=======================================================

***Geração de Relatório com JasperReports e iReport***

=======================================================

# **Introdução**

Basicamente, o processo de geração de relatório resume-se na definição do design e mapeamento de dados para campos dentro de um layout definido. Nesse contexto, surgiram ferramentas comerciais com intuito de auxiliar neste processo.

# **JasperReports**

JasperReports é um poderoso framework open-source para geração de relatórios. Escrito em Java, essa biblioteca apresenta grande habilidade na organização e apresentação de conteúdo, permitindo a geração dinâmica de relatórios em diversos formatos, como PDF, HTML, XLS, CSV e XML, podendo ainda ser utilizada em qualquer aplicação Java, incluindo aplicações desktop, Web e distribuídas.

# **Funcionamento**

O design do relatório, incluindo a localização dos campos a serem preenchidos e seus respectivos nomes, para futuro mapeamento, são definidos em um arquivo **XML** através de tags **XML** que obedecem a uma estrutura, vocabulário e restrições declarados em um arquivo **DTD (jasperreports.dtd).**

Usando **XML**, o designer pode definir textos estáticos, imagens, linhas, formas geométricas, como retângulos e elipses, e suas localizações dentro do relatório. Pode-se, ainda, e principalmente, definir os campos que serão preenchidos dinamicamente a partir de uma base de dados.

O arquivo **XML** precisa ser compilado, gerando um arquivo **.jasper**, isto é, contendo a versão compilada do código **XML**. Isto implica na compilação de todas as expressões Java definidas no arquivo **XML**, sendo realizadas várias verificações em tempo de compilação.

Diferentes objetos JasperReports são usados para representar as etapas do processo de geração de relatório:

* **JasperDesign:** Representa a definição do relatório. A partir do template XML é criado um JasperDesign.
* **JasperReport:** Representa o JasperDesign compilado. O processo de compilação verifica o design do relatório e compila o design em um objeto JasperReport.
* **JasperPrint:** Representa o relatório gerado. É criado um JasperPrint a partir de um JasperReport, contendo o relatório preenchido.

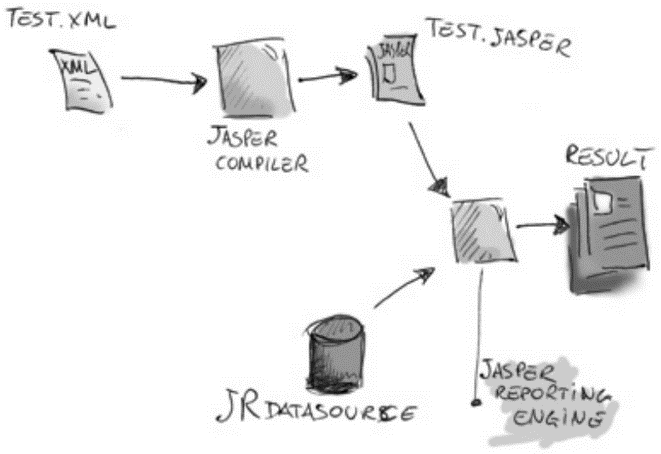
# **Dados**

Para produzir um relatório precisamos fornecer dados ao Jasper. Estes dados podem ser recuperados de diferentes lugares, como de uma base de dados em um SGBD ou em um arquivo XML.

Para recuperarmos informações de um banco de dados relacional, precisamos realizar uma consulta (**query**) em linguagem SQL. Essa query pode ser inserida ao código XML ou ser realizada por uma classe Java, gerando um objeto ResulSet, que será passado às classes do Jasper para o preenchimento do relatório.

O JasperReports suporta vários tipos de **datasources** (fonte de dados) através de uma interface específica chamada **JRDataSource**.

Há uma implementação padrão desta interface para objetos ResultSet, chamada **JRResultSetDataSource**, ou seja, é possível realizar consultas, gerando objetos ResultSet e passando ao JasperReports para o preenchimento do relatório. Quando a fonte de dados é um ResultSet, este objeto deve conter todas as colunas a serem mapeadas para seus campos correspondentes no relatório.



