

JasperReport desde Java



JasperReports

Índice

Utilizar JasperReport desde Java.....	3
Librerías.....	3
Ruta de los archivos.....	4
1.- Informe plano.....	6
2.- Informe con datos recogidos de una BBDD.....	7
3.- Editar el informe en tiempo de ejecución.....	7
Funciones útiles que se pueden hacer con un informe.....	9
Anexo: Clases utilizadas en la guía.....	11

Utilizar JasperReport desde Java

Guía para Previsualizar/Generar informes creados con **JasperSoft Studio** desde un programa escrito en Java.

Librerías

Para poder utilizar JasperReport desde Java se necesitan importar varias librerías:

- commons-beanutils-1.9.0.jar
- commons-collections-3.2.1.jar
- commons-designer-2.1.jar
- commons-logging-1.1.1.jar
- ecj-4.3.1.jar
- itextpdf-5.5.0.jar
- itext-pdfa-5.5.0.jar
- jasperreports-6.0.0.jar
- jasperreports-fonts-6.0.0.jar
- jasperreports-functions-6.0.0.jar
- joda-time-2.4.jar

Las librerías itextpdf y itext-pdfa son necesarias solamente si se va a generar y guardar el informe en formato PDF.

La librería jasperreports-functions es necesaria en caso de que el informe contenga fórmulas matemáticas (PRODUCT(), SUM(), FALSE(), etc.).

	commons-beanutils-1.9.0.jar	07/12/2013 15:30	Executable Jar File	226 KB
	commons-collections-3.2.1.jar	17/01/2014 17:55	Executable Jar File	562 KB
	commons-digester-2.1.jar	19/08/2013 17:12	Executable Jar File	193 KB
	commons-logging-1.1.1.jar	19/08/2013 17:12	Executable Jar File	60 KB
	ecj-4.3.1.jar	03/06/2014 10:36	Executable Jar File	1.788 KB
	itextpdf-5.5.0.jar	06/08/2014 14:23	Executable Jar File	2.066 KB
	itext-pdfa-5.5.0.jar	06/08/2014 14:23	Executable Jar File	54 KB
	jasperreports-6.0.0.jar	25/11/2014 9:38	Executable Jar File	5.024 KB
	jasperreports-fonts-6.0.0.jar	25/11/2014 9:38	Executable Jar File	2.421 KB
	jasperreports-functions-6.0.0.jar	14/02/2019 19:36	Executable Jar File	32 KB
	joda-time-2.4.jar	27/07/2014 17:42	Executable Jar File	573 KB

Nota: Las versiones pueden variar, pero esta combinación funciona.

Nota2: En caso de estar trabajando en Linux es posible que no reconozca alguna de las fuentes utilizadas en el informe. Para solucionarlo hay que instalar el paquete mscorefonts:
sudo apt-get install ttf-mscorefonts-installer

Ruta de los archivos

A la hora de trabajar con los informes es muy importante saber donde hay que poner los archivos **.jrxml**, los **.jasper** y las imágenes que utilizan los informes.

En el archivo **.jrxml** generado por cada uno de los informes no se establecen rutas a la hora de referenciar las imágenes, por lo que al compilar el programa **Java** solamente busca en el directorio desde el que se compila, que en el caso de **Eclipse** es el directorio del proyecto, por lo que se deberían poner ahí las imágenes o editar el archivo **.jrxml** y establecer la ruta hacia la imagen.

