletSum" class="java.math.BigDecimal" resetType="Group" resetGroup="➥
               kategoriaGroup" calculation="Sum">
28                 <variableExpression><![CDATA[$F{TERULET}]]></variableExpression>
29         </variable>
30         <variable name="orszGrpTeruletSum" class="java.math.BigDecimal" resetType="Group" resetGroup="➥
               orszagnevGroup" calculation="Sum">
31                 <variableExpression><![CDATA[$F{TERULET}]]></variableExpression>
32         </variable>
33         <group name="kategoriaGroup">
34                 <groupExpression><![CDATA[$F{KAT}]]></groupExpression>
35                 <groupHeader>
36                         <band height="34">
37                                 <rectangle>
38                                         <reportElement uuid="1fd2eb7a-fd09-4849-9f56-50a078657bec" mode="Opaque"➥
                                               x="0" y="0" width="555" height="34" backcolor="#FF3300"/>
39                                         <graphicElement>
40                                                 <pen lineWidth="0.0"/>
41                                         </graphicElement>
42                                 </rectangle>
43                                 <textField>
44                                         <reportElement uuid="fc5f354c-3535-4035-a396-84619bc058d3" x="121" y➥
                                               ="10" width="100" height="20"/>
45                                         <textElement>
46                                                 <font isBold="true"/>
47                                         </textElement>
48                                         <textFieldExpression><![CDATA[$F{KAT}]]></textFieldExpression>
49                                 </textField>
50                                 <staticText>
51                                         <reportElement uuid="89aa5dcd-568a-4c24-b2e7-cc6c13eda27c" x="30" y="10"➥
                                               width="77" height="20"/>
52                                         <textElement>
53                                                 <font isBold="true"/>
54                                         </textElement>
55                                         <text><![CDATA[Új kategória:]]></text>
56                                 </staticText>
57                         </band>
58                 </groupHeader>
59                 <groupFooter>
60                         <band height="34">
61                                 <rectangle>
62                                         <reportElement uuid="966234e7-d837-409d-812e-b923acae5fdd" x="0" y="0" ➥
                                               width="555" height="34" backcolor="#FF3300"/>
63                                         <graphicElement>
64                                                 <pen lineWidth="0.0"/>
65                                         </graphicElement>
66                                 </rectangle>
67                                 <textField>
68                                         <reportElement uuid="0192bb82-605b-42a6-ad30-0de909bd1bc8" x="90" y="7" ➥
                                               width="100" height="20"/>
69                                         <textElement>
```

```
70                                              <font isBold="true"/>
71                                      </textElement>
72                                      <textFieldExpression><![CDATA[$F{KAT}]]></textFieldExpression>
73                              </textField>
74                              <textField>
75                                      <reportElement uuid="bc399f7c-38d0-4586-8b17-c16d1654e057" x="169" y="7"➥
                                            width="40" height="20"/>
76                                      <textElement textAlignment="Right">
77                                              <font isBold="true"/>
78                                      </textElement>
79                                      <textFieldExpression><![CDATA[$V{kategoriaGroup_COUNT}]]></➥
                                            textFieldExpression>
80                              </textField>
81                              <staticText>
82                                      <reportElement uuid="2922ed41-e2d2-4d63-8691-7e48226b3507" x="231" y="7"➥
                                            width="100" height="20"/>
83                                      <textElement>
84                                              <font isBold="true"/>
85                                      </textElement>
86                                      <text><![CDATA[ország]]></text>
87                              </staticText>
88                              <textField pattern="#,##0.00;-#,##0.00">
89                                      <reportElement uuid="124d15ae-9111-43c6-aad3-69eb40edf4ea" x="431" y="7"➥
                                            width="100" height="20"/>
90                                      <textElement textAlignment="Right">
91                                              <font isBold="true"/>
92                                      </textElement>
93                                      <textFieldExpression><![CDATA[$V{katGrpTeruletSum}]]></➥
                                            textFieldExpression>
94                              </textField>
95                              <staticText>
96                                      <reportElement uuid="89aa5dcd-568a-4c24-b2e7-cc6c13eda27c" x="21" y="7" ➥
                                            width="57" height="20"/>
97                                      <textElement>
98                                              <font isBold="true"/>
99                                      </textElement>
100                                     <text><![CDATA[Kategória:]]></text>
101                             </staticText>
102                             <staticText>
103                                     <reportElement uuid="91eec85b-5f4f-4c52-9478-fd8d364dc218" x="341" y="7"➥
                                            width="70" height="20"/>
104                                     <textElement>
105                                             <font isBold="true"/>
106                                     </textElement>
107                                     <text><![CDATA[Összterület:]]></text>
108                             </staticText>
109                     </band>
110             </groupFooter>
111     </group>
112     <group name="orszagnevGroup">
113             <groupExpression><![CDATA[$F{ORSZAG}.substring( 0, 1 )]]></groupExpression>
114             <groupHeader>
115                     <band height="34">
116                             <rectangle>
117                                     <reportElement uuid="be6745e6-20b7-4564-ba40-85a5629f48d5" x="21" y="0" ➥
                                            width="534" height="34" backcolor="#FFFF00"/>
118                                     <graphicElement>
119                                             <pen lineWidth="0.0"/>
120                                     </graphicElement>
121                             </rectangle>
122                             <staticText>
123                                     <reportElement uuid="3fd04ae0-4e85-45f6-b4f4-f116b8c586e3" x="41" y="10"➥
                                            width="100" height="20"/>
124                                     <textElement>
125                                             <font isBold="true"/>
126                                     </textElement>
127                                     <text><![CDATA[Kezdőbetű:]]></text>
128                             </staticText>
129                             <textField>
130                                     <reportElement uuid="a06845a7-7346-4039-9d04-3293106101b7" x="141" y➥
                                            ="10" width="100" height="20"/>
131                                     <textElement>
132                                             <font isBold="true"/>
133                                     </textElement>
134                                     <textFieldExpression><![CDATA[$F{ORSZAG}.substring( 0, 1 )]]></➥
                                            textFieldExpression>
135                             </textField>
136                             <rectangle>
137                                     <reportElement uuid="114e71bd-1fb9-41e9-b3fd-c0d95107ceb2" x="11" y="0" ➥
                                            width="10" height="34" backcolor="#FFFF00"/>
138                                     <graphicElement>
139                                             <pen lineWidth="0.0"/>
140                                     </graphicElement>
141                             </rectangle>
142                             <rectangle>
143                                     <reportElement uuid="114e71bd-1fb9-41e9-b3fd-c0d95107ceb2" x="0" y="0" ➥
                                            width="10" height="34" backcolor="#FF3300"/>
144                                     <graphicElement>
```

```
145                                         <pen lineWidth="0.0"/>
146                                 </graphicElement>
147                         </rectangle>
148                 </band>
149         </groupHeader>
150         <groupFooter>
151                 <band height="34">
152                         <rectangle>
153                                 <reportElement uuid="5cd22b40-f8a5-4fa2-bde8-efcb1eff0050" x="21" y="0" ➥
                                        width="534" height="34" backcolor="#FFFF00"/>
154                                 <graphicElement>
155                                         <pen lineWidth="0.0"/>
156                                 </graphicElement>
157                         </rectangle>
158                         <textField>
159                                 <reportElement uuid="53fc109c-723b-4c65-99c7-bd5127db3c57" x="169" y="7"➥
                                        width="52" height="20"/>
160                                 <textElement>
161                                         <font isBold="true"/>
162                                 </textElement>
163                                 <textFieldExpression><![CDATA[$V{orszagnevGroup_COUNT}]]></➥
                                        textFieldExpression>
164                         </textField>
165                         <textField pattern="#,##0.00;-#,##0.00">
166                                 <reportElement uuid="0f65e246-c037-4e86-853c-6f73290d177a" x="431" y="8"➥
                                        width="100" height="20"/>
167                                 <textElement>
168                                         <font isBold="true"/>
169                                 </textElement>
170                                 <textFieldExpression><![CDATA[$V{orszGrpTeruletSum}]]></➥
                                        textFieldExpression>
171                         </textField>
172                         <textField>
173                                 <reportElement uuid="7253351b-8151-4f0a-9a7f-e46cfe27c8f5" x="29" y="8" ➥
                                        width="26" height="20"/>
174                                 <textElement>
175                                         <font isBold="true"/>
176                                 </textElement>
177                                 <textFieldExpression><![CDATA[$F{ORSZAG}.substring( 0, 1 )]]></➥
                                        textFieldExpression>
178                         </textField>
179                         <staticText>
180                                 <reportElement uuid="cc31ccae-e521-423a-8234-73c610375318" x="55" y="8" ➥
                                        width="100" height="20"/>
181                                 <textElement>
182                                         <font isBold="true"/>
183                                 </textElement>
184                                 <text><![CDATA[betű:]]></text>
185                         </staticText>
186                         <staticText>
187                                 <reportElement uuid="08ec4b24-2637-4453-a234-9af012fa049e" x="231" y="8"➥
                                        width="100" height="20"/>
188                                 <textElement>
189                                         <font isBold="true"/>
190                                 </textElement>
191                                 <text><![CDATA[ország]]></text>
192                         </staticText>
193                         <staticText>
194                                 <reportElement uuid="a3b0844d-68bb-4b1e-911f-990eea29a03c" x="341" y="7"➥
                                        width="100" height="20"/>
195                                 <textElement>
196                                         <font isBold="true"/>
197                                 </textElement>
198                                 <text><![CDATA[Összterület:]]></text>
199                         </staticText>
200                         <rectangle>
201                                 <reportElement uuid="114e71bd-1fb9-41e9-b3fd-c0d95107ceb2" x="0" y="0" ➥
                                        width="10" height="34" backcolor="#FF3300"/>
202                                 <graphicElement>
203                                         <pen lineWidth="0.0"/>
204                                 </graphicElement>
205                         </rectangle>
206                         <rectangle>
207                                 <reportElement uuid="114e71bd-1fb9-41e9-b3fd-c0d95107ceb2" x="11" y="0" ➥
                                        width="10" height="34" backcolor="#FFFF00"/>
208                                 <graphicElement>
209                                         <pen lineWidth="0.0"/>
210                                 </graphicElement>
211                         </rectangle>
212                 </band>
213         </groupFooter>
214 </group>
215 <background>
216         <band splitType="Stretch"/>
217 </background>
218 <title>
219         <band height="79" splitType="Stretch">
220                 <staticText>
```

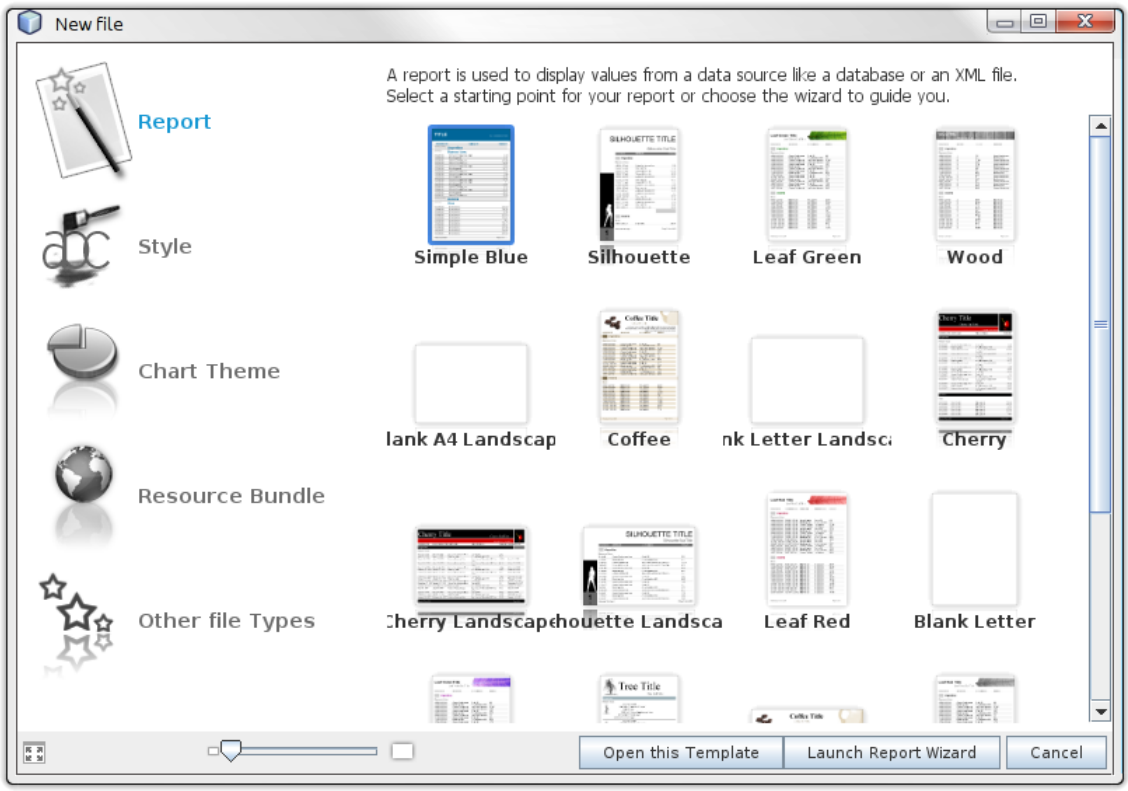
```
221                                 <reportElement uuid="b4880372-f0bd-40c2-98d9-9f0fe433dc21" x="55" y="14" width
                                        ="405" height="31"/>
222                                 <textElement>
223                                         <font size="24"/>
224                                 </textElement>
225                                 <text><![CDATA[Az országok legfontosabb adatai]]></text>
226                         </staticText>
227                 </band>
228         </title>
229         <pageHeader>
230                 <band height="26" splitType="Stretch">
231                         <staticText>
232                                 <reportElement uuid="deb81fed-e34f-4bd0-ae94-0508af09ff25" x="444" y="0" width
                                        ="100" height="20" forecolor="#FF0066"/>
233                                 <textElement textAlignment="Right">
234                                         <font isBold="true"/>
235                                 </textElement>
236                                 <text><![CDATA[Titkos!]]></text>
237                         </staticText>
238                 </band>
239         </pageHeader>
240         <columnHeader>
241                 <band height="23" splitType="Stretch">
242                         <staticText>
243                                 <reportElement uuid="ce8a11bf-2f11-474c-bb67-6e8802752458" x="29" y="2" width
                                        ="100" height="20"/>
244                                 <textElement>
245                                         <font isBold="true"/>
246                                 </textElement>
247                                 <text><![CDATA[Ország]]></text>
248                         </staticText>
249                         <staticText>
250                                 <reportElement uuid="491cf4f0-eeaa-4d9e-909a-8ee49518b24e" x="131" y="0" width
                                        ="100" height="20"/>
251                                 <textElement>
252                                         <font isBold="true" pdfEncoding="Cp1250" isPdfEmbedded="true"/>
253                                 </textElement>
254                                 <text><![CDATA[Főváros]]></text>
255                         </staticText>
256                         <staticText>
257                                 <reportElement uuid="ff2efeea-6633-4b8a-8a04-9328b0ac56da" x="241" y="3" width
                                        ="100" height="20"/>
258                                 <textElement>
259                                         <font isBold="true"/>
260                                 </textElement>
261                                 <text><![CDATA[Földrész]]></text>
262                         </staticText>
263                         <staticText>
264                                 <reportElement uuid="0dca2bff-1bc1-439d-9913-b76866beb8da" x="371" y="3" width
                                        ="81" height="20"/>
265                                 <textElement textAlignment="Right">
266                                         <font isBold="true"/>
267                                 </textElement>
268                                 <text><![CDATA[Terület]]></text>
269                         </staticText>
270                         <staticText>
271                                 <reportElement uuid="f043f1c1-df37-4c5a-9f8a-977b9a2c88d5" x="477" y="0" width
                                        ="67" height="20"/>
272                                 <textElement>
273                                         <font isBold="true"/>
274                                 </textElement>
275                                 <text><![CDATA[GDP]]></text>
276                         </staticText>
277                 </band>
278         </columnHeader>
279         <detail>
280                 <band height="34" splitType="Stretch">
281                         <textField>
282                                 <reportElement uuid="a2856d02-9cef-48e3-9240-cf0a11b76488" x="29" y="12" width
                                        ="100" height="20"/>
283                                 <textElement/>
284                                 <textFieldExpression><![CDATA[$F{ORSZAG}]]></textFieldExpression>
285                         </textField>
286                         <textField>
287                                 <reportElement uuid="5e6ff35e-58db-418b-9bcb-e4f4044185e0" x="131" y="12" width
                                        ="100" height="20"/>
288                                 <textElement/>
289                                 <textFieldExpression><![CDATA[$F{FOVAROS}]]></textFieldExpression>
290                         </textField>
291                         <textField>
292                                 <reportElement uuid="be169129-86a9-47c3-829a-4196decf271c" x="241" y="12" width
                                        ="83" height="20"/>
293                                 <textElement/>
294                                 <textFieldExpression><![CDATA[$F{FOLDR_HELY}]]></textFieldExpression>
295                         </textField>
296                         <textField>
297                                 <reportElement uuid="a583d337-91a5-4ee2-b1cf-3bcd45429f8a" x="362" y="12" width
                                        ="90" height="20"/>
```