No linguajar “Jasper", um **datasource** somado a um arquivo **.jasper** gera um **"print"**, que pode ser **"exportado"** para os formatos PDF, HTML, XML, CVS ou XLS.

# **Campos, Parâmetros, Variáveis e Expressões**

**Campos (Fields)** são “áreas específicas” que receberão diretamente os dados das respectivas colunas referenciadas. O relatório deve conter um campo com o mesmo nome da coluna a qual faz referência.

Por exemplo, para os dados da coluna **Nome** do tipo VARCHAR, da tabela **Cliente**, serem mapeados para o relatório, um campo **Nome** deve ser definido no arquivo XML da seguinte forma:

<field name = “Nome” class = “java.lang.String”/>

**Parâmetros** são dados passados para a operação de preenchimento, que não podem ser encontrados normalmente na fonte de dados. São declarados, por exemplo, da seguinte forma:

<parameter name = “TituloDoRelatorio” class = “java.lang.String”/>

E passados via código Java, através da classe HashMap:

Map parametros = new HashMap( );

parametros.put( “Cliente”, “Carlos Paixão” );

Outra importante utilização de parâmetros é na query do relatório. Por exemplo:

Select \* FROM CLIENTE WHERE CLIENTE=$P{Cliente}

O relatório será gerado apenas para o cliente passado por parâmetro.

**Variáveis** são utilizadas para simplificar o projeto do relatório. Através de uma variável podemos definir somente uma vez uma **expressão**, que seja usada frequentemente durante o design do relatório, chamando-a quando precisarmos daquela funcionalidade. Elas podem referenciar tipos internos de cálculos, como contagem (count), soma (sum), média (average), menor (lowest), maior (highest), etc. Por exemplo, o cálculo do valor total da compra:

Em uma expressão, uma variável pode referenciar outras variáveis do relatório, mas somente se aquelas variáveis foram definidas previamente no projeto do relatório. Assim, a ordem em que as variáveis são declaradas no relatório é importante.

Para as variáveis que executam o cálculo nós podemos especificar o nível em que devem ser reinicializadas. O nível Report (de relatório) significa que a variável será inicializada somente uma vez, no começo do relatório, e que executa o cálculo especificado até que o fim do relatório seja alcançado.

Mas nós podemos escolher executar o cálculo em nível de página, coluna ou de grupo. O exemplo abaixo demonstra o mesmo cálculo anterior em nível de página. Nossa variável será inicializada com zero no começo de cada nova página:

<variable name = "ValorTotalCompraSum” class = "java.lang.Double” resetType = "Page” calculation = "Sum”>

<variable expression> ${ValorProduto} </variable expression>

<initialValueExpression> new Double( 0 ) </initialValueExpression>

</variable>

Existem também variáveis internas da ferramenta, com nomes “autoexplicativos”, prontas para o uso nas expressões: PAGE\_NUMBER, COLUMN\_NUMBER, REPORT\_COUNT, PAGE\_COUNT, COLUMN\_COUNT, GroupName\_COUNT.

**Expressões (Expressions)** são utilizadas para especificar o conteúdo de campos de texto, na realização de cálculos frequentes, por exemplo. Todas elas são expressões Java que podem conter em sua sintaxe: campos, parâmetros e variáveis de relatório. Por exemplo:

<textFieldExpression>

“Sr.(a) ” + $F{Cliente} + " realizou um total de compras no valor de " + $V{ValorTotalCompraSum} + " no dia " + (new SimpleDateFormat("dd/MM/yyyy")).format($F{DataCompra}) + "."

</textFieldExpression>

# **Layout**

Para a melhor organização e definição do design do relatório, o JasperReports divide o layout em áreas “pré-definidas”, chamadas seções. Ao projetar um relatório nós necessitamos definir a posição do conteúdo dentro de uma seção, levando em consideração o que ela representa na estrutura visual de um relatório. A seguir estão as seções em que é baseado o layout de relatório: **title, pageHeader, columnHeader, groupHeader, detail, groupFooter, columnFoter, pageFooter, summary.**

=======================================================

***iReport***

=======================================================

Criar o design do relatório diretamente em XML pode ser uma tarefa custosa. Necessitava-se, então, de uma ferramenta que automatizasse esse processo. O iReport veio preencher essa lacuna, permitindo e definir o design do relatório dentro de um ambiente gráfico, contento “todos” os recursos que a biblioteca Jasper oferece.