📁 .settings	14/02/2019 15:52	Carpeta de archivos	
📁 bin	14/02/2019 20:03	Carpeta de archivos	
📁 src	14/02/2019 19:42	Carpeta de archivos	
📄 .classpath	14/02/2019 20:03	Archivo CLASSPATH	2 KB
📄 .project	14/02/2019 15:52	Archivo PROJECT	1 KB
📄 factura_Subreport.jasper	14/02/2019 15:48	Archivo JASPER	31 KB
📄 facturaJardineria.jasper	14/02/2019 15:48	Archivo JASPER	45 KB
🖼️ leaf_banner_violet.png	14/02/2019 19:02	Archivo PNG	502 KB
🖼️ tree1.png	12/02/2019 8:08	Archivo PNG	84 KB
🖼️ tree2.png	12/02/2019 8:08	Archivo PNG	35 KB

facturaJardineria.jrxml ✕

```

101      </groupHeader>
102      <groupFooter>
103          <band/>
104      </groupFooter>
105  </group>
106  <background>
107      <band splitType="Stretch"/>
108  </background>
109  <title>
110      <band height="132" splitType="Stretch">
111          <image>
112              <reportElement x="2" y="0" width="118" height="122" uuid="d47168ce-aae
113              <imageExpression><![CDATA["tree1.png"]]></imageExpression>
114          </image>
115          <staticText>
116              <reportElement style="Title" x="130" y="0" width="243" height="79" uui
117              <textElement textAlignment="Center">
118                  <font size="60" isBold="false"/>
119              </textElement>
120              <text><![CDATA[Factura]]></text>
121          </staticText>
122      <!--
123      <componentElement>
124          <reportElement x="379" y="0" width="176" height="131" uuid="8e26740e-2
125          <jr:QRCode xmlns:jr="http://jasperreports.sourceforge.net/jasperreport
126              <jr:codeExpression><![CDATA["Pedido: " + ${CodigoPedido} + "\nFec

```

Al utilizar Subreports, pasa exactamente lo mismo con los archivos **.jasper**, que los busca solamente en el directorio desde el que se compila el programa, por lo que la solución es la misma.

```
facturaJardineria.jrxml
296 <textElement>
297   <font size="16" isBold="true"/>
298 </textElement>
299 <text><![CDATA[Cliente:]]></text>
300 </staticText>
301 <textField isStretchWithOverflow="true" isBlankWhenNull="true">
302   <reportElement style="Detail" positionType="Float" x="130" y="30" width="150" height="20" uuid="
303   <textElement textAlignment="Left">
304     <font size="12" isBold="true"/>
305   </textElement>
306   <textFieldExpression><![CDATA[$F{CodigoPedido}]]></textFieldExpression>
307 </textField>
308 <frame>
309   <reportElement mode="Opaque" x="0" y="166" width="555" height="24" forecolor="#B89F7D" backcol
310 </frame>
311 <subreport isUsingCache="false" overflowType="Stretch">
312   <reportElement x="-20" y="170" width="552" height="336" uuid="9db00aaf-19c0-4f3b-91ce-87f5425a
313   <subreportParameter name="CodigoPedido">
314     <subreportParameterExpression><![CDATA[$P{CodigoPedido}]]></subreportParameterExpression>
315   </subreportParameter>
316   <connectionExpressions><![CDATA[$P{REPORT_CONNECTION}]]></connectionExpressions>
317   <subreportExpression><![CDATA["factura_Subreport.jasper"]]></subreportExpression>
318 </subreport>
319 <staticText>
320   <reportElement positionType="Float" x="-20" y="100" width="13" height="140" uuid="5e90d2af-71a
321   <textElement rotation="Left"/>
322   <text><![CDATA[Formato Factura 100/2019]]></text>
323 </staticText>
324 </hands>
```

Los archivos **.jrxml** son los que tendremos que referenciar en el programa **Java** a la hora de trabajar con el informe, por lo que pueden estar en la ruta deseada, pero a la hora de trabajar con ellos hay que indicar dicha ruta.