```
298                                 <textElement textAlignment="Right"/>
299                                 <textFieldExpression><![CDATA[$F{TERULET}]]></textFieldExpression>
300                             </textField>
301                             <textField>
302                                 <reportElement uuid="ad96b438-b89c-40bf-80bd-53db93c7a2ab" x="477" y="12" width
                                        ="71" height="20"/>
303                                 <textElement/>
304                                 <textFieldExpression><![CDATA[$F{GDP}]]></textFieldExpression>
305                             </textField>
306                             <rectangle>
307                                 <reportElement uuid="114e71bd-1fb9-41e9-b3fd-c0d95107ceb2" x="0" y="0" width
                                        ="10" height="34" backcolor="#FF3300"/>
308                                 <graphicElement>
309                                     <pen lineWidth="0.0"/>
310                                 </graphicElement>
311                             </rectangle>
312                             <rectangle>
313                                 <reportElement uuid="114e71bd-1fb9-41e9-b3fd-c0d95107ceb2" x="11" y="0" width
                                        ="10" height="34" backcolor="#FFFF00"/>
314                                 <graphicElement>
315                                     <pen lineWidth="0.0"/>
316                                 </graphicElement>
317                             </rectangle>
318                     </band>
319             </detail>
320             <columnFooter>
321                     <band height="45" splitType="Stretch"/>
322             </columnFooter>
323             <pageFooter>
324                     <band height="54" splitType="Stretch">
325                         <textField>
326                             <reportElement uuid="238f2dfb-d246-46b7-97b1-90fbdf7d869e" x="241" y="19" width
                                    ="80" height="20"/>
327                             <textElement textAlignment="Right"/>
328                             <textFieldExpression><![CDATA[$V{PAGE_NUMBER}+" /"]]></textFieldExpression>
329                         </textField>
330                         <textField evaluationTime="Report">
331                             <reportElement uuid="45046766-1c1b-4a3b-8380-9ec268d6f7df" x="331" y="19" width
                                    ="40" height="20"/>
332                             <textElement/>
333                             <textFieldExpression><![CDATA[$V{PAGE_NUMBER}]]></textFieldExpression>
334                         </textField>
335                     </band>
336             </pageFooter>
337             <summary>
338                     <band height="42" splitType="Stretch"/>
339             </summary>
340 </jasperReport>
```

A 8-10 sorok a már említett rendezettségű SQL select-et tartalmazzák. A 11-26 sorok annyi mezőt definiálnak, ahány oszlopa van a select parancsnak, noha mi ezekből csak párat használunk. A 27-32 sorok között található a 2 csoportra állított összegző változó. Emlékeztetjük az olvasót, hogy van még 2 rejtett változó is, amik automatikusan jöttek létre a csoporttal, annak sorait számlálják. A 33-111 sorok között a külső, míg a 112-214 sorok között a belső csoportosítás teljes leírása látható. Az XML fájl teljes tartalma az iReport eszközzel készült, azt manuálisan elkészíteni elég nagy munka lett volna. Ez a RAD eszköz az, amit minden esetben használnunk kell, hiszen a formára is jól kinéző és gyorsan elkészíthető riport alapkövetelmény. Ezt megfogadva a következő cikkben az iReport lehetőségeit tekintjük át, azonban megjegyezzük, hogy annak minden részletét nem tudjuk leírni. Célunk egy olyan áttekintés adása, amivel utána mindenki könnyebben használhatja és maga is megtalálhatja a kívánt megoldáshoz vezető utat. Ebben a részben nézzük át a komponens palettát és felfedezzük az eddig használt alapelemeken kívül fellelhető további vizuális komponenseket is.

8. Az iReport - A riport külalakjának megtervezése

Ebben a cikkben a *jrxml* design fájlok elsődleges vizuális szerkesztő programját, az *iReport* eszközt fogjuk bemutatni. A jrxml template-eket mindenképpen ezzel érdemes elkészíteni, mert ez egy RAD eszköz, ami hatékony fejlesztést biztosít. Az ismerkedés közben további érdekes *JasperReports* lehetőségeket is bemutatunk.



8.1. ábra: Az iReport varázsló indítása

Az iReport főbb részei

Egy új riport designt legegyszerűbben az *iReport* varázsló (8.1. ábra, *File* → *New...*) segítségével kezdhetünk el elkészíteni. Ez végigvezet bennünket az

- adatforrás query,
- a mezők kiválasztása,
- és csoportosítás

lépéseken, de előtte kiválaszthatunk egy jelentés kinézet témát is, ha szeretnénk. A riport készítése során van néhány központi ablak, amin keresztül alapvető funkciókat érhetünk el. Ezeket a *Window* főmenüpontból tudjuk kiválasztani és ezek a legfontosabbak (korábban már mindegyik képet láttuk):

- *Report Inspector* (7.1. ábra): A feladata az, hogy a jelentés elemein könnyen tudjunk navigálni, miközben átlátjuk a tel-

jes riport struktúráját. Amikor kiválasztunk egy sort rajta, akkor lehetőségünk van ugyanazt a property halmazt szerkeszteni, mint amikor a rajzvásznon tesszük ugyanezt. Ugyanakkor itt a riport nem vizuális részeit (változók, mezők, stílusok, ...) is elérjük.

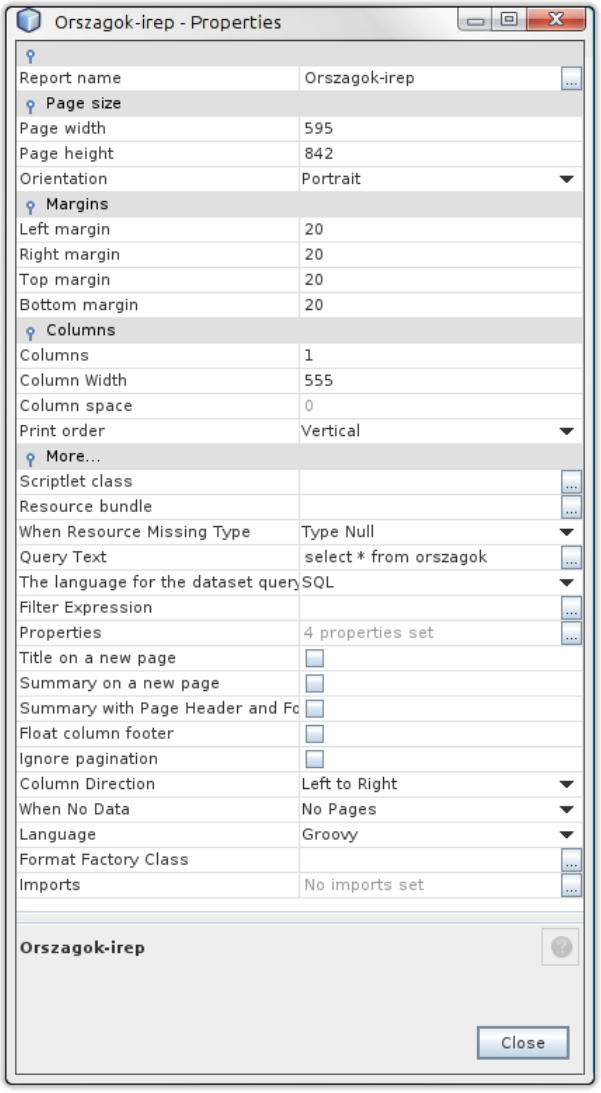
- iReport *Palette*: Az 1.4. ábra a teljes iReport felülettel együtt mutatja. Innen tudjuk azokat a komponenseket elérni, amiket a tervezővászonra tehetünk. Eddig tipikusan a *textField* és *staticText* elemeket használtuk, az alapriportokhoz ezen kívül nem nagyon van másra szükség.
- iReport *Field*: A 2.2. ábra az *ORSZAG* mező jellemzőinek beállítási lehetőségeit jeleníti meg. Ezek legnagyobb része a kinézettel és megjelenéssel foglalkozik.
- *Report Query* (2.1. ábra): Tekintettel arra, hogy minden riport mögött egy adatforrás található, azok megadása és kezelése kiemelten fontos.
- *Report Expression* (6.3. ábra): A 6. cikk részletesen foglalkozik a kifejezések megadási lehetőségeivel.

Riportszintű beállítások

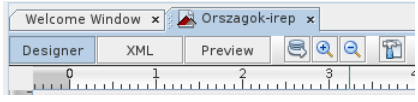
Amikor a *Report Inspector* gyökér elemére (azaz a teljes riportra) jobb egérgombbal kattintunk, akkor kiválaszthatjuk a riportszintű jellemzőket (8.2. ábra). Természetesen minden beállítás itt is a jrxml fájlt változtatja, de lévén az iReport 2-útas eszköz, a jrxml-beli változtatások is látszódnak. A 8.3. ábra egy részlet a felületből, azt mutatja, hogy a munka közben 3 nézet között kapcsolgathatunk:

- *Designer*: A vizuális tervezés nézete.
- *XML*: A jrxml szintű kódírás nézete (8.4. ábra).
- *Preview*: A riport „röptében” való legenerálása és megtekintése.

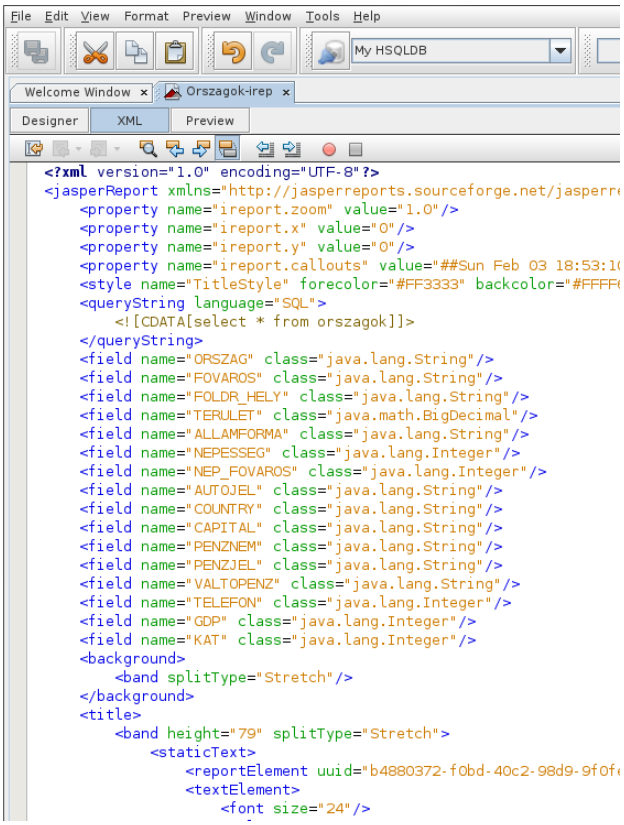
Az egyes jellemzők jelentése általában egyértelmű, ezért most csak azt nézzük meg, amihez kis magyarázatot célszerű tenni.



8.2. ábra: Riportszintű beállítások



8.3. ábra: 2-útas eszköz

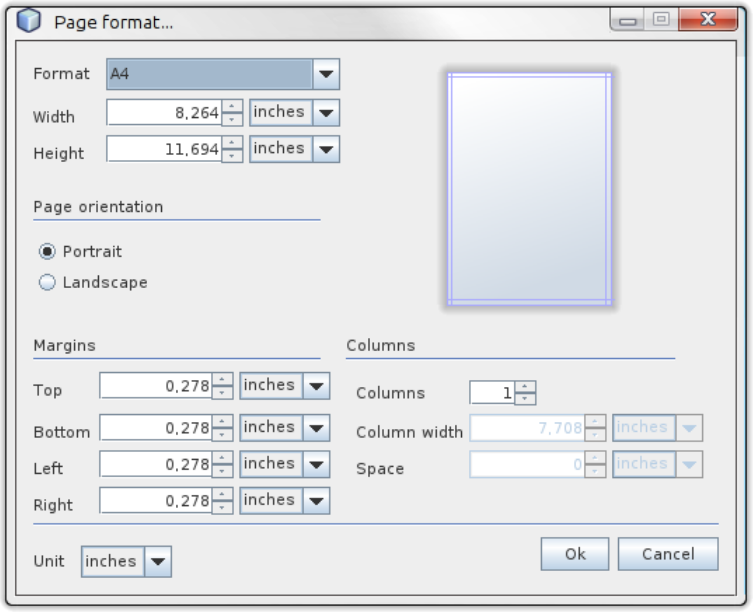


8.4. ábra: A jrxml XML nézet

Nézzük ezeket a jellemzőket!

- *Columns*: A JasperReports képes több oszlopos szedést is megvalósítani, itt ezek száma adható meg. Ide tartozik a *Column space*, ami 2 oszlop távolságát jelenti.
- *Filter Expression*: Az adatforrás rekordjait tudjuk szűrni az itt megadott logikai feltétellel.
- *Title on a new page*: A riport *title* sávját új oldalra nyomtatja.
- *Summary on a new page*: A riport *summary* sávját új oldalra nyomtatja.
- *Ignore Pagination*: Amennyiben igaz, úgy az egész riport 1 oldalra kerül. Ennek a *html* output esetén van meg az igazi értelme.
- *When no data*: Az adatforrás nem mindig hoz sorokat, ebben az esetben elmaradhat a riport generálása. Itt több lehetséges értékből választhatunk, hogy ilyenkor mi történjen.
- *Language*: A kifejezéseket az itt megadott nyelven kell megadni (default érték: *Groovy* nyelv).

Szintén az egész riportra vonatkozó beállítás a *Page Format...* helyi menü kiválasztására bejövő ablak, amit a 8.5. ábrán tekinthetünk meg.

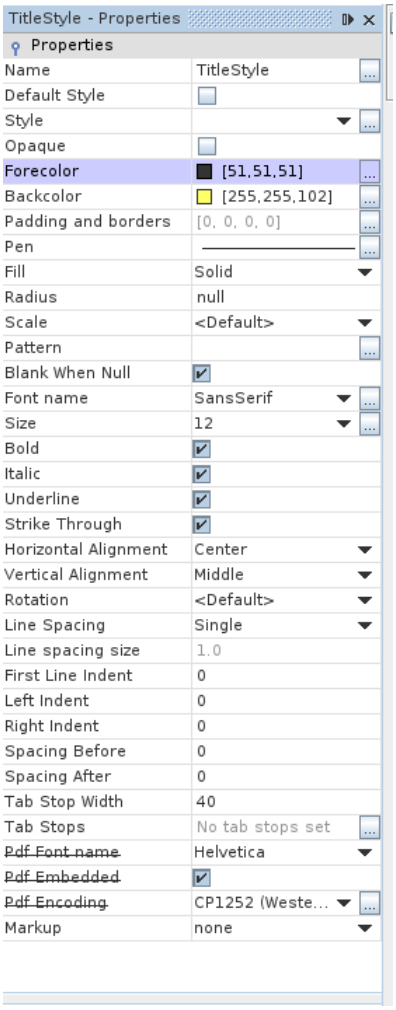


8.5. ábra: Page Format...

A stílusok használata

Az egyes komponensek sok vizuális beállítási lehetőséggel rendelkeznek, ami nagyon rugalmas a riportok külalakjának beállításakor, azonban sok vezérlő esetén már kényelmetlen azok egyenkénti konfigurálása. Mint más környezetekben, itt is a stílus fogalmának bevezetése és használata siet segítségünkre. Amikor a *Report Inspector Styles* csomópontján egy jobb egérklikket csinálunk,

akkor az *Add* menüpont segítségével új stílusneveket hozhatunk létre, amiknek konkrét jellemzőit a 8.6. ábra ablaka segítségével állíthatjuk be. A példában éppen egy *TitleStyle* nevű stílus kialakítását láthatjuk.



8.6. ábra: A stílus property lap

Mindez a következő XML részletet fogja a jrxml fájlba generálni:

```
<style name="TitleStyle" forecolor="#FF3333" backcolor➥
    ="#FFFF66" fill="Solid" hAlign="Center" vAlign="➥
    Middle" fontSize="12"/>
```

A stílusok használata egyszerű, amennyiben egy komponens támogatja, úgy annak van egy *style* nevű jellemzője, ott kell beállítani a megfelelő stílusnevet.

A riport háttér és a vízjel

Lehetőségünk van, hogy a riport hátterét megadjuk, amit leggyakrabban a következő okokból szoktunk használni:

- Látványosabb legyen a jelentés
- Egy vízjelet (watermark) szeretnénk megjeleníteni
- Valamilyen előre tervezett lap lesz a háttér és ahhoz képes rendezzük el a riport elemeit.

Mindezt a inspector *Background* csomópontjánál lehet elvégezni. A háttér egy külön sávon fog megjelenni és jellemzően egy kép komponenst (de akár egyszerű *textField*-et is) teszünk rá, ami a hátteret adja. A kép hozzáadása után a jrxml-ben ez az XML részlet keletkezett:

```
<background>
 <band height="802" splitType="Stretch">
  <image scaleImage="FillFrame">
        <reportElement positionType="Float" x="0" y➥
            ="0" width="555" height="800"/>
    <imageExpression>
        <![CDATA["/home/tanulas/imre-halvany.jpg"]]>
    </imageExpression>
  </image>
 </band>
</background>
```

Van néhány beállítási lehetőség, de mi ezek közül kettőre szeretnénk felhívni a figyelmet:

1. *Print When Expression*: Megadható egy logikai kifejezés, ami dinamikusan értékelődik ki és ettől függően lesz vagy nem lesz háttérkép.
2. *Image Expression*: Amennyiben képet teszünk háttérnek, úgy annak egy fájlban kell lennie, azonban a fájl neve – és ezzel a laponkénti háttérkép – dinamikusan változhat (A példánkban most éppen az *imre-halvany.jpg* fájlt tettük konstans kifejezésként).

A *Preview* fülre kattintva elkészül a riport, aminek egy részletét mutatja a 8.7. ábra.

8.7. ábra: Háttérkép a riporthoz

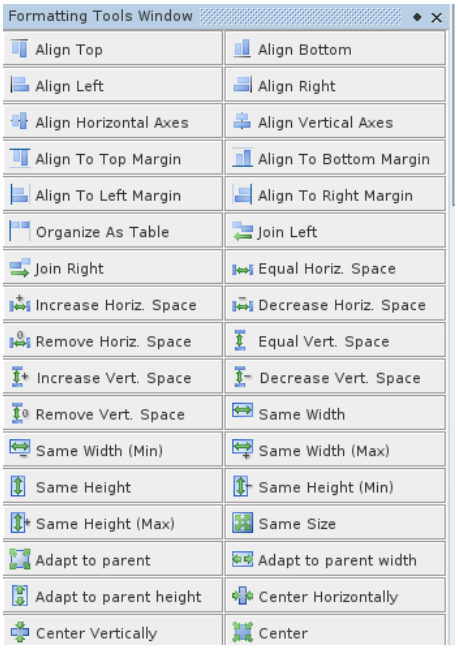
A példában használtuk az image komponens *Scale Image* jellemzőjét, azt *Fill Frame* értékre tettük, aminek hatására a kép kitöltötte a hátteret. Természetesen a képet megadott pozícióra és méretben is kitehetjük háttérként.

A riport elemek formázása

A formázás leggyakoribb műveletei a következőek:

- pozíció megadás,
- méretezés,
- előtér és háttér színek megadása,
- egymáshoz képest való elhelyezés.

Amikor kijelölünk egy komponenst, akkor a *property sheet*-jén mindezt megadhatjuk, aminek jelentős része a *jrxml reportElement* tag attribútumaként fog megjelenni. Ez a címke minden komponens fontos része. A *Window* főmenüből választható ki a *Formatting Tools Window* ablak (8.8. ábra), amivel igen kifinomultan lehet a komponensek lapon belüli és egymáshoz képesti elrendezését gyorsan beállítani. Amikor csinosítjuk jelentésünket, erről az eszközről soha nem szabad megfeledkeznünk.



8.8. ábra: Gyors elrendezési lehetőség

A keretek használata

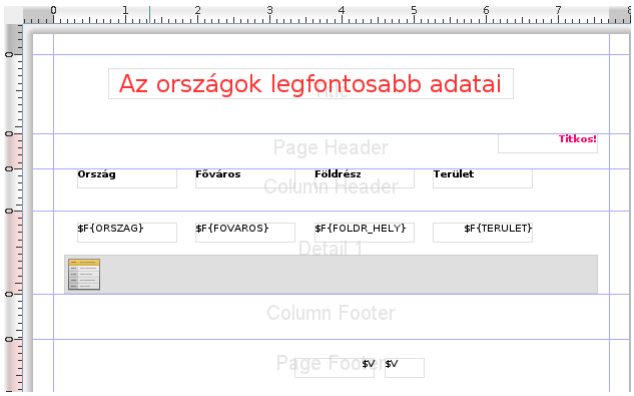
A keretek bevezetése tovább javította a JasperReport template tervezési lehetőségeit. A palettáról egy keretet elhelyezve egy konténert kapunk, ami egy téglalap alakú rész és tetszőleges számú komponenst tehetünk bele, aminek a stílusa és kezelése a keretre nézve egységes lesz. Az iReport-ban a keretet egy egységként tudjuk mozgatni, pozicionálni és színezni, noha több komponenst tartalmaz. A jrxml fájlban a keret a *<frame>* tag lesz, ez fogadja be a további elemeket (például a *textElement* komponenst):

```
<frame>
   <reportElement x="42" y="23" width="337" height➥
        ="114"/>
   <staticText>
     <reportElement x="46" y="31" width="173" height➥
          ="20"/>
     <textElement/>
       <text><![CDATA[Az országok ezek voltak]]></text➥
            >
     </staticText>
</frame>
```

9. Az alriportok (subreports) használata

A JasperReports egy hatékony és kellemes lehetősége az alriportok használata. Képzeljük el, hogy egy olyan jelentést kell készítenünk aminek a logikai szerkezete inkább több riport együtteseként adódik. Ekkor a design megfogalmazása sokkal egyszerűbb a subreport mechanizmus használattal, ami egy kiváló absztrakciós szintnek mutatkozik.

A *subreport* (alriport) egy teljes riport egy másikba ágyazva. A beágyazott riportot emiatt *detail* vagy *child* szóval, míg a beágyazó főriportot *master* vagy *parent* jelzővel szokták illetni. Bármilyen, már elkészült riport lehet subreport, a lényeg csak az, hogy a főriport hívja meg annak a hatására, hogy a komponens palettáról kihelyeztünk a rajzvászonra egy *Subreport* elemet.



9.1. ábra: A master/parent riport

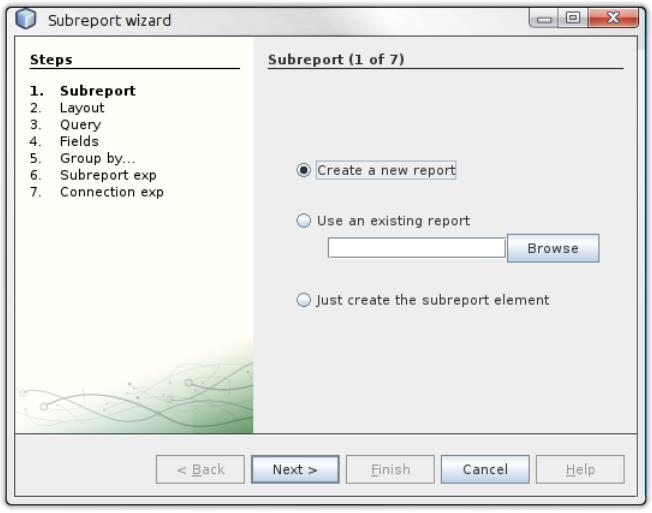
Mikor használjunk alriportokat?

Van néhány szituáció, amikor érdemes meggondolni az alriport használatának lehetőségét, mert segítheti vagy éppen megvalósíthatóvá teszi a feladat elkészítését:

- Amikor Master/Detail adatszerkezetben tárolt adatokat szeretnénk jelentés formájában megjeleníteni.
- Amikor egymással kicsi vagy semmilyen kapcsolatban lévő táblák (adatforrások) tartalmát együtt kell megmutatnunk.
- Az adatokat egy jelentésen belül különféle nézetekben kell prezentálnunk.

Az alriport használata

Ebben a pontban bemutatjuk azt a technikát, hogy egy meglévő riportba (9.1. ábra) hogy ágyazhatunk be egy tetszőleges másikat. A komponens palettáról húzzuk be a *Subreport* elemet, esetünkbe most a master riport *detail* sávjára. Nincs megkötés arra, hogy az alriportot melyik sávra tesszük, de most a feladatunk az lesz, hogy minden egyes országhoz alriportként megvalósítva még megjelenítjük a pénznem, államforma, autó jel és telefon információkat is, emiatt a subreport természetes helye ezen a sávon van. Az ábrán a kis riport ikonnal felszerelt szürke téglalap jelképezi a *Subreport* komponenst.



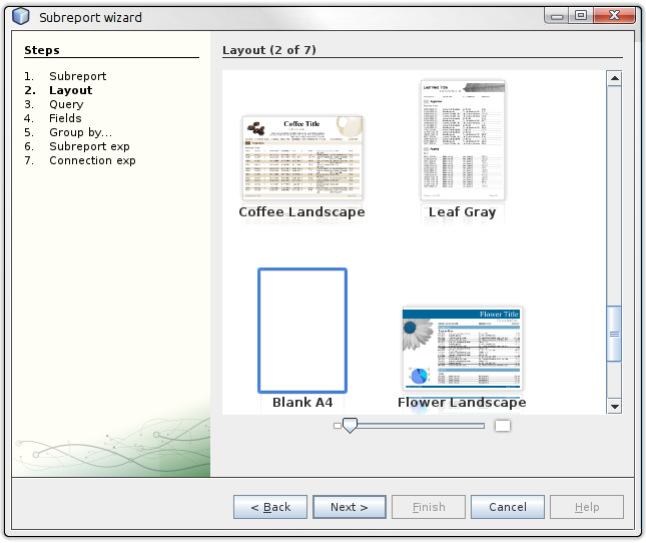
9.2. ábra: Subreport varázsló - 1

A komponens elhelyezése után automatikusan elindul a *Subreport wizard*, aminek most nézzük meg együtt mind a 7 lépését! Az induló ablakban (9.2. ábra) eldönthetjük, hogy az alábbi 3 lehetőségből melyiket szeretnénk:

1. Egy új riport létrehozása, ami az alriportunk lesz majd.

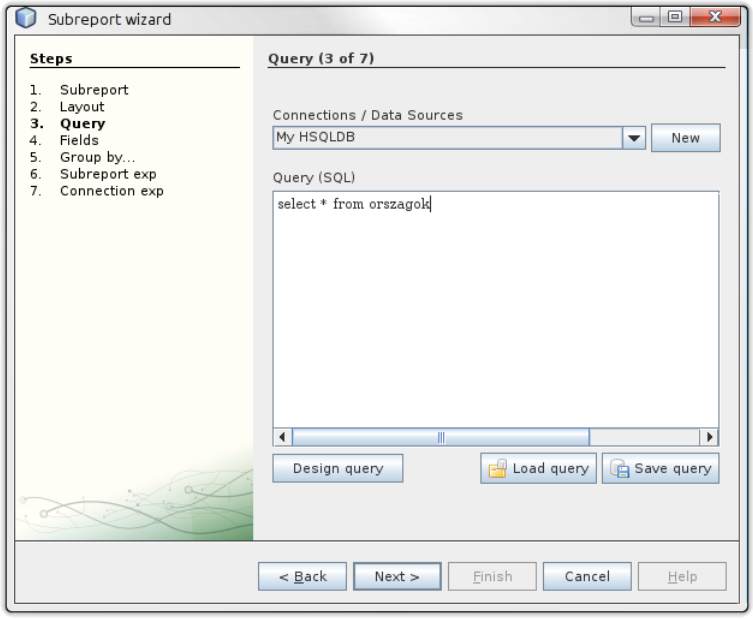
2. Egy már korábban elkészített jelentés template-et szeretnénk használni subreport-ként vagy

3. A master riportba most kitett *Subreport* elemhez most még nem akarunk alriportot megadni.



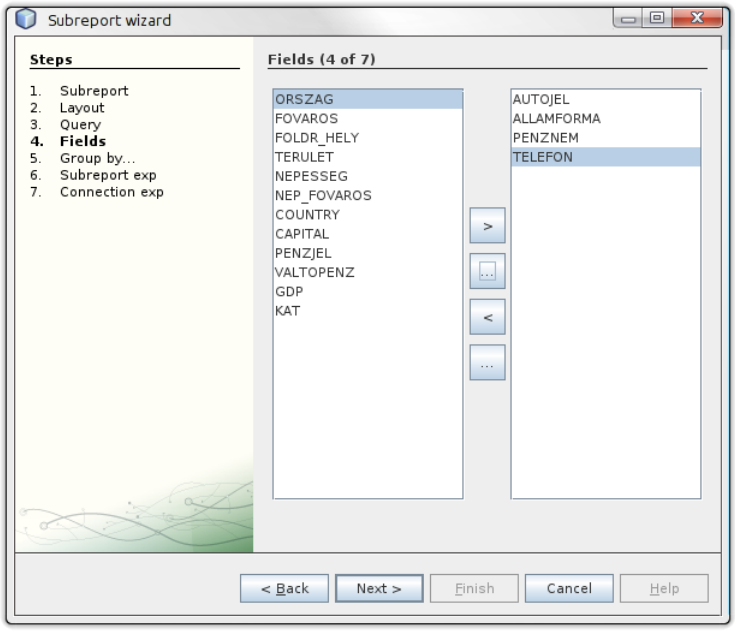
9.3. ábra: Subreport varázsló - 2

A *Next* gomb a következő ablakhoz (9.3. ábra) vezet, ami a már ismerős *Layout* kiválasztó funkció. Mi a *Blank A4* lehetőséget választottuk, majd továbbmentünk a következő panelre (9.4. ábra). Itt adhatjuk meg az alriport adatforrását, ami nekünk még mindig a HSQLDB adatbázisunkra kiadott select.

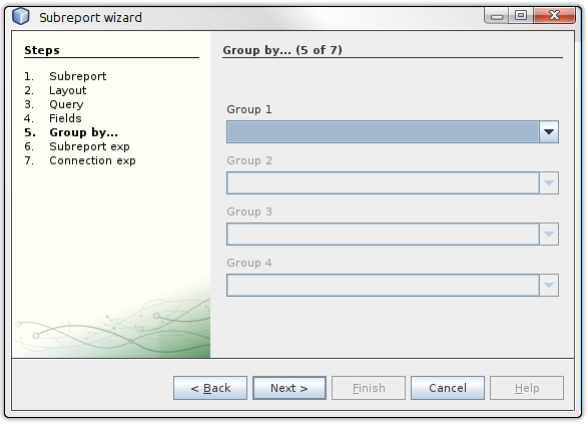


9.4. ábra: Subreport varázsló - 3

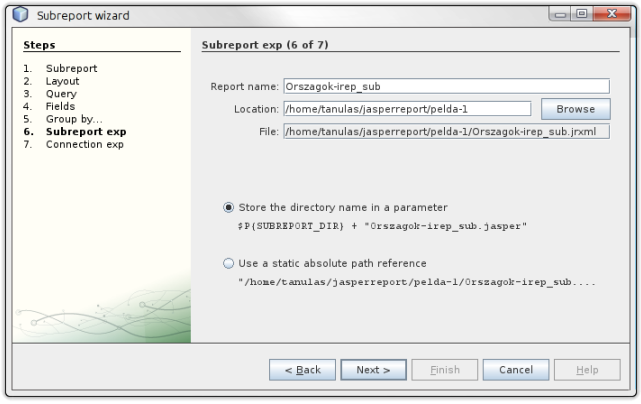
Ezután *Next* és áttérünk a következő képernyőre (9.5. ábra), ahol a select összes oszlopa felkínálódik, azonban mi csak azt a 4 oszlopot tettük át a jobb oldalra, amit az alriportban szerepeltetni szeretnénk.



9.5. ábra: Subreport varázsló - 4

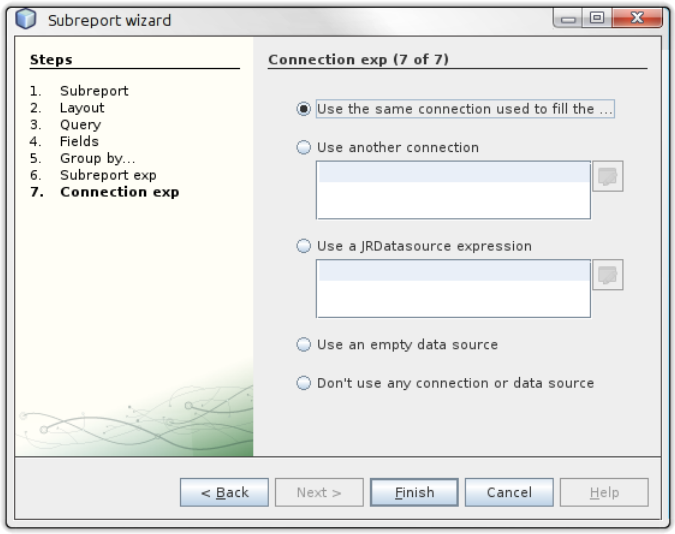


9.6. ábra: Subreport varázsló - 5

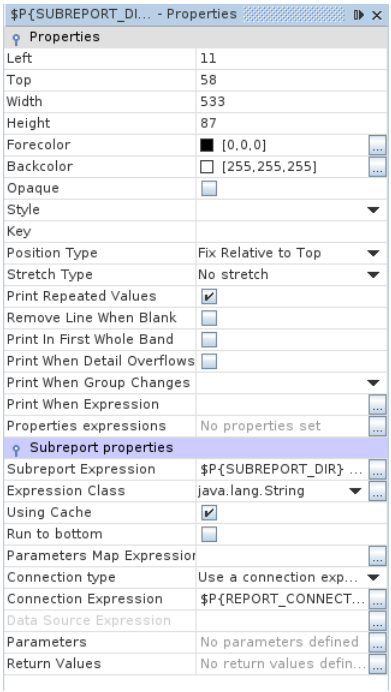


9.7. ábra: Subreport varázsló - 6

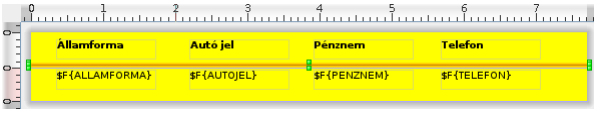
A 9.6. ábra ablaka most nem fontos nekünk, mert nem szeretnénk a child riportunkba semmilyen csoportosítást, bár volna rá lehetőség. Emiatt menjünk a következő, 9.7. ábra ablakára, ahol az alriportnak egy fájl nevet adhatunk és rendelkezhetünk arról, hogy mely könyvtárba tárolódjon. Az utolsó varázsló lépéshez ismét nyomjunk *Next* gombot (9.8. ábra), ahol az adatforrás elérhetőségét lehet beállítani. Az ábrán is látszik, hogy mi ugyanazt a *Connection*-t fogjuk használni, amit a parent riport is használ. A *Finish* gomb megnyomására végül legenerálódik a *subreport* induló váza. Erre kitettük a generált 4 mezőt, kitöröltük a felesleges sávokat, azaz csak a *column header* és *detail* maradt meg, amiket egy-egy *rectangle* komponens segítségével sárgára színeztünk, így kaptuk a 9.10. ábrán látható design kinézetet.



9.8. ábra: Subreport varázsló - 7



9.9. ábra: A subreport komponens jellemzői



9.10. ábra: Az alriport

A következő lépés az lesz, hogy a master riport *Subreport* komponensének jellemzőit kell beállítanunk, amit a 9.9. ábra ablakán keresztül érhetünk el. Itt a *Subreport properties* rész az, ami kimondottan alriport komponens specifikus beállító cellákat tartalmaz. Nézzük meg őket egyesével:

- *Subreport Expression*: Értéke *$P{SUBREPORT_DIR} + "Orszagok-irep_sub.jasper"*.
- *Connection type*: 3 értékből lehet választani aszerint, hogy connection-t vagy adatforrást szeretnék használni, de az is lehet, hogy nem szeretnénk adatokat kapni a riporthoz.
- *Connection Expression*: Az érték most *$P{REPORT_CONNECTION}*, azaz a mester riport kapcsolata.
- *Parameters*: Itt adhatjuk meg, hogy milyen néven melyik paramétereket szeretnénk átadni a subreport részére-

A fentieket a jrxml fájl is tartalmazza, amiket a 9-1. és 9-2. Programlistákon láthatunk. A 9-1. Programlista az alriport releváns részét mutatja. A 3. sor a *parOrszag* paramétert határozza meg, mert ez az érték majd a főriporttól fog jönni. A szerepe az, hogy a 6. sor *select* utasításának *where* feltételét kiegészítse. Ezzel biztosítható, hogy az alriport mindig csak a paraméterül kapott országra fog lefutni. A 9-2. Programlista a főriport lényeges részeit mutatja. A *detail* sávnál észrevehetjük a 14-27 sorok között megadott *<subreport>* tag-et. Erre úgy kell gondolni, mint egy rutinhívásra, de most ez egy alriport meghívása lesz. Az egyetlen paraméter átadása a 16-20 sorok között történik, ez most éppen a *$F{ORSZAG}* pillanatnyi értéke. Tekintettel arra, hogy a *Subreport* komponens a *detail* sávon van, ezért annak meghívására annyi alkalommal fog sor kerülni, ahány sor kerül a főriportba. A 21-23 sorok közötti *connectionExpression* tartalmazza a hivatkozást a főriport kapcsolatára, amit ily módon tudunk átpasszolni az alriportnak. Végül a *subreportExpression* (24-26 sorok) tartalmazza a meghívandó alriport lefordított jasper fájljának az elérhetőségét, hiszen erről tudnunk kell, hogy hol van. Itt HTTP URL is megadható.

A főriport *Preview* fülre kattintva a 9.11. ábra mutatja az elkészült jelentésünket, aminek sárga részei a beágyazott alriportot jelentik.