É possível definir relatórios com designs modernos e complexos sem se quer escrever uma linha de código XML, que é todo gerado automaticamente. O ambiente ainda oferece atalhos para tarefas de compilação e visualização do relatório, permitindo a realização de testes, acelerando assim o processo de design.

É importante salientar que existem outras ferramentas com o mesmo objetivo que o iReport, mas que não são suficientemente maduras, no que diz respeito a facilidade de uso, e principalmente, no suporte as tags XML do JasperReports.

# **Fonte de Dados**

O iReport dá suporte a conexões JDBC, ODBC, e à 4 tipos de datasources:

**1 – Empty data source (JREmptyDatasource):** é um especial datasource usado para preencher relatórios que não possuem registros ou dados recuperados. Este datasource é usado quando é pressionado o botão “run”:

**2 – XML DataSource:** é um datasource capaz de empacotar um arquivo XML e normalizar seu conteúdo. As únicas informações necessárias para criar este tipo de datasource são: o nome do datasource e o nome do arquivo XML.

**3 – JavaBeans Set Datasource:** é um datasource capaz de empacotar uma Collection ou um Array de JavaBeans. É necessário uma classe especial de fábrica (factory) que forneça um método estático para gerar a coleção ou um array de JavaBeans. Para criar este datasource você precisa de um nome para o datasource, o nome da classe que fornece o método estático para recuperar o Array/Collection de objetos e o nome deste método, que terá uma definição como esta:

public static Collection createMyCollection( )

ou

public static Object[ ] createMyArray( )

É necessário setar o tipo de resultado (Collection ou Array).

**4 – Custom Datasource:** este tipo de datasource é genérico. O iReport não sabe como a interface JRDataSource é implementada por esta conexão particular, mas isto não é importante. É necessário uma classe especial de fábrica (factory) que forneça um método estático que retorne um JRDataSource. Para criar este datasource você precisa do nome do datasource, do nome da classe que fornece o método estático para recuperar o JRDataSource, e do nome deste método que terá uma definição como esta:

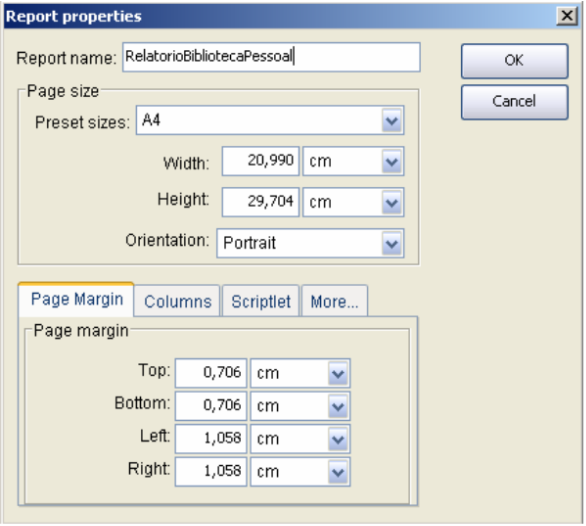
public static JRDataSource createMyJRDataSource( )

# **Criando o Design do Relatório**

Considerando que o teste de conexão com o banco foi bem-sucedido iniciamos a etapa de design do relatório.

**1.** Vá ao menu **“ File >> New Document ”** ou clique no botão:

**2.** Irá aparecer a janela **“Report Properties”** onde damos um nome ao nosso relatório **(Report Name)** chamando-o de **“RelatorioBibliotecaPessoal”** (não pode haver espaço entre as palavras). Ainda na mesma janela podemos configurar o tamanho da página, a sua orientação, as margens, o comprimento, largura e o espaço entre as colunas, além de outras opções mais avançadas.