Ejemplo: El archivo **Leaf_Violet.jrxml** se ha situado en el directorio **/src/Informes** dentro del proyecto de Eclipse.

```
14 public class Report2PDFnoBBDD {
15   // Atributos
16   private static final String ARCHIVO_JRXML = "./src/Informes/Leaf_Violet.jrxml";
17   private static final String ARCHIVO_DESTINO = "./src/Informes/Leaf_Violet.pdf";
18
19   // Main
20   public static void main(String[] args) {
21     try {
22       // Primero se compila el archivo JRXML
23       JasperReport jasperReport = JasperCompileManager.compileReport(ARCHIVO_JRXML);
24
25       // Parametros para el informe
26       Map<String, Object> parameters = new HashMap<String, Object>();
27
28       // Fuente de datos
29       JRDataSource dataSource = new JREmptyDataSource();
30
31       // Print
32       JasperPrint print = JasperFillManager.fillReport(jasperReport, parameters, dataSource);
33
34       // Previsualiza el diseño del informe
35       JasperDesignViewer.viewReportDesign(jasperReport);
36
37       // Previsualiza el informe generado
38       JasperViewer.viewReport(print);
39
40       // Esportador a PDF
41       JasperExportManager.exportReportToPdfFile(print, ARCHIVO_DESTINO);
42     }
43   }
44 }
```

1.- Informe plano

Vamos a trabajar con un Informe creado con **JasperSoft Studio** desde una clase **Java**.

Lo primero que hay que hacer es compilar el archivo **informe.jrxml**:

```
JasperReport jasperReport = JasperCompileManager.compileReport(ARCHIVO_JRXML);
```

*ARCHIVO_JRXML es la ruta del propio archivo

Después, hay que definir un objeto **Map<String, Object>** a través del cual se le pasarán los parámetros al informe. En caso de que el informe no espere recibir ningún parámetro, se instancia el objeto igual pero no se le añade ningún parámetro.

```
Map<String, Object> parameters = new HashMap<String, Object>();  
parameters.put("CodigoPedido", 2);
```

El primer elemento que se le pasa al método `put(String, Object)` de la clase **Map** es el nombre del parámetro, y el segundo elemento es el valor que va a recibir, funciona de la misma manera que un 'diccionario' en otros lenguajes de programación.

En el siguiente paso tenemos que instanciar un objeto de la clase **JRDataSource** (en el caso de que no se utilice una base de datos para generar los datos del informe, en cuyo caso ver el apartado siguiente):

```
JRDataSource dataSource = new JREmptyDataSource();
```

Por último le tenemos que definir una "impresora" para este informe, a la que le pasamos el informe compilado, los parámetros que va a recibir y la fuente de datos por el constructor:

```
JasperPrint print = JasperFillManager.fillReport(jasperReport, parameters,  
dataSource);
```

Esta "impresora" es lo utilizaremos para trabajar con el informe, se pueden ver algunas de las funciones en el último apartado de este documento.

2.- Informe con datos recogidos de una BBDD

La base es la misma que en el anterior caso, solo que cambiando algunos detalles.

En este caso concreto, lo primero que hay que hacer es cargar el Driver del gestor de bases de datos que se vaya a utilizar y conectar con la base de datos que contiene la información.

```
Class.forName(DRIVER);
Connection conexion = DriverManager.getConnection(CONECTOR_BBDD, "usuario",
"usuario");
```

Después de haber realizado la conexión con la base de datos hay que preparar el informe para trabajar con él de la misma manera que hicimos en el caso anterior, solo que no hace falta instanciar el objeto **JRDataSource** ya que **Connection** realiza la misma función. Al instanciar **JasperPrint** se le pasará el objeto **Connection** en lugar del **JRDataSource**.

```
JasperPrint print = JasperFillManager.fillReport(jasperReport, parameters,
conexion);
```

3.- Editar el informe en tiempo de ejecución

Para poder editar un informe en tiempo de ejecución, antes de compilar el archivo **informe.jrxml** hay que instanciar un objeto de la clase **JasperDesign**:

```
JasperDesign jasperDesign = JRXmLloader.Load(ARCHIVO_JRXML);
```

Utilizando el objeto **jasperDesign** podemos obtener mucha información, como la consulta que utiliza el informe, los márgenes, etc. Entre ellos tiene el método **getAllBands()** que devuelve un objeto **JRBand[]** que contiene todas las bandas del informe.