```
1  // 9-1. Programlista: Az Orszagok-irep_sub.jrxm - részlet
2  ...
3  <parameter name="parOrszag" class="java.lang.String"/>
4
5  <queryString language="SQL">
6          <![CDATA[select * from orszagok where orszag=$P{parOrszag}]]>
7  </queryString>
8
9  <field name="AUTOJEL" class="java.lang.String"/>
10 <field name="ALLAMFORMA" class="java.lang.String"/>
11 <field name="PENZNEM" class="java.lang.String"/>
12 <field name="TELEFON" class="java.lang.Integer"/>
13 ...
```

```
1  // 9-2. Programlista: Az Orszagok-irep.jrxm - részlet
2  ...
3  <style name="TitleStyle" forecolor="#FF3333" backcolor="#FFFF66" fill="Solid" hAlign="Center" vAlign="Middle" ➥
       fontSize="12"/>
4    <parameter name="SUBREPORT_DIR" class="java.lang.String" isForPrompting="false">
5      <defaultValueExpression><![CDATA["/home/tanulas/jasperreport/pelda-1/"]]></defaultValueExpression>
6    </parameter>
7
8  <queryString language="SQL">
```

```
9     <![CDATA[select * from orszagok]]>
10  </queryString>
11  ...
12  <detail>
13  ...
14       <subreport>
15          <reportElement x="11" y="44" width="533" height="39"/>
16          <subreportParameter name="parOrszag">
17              <subreportParameterExpression>
18                  <![CDATA[$F{ORSZAG}]]>
19              </subreportParameterExpression>
20          </subreportParameter>
21          <connectionExpression>
22              <![CDATA[$P{REPORT_CONNECTION}]]>
23          </connectionExpression>
24          <subreportExpression>
25             <![CDATA[$P{SUBREPORT_DIR} + "Orszagok-irep_sub.jasper"]]>
26          </subreportExpression>
27       </subreport>
28  ...
29  </detail>
30  ...
```

Az országok legfontosabb adatai

Titkos!

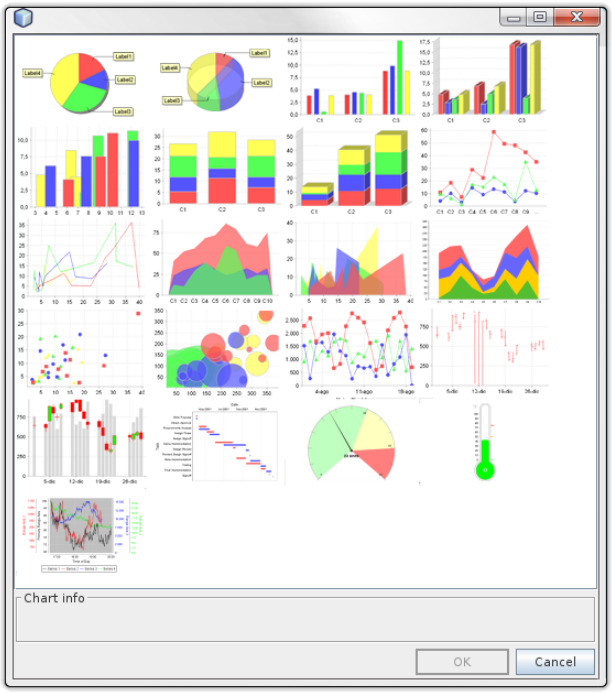
Ország	Főváros	Földrész	Terület
SPANYOLORSZÁG	MADRID	Dél-Európa:Ibériai-félsziget	504782.00
Államforma	**Autó jel**	**Pénznem**	**Telefon**
alkotmányos	E	euró	34
PORTUGÁLIA	LISSZABON	Dél-Európa:Ibériai-félsziget	92082.00
Államforma	**Autó jel**	**Pénznem**	**Telefon**
köztársaság	P	euró	351
FRANCIAORSZÁG	PÁRIZS	Nyugat-Európa	547026.00
Államforma	**Autó jel**	**Pénznem**	**Telefon**
köztársaság	F	euró	33
NAGY-BRITANNIA	LONDON	Európa:Britt-szigetek	244046.00

9.11. ábra: Az elkészült riport

10. Grafikonok használata

Egy korszerű jelentés az adatok szöveges elrendezése, csoportosítása és összegzése mellett grafikonokat is tartalmazhat. A leíró statisztika sokféle adatábrázolási formát ismer, amiknek jelentős részét a JasperReports is tudja. Ezzel segíteni tudjuk az adatok értelmezését és a közöttük lévő rejtett összefüggések feltárását. Ebben a cikkben ezeket a lehetőségeket tekintjük át.

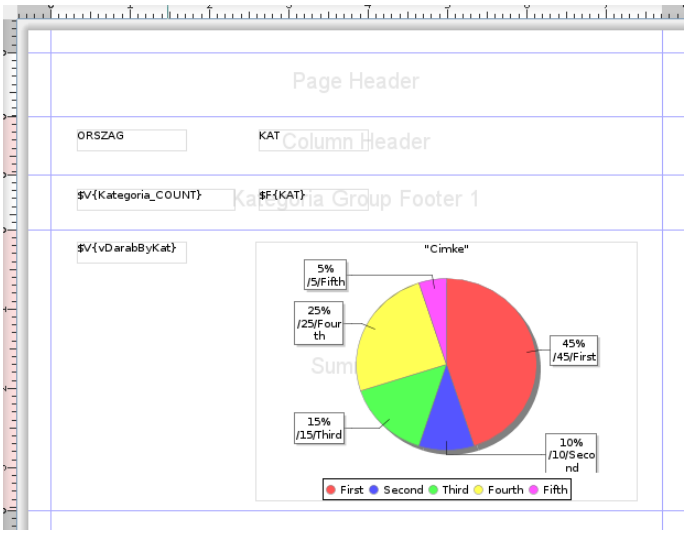
A JasperReports képes a riportokba grafikonokat is generálni. Mindezt az ismert *JFreeChart* library (webhelye: `http://www.jfree.org/jfreechart/`) segítségével teszi. Előnyös és érdekes lehetőség, hogy a grafikonok testreszabását egészen a *JFreeChart* csomag közvetlen használatáig visszanyúlva is el lehet végezni, amennyiben implementáljuk a *JRChartCustomizer* Java interface-t. Ez utóbbi lehetőségre rövidesen példát is mutatunk.



10.1. ábra: Az elérhető grafikon típusok

Egy új grafikont a palettáról a tervező vászonra behúzott *Chart* komponens segítségével tehetünk be a jelentésbe, amikor a 10.1. ábra ablaka jelenik meg. Miután kiválasztottuk a grafikon típusát (például kördiagram), a komponens felkerül tervező felületre és azt tovább konfigurálhatjuk (példa: 10.2. ábra).

Kördiagram

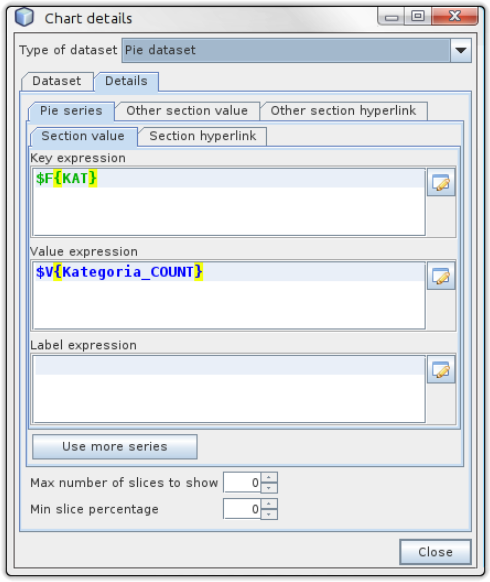


10.2. ábra: Kördiagram

A kördiagram a statisztikai sokaság egymástól megkülönböztetendő elemtípusainak a relatív gyakoriságát (eloszlását) képes szemléletesen ábrázolni. A példánk ennek megfelelően legyen az, hogy az egyes országok kategóriabesorolása szerint mutatjuk meg azok gyakoriságának alakulását. Annyi körcikk lesz, ahány kategória (az *orszagok* tábla jelenleg 3 kategóriát tartalmaz). A körcikkek mérete az adott kategóriába tartozó országok számából fog jönni. Ennek megfelelően ehhez ezt az adatforrás SQL *select*-et fogjuk

használni, azaz az eredménytábla rendezett kell legyen a *KAT* oszlopra (10-1. Programlista 11-13 sorok):

```
select * from orszagok order by kat
```



10.3. ábra: A kördiagram adathalmaza

A 10.2. ábra mutatja az elkészített elrendezést, aminek részletei:

- *A* kördiagram a *summary* sávba került.
- Felvettünk egy *Kategoria* nevű group-ot (35-41 sorok), ugyanis a kategóriákba tartozó országok számát csoportszinten meg is szeretnénk jeleníteni, amit a kördiagramba tett értékkel hasonlítunk majd össze. A csoportnak csak a *footer* sávját hagytuk meg, ott láthatóak majd a részösszegek.
- A *detail* sávra nincs szükségünk, azt töröltük. Ennek megfelelően a riportba sem jelennek meg az adatsorok, de a csoport összegek igen.
- Definiáltunk egy *vDarabByKat* nevű változót (31-33 sorok), ami számolja az összes országot, így a *summary* sávba a diagram mellé kikerül az összes ország száma is.

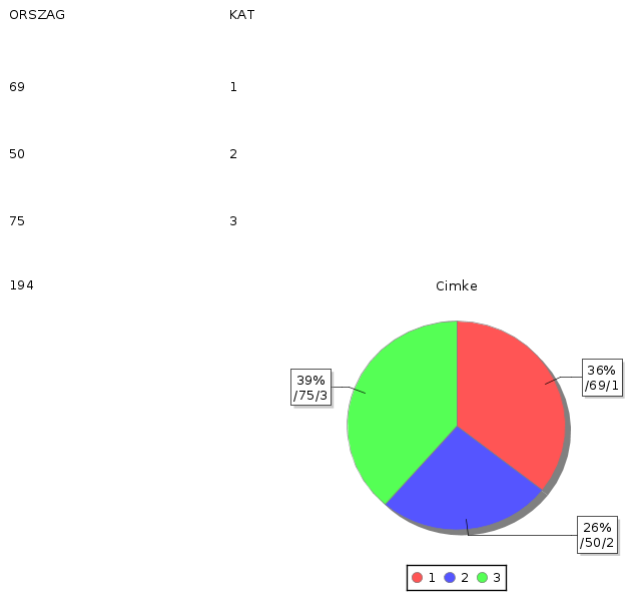
A kördiagramra jobb egérgombbal kattintva elérhető a *Chart Data* helyi menü, ahonnan a 10.3. ábra beállító ablakához jutunk, amennyiben a *Details* fület választjuk. A *Key expression* egy olyan kifejezés, aminek minden egyes eltérő értéke egy-egy körcikket fog eredményezni a kördiagramon. Ez esetünkben most *$F{KAT}*, azaz ahogy jönnek a sorok az adatforrásokból, úgy mindig a bennük lévő kategória kód szerint lesz építve a diagram. A *Value expression* azt adja meg, hogy a különböző körcikkekhez az értékeket milyen kifejezéssel halmozzuk hozzá. Itt most a *Kategoria* csoporthoz automatikusan létrejött *Kategoria_COUNT* számláló változót adtuk meg. A körcikkekhez tartozó feliratot most nem itt adtuk meg, hanem a 94. sorban látható *{2}/{1}/{0}* helytartó szimbólumokkal. Azonban megjegyezzük, hogy amennyiben itt adjuk meg, úgy annak nagyobb a prioritása a megjelenítés szempontjából. A kördiagram teljes megadását a 81-98 sorok között tanulmányozhatjuk.