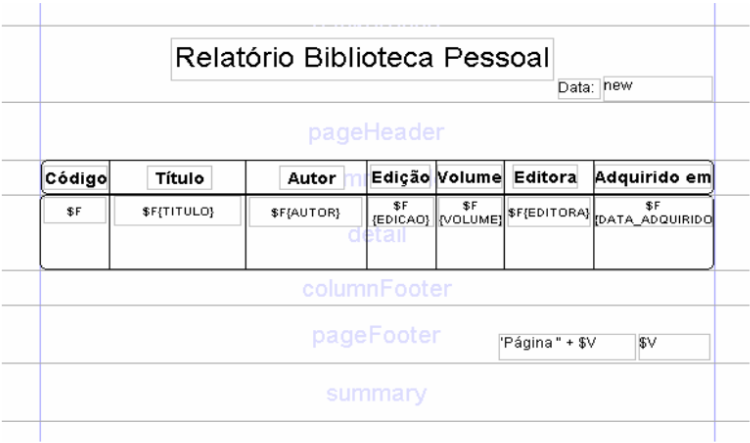


**3.** Agora podemos criar o design de nosso relatório (fig. 8), utilizando a barra de ferramentas:

Inserimos os elementos do relatório, bastando clicarmos uma vez sobre o respectivo botão e logo após no relatório. O conteúdo dos elementos de texto e as configurações de fonte estão disponíveis através de um duplo clique sobre o elemento, assim como as configurações para os elementos geométricos.

Na figura abaixo, os textos **“Relatório Biblioteca Pessoal”, “Data:”, “Código”, “Título”, “Volume", "Edição”, “Editora”, “Autor”** e **“Adquirido em”** são campos de texto estático:

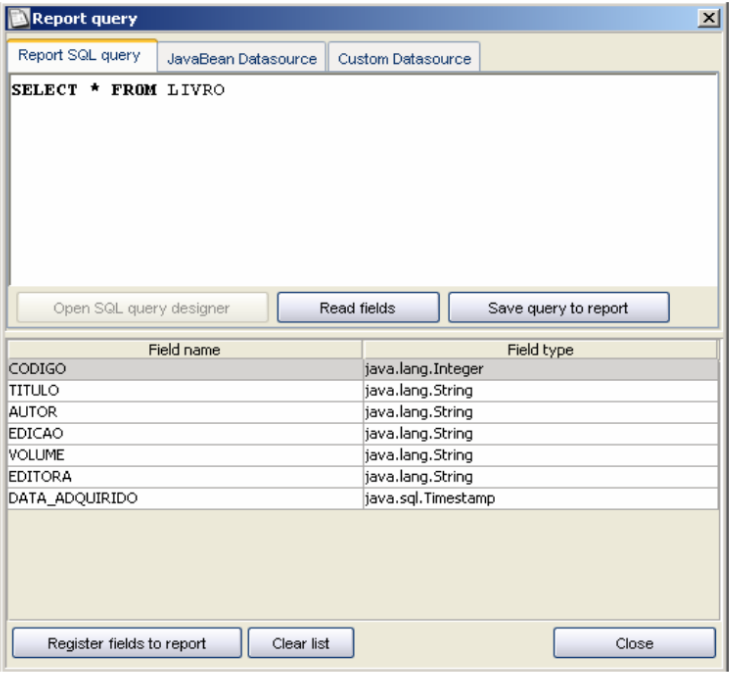
A tabela é constituída de dois Retângulos “arredondados”: , enquanto os outros elementos são todos campos (fields).



**4.** Para definir os campos (fields) para preenchimento , precisamos conhecer os nomes das colunas das tabelas de onde recuperaremos os dados. O iReport permite a inserção do SQL no código XML através do menu **“Datasource >> Report query”** ou clicando no botão:

**5.** No campo da janela que aparece, na aba **Report SQL Query**, digitamos nossa query. Levando em consideração que o resultado dessa consulta deve conter os dados que preencherão o relatório, o SQL deve ser bem estruturado.

