Utilización de JasperReports para emitir informes

El ejercicio final consiste en emitir un informe con JasperReports a partir de los datos obtenidos mediante consulta SQL de una base de datos.
Los pasos a dar son:

- Descargar JasperReports e integrarlo en el IDE de NetBeans.
- Diseñar y compilar la plantilla (jrxml)
- Crear el proyecto java

Descarga de JasperReports e integración en el IDE de NetBeans.

Lo descargamos en cualquier sitio y después:
Tools>Libraries>añadir librerías. Damos el nombre p.e. JasperReport374 y dejamos librería de clases.
En la pestaña Classpath tab añadimos el fichero jasperreports 3.7.4 de la carpeta dist y los cinco ficheros señalados en la imagen de la carpeta lib:

- ...beanutils-1.8.0.jar
- ...collections-2.1.1.jar
- ...digester-2.1.jar
- ...javaflow-20060411.jar
- …logging-1.1.1.jar

En la pestaña Sources añadimos el directorio src.
En la pestaña Javadoc añadimos la carpeta api que hay en la ruta
..jasperreports3.7.4\dist\docs\api
Y hemos terminado de integrar JasperReports en NetBeans.

Diseñar y compilar la plantilla.

Tools>Options>Miscelaneus>Files>New y ahí añadimos la nueva extensión jrxml.

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport
  PUBLIC "-//JasperReports//DTD Report Design//EN"
  "http://JasperReports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport name="HelloReportWorld">
   <detail>
      <band height="200">
         <staticText>
            <reportElement x="0" y="10" width="500" height="20"/>
            <text><![CDATA[Informe !Hola Mundo!]]></text>
         </staticText>
      </band>
   </detail>
</jasperReport>
```

Crear un nuevo proyecto

Añadir las librerías de JasperReport

Sobre la carpeta raíz del proyecto creamos un directorio llamado report y luego dos subdirectorios: templates y results.

-En la pestaña Files BD sobre el nombre del proyecto, elegimos File > new folder y ponemos el nombre a la carpeta report y seguimos el mismo proceso para templates y results.

Ahora en la pestaña Files BD sobre la carpeta templates y elegimos File >File/Folder del menú contextual y seleccionamos fichero vacío o sea Empty File

Le damos el nombre HolaMundo.jrxml

Añadimos el código html correspondiente.

A continuación nos vamos a la main hacemos los import:

```
Import java.util.*;
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.view.JasperViewer;
   public static void main(String[] args) {
      String reportSource="./reports/templates/HolaMundo.jrxml";
      String reportDest="./reports/results/HolaMundo11.html";
       Map<String, Object> params=new HashMap<String, Object>();
      try
      {
         JasperReport jasperReport=
               JasperCompileManager.compileReport(reportSource);

         JasperPrint jasperPrint=
               JasperFillManager.fillReport(
                jasperReport, params, new JREmptyDataSource());
                  JasperExportManager.exportReportToHtmlFile(
               jasperPrint,reportDest);

         JasperViewer.viewReport(jasperPrint);
      }
      catch(JRException ex)
      {
         System.out.println(ex.getMessage());
      }
   }
```

Explicación de las sentencias utilizadas:

JasperCompileManager.compileReport

El método compileReport toma la plantilla XML JasperReports y la compila a format byte code (una instancia JasperReports), que puedes serializar a disco como un fichero .jasper.

Se pueden reutilizar las plantillas compiladas para generar informes sin compilarlas a menos que hayas cambiado el fuente de la plantilla.

JasperFillManager.fillReport

El método fillReport toma el informe compilado (la instancia JasperReports), unos pocos parámetrosdefinidos por el usuario y una fuente de datos JasperReports, y rellena la instancia

informe con parámetros y datos. En este sencillo ejemplo de Hola Mundo, no rellena nada pero retorna una instancia JasperPrint.
La instancia JasperPrint es una representación del objeto del informe completo. Ahora todo lo que hay que hacer es instruir a JasperReports para que escriba a disco en los formatos deseados.

JasperExportManager.exportReportToHtmlFile

El método exportReport toma la instancia JasperPrint y una ruta de destino de fichero y crea un fichero HTML con el contenido del informe. Además de HTML, JasperReports framework soporta exportar a PDF, XML, CSV, XLS, RTF, fichero de texto. También se pueden crear formatos de fichero personalizados para exportar con la API extensible de JasperReports.

JasperViewer.viewReport

El método viewReport muestra el informe generado en el visor JasperReport Viewer

Continuación del ejemplo 2

Ahora vamos a hacer que obtenga datos de una base de datos. En concreto de la base de datos derby que se incluye al instalar el jdk.

Modificar la platilla .jrxml

Añadimos a la plantilla HolaMundo.jrxml el título, autor y fecha.