```
1  // 10-1. Programlista: Az Orszagok-Chart.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/➥
       XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://➥
       jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-Chart" language="groovy" pageWidth="595"➥
        pageHeight="842" columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20" uuid=➥
       "51030eb1-4516-48e2-a63d-04d9526ab437">
5          <property name="ireport.zoom" value="1.0"/>
6          <property name="ireport.x" value="0"/>
7          <property name="ireport.y" value="0"/>
8
9          <import value="org.cs.test.*"/>
10
11         <queryString>
12                 <![CDATA[select * from orszagok order by kat]]>
13         </queryString>
```

```
14          <field name="ORSZAG" class="java.lang.String"/>
15          <field name="FOVAROS" class="java.lang.String"/>
16          <field name="FOLDR_HELY" class="java.lang.String"/>
17          <field name="TERULET" class="java.math.BigDecimal"/>
18          <field name="ALLAMFORMA" class="java.lang.String"/>
19          <field name="NEPESSEG" class="java.lang.Integer"/>
20          <field name="NEP_FOVAROS" class="java.lang.Integer"/>
21          <field name="AUTOJEL" class="java.lang.String"/>
22          <field name="COUNTRY" class="java.lang.String"/>
23          <field name="CAPITAL" class="java.lang.String"/>
24          <field name="PENZNEM" class="java.lang.String"/>
25          <field name="PENZJEL" class="java.lang.String"/>
26          <field name="VALTOPENZ" class="java.lang.String"/>
27          <field name="TELEFON" class="java.lang.Integer"/>
28          <field name="GDP" class="java.lang.Integer"/>
29          <field name="KAT" class="java.lang.Integer"/>
30
31          <variable name="vDarabByKat" class="java.lang.Integer" calculation="Sum">
32                  <variableExpression><![CDATA[1]]></variableExpression>
33          </variable>
34
35          <group name="Kategoria">
36                  <groupExpression><![CDATA[$F{KAT}]]></groupExpression>
37                  <groupFooter>
38                          <band height="50">
39                                  <textField>
40                                          <reportElement uuid="79aa4ae2-75dd-4ffa-a85d-4953855c6155" x="24" y="13"➥
                                                width="144" height="20"/>
41                                          <textElement/>
42                                          <textFieldExpression><![CDATA[$V{Kategoria_COUNT}]]></➥
                                                textFieldExpression>
43                                  </textField>
44                                  <textField>
45                                          <reportElement uuid="e1a29491-1e8b-403e-964b-1e6f69d9099e" x="189" y="13➥
                                                " width="100" height="20"/>
46                                          <textElement/>
47                                          <textFieldExpression><![CDATA[$F{KAT}]]></textFieldExpression>
48                                  </textField>
49                          </band>
50                  </groupFooter>
51          </group>
52
53          <pageHeader>
54                  <band height="58"/>
55          </pageHeader>
56
57          <columnHeader>
58                  <band height="53">
59                          <staticText>
60                                  <reportElement uuid="51f6734d-3ab5-40f3-85a4-8658815af27c" x="24" y="12" width="➥
                                        100" height="20"/>
61                                  <textElement/>
62                                  <text><![CDATA[ORSZAG]]></text>
63                          </staticText>
64                          <staticText>
65                                  <reportElement uuid="6af86c26-5f37-4659-a9a6-b62643af93ae" x="189" y="12" width=➥
                                        "100" height="20"/>
66                                  <textElement/>
67                                  <text><![CDATA[KAT]]></text>
68                          </staticText>
69                  </band>
70          </columnHeader>
71          <columnFooter>
72                  <band/>
73          </columnFooter>
74          <summary>
75                  <band height="255">
76                          <textField>
77                                  <reportElement uuid="e565cb96-04c6-460c-8b77-3a1f0096a3f7" x="24" y="11" width="➥
                                        100" height="20"/>
78                                  <textElement/>
79                                  <textFieldExpression><![CDATA[$V{vDarabByKat}]]></textFieldExpression>
80                          </textField>
81                          <pieChart>
82                                  <chart evaluationTime="Report" customizerClass="org.cs.test.MyPieChartCustomizer➥
                                        ">
83                                          <reportElement uuid="fa7fee62-d2f6-4419-974e-55fd27e0bcbd" stretchType="➥
                                                RelativeToBandHeight" x="186" y="11" width="347" height="236"/>
84                                          <chartTitle>
85                                                  <titleExpression><![CDATA["Cimke"]]></titleExpression>
86                                          </chartTitle>
87                                          <chartSubtitle/>
88                                          <chartLegend/>
89                                  </chart>
90                                  <pieDataset>
91                                          <keyExpression><![CDATA[$F{KAT}]]></keyExpression>
92                                          <valueExpression><![CDATA[$V{Kategoria_COUNT}]]></valueExpression>
93                                  </pieDataset>
```

```
94                                  <piePlot isCircular="false" labelFormat="{2}/{1}/{0}">
95                                      <plot/>
96                                      <itemLabel/>
97                                  </piePlot>
98                              </pieChart>
99                      </band>
100             </summary>
101     </jasperReport>
```

Az *iReport*-ban álljunk a *Preview* fülre, azaz készítsük el a jelentést, aminek eredményét a 10.4. ábra mutatja.



10.4. ábra: Az elkészült riport

Látható, hogy nincsenek részletező adatok, azonban az egyes csoportokhoz, azaz kategóriákhoz tartozó ország darabszámok fel vannak tüntetve, ez a mostani konkrét esetre így alakult:

- 1. kategória → 69 darab ország
- 2. kategória → 50 darab ország
- 3. kategória → 75 darab ország

Összesen 194 ország van az *orszagok* táblában, ezt a *summary* sáv mutatja. A kördiagram pontosan ezt az eloszlást mutatja. Az egyes címkék jelentése a megadott *{2}/{1}/{0}* felépítésnek megfelelően ez:

- {2} → A körcikknek megfelelő százalék
- {1} → A halmozott összeg
- {0} → A körcikk elem kulcsneve

Például a kék körcikk az országok 26%-át tartalmazza, ami a 2. kategóriás országokat jelenti és ebből 50 darab van. Van azonban a diagrammal egy kis gond. A százalékok összege 101%, ami érthető, de nem szép. Amennyiben a tizedesjegyeket is feltüntettük volna, úgy ezzel nem lenne gond, de egészre kerekítve pont az lenne a véletlen, ha pont kijött volna a 100%. A következő részben bemutatjuk a %-ok pontosabb kiírási lehetőségét, amihez felhasználjuk az alkalmat a *JRChartCustomizer* bemutatására is.

Kördiagram - testre szabva

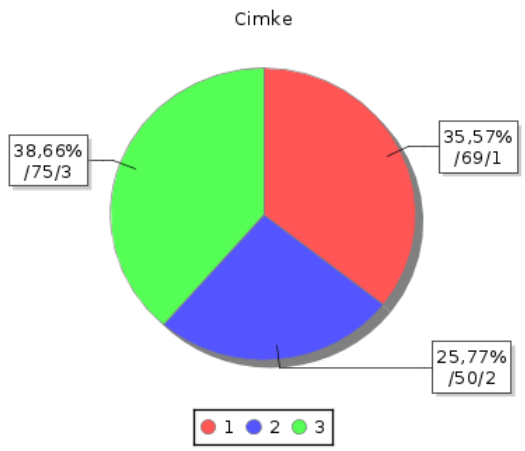
A *JRChartCustomizer* interface egy annyira rugalmas lehetőség, hogy a *JFreeChart* könyvtár teljes tudását elérhetővé teszi. A JasperReports persze igyekszik a tervezőfelületre kivezetni a grafikonok beállítási lehetőségeit, de ez nyilván nem sikerülhet teljes mértékben. Erre jó példa, hogy a körcikkek % értékének tizedesjegyeit sem állíthatjuk be, de ekkor kisegíthet minket egy osztály, ami implementálja ezt az interface-t. Az osztályunk neve most *MyPieChartCustomizer* lesz és így mondjuk meg a jrxml *chart* tagnél, hogy a generátor használja őt:

```
...
<chart evaluationTime="Report" customizerClass="org.cs➥
    .test.MyPieChartCustomizer">
...
```

A 10-2. Programlista a class megvalósítását tartalmazza, láthatjuk, hogy egyáltalán nincs szó bonyolult dolgokról. Egyedül a *customize()* metódus megvalósításáról szól ez a technika,

aminek az az ötlete, hogy itt az implementációhoz ajándékba kapjuk a *JFreeChart* (JFreeChart reprezentáció) és *JRChart* (JasperReport reprezentáció) konkrét objektumokat. Ezzel elérhetjük a *jasperChart* objektum által tárolt beállításokat és manipulálhatjuk a *chart* objektumot is. Számunkra a 25. sor hordozza a lényeges lépést, ahol beállítjuk, hogy a százalékok 2 tizedesjeggyel legyenek kiírva.

```
1  // 10-2. Programlista: MyPieChartCustomizer.java
2
3  package org.cs.test;
4
5  import net.sf.jasperreports.engine.JRChart;
6  import net.sf.jasperreports.engine.JRChartCustomizer;
7  import org.jfree.chart.JFreeChart;
8  import org.jfree.chart.labels.StandardPieSectionLabelGenerator;
9  import org.jfree.chart.plot.PiePlot;
10 import org.jfree.chart.plot.Plot;
11
12 /**
13  *
14  * @author inyiri
15  */
16 public class MyPieChartCustomizer implements JRChartCustomizer
17 {
18     @Override
19     public void customize(JFreeChart chart, JRChart jasperChart)
20     {
21         Plot plot = chart.getPlot();
22         if (plot instanceof PiePlot)
23         {
24             PiePlot piePlot = (PiePlot) plot;
25             ((StandardPieSectionLabelGenerator) piePlot.getLabelGenerator()).getPercentFormat().➥
                   setMaximumFractionDigits(2);
26         }
27     }
28 }
```



10.5. ábra: Javított kördiagram

Futtassuk le a riportot, az eredményt a 10.5. ábra mutatja. A százalékokat összeadva most kijön a 100% is, ugyanakkor látjuk, hogy itt miért kerekített a generátor mindenhol felfelé.

Kördiagram csoportszinten

Az előző kördiagram a *summary* sávban jelent meg. A következőekben a kategória csoportokhoz is rendelünk diagramokat azért, hogy egy-egy kategórián belül láthassuk azok terület csoportonkénti megoszlását is. Az elkészítendő riport továbbra sem tartalmaz részletező sávot (mert most csak a riport szerkezetébe illesztett grafikont szeretnénk bemutatni), de 2 egymásba ágyazott *group*-ot igen, ahogy azt a 10.6. ábrán

látható *Report inspector* részlet is mutatja.

10.6. ábra: 2 group egymásba ágyazva

Esetünkben most a területnagyság (neve: *TeruletNagysag*) csoportot a következő kifejezés fogja meghatározni:

```
(int)($F{TERULET}.intValue()/2000000)
```

Ez más szavakkal azt jelenti, hogy ezek a beágyazott csoportok lesznek, ahol a számok a négyzetkilométert jelentik:

- (0 – 2000000)
- [2000000 – 4000000)
- [4000000 – 6000000)
- ...

A kategória group természetesen maradt a régi (*$F{KAT}*). A kategória csoport kördiagramjai mutatni fogják a belső csoportnak megfelelő területnagyságok eloszlását, így azokat most a kategória csoport *footer* sávjára tettük, hiszen a kategóriaváltásoknál van értelme megjeleníteni őket (10.7. ábra). A részletező kördiagramok beállítását a 10.8. ábrán tekinthetjük meg. A jelentés végső kinézetét a 10.9. ábra tartalmazza. Látható, hogy a riport eredményének szerkezete megegyezik az előzővel, de az – ahogy terveztük – kategóriánként tartalmaz még egy-egy kördiagramot is. Nézzük meg például a részletező 2. kördiagram körcikkeit:

- Láthatjuk, hogy a 2. kategóriában 2 területnagyság intervallum van: 0 (piros) és 1 (kék), azaz 4000000 négyzetkilométernél nincs nagyobb ország ebben a kategóriában.
- A 0 érték száma 47 db, míg az 1 érték 3 db, így a már ismert 50 db ország ki is jön.
- A piros 94%, a kék pedig 6% az országoknak, ha a 2. kategóriába belül vizsgálódunk.

A továbbiakban még nézzük meg a riport template lényegesebb részeit is röviden, amit a 10-3. Programlista tartalmaz!

```
1  // 10-3. Programlista: Az Orszagok-Chart.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport ...
5
6          <import value="org.cs.test.*"/>
7
8          <queryString>
9                  <![CDATA[select * from orszagok order by kat, terulet]]>
10         </queryString>
11
12         <field name="ORSZAG" class="java.lang.String"/>
13         <field name="FOVAROS" class="java.lang.String"/>
14         <field name="FOLDR_HELY" class="java.lang.String"/>
15         <field name="TERULET" class="java.math.BigDecimal"/>
16 ...
17         <field name="KAT" class="java.lang.Integer"/>
18
19         <variable name="vDarabByKat" class="java.lang.Integer" calculation="Sum">
20                 <variableExpression><![CDATA[1]]></variableExpression>
21         </variable>
22
23         <group name="Kategoria">
24                 <groupExpression><![CDATA[$F{KAT}]]></groupExpression>
25                 <groupFooter>
26                         <band height="158">
27                                 <textField>
28                                         <reportElement x="16" y="13" width="108" height="20"/>
29                                         <textElement/>
```

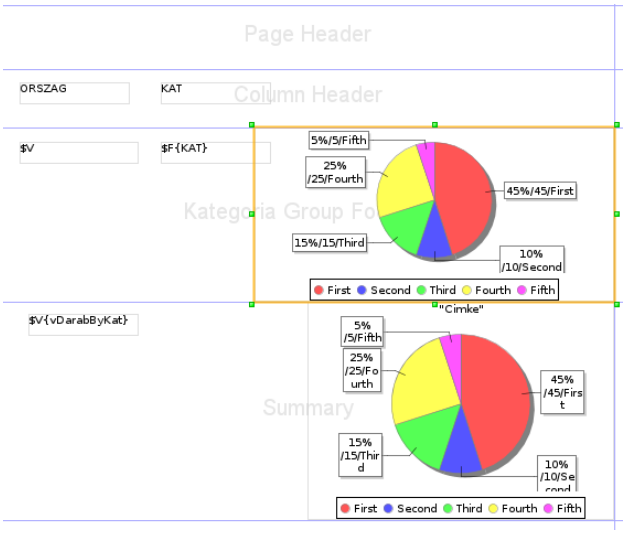
```
30                                      <textFieldExpression><![CDATA[$V{Kategoria_COUNT}]]></➥
                                            textFieldExpression>
31                                  </textField>
32                                  <textField>
33                                          <reportElement x="144" y="13" width="100" height="20"/>
34                                          <textElement/>
35                                          <textFieldExpression><![CDATA[$F{KAT}]]></textFieldExpression>
36                                  </textField>
37                                  <pieChart>
38                                          <chart>
39                                                  <reportElement x="229" y="0" width="326" height="158"/>
40                                                  <chartTitle/>
41                                                  <chartSubtitle/>
42                                                  <chartLegend/>
43                                          </chart>
44                                          <pieDataset>
45                                                  <dataset resetType="Group" resetGroup="Kategoria"/>
46                                                  <keyExpression><![CDATA[(int)($F{TERULET}.intValue()/2000000)]]>➥
                                                        </keyExpression>
47                                                  <valueExpression><![CDATA[$V{TeruletNagysag_COUNT}]]></➥
                                                        valueExpression>
48                                          </pieDataset>
49                                          <piePlot labelFormat="{2}/{1}/{0}">
50                                                  <plot/>
51                                                  <itemLabel/>
52                                          </piePlot>
53                                  </pieChart>
54                          </band>
55                  </groupFooter>
56          </group>
57
58          <group name="TeruletNagysag">
59                  <groupExpression><![CDATA[(int)($F{TERULET}.intValue()/2000000)]]></groupExpression>
60          </group>
61
62  ...
63
64          <summary>
65                  <band height="198">
66                          <textField>
67                                  <reportElement x="24" y="11" width="100" height="20"/>
68                                  <textElement/>
69                                  <textFieldExpression><![CDATA[$V{vDarabByKat}]]></textFieldExpression>
70                          </textField>
71                          <pieChart>
72                                  <chart evaluationTime="Report" customizerClass="org.cs.test.MyPieChartCustomizer➥
                                        ">
73                                      <reportElement  stretchType="RelativeToBandHeight" x="277" y="0" width="➥
                                            278" height="198"/>
74                                      <chartTitle>
75                                              <titleExpression><![CDATA["Cimke"]]></titleExpression>
76                                      </chartTitle>
77                                      <chartSubtitle/>
78                                      <chartLegend/>
79                                  </chart>
80                                  <pieDataset>
81                                          <keyExpression><![CDATA[$F{KAT}]]></keyExpression>
82                                          <valueExpression><![CDATA[$V{Kategoria_COUNT}]]></valueExpression>
83                                  </pieDataset>
84                                  <piePlot isCircular="true" labelFormat="{2}/{1}/{0}">
85                                          <plot/>
86                                          <itemLabel/>
87                                  </piePlot>
88                          </pieChart>
89                  </band>
90          </summary>
91  </jasperReport>
```

A 8-10 sorok közötti *queryString* alakja:

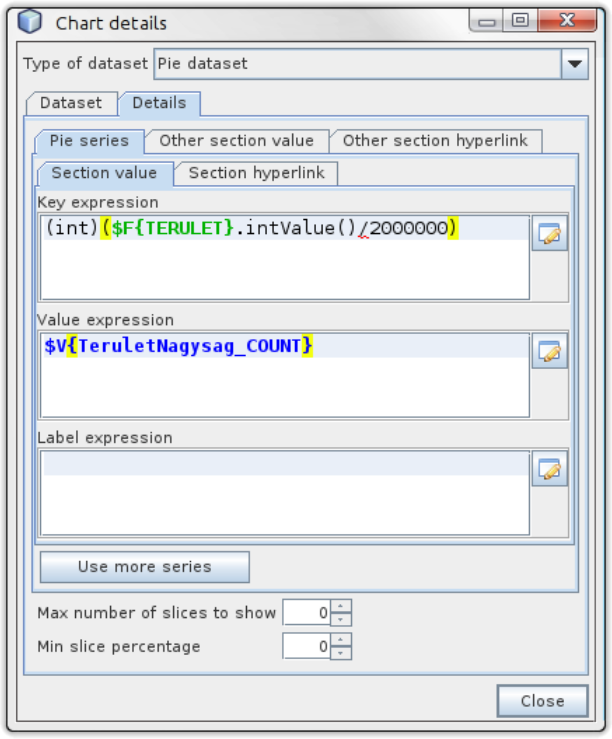
```
select * from orszagok order by kat, terulet
```

Itt az *order by* rész után a terület is része lett a rendezettségi feltételnek, hiszen a 2. (beágyazott) csoport képezi a kördiagram kulcsképző szempontját, ami viszont a *TERULET* mező függvénye. Az új csoportunk az 58-60 sorok között van, szerepe csak annyi, hogy hivatkozni lehessen rá, illetve felhasználjuk az automatikusan generált *$V{TeruletNagysag_COUNT}* értéket is, hiszen az adja a körcikkek méretét. Az eredeti – és most már beágyazó – csoport *footer* sávja azzal egészült ki, hogy itt helyeztük el 37-51 sorok között látható kördiagram tervet (az adatainak beállítását a 10.8. ábra mutatja). A 19-21 sorok közötti *vDarabByKat* változó most

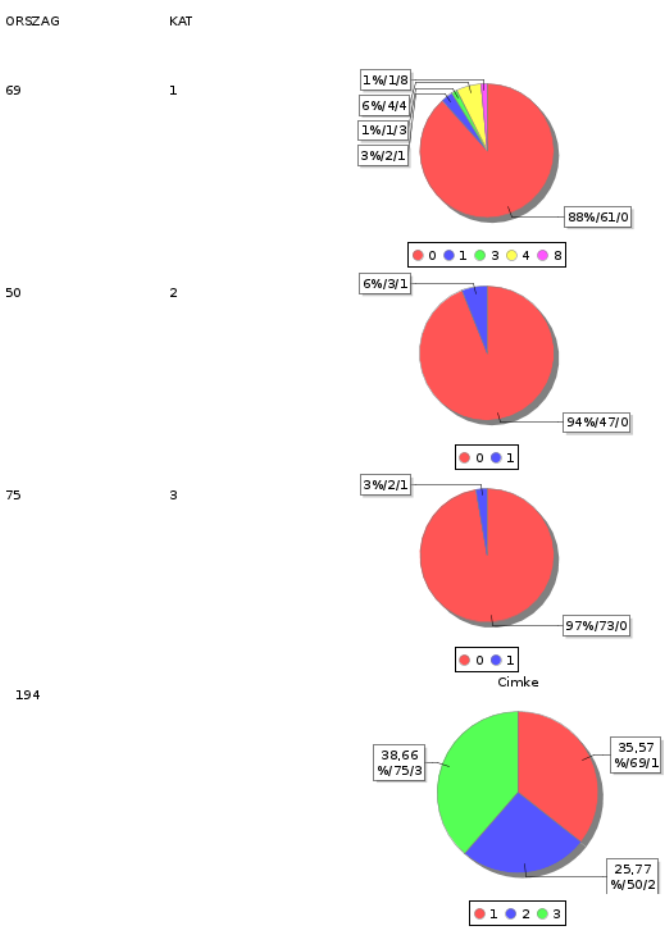
éppen nincs használva, de ha az országok darabszáma helyett a területek összegét akarnánk a kördiagramba tenni, akkor ezt használnánk *Value expression* gyanánt.



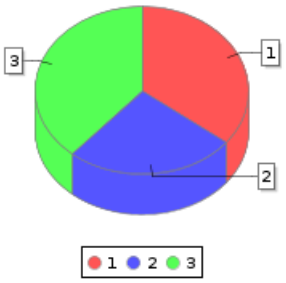
10.7. ábra: Terület csoportonkénti diagram - 1



10.8. ábra: Terület csoportonkénti diagram - 2



10.9. ábra: Az elkészült jelentés



10.10. ábra: 3D Kördiagram a summary sávban

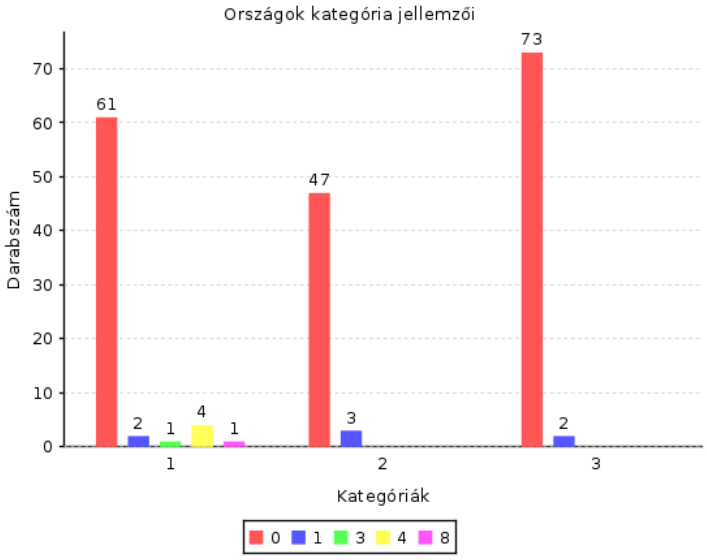
A 2D kördiagramból könnyen készíthető 3D diagram, az *iReport* automatikusan képes ezt a konverziót elvégezni, aminek eredményeképpen a jrxml-be ezek a változtatások történnek a grafikonra:

- *<pieChart>* helyett *<pie3DChart>* tag
- *<piePlot>* helyett *<pie3DPlot>* tag

Ekkor a 10.10. ábra mutatja a *summary* sáv 3D kördiagramját, ahol most csak a kategória értékek lettek a címkék.

Oszlopdiagram

Az oszlopdiagram is az egyes értékek közötti arányokat mutatja, de ugyanakkor azok abszolút értékét is jól ábrázolja. Általános esetben több kategóriát is tudunk ábrázolni, így a 10.9. ábra 3 részletező kördiagramját egyetlen oszlopdiagrammal is kiválthatjuk, ahogy az a 10.11. ábráról kitűnik. A *X* tengely mutatja most az egyes ország kategóriákat (1, 2 és 3), az Y pedig a kategórián belüli területnagyság mértékének eloszlását darab egységben kifejezve. Amennyiben a kategóriákon belüli számokat összeadjuk, megkapjuk a már ismert 69, 50, 75 ország számokat.



10.11. ábra: Oszlopdiagram

A 10.11. ábra a *summary* sávban van, azt a 10-4. Programlista 8-31 sorok közötti része állította elő, így érdemes azt röviden áttekinteni.