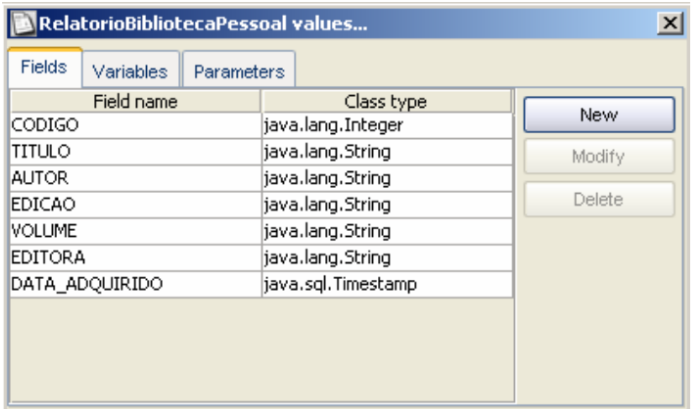
A mesma janela nos permite visualizar os nomes dos campos que fazem parte do resultado da consulta. Por exemplo, se digitarmos a consulta **SELECT \* FROM LIVRO** e em seguida clicarmos no botão **“Read Fields”** visualizaremos todas as colunas da tabela LIVRO de nosso banco, e seus respectivos tipos. Para inserir esta consulta no código XML devemos clicar no botão **“Save query to report”**.



**Obs.:** Em nosso exemplo não iremos inserir o SQL no código XML. Realizaremos a conexão com o banco e a consulta em nossa classe Java, passando o objeto ResultSet gerado para o JasperReports.

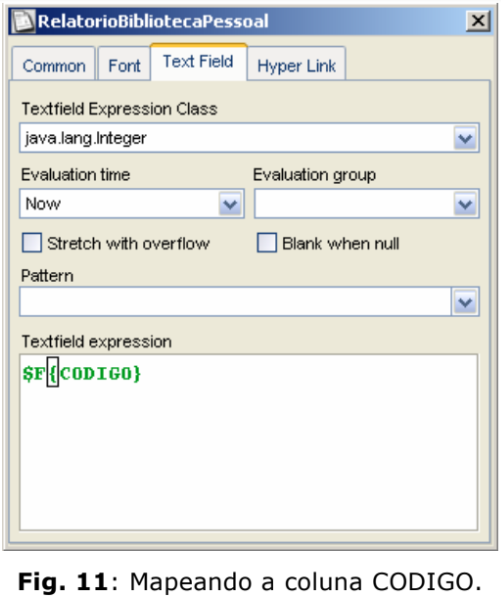
**6.** Agora que conhecemos os nomes das colunas e seus tipos, podemos definir nossos campos (fields) no relatório, **registrando-os** para o mapeamento dos dados. Para tanto, na mesma janela selecione o primeiro campo (coluna) que aparece e, segurando a tecla Shift, clique no último campo, pressionando o botão **“Register fields to report”**. A esma tarefa pode ser realizada indo ao menu **“View >> Report fields”** ou clicando no botão:

A janela que aparece permite o registro de campos (fields), parâmetros e a criação de variáveis no relatório:



**7.** Após registrarmos as colunas a serem mapeadas, devemos inserir elementos fields que farão referência a elas para serem preenchidos.

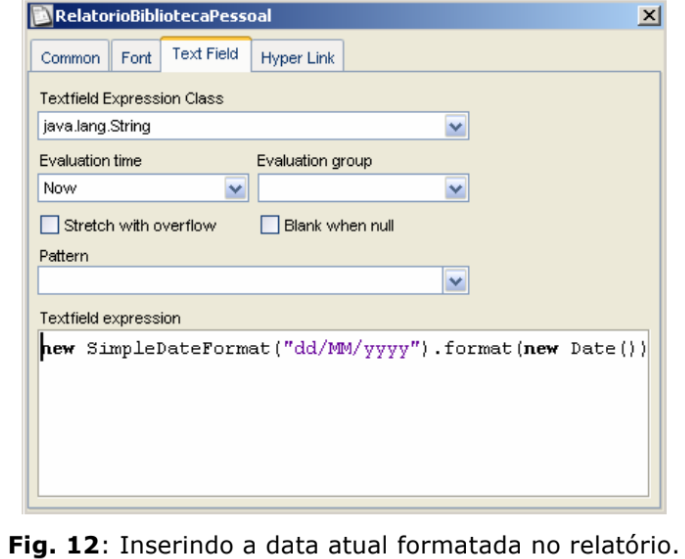
Para isso, depois de inserir um **“field”**, damos um duplo clique nele, surgindo assim uma janela onde, na aba **“Text Field”**, em **“TextField Expression Class”**, configuraremos a classe referente ao tipo da coluna que queremos mapear; e em **“Textfield expression”**, o nome desta coluna entre **$F{** e **}**. Por exemplo, para mapear a coluna **CODIGO** que é do tipo **INTEGER**, selecionamos em **“TextField Expression Class”**, a classe **java.lang.Integer**, e em **“Textfield expression”** digitamos **$F{CODIGO}**. Realize o mesmo processo para as outras colunas (TITULO, AUTOR, VOLUME, EDICAO, EDITORA, DATA\_ADQUIRIDO), atentando para o tipo da coluna.