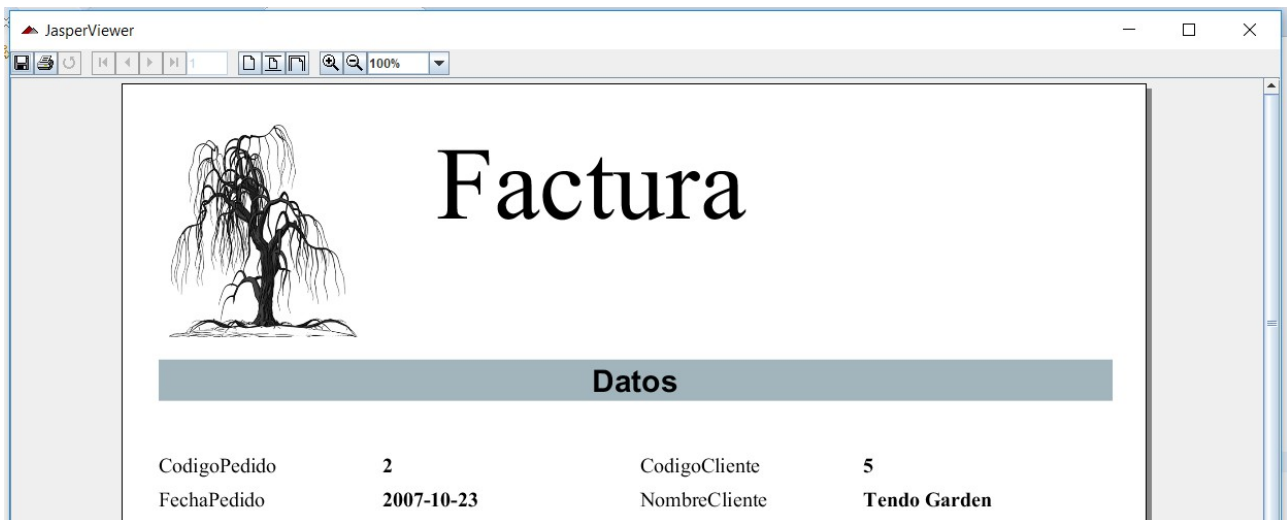
```
JRBand[] bands = jasperDesign.getAllBands();
```

De cada una de las bandas podemos obtener todos los elementos o un elemento a partir de su **key**. Una vez tenemos un elemento, se le puede hacer un casting al tipo de elemento que es (por ejemplo **JRTextElement**) y cambiar el tamaño, color, tipo de letra, etc.