```
1  // 10-4. Programlista: Az Orszagok-Chart-BAR.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport ...
5  ...
6    <summary>
7      <band height="398">
8        <barChart>
9          <chart>
10           <reportElement x="99" y="56" width="422" height="319"/>
11           <chartTitle position="Top">
12             <titleExpression><![CDATA["Országok kategória jellemzői"]]></titleExpression>
13           </chartTitle>
14           <chartSubtitle/>
15           <chartLegend/>
16         </chart>
17         <categoryDataset>
18           <categorySeries>
19             <seriesExpression><![CDATA[(int)($F{TERULET}.intValue()/2000000)]]></seriesExpression>
20             <categoryExpression><![CDATA[$F{KAT}]]></categoryExpression>
21             <valueExpression><![CDATA[$V{TeruletNagysag_COUNT}]]></valueExpression>
22             <labelExpression><![CDATA[$V{TeruletNagysag_COUNT}.toString()]]></labelExpression>
23           </categorySeries>
24         </categoryDataset>
25         <barPlot isShowLabels="true">
26           <plot/>
27           <itemLabel/>
28           <categoryAxisLabelExpression><![CDATA["Kategóriák"]]></categoryAxisLabelExpression>
29           <valueAxisLabelExpression><![CDATA["Darabszám"]]></valueAxisLabelExpression>
30         </barPlot>
31       </barChart>
32     </band>
33   </summary>
34 ...
35 </jasperReport>
```

A 12. sorban látható a grafikon címének beállítása. A 28. és 29. sorok pedig az *X* és *Y* tengelyek megnevezését adja meg. A leglényegesebb részt 19-22 sorok között találjuk, itt – oszlopdiagram esetén – 4 különböző dolgot kell megadnunk:

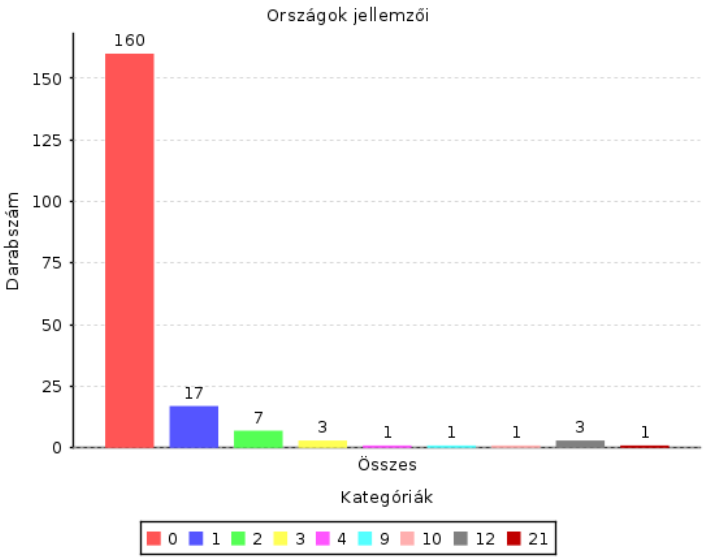
- 19. sor: Itt a már ismert terület osztály meghatározó kifejezést találjuk. Ez ad egy adatsorozatot egy-egy oszlopdiagram kategórián belül.
- 20. sor: Az X tengelyen megjelenő kategóriákat adó kifejezés, esetünkben most az országok kategória besorolása az.
- 21. sor: A kategóriánként újraszámító *TeruletNagysag_COUNT* változó értéke, ami az egyes oszlopok magasságát (esetünkben darabszám) jelenti. A *TeruletNagysag* group értékváltozásánál történik a reset-je.
- 22. sor: A 21. sor értékeit meg is jelenítjük a grafikonon, ez most az oszlop felett megjelenő szám.

Szeretnénk 2 megjegyzést tenni:

1. Bármelyik oszlopdiagram 1 attribútummal vertikálisról horizontálissá változtatható
2. A 2D helyett itt is kérhetjük 3D megjelenést.

Oszlopdiagram - csak 1 kategória

Néha nem akarunk különböző kategóriákat az *X* tengelyre, mert csak a teljes sokaság eloszlását szeretnénk oszlopdiagram formájában ábrázolni. Ilyenkor egyszerű a megoldás, a *categoryExpression* tag-hez egy konstanst írunk. A 10.12. ábra kördiagramja ország kategóriától függetlenül, azaz az összes ország terület besorolása szerinti darabszám eloszlást mutatja. Most a diagramot is tartalmazó teljes jrxml tartalmat bemutatjuk a 10-5. Programlista segítségével.



10.12. ábra: Oszlopdiagram - 1 kategória

```
1  // 10-5. Programlista: Az Orszagok-Chart-BAR.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/➥
       XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports␣http://➥
       jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-Chart" language="groovy" pageWidth="595"➥
        pageHeight="842" columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20" uuid=➥
       "51030eb1-4516-48e2-a63d-04d9526ab437">
5    <property name="ireport.zoom" value="1.0"/>
6    <property name="ireport.x" value="0"/>
7    <property name="ireport.y" value="0"/>
8
9    <queryString>
10     <![CDATA[select * from orszagok order by terulet]]>
11   </queryString>
12   <field name="ORSZAG" class="java.lang.String"/>
13   <field name="FOVAROS" class="java.lang.String"/>
14   <field name="FOLDR_HELY" class="java.lang.String"/>
15   <field name="TERULET" class="java.math.BigDecimal"/>
16   <field name="ALLAMFORMA" class="java.lang.String"/>
17   <field name="NEPESSEG" class="java.lang.Integer"/>
18   <field name="NEP_FOVAROS" class="java.lang.Integer"/>
```

```
19    <field name="AUTOJEL" class="java.lang.String"/>
20    <field name="COUNTRY" class="java.lang.String"/>
21    <field name="CAPITAL" class="java.lang.String"/>
22    <field name="PENZNEM" class="java.lang.String"/>
23    <field name="PENZJEL" class="java.lang.String"/>
24    <field name="VALTOPENZ" class="java.lang.String"/>
25    <field name="TELEFON" class="java.lang.Integer"/>
26    <field name="GDP" class="java.lang.Integer"/>
27    <field name="KAT" class="java.lang.Integer"/>
28    <variable name="vDarabByKat" class="java.lang.Integer" calculation="Sum">
29      <variableExpression><![CDATA[1]]></variableExpression>
30    </variable>
31    <variable name="vTerCounter" class="java.lang.Integer" resetType="Group" resetGroup="TeruletNagysag" ➥
          calculation="Count">
32      <variableExpression><![CDATA[$F{TERULET}]]></variableExpression>
33    </variable>
34    <group name="TeruletNagysag">
35      <groupExpression><![CDATA[(int)($F{TERULET}.intValue()/800000)]]></groupExpression>
36    </group>
37    <pageHeader>
38      <band height="58"/>
39    </pageHeader>
40    <columnHeader>
41      <band height="53">
42        <staticText>
43          <reportElement x="16" y="12" width="100" height="20"/>
44          <textElement/>
45          <text><![CDATA[ORSZAG]]></text>
46        </staticText>
47        <staticText>
48          <reportElement  x="144" y="12" width="100" height="20"/>
49          <textElement/>
50          <text><![CDATA[KAT]]></text>
51        </staticText>
52      </band>
53    </columnHeader>
54    <columnFooter>
55      <band/>
56    </columnFooter>
57    <summary>
58      <band height="398">
59        <barChart>
60          <chart>
61            <reportElement  x="99" y="56" width="422" height="319"/>
62            <chartTitle position="Top">
63              <titleExpression><![CDATA["Országok jellemzői"]]></titleExpression>
64            </chartTitle>
65            <chartSubtitle/>
66            <chartLegend/>
67          </chart>
68          <categoryDataset>
69            <categorySeries>
70              <seriesExpression><![CDATA[(int)($F{TERULET}.intValue()/800000)]]></seriesExpression>
71              <categoryExpression><![CDATA["Összes"]]></categoryExpression>
72              <valueExpression><![CDATA[$V{vTerCounter}]]></valueExpression>
73            </categorySeries>
74          </categoryDataset>
75          <barPlot isShowLabels="true">
76            <plot/>
77            <itemLabel/>
78            <categoryAxisLabelExpression><![CDATA["Kategóriák"]]></categoryAxisLabelExpression>
79            <valueAxisLabelExpression><![CDATA["Darabszám"]]></valueAxisLabelExpression>
80          </barPlot>
81        </barChart>
82      </band>
83    </summary>
84  </jasperReport>
```

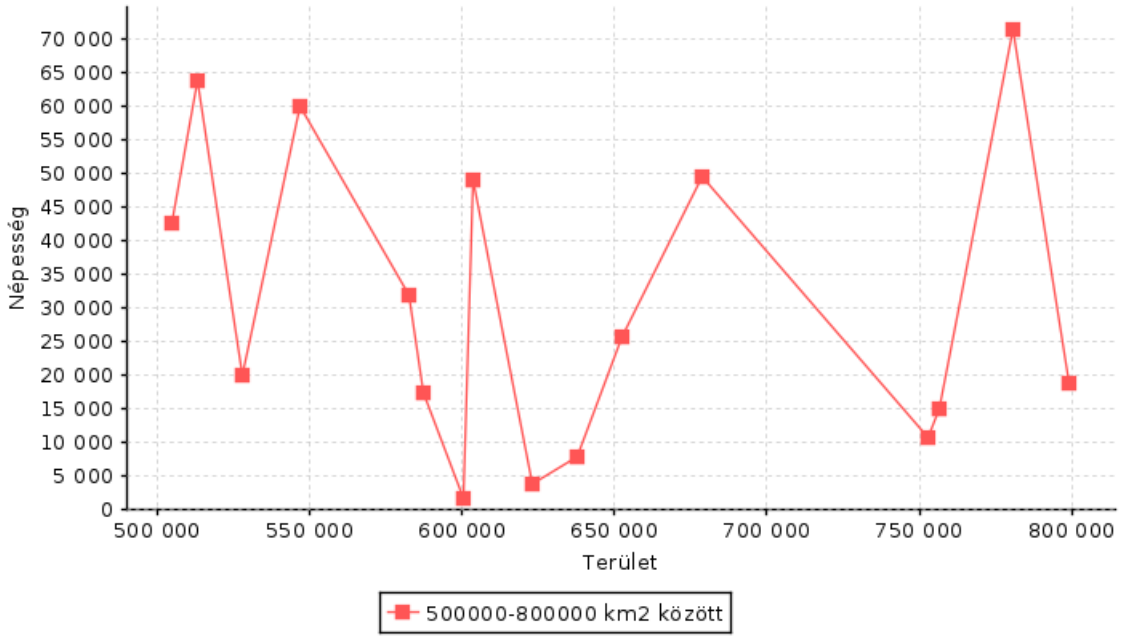
A 10. sor *queryString* tag-je, mutatja, hogy a *select* most természetesen csak a *TERULET* szerint rendezett, hiszen a *KAT* most nem játszik szerepet. Lényeges elem a 34-36 sorok közötti *TeruletNagysag* group, aminek érték ugrásait most 800.000-re csökkentettünk, hogy finomabb eloszlást láthassunk a diagramon. A diagram most is a *summary* sávban van, konfigurálását az 59-81 sorok között láthatjuk. Kiemelendő változás, hogy most az oszlop magasságát *$V{vTerCounter}* kifejezés adja, aminek a változóját a 31-33 sorok között definiáltuk és a számlálás működését a *TeruletNagysag* csoporttal szinkronizáltuk.

X-Y Vonaldiagram

Utolsó példaként nézzünk meg a X-Y vonaldiagram használatát, amit a 10-6. Programlista mutat. A riport futási eredményének diagram részét pedig a 10.13. ábráról tekinthetjük meg. Itt ismét a teljes *jrxml* fájlt közöltük, így remélhetőleg könnyebb lesz a teljes riport működésének az áttekintése is. Az adatsorokat szolgáltató *SQL select* 8-10 sorok között tanulmányozható.