```
<jasperReport name="HelloReportWorld">
<parameter name="reportTitle" class="java.lang.String"/>
<parameter name="author" class="java.lang.String"/>
<parameter name="startDate" class="java.lang.String"/>
```

Se añade una sección <title/> entre las secciones <parameter> y <detail>

```
<title>
  <band height="60">
    <textField>
      <reportElement x="0" y="10" width="500" height="40"/>
      <textElement textAlignment="Center">
        <font size="24"/>
      </textElement>
      <textFieldExpression class="java.lang.String">
        <![CDATA[$P{reportTitle}]]>
      </textFieldExpression>
    </textField>
    <textField>
      <reportElement x="0" y="40" width="500" height="20"/>
      <textElement textAlignment="Center"/>
      <textFieldExpression class="java.lang.String">
        <![CDATA["Ejecutado por: " + $P{author}
          + " on " + $P{startDate}]]>
      </textFieldExpression>
    </textField>
  </band>
</title>
```

A continuación añadimos el código que sigue al método main de la clase Ppal.

```
    params.put("reportTitle", "Informe Hola Mundo");
    params.put("author", "Fulano Mengano Zutano");
    params.put("startDate", (new java.util.Date()).toString());
```

A continuación nos conectamos a la base de datos

En la pestaña Services nos aparecen las conexiones a bases de datos. Conectamos con la base de datos derby con BD y connect.

Vemos las tablas de APP, una de las cuales es CUSTOMER

Añadimos a la plantilla los campos necesarios:

Añadimos CUSTOMER_ID,NAME,CITY y STATE, y la consulta entre las secciones <parameter> y <title> en l la plantilla del informe, o sea en HolaMundo.jrxml.

El orden de las secciones es importante. Un orden incorrecto producirá errores. Cuando se añade la sección queryString hay que asegurarse de insertarla después de la sección parameter y antes de los campos.

```
<queryString>
   <![CDATA[SELECT * FROM CUSTOMER ORDER BY STATE, CITY]]>
</queryString>
<field name="CUSTOMER_ID" class="java.lang.Integer"/>
<field name="NAME" class="java.lang.String"/>
<field name="CITY" class="java.lang.String"/>
<field name="STATE" class="java.lang.String"/>
```

Añadimos la sección <columnHeader> entre las secciones <title> y <detail>

```
<columnHeader>
   <band height="30">
      <rectangle>
         <reportElement x="0" y="0" width="500" height="25"/>
         <graphicElement/>
      </rectangle>
      <staticText>
         <reportElement x="5" y="5" width="50" height="15"/>
         <textElement/>
         <text><![CDATA[ID]]></text>
      </staticText>
      <staticText>
         <reportElement x="55" y="5" width="150" height="15"/>
         <text><![CDATA[Nombre]]></text>
      </staticText>
      <staticText>
         <reportElement x="205" y="5" width="255" height="15"/>
         <text><![CDATA[Ciudad, Estado]]></text>
      </staticText>
   </band>
</columnHeader>
```

Seccion detalle

Sustituimos la actual sección <detail> por la siguiente con los nuevos campos que estamos añadiendo

```
<detail>
```

```
  <band height="20">
    <textField>
      <reportElement x="5" y="0" width="50" height="15"/>
      <textElement/>
      <textFieldExpression class="java.lang.Integer">
        <![CDATA[$F{CUSTOMER_ID}]]>
      </textFieldExpression>
    </textField>
    <textField>
      <reportElement x="55" y="0" width="150" height="15"/>
      <textElement/>
      <textFieldExpression class="java.lang.String">
        <![CDATA[$F{NAME}]]>
      </textFieldExpression>
    </textField>
    <textField>
      <reportElement x="205" y="0" width="255" height="15"/>
      <textElement/>
      <textFieldExpression class="java.lang.String">
        <![CDATA[$F{CITY} + ", " + $F{STATE}]]>
      </textFieldExpression>
    </textField>
  </band>
</detail>
```

Añadir la librería Java DB Driver

Por último tenemos que añadir la librería JavaDB al proyecto en Tools y luego Libraries JabaDB.

Observando el código

Sustituir el bloque de código que teníamos entre el try y el catch en el main de la clase Main con el código que se ve en la imagen.

Añadir los imports necesarios. O con el bd en el editos y Fix Imports o en el editor con Alt_Shift_F

Sustituimos el código que teníamos entre el try y el catch por el siguiente

```
try
    {
      //Compilar la plantilla
      JasperReport jasperReport=
           JasperCompileManager.compileReport(reportSource);

      //Habilitar el driver necesario
      Class.forName("org.apache.derby.jdbc.ClientDriver");

      //Crear una conexion para pasar el informe
```

```
        java.sql.Connection conn=DriverManager.getConnection(
              "jdbc:derby://localhost:1527/sample","app","app");
        //Sustituir el parametro datasource JR vacio por
        //el parametro de conexion conn
        JasperPrint jasperPrint=
              JasperFillManager.fillReport(
                jasperReport, params, conn);

        JasperExportManager.exportReportToHtmlFile(
              jasperPrint,reportDest);

        JasperViewer.viewReport(jasperPrint);
      }
      catch(JRException ex)
```

En realidad lo que hace es añadir la conexión a apache derby y sustituir:

```
JasperPrint jasperPrint=
              JasperFillManager.fillReport(
                jasperReport, params, new JREmptyDataSource());
por
        JasperPrint jasperPrint=
              JasperFillManager.fillReport(
                jasperReport, params, conn);
```