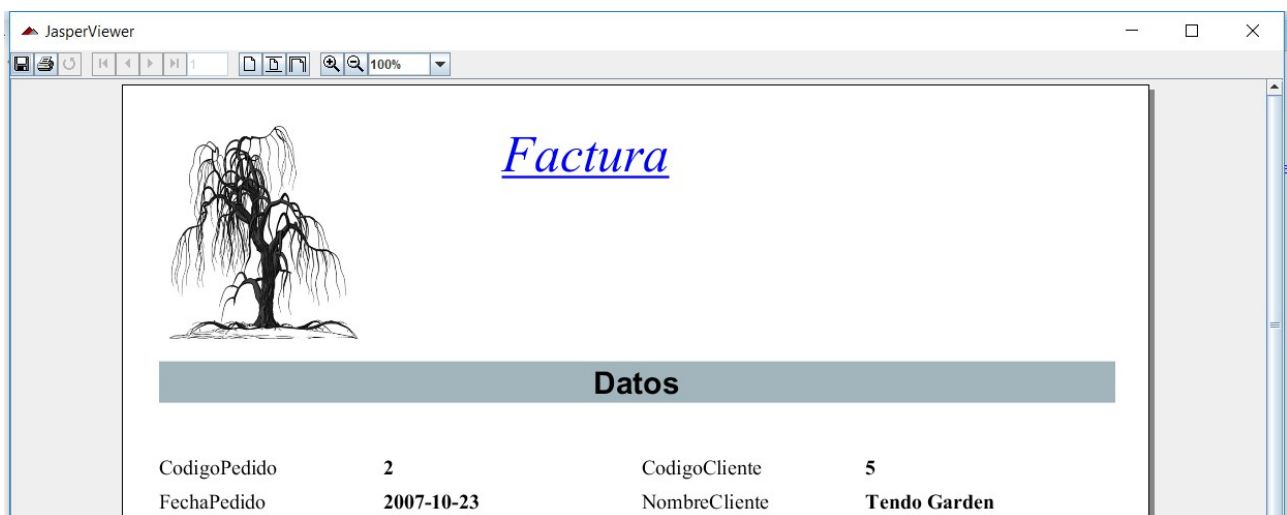
```
for(JRBand band : bands) {
    JRElement[] elements = band.getElements();
    for(JRElement element : elements) {
        if(element.getUUID().toString().equals("f49d68fd-a2c7-44ec-a5fc-
9300d10c85a1")) {
            JRTextElement txt = (JRTextElement) element;
            txt.setItalic(true);
            txt.setUnderline(true);