```
1  // 10-6. Programlista: Az Orszagok-Chart-XYLine.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/➥
       XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports␣http://➥
       jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-Chart" language="groovy" pageWidth="595"➥
        pageHeight="842" columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20" uuid➥
       "51030eb1-4516-48e2-a63d-04d9526ab437">
5    <property name="ireport.zoom" value="1.0"/>
6    <property name="ireport.x" value="0"/>
7    <property name="ireport.y" value="0"/>
8    <queryString>
9      <![CDATA[select * from orszagok where terulet < 800000 and terulet > 500000 order by terulet]]>
10   </queryString>
11   <field name="ORSZAG" class="java.lang.String"/>
12   <field name="FOVAROS" class="java.lang.String"/>
13   <field name="FOLDR_HELY" class="java.lang.String"/>
14   <field name="TERULET" class="java.math.BigDecimal"/>
15   <field name="ALLAMFORMA" class="java.lang.String"/>
16   <field name="NEPESSEG" class="java.lang.Integer"/>
17   <field name="NEP_FOVAROS" class="java.lang.Integer"/>
18   <field name="AUTOJEL" class="java.lang.String"/>
19   <field name="COUNTRY" class="java.lang.String"/>
20   <field name="CAPITAL" class="java.lang.String"/>
21   <field name="PENZNEM" class="java.lang.String"/>
22   <field name="PENZJEL" class="java.lang.String"/>
23   <field name="VALTOPENZ" class="java.lang.String"/>
24   <field name="TELEFON" class="java.lang.Integer"/>
25   <field name="GDP" class="java.lang.Integer"/>
26   <field name="KAT" class="java.lang.Integer"/>
27   <variable name="vDarabByKat" class="java.lang.Integer" calculation="Sum">
28     <variableExpression><![CDATA[1]]></variableExpression>
29   </variable>
30   <variable name="vTerCounter" class="java.lang.Integer" resetType="Group" resetGroup="TeruletNagysag" ➥
         calculation="Count">
31     <variableExpression><![CDATA[$F{TERULET}]]></variableExpression>
32   </variable>
33   <group name="TeruletNagysag">
34     <groupExpression><![CDATA[(int)($F{TERULET}.intValue()/800000)]]></groupExpression>
35   </group>
36   <pageHeader>
37     <band height="58"/>
38   </pageHeader>
39   <columnHeader>
40     <band height="53">
41       <staticText>
42         <reportElement x="16" y="12" width="100" height="20"/>
43         <textElement/>
44         <text><![CDATA[ORSZAG]]></text>
45       </staticText>
46       <staticText>
47         <reportElement x="144" y="12" width="100" height="20"/>
48         <textElement/>
49         <text><![CDATA[KAT]]></text>
50       </staticText>
51     </band>
52   </columnHeader>
53   <columnFooter>
54     <band/>
55   </columnFooter>
56   <summary>
57     <band height="398">
58       <xyLineChart>
59         <chart>
60           <reportElement x="16" y="45" width="539" height="299"/>
61           <chartTitle/>
62           <chartSubtitle/>
63           <chartLegend/>
64         </chart>
65         <xyDataset>
66           <xySeries>
67             <seriesExpression><![CDATA["500000-800000 km2 között"]]></seriesExpression>
68             <xValueExpression><![CDATA[$F{TERULET}]]></xValueExpression>
69             <yValueExpression><![CDATA[$F{NEPESSEG}]]></yValueExpression>
70           </xySeries>
```

```
71          </xyDataset>
72          <linePlot isShowLines="true" isShowShapes="true">
73            <plot/>
74            <categoryAxisLabelExpression><![CDATA["Terület"]]></categoryAxisLabelExpression>
75            <valueAxisLabelExpression><![CDATA["Népesség"]]></valueAxisLabelExpression>
76          </linePlot>
77        </xyLineChart>
78      </band>
79    </summary>
80  </jasperReport>
```



10.13. ábra: X-Y Vonaldiagram

Az 56. sortól kezdődő *summary* sávra tettük a diagramot, aminek a leírása az 58-77 sorok között található és viszonylag egyből megérthető. Mindig 2 összetartozó számsorozatot kell előállítani, amiket az *xValueExpression* és *yValueExpression* tag-ek között adhatunk meg (68, 69. sorok). Ennek a nevét a *seriesExpression* tartalmazza, amint látható ez a legend résznél is megjelenik. Az X és Y tengelyek címkéit pedig a 74-75 sorokban látható módon lehetett megadni.

További diagramok

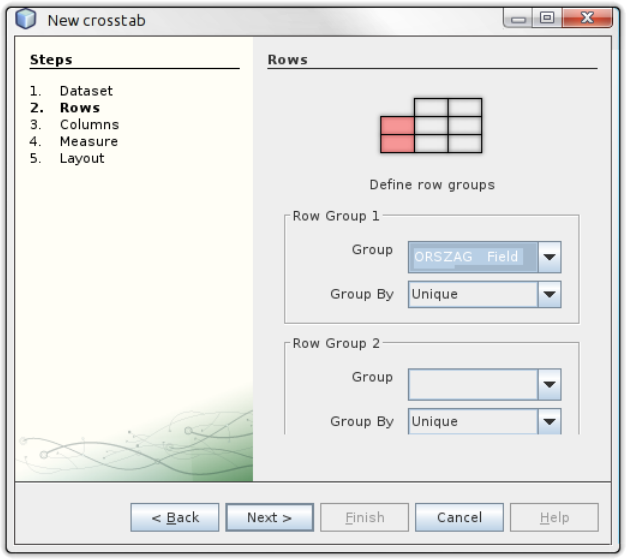
A *JFreeChart* könyvtár sok diagram típust ismer, ezek nagy része közvetlenül is elérhető, de testreszabással be is építhető a *JasperReports*-ba: Stacked Bar Chart, Area Chart, Scatter Plot Chart, Bubble Chart, Time Series Chart, Candlestick Chart. Tervezzük, hogy a szaklap további számaiban ezen diagram fajtákat is bemutatjuk.

11. Táblázatos riportok készítése (Crosstab)

A JasperReport az 1.1 verzió óta képes olyan kimutatásokat készíteni, amelyek egy táblázatnak felelnek meg. Ez sorokból és oszlopokból áll és ezek találkozásánál olyan cellák vannak, ahova a jellemzők darabszámát, összegét, átlagát, stb. tudjuk generálni. Ebben a cikkben bemutatjuk azt is, hogy egy riport alkalmazást milyen módon készíthetünk el webalkalmazásként (servletként).

A táblázat komponens

Néha egy jelentés fontos része, hogy tartalmazzon egy vagy több táblázatot, amit a JasperReports *Crosstab* komponense segítségével tudunk könnyen megvalósítani. A komponens palettáról behúzhatjuk bármelyik sávra a riport igényelt szerkezetétől függően. Így csoportonként több táblázat is megjeleníthető. A 11.1. ábra mutatja a komponens behúzása után automatikusan elinduló crosstab varázsló 2. ablakát.



11.1. ábra: Crosstab varázsló

Amikor egy új táblázatot akarunk beszúrni, akkor használhatjuk a fő *dataset*-et, ami az eddig ismert *queryString* és adatforrás. Már a diagramoknál is említtethettük volna, de rugalmassága miatt itt mindenképpen szeretnénk kiemelni, hogy egy *jrxml* fájlban tetszőleges számú, egyedi névvel azonosítható további adatforrást is létrehozhatunk, amit a *subDataset* tag segítségével tehetünk meg, ahogy a következő példa is mutatja:

```
<subDataset name="MyDataset">
  <queryString language="SQL">
    <![CDATA[select * from orszagok]]>
    </queryString>
      <field name="ORSZAG" class="java.lang.String"/>
      <field name="FOVAROS" class="java.lang.String"/>
</subDataset>
```

Itt a dataset neve *MyDataset* és ugyanúgy adható meg, mint az eddig használt fő dataset. A crosstab varázsló az 1. ablakban azt kérdezi meg, hogy melyik adatforrást szeretnénk használni. Természetesen a main dataset mindig rendelkezésre áll, de innentől kezdve a *MyDataset* is használható. A varázsló 2. képernyője azt is mutatja, hogy a táblázat sorainak kialakításánál több dimenziót is megadhatunk, ahogy az a valódi statisztikai tábláknál gyakorta előfordul. A 3. képernyőn megadható oszlop jellemzők hasonlóan többdimenziósak. A 4. képernyőn azt a gyűjtött adatforrásra épülő kifejezést kell megadnunk, aminek számszerű és aggregált értékei kerülnek az egyes cellákba. Érdemes meggondolni, hogy az is egy hatékony lehetőség, hogy az egyes táblákat alriportként szúrjuk be a fő-riportba, bár erre ritkán van valóban szükség, ugyanis a *subDataset* segítségével hatékonyan lehet az egyedi, a riport főszerkezetétől eltérő adatforrásokat megfogalmazni. Ennek erejét mutatja, hogy még paramétereket is átadhatunk. A *MyDataset2* mögött lévő lekérdezés dinamikusan csak azokat az országokat fogja legyűjteni, ahol az átadott méretnél kisebb az ország:

```
<subDataset name="MyDataset2">
  <parameter name="parSize" class="java.lang.Number"/>
  <queryString language="SQL">
    <![CDATA[select * from orszagok where terulet < $P➥
        {parSize}]]>
    </queryString>
      <field name="ORSZAG" class="java.lang.String"/>
      <field name="FOVAROS" class="java.lang.String"/>
</subDataset>
```

Táblázat az országokról

A használat könnyebb megértése érdekében készítsünk egy olyan táblázatot, ahol a sorok a földrajzi területek, míg az oszlopok az ország kategória szerint vannak csoportosítva. A kimutatásban az érdekel bennünket, hogy az egyes földrajzi helyekbe az egyes kategóriákból hány ország esik, ezért az aggregálás típusa *count*, azaz leszámlálás lesz. Itt – lévén statisztikáról van szó – sok minden hasonlít a diagramoknál írtakhoz, de most nem egy grafikon lesz az eredmény, hanem egy táblázat. A táblázat csak azokat az országokat vonja be a vizsgálatba, ahol a $terulet > 2000000\,km^2$. A *Crosstab* komponenst most a *summary* sávba húzzuk be, az eredményt a 11.2. ábra mutatja.

	1	2	3	Total KAT
Ausztrália	1	0	0	1
Dél-Amerika	2	0	0	2
Eurázsia	1	0	0	1
Közép-Afrika	0	0	1	1
Ázsia	1	1	0	2
Ázsia:Arab-félsziget	0	1	0	1
Ázsia: Hindusztáni-	1	0	0	1
Észak-Afrika	0	1	0	1
Észak-Amerika	2	0	0	2
Észak-Kelet-Afrika	0	0	1	1
Total FOLDR_HELY	8	3	2	13

11.2. ábra: Táblázat - Földrajzi hely/Kategória

A 11-1. Programlista a teljes, létrehozott *jrxml* fájlt mutatja. A varázsló által generált *crosstab* részt a 32-142 sorok között láthatjuk.

```
1  // 11-1. Programlista: Az Orszagok-Chart-Crosstab.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/➥
       XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports␣http://➥
       jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-Chart-Crosstab" language="groovy" ➥
       pageWidth="595" pageHeight="842" columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" ➥
       bottomMargin="20" >
5    <property name="ireport.zoom" value="1.0"/>
6    <property name="ireport.x" value="0"/>
7    <property name="ireport.y" value="0"/>
8    <style name="Crosstab␣Data␣Text" hAlign="Center"/>
9    <queryString>
10     <![CDATA[select * from orszagok where terulet > 2000000 order by FOLDR_HELY, KAT]]>
11   </queryString>
12   <field name="ORSZAG" class="java.lang.String"/>
13   <field name="FOVAROS" class="java.lang.String"/>
14   <field name="FOLDR_HELY" class="java.lang.String"/>
15   <field name="TERULET" class="java.math.BigDecimal"/>
16   <field name="ALLAMFORMA" class="java.lang.String"/>
17   <field name="NEPESSEG" class="java.lang.Integer"/>
18   <field name="NEP_FOVAROS" class="java.lang.Integer"/>
19   <field name="AUTOJEL" class="java.lang.String"/>
20   <field name="COUNTRY" class="java.lang.String"/>
21   <field name="CAPITAL" class="java.lang.String"/>
22   <field name="PENZNEM" class="java.lang.String"/>
23   <field name="PENZJEL" class="java.lang.String"/>
24   <field name="VALTOPENZ" class="java.lang.String"/>
25   <field name="TELEFON" class="java.lang.Integer"/>
26   <field name="GDP" class="java.lang.Integer"/>
27   <field name="KAT" class="java.lang.Integer"/>
28
29   <summary>
30     <band height="398">
31       <crosstab>
32         <reportElement x="16" y="17" width="481" height="347"/>
33         <rowGroup name="FOLDR_HELY" width="70" totalPosition="End">
34           <bucket class="java.lang.String">
35             <bucketExpression><![CDATA[$F{FOLDR_HELY}]]></bucketExpression>
```

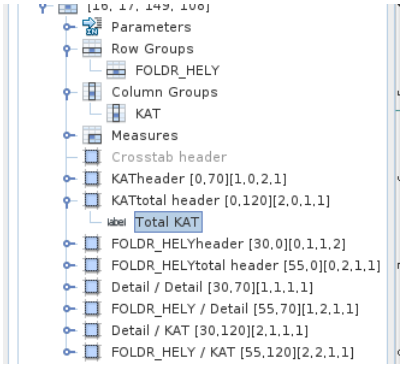
```
36            </bucket>
37            <crosstabRowHeader>
38              <cellContents backcolor="#F0F8FF" mode="Opaque">
39                <box>
40                  <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
41                </box>
42                <textField>
43                  <reportElement style="Crosstab Data Text" x="0" y="0" width="70" height="25"/>
44                  <textElement/>
45                  <textFieldExpression><![CDATA[$V{FOLDR_HELY}]]></textFieldExpression>
46                </textField>
47              </cellContents>
48            </crosstabRowHeader>
49            <crosstabTotalRowHeader>
50              <cellContents backcolor="#BFE1FF" mode="Opaque">
51                <box>
52                  <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
53                </box>
54                <staticText>
55                  <reportElement  x="0" y="0" width="70" height="25"/>
56                  <textElement textAlignment="Center" verticalAlignment="Middle"/>
57                  <text><![CDATA[Total FOLDR_HELY]]></text>
58                </staticText>
59              </cellContents>
60            </crosstabTotalRowHeader>
61          </rowGroup>
62          <columnGroup name="KAT" height="30" totalPosition="End">
63            <bucket class="java.lang.Integer">
64              <bucketExpression><![CDATA[$F{KAT}]]></bucketExpression>
65            </bucket>
66            <crosstabColumnHeader>
67              <cellContents backcolor="#F0F8FF" mode="Opaque">
68                <box>
69                  <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
70                </box>
71                <textField>
72                  <reportElement  style="Crosstab Data Text" x="0" y="0" width="50" height="30"/>
73                  <textElement/>
74                  <textFieldExpression><![CDATA[$V{KAT}]]></textFieldExpression>
75                </textField>
76              </cellContents>
77            </crosstabColumnHeader>
78            <crosstabTotalColumnHeader>
79              <cellContents backcolor="#BFE1FF" mode="Opaque">
80                <box>
81                  <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
82                </box>
83                <staticText>
84                  <reportElement  x="0" y="0" width="50" height="30"/>
85                  <textElement textAlignment="Center" verticalAlignment="Middle"/>
86                  <text><![CDATA[Total KAT]]></text>
87                </staticText>
88              </cellContents>
89            </crosstabTotalColumnHeader>
90          </columnGroup>
91          <measure name="GDPMeasure" class="java.lang.Integer" calculation="Count">
92            <measureExpression><![CDATA[$F{GDP}]]></measureExpression>
93          </measure>
94          <crosstabCell width="50" height="25">
95            <cellContents>
96              <box>
97                <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
98              </box>
99              <textField>
100               <reportElement style="Crosstab Data Text" x="0" y="0" width="50" height="25"/>
101               <textElement/>
102               <textFieldExpression><![CDATA[$V{GDPMeasure}]]></textFieldExpression>
103             </textField>
104           </cellContents>
105         </crosstabCell>
106         <crosstabCell height="25" rowTotalGroup="FOLDR_HELY">
107           <cellContents backcolor="#BFE1FF" mode="Opaque">
108             <box>
109               <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
110             </box>
111             <textField>
112               <reportElement style="Crosstab Data Text" x="0" y="0" width="50" height="25"/>
113               <textElement/>
114               <textFieldExpression><![CDATA[$V{GDPMeasure}]]></textFieldExpression>
115             </textField>
116           </cellContents>
117         </crosstabCell>
118         <crosstabCell width="50" columnTotalGroup="KAT">
119           <cellContents backcolor="#BFE1FF" mode="Opaque">
120             <box>
121               <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
122             </box>
123             <textField>
```

```
124                 <reportElement  style="Crosstab_Data_Text" x="0" y="0" width="50" height="25"/>
125                 <textElement/>
126                 <textFieldExpression><![CDATA[$V{GDPMeasure}]]></textFieldExpression>
127               </textField>
128             </cellContents>
129           </crosstabCell>
130           <crosstabCell rowTotalGroup="FOLDR_HELY" columnTotalGroup="KAT">
131             <cellContents backcolor="#BFE1FF" mode="Opaque">
132               <box>
133                 <pen lineWidth="0.5" lineStyle="Solid" lineColor="#000000"/>
134               </box>
135               <textField>
136                 <reportElement  style="Crosstab_Data_Text" x="0" y="0" width="50" height="25"/>
137                 <textElement/>
138                 <textFieldExpression><![CDATA[$V{GDPMeasure}]]></textFieldExpression>
139               </textField>
140             </cellContents>
141           </crosstabCell>
142         </crosstab>
143       </band>
144     </summary>
145 </jasperReport>
```

Jó sok sor generálódott, de mindez csak pár kattintás volt a Crosstab varázsló használatakor, ahol ezeket a jellemzőket adtuk meg:

- FOLDR_HELY: A sorok csoportosítási szempontja.
- KAT: Az oszlopok csoportosítási szempontja.
- GDP: Lehetett volna bármelyik másik oszlop is, mert itt csak az előfordulás számát számoljuk (és GDP-je mindenkinek van).

A 11.3. ábra a Report Inspector Crosstab részletét mutatja, ott jól áttekinthető a generált szerkezet.



11.3. ábra: Crosstab - Report Inspector

Riportkészítő webalkalmazás

Befejezésül egy érdekes riporthívási lehetőséget szeretnénk bemutatni, amihez példaképpen a fenti riportot fogjuk használni. A mai világban fontos az egyes funkciók böngészős elérése, így ezt mutatjuk be a következőkben. Szeretnénk kiemelni, hogy a bemutatott megoldás hosszú futású idejű riportoknál nem ajánlott, mert a HTTP request-response mechanizmusra épül, ami a beállított time-out érték után megszakad, így a riportunkhoz sem fogunk tudni hozzáférni. Ilyenkor is lehet Servlet-et használni, de ekkor az külön futási szálon (Thread-en) indítsa el a riport elkészítését. A 11-2. Programlista a feladat megoldását adó Java servlet, amit *Tomcat*, *JBoss* és *Weblogic* környezetben is kipróbáltunk, hibátlanul működik. A teszteléshez az alábbi szerverek futottak a fejlesztői gépen:

- Java Webserver (Tomcat vagy JBoss vagy Weblogic)
- HSQLDB adatbázis-kezelő (ami tartalmazza az *Orszagok* táblát is)

Érdemes röviden átnézni a Servlet működését, lehet, hogy nem mindenki ismerős ebben a világban. A lényeg a 32-69 sorok között megvalósított *processRequest()* metóduson van, azt hívja meg a HTTP kérésre a Webserver is. A kód vázlatát Netbeans környezetben generáltuk, de az ismert

Eclipse-ben is hasonló lett volna. A *request* és *response* metódus paraméterek a kérést és a választ reprezentáló objektumok. A 37. sorban lekérünk egy olyan *OutputStream*-et (byte folyam, ami visszamegy a böngészőnek), amibe a generált PDF byte-jait tesszük majd. A 38-39 sorokban a riport template-hez (azaz a *jasper* fájl) jutunk hozzá, amit a *reportStream* változóba töltünk. Ehhez persze az kellett, hogy a web alkalmazáshoz csomagoljuk hozzá a jasper fájlt is. A 43-44 sorokban a *HSQLDB* adatbázishoz kapcsolódunk. A 45. sorban legeneráljuk a riportot. Használhattuk volna az eddigi módszereinket is, de a JasperReports tartalmaz néhány utility metódust , ilyen a *runReportToPdfStream()* is. A 46. sorban beállítjuk, hogy a visszaadott tartalom (content) PDF MIME típusú.

```
1  // 11-2. Programlista: JasperServlet.java
2
3  package org.cs.test;
4
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.PrintWriter;
8  import java.io.StringWriter;
9  import java.sql.Connection;
10 import java.sql.DriverManager;
11 import java.sql.SQLException;
12 import java.util.HashMap;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15 import javax.servlet.ServletContext;
16 import javax.servlet.ServletException;
17 import javax.servlet.ServletOutputStream;
18 import javax.servlet.http.HttpServlet;
19 import javax.servlet.http.HttpServletRequest;
20 import javax.servlet.http.HttpServletResponse;
21
22 import net.sf.jasperreports.engine.JRException;
23 import net.sf.jasperreports.engine.JasperRunManager;
24
25 /**
26  *
27  * @author inyiri
28  */
29 public class JasperServlet extends HttpServlet
30 {
31
32     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33             throws ServletException, IOException
34     {
35
36         Connection conn = null;
37         ServletOutputStream servletOutputStream = response.getOutputStream();
38         ServletContext ctxServlet = getServletConfig().getServletContext();
39         InputStream reportStream = ctxServlet.getResourceAsStream("/reports/Orszagok-Chart-➥
               Crosstab.jasper");
40
41         try
42         {
43             Class.forName( "org.hsqldb.jdbcDriver" );
44             conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "SA", "");
45             JasperRunManager.runReportToPdfStream(reportStream, servletOutputStream, new HashMap➥
                   (), conn);
46             response.setContentType("application/pdf");
47             servletOutputStream.flush();
48             servletOutputStream.close();
```

```
49          }
50          catch (Exception e)
51          {
52              // display stack trace in the browser
53              StringWriter stringWriter = new StringWriter();
54              PrintWriter printWriter = new PrintWriter(stringWriter);
55              e.printStackTrace(printWriter);
56              response.setContentType("text/plain");
57              response.getOutputStream().print(stringWriter.toString());
58          }
59          finally
60          {
61              try
62              {
63                  conn.close();
64              } catch (Exception ex)
65              {
66                  ;
67              }
68          }
69      }
70
71      @Override
72      protected void doGet(HttpServletRequest request, HttpServletResponse response)
73              throws ServletException, IOException
74      {
75          processRequest(request, response);
76      }
77
78      @Override
79      protected void doPost(HttpServletRequest request, HttpServletResponse response)
80              throws ServletException, IOException
81      {
82          processRequest(request, response);
83      }
84
85      @Override
86      public String getServletInfo()
87      {
88          return "Short description";
89      }// </editor-fold>
90  }
```

	1	2	3	Total KAT
Ausztrália	1	0	0	1
Dél-Amerika	2	0	0	2
Eurázsia	1	0	0	1
Közép-Afrika	0	0	1	1
Ázsia	1	1	0	2
Ázsia:Arab-félsziget	0	1	0	1
Ázsia:Hindusztáni-	1	0	0	1
Észak-Afrika	0	1	0	1
Észak-Amerika	2	0	0	2
Észak-Kelet-Afrika	0	0	1	1
Total FOLDR_HELY	8	3	2	13

11.4. ábra: JasperServlet

12. Képek és rajzok beszúrása

A kiadvány utolsó cikkében röviden bemutatjuk a riportokban használható grafikai elemek használatát. Ilyet már a korábbi cikkekben is használtunk (vízjel, téglalap).

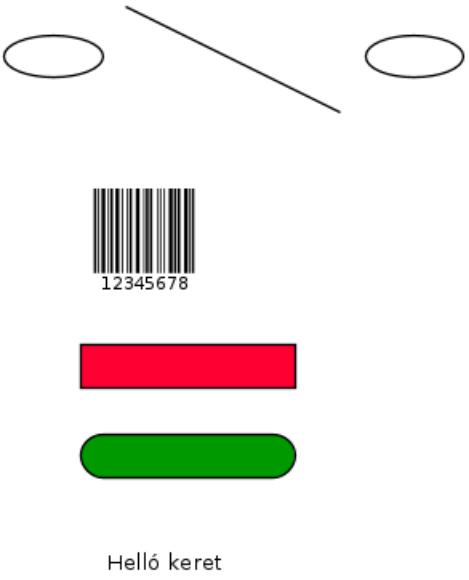
Ebben a rövid cikkben csak bemutatni szeretnénk néhány lehetőséget, külön magyarázat nélkül. A 12-1. Programlista egy olyan *jrxml* fájl, amibe néhány tipikus grafikus elemet tettünk, ezeket a JasperReports out-of-box szolgáltatja:

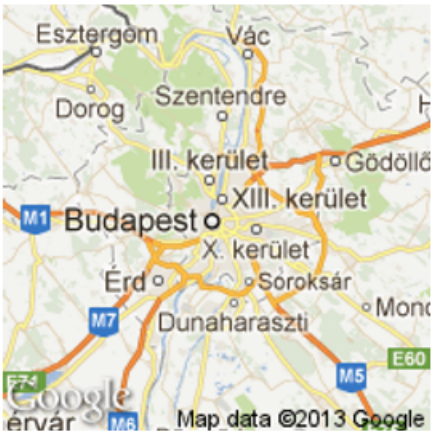
- vonal, téglalap, ellipszis,
- egy kép megjelenítése,
- különféle vonalkódok megjelenítése,
- a frame (keret) egy speciális lehetőség az összetartozó elemek együttes kezelésére,
- Google térkép
- HTML tartalom részlet.

Az előállított riportot (annak *summary* sávját) a 12.1. ábra mutatja.

```
1  // 12-1. Programlista: Az Orszagok-Image.jrxm
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/➥
       XMLSchema-instance" xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://➥
       jasperreports.sourceforge.net/xsd/jasperreport.xsd" name="Orszagok-Chart-Crosstab" language="groovy" ➥
       pageWidth="595" pageHeight="842" columnWidth="535" leftMargin="20" rightMargin="20" topMargin="20" ➥
       bottomMargin="20" uuid="51030eb1-4516-48e2-a63d-04d9526ab437">
5    <property name="ireport.zoom" value="1.0"/>
6    <property name="ireport.x" value="0"/>
7    <property name="ireport.y" value="0"/>
8    <style name="Crosstab Data Text" hAlign="Center"/>
9    <subDataset name="dataset1" uuid="094aae7d-c577-4861-9d68-584a1e5d70c5"/>
10   <queryString>
11     <![CDATA[select * from orszagok where terulet > 2000000 order by FOLDR_HELY, KAT]]>
12   </queryString>
13   <summary>
14     <band height="704">
15       <ellipse>
16         <reportElement uuid="f711eac1-8a6d-4c95-9c41-a7ba1596645f" x="18" y="40" width="46" height="20"/>
17       </ellipse>
18       <ellipse>
19         <reportElement uuid="755958dd-7fa3-478e-af3c-17eeed1ac7e5" x="187" y="40" width="46" height="20"/>
20       </ellipse>
21       <componentElement>
22         <reportElement uuid="0099af29-4890-4b26-92c8-b1f3b0e6ae45" x="54" y="112" width="200" height="50"/>
23         <jr:Code128 xmlns:jr="http://jasperreports.sourceforge.net/jasperreports/components" xsi:schemaLocation=➥
               "http://jasperreports.sourceforge.net/jasperreports/components http://jasperreports.sourceforge.net➥
               /xsd/components.xsd" textPosition="bottom">
24           <jr:codeExpression><![CDATA[12345678]]></jr:codeExpression>
25         </jr:Code128>
26       </componentElement>
27       <rectangle radius="10">
28         <reportElement uuid="7d9510d1-3a1e-476a-88f1-b64b8920c6d5" x="54" y="227" width="100" height="20" ➥
               backcolor="#009900"/>
29       </rectangle>
30       <line>
31         <reportElement uuid="e330f4e3-d7a8-4a7c-bcb8-c8a067c116b5" x="75" y="27" width="100" height="49"/>
32       </line>
33       <rectangle>
34         <reportElement uuid="f2631a4e-b597-4aec-b683-fa5166cf8465" x="54" y="185" width="100" height="20" ➥
               backcolor="#FF0033"/>
35       </rectangle>
36       <componentElement>
37         <reportElement uuid="c26848c4-552f-4489-bb75-d463ed252d34" x="308" y="27" width="200" height="200"/>
38         <mp:map xmlns:mp="http://jasperreports.sourceforge.net/jasperreports/components" xsi:schemaLocation="➥
               http://jasperreports.sourceforge.net/jasperreports/components http://jasperreports.sourceforge.net/➥
               xsd/components.xsd">
39           <mp:latitudeExpression><![CDATA[47.5011151657f]]></mp:latitudeExpression>
40           <mp:longitudeExpression><![CDATA[19.0531965145f]]></mp:longitudeExpression>
41         </mp:map>
42       </componentElement>
43       <frame>
```

```
44          <reportElement uuid="782e7b7b-4aed-4ba8-875b-ddd916d9b99d" stretchType="RelativeToTallestObject" x="64" y="264" width="122" height="44" forecolor="#3300FF"/>
45          <staticText>
46            <reportElement uuid="57a2400c-5c52-4564-b8b2-7e19347eeac1" x="14" y="14" width="100" height="20"/>
47            <textElement/>
48            <text><![CDATA[Static text]]></text>
49          </staticText>
50        </frame>
51        <componentElement>
52          <reportElement uuid="444639ea-5513-4531-bf94-c781a48e1d31" x="275" y="281" width="200" height="200"/>
53          <hc:html xmlns:hc="http://jasperreports.sourceforge.net/htmlcomponent" xsi:schemaLocation="http://jasperreports.sourceforge.net/htmlcomponent http://jasperreports.sourceforge.net/xsd/htmlcomponent.xsd" scaleType="RetainShape" horizontalAlign="Left" verticalAlign="Middle">
54            <hc:htmlContentExpression><![CDATA["<h1>Alma</h2>"]]></hc:htmlContentExpression>
55          </hc:html>
56        </componentElement>
57        <image>
58          <reportElement uuid="01b10e2d-37e2-48b8-9e95-ab5a93348b80" x="22" y="327" width="211" height="154"/>
59          <imageExpression><![CDATA["/home/laz.jpg"]]></imageExpression>
60        </image>
61      </band>
62    </summary>
63  </jasperReport>
```



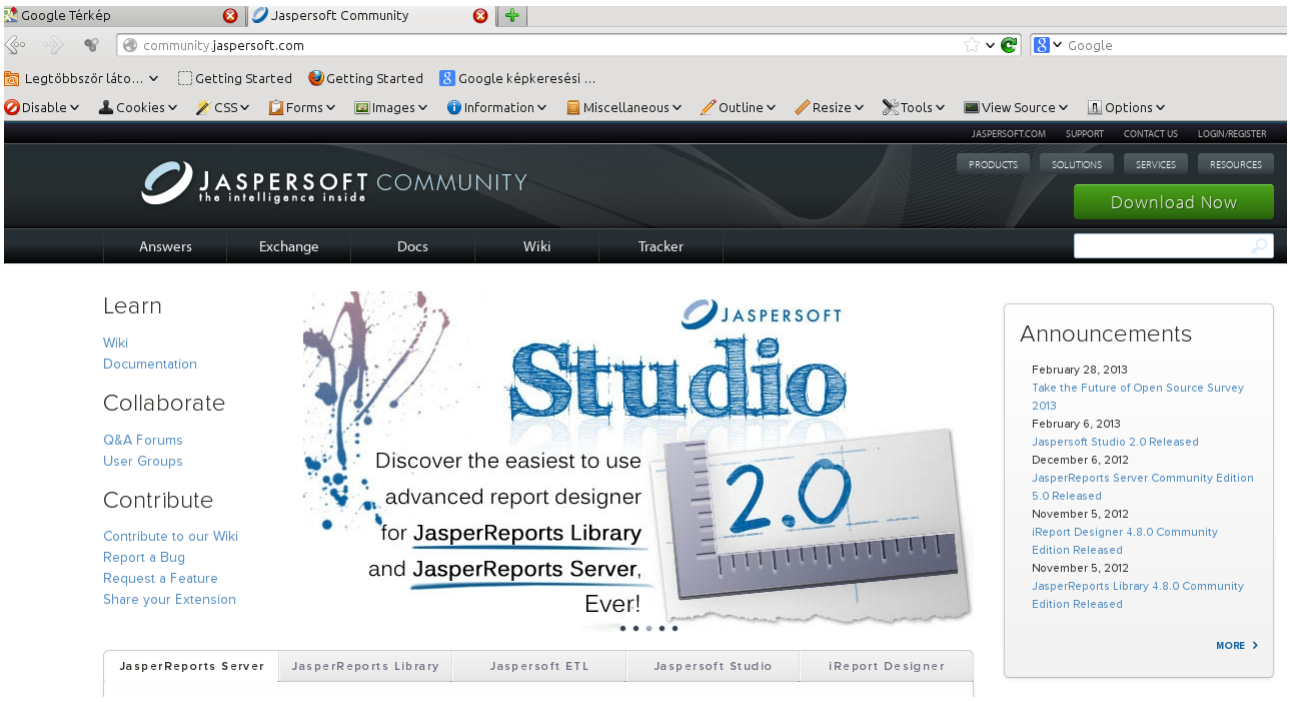


Alma

12.1. ábra: Grafikai elemek

Néhány kiegészítést érdemes tenni a fenti riportban lévő komponensek mostani konkrét használatáról:

- A 2 darab ellipszist a 15-20 sorok között adtuk meg.
- A vonalat a 30-32 sorok között definiáltuk.
- A vonaldiagram megadását a 21-26 sorok között látjuk. A megjelent 12345678 értéket a 24. sor tartalmazza, de itt bármilyen ismert JasperReports kifejezés is megadható lett volna.
- A zöld lekerekített téglalap a 27-29 sorok eredménye.
- A piros téglalap specifikációjának helye: 33-35 sorok.
- A *Google Maps* elem megadását a 36-42 sorok tartalmazzák, látható ahogy megadtuk a koordinátákat is, amik természetesen szintén lehetnek dinamikusan kiszámított értékek. A térkép komponensnek több más paramétere is van, például a zoom, de mi most csak az alapértelmezést használtuk.
- Az 51-56. sorok között található a HTML komponens.
- Végezetül a beszúrt kép az 57-60 sorok terméke. Az 59. sor egy kifejezésként tartalmazza a képfájl helyét, így itt is dinamikusan, akár soronként is változtatható egy-egy kép, ami például egy termék katalógus jelentésnél biztosan nagyon jól jön.



12.2. ábra: http://community.jaspersoft.com/