            txt.setFontSize(30);
            txt.setForeground(Color.BLUE);
            txt.setBackgroundColor(Color.BLUE);
        }
    }
}
```

Nota: El método **setFontSize(int)** está **deprecated** pero por ahora funciona igualmente.

Este es el título del informe sin modificar:



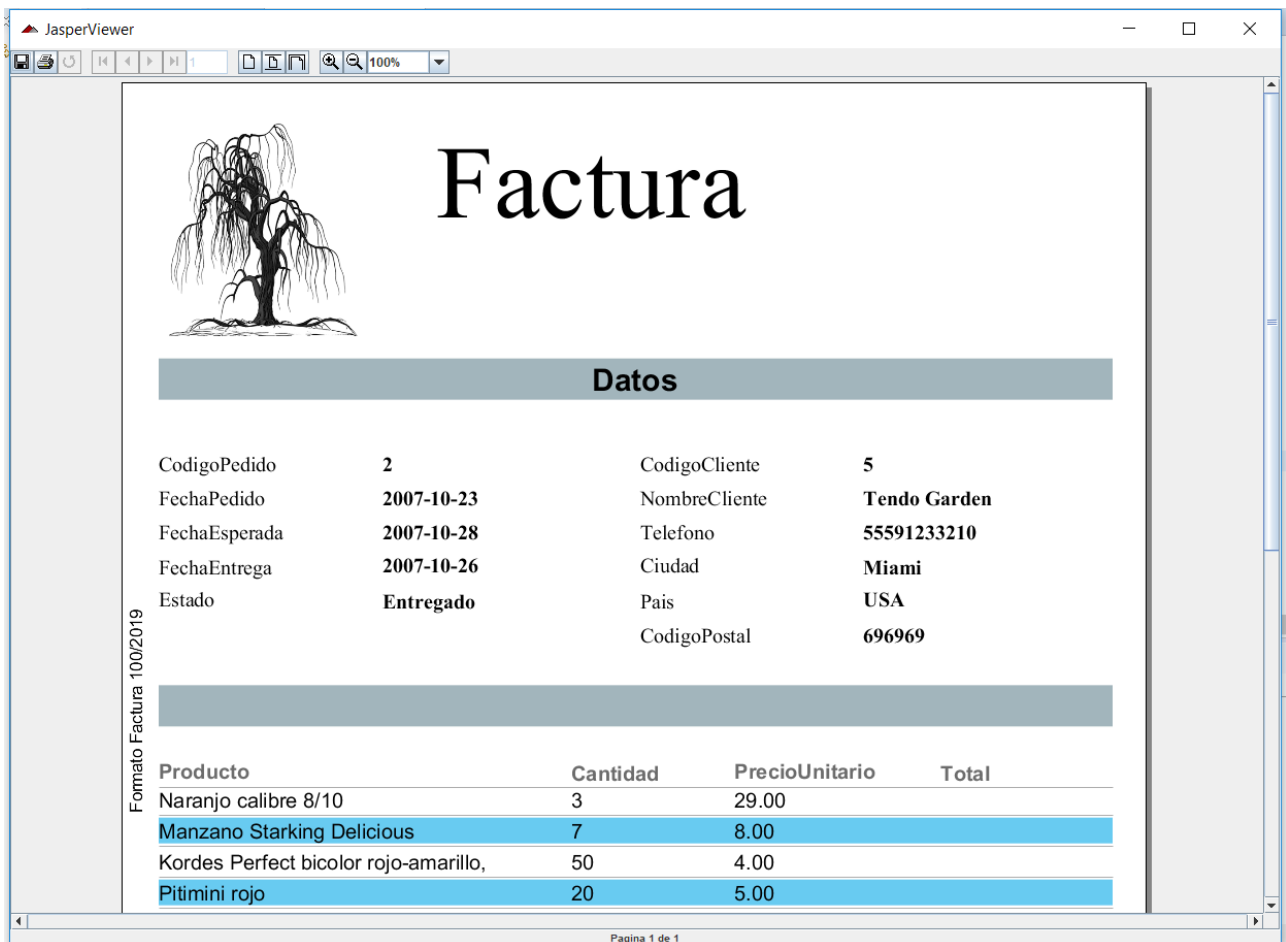
Y este es el título del informe después de ejecutar el código anterior:



Funciones útiles que se pueden hacer con un informe

- Previsualizar el diseño del informe:
`JasperDesignViewer.viewReportDesign(jasperReport);`
- Previsualizar el resultado del informe generado:
`JasperViewer.viewReport(print);`
- Exportar el informe generado en formato PDF:
`JasperExportManager.exportReportToPdfFile(print, ARCHIVO_DESTINO);`

Previsualización de la factura de la Jardinería generada en los ejemplos utilizados en esta guía. Desde esta ventana se puede guardar el fichero en muchos formatos, entre ellos PDF, ODT, DOCX y HTML.



Factura

Datos

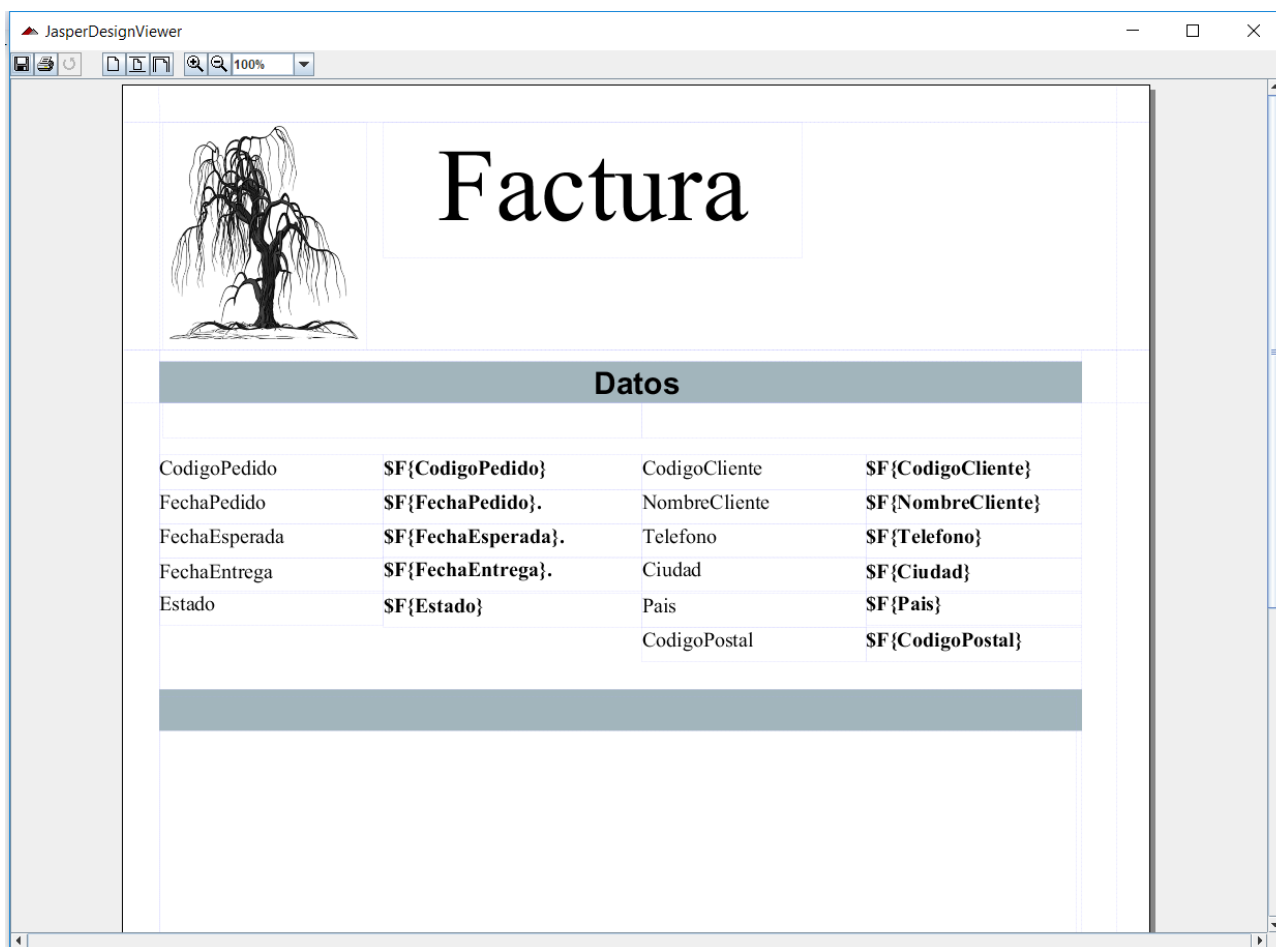
CodigoPedido	2	CodigoCliente	5
FechaPedido	2007-10-23	NombreCliente	Tendo Garden
FechaEsperada	2007-10-28	Telefono	55591233210
FechaEntrega	2007-10-26	Ciudad	Miami
Estado	Entregado	Pais	USA
		CodigoPostal	696969

Producto	Cantidad	PrecioUnitario	Total
Naranja calibre 8/10	3	29.00	
Manzano Starking Delicious	7	8.00	
Kordes Perfect bicolor rojo-amarillo,	50	4.00	
Pitiminí rojo	20	5.00	

Formato Factura 100/2019

Página 1 de 1

Previsualización del diseño del mismo informe anterior, también puede guardarse en muchas extensiones de archivo diferentes.



Anexo: Clases utilizadas en la guía

JasperReport:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JasperReport.html>

JasperCompileManager:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JasperCompileManager.html>

Map:

<https://docs.oracle.com/javase/8/docs/api/java/util/Map.html>

JRDataSource:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRDataSource.html>

JasperPrint:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JasperPrint.html>

JasperDesign:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/design/JasperDesign.html>

JRBand:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRBand.html>

JRElement:

<http://javadocx.com/net.sf.jasperreports/jasperreports/5.5.0/net/sf/jasperreports/engine/JRElement.html>

JRTextElement:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JRTextElement.html>

JasperDesignViewer:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/view/JasperDesignViewer.html>

JasperViewer:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/view/JasperViewer.html>

JasperExportManager:

<http://jasperreports.sourceforge.net/api/net/sf/jasperreports/engine/JasperExportManager.html>