**8.** Podemos também criar expressões para efetuar cálculos ou formatações sobre os dados provenientes das colunas mapeadas, assim como, optar a que nível executar esta expressão.

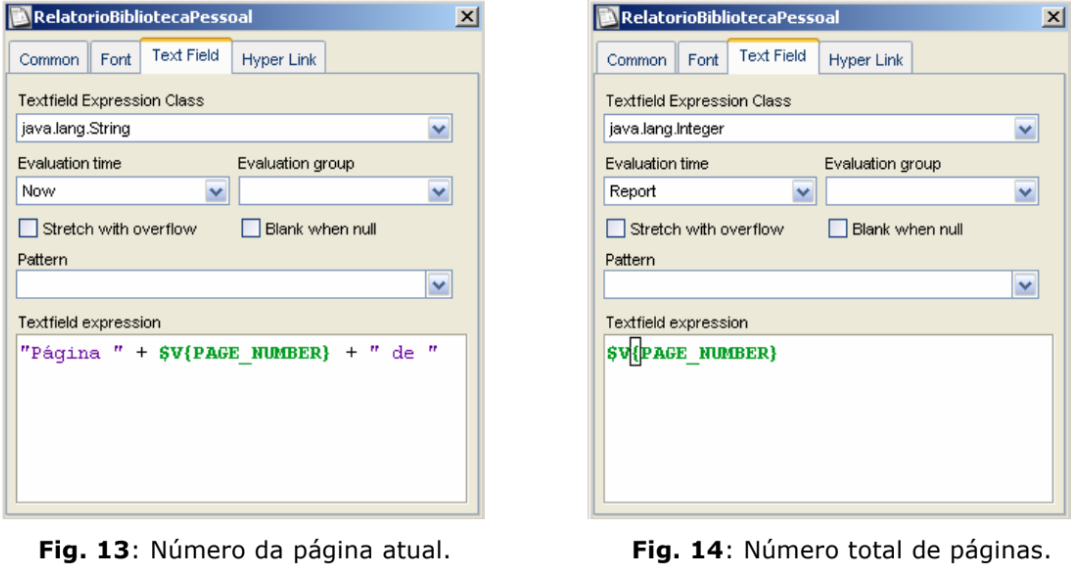
Em nosso exemplo inserimos a dada atual de “impressão” do relatório, formatando-a de acordo com o padrão brasileiro. Para isso, inserimos um “field”, damos um duplo clique sobre ele, e na janela que aparece, na aba “Text Field”, digitamos a seguinte expressão Java no campo “TextField Expression”:

new SimpleDateFormat( "dd/MM/yyyy" ).format( new Date( ) )

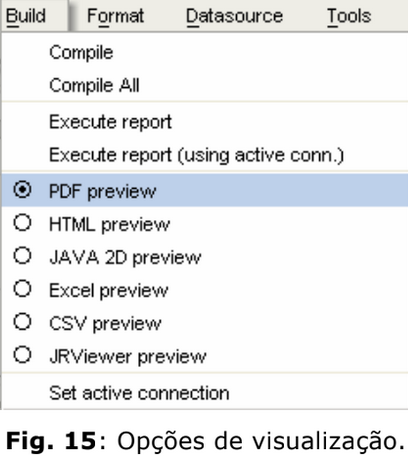


Outro exemplo pode ser visto ao inserirmos o número das páginas de nosso relatório. Para isso, utilizamos a variável interna **$V{PAGE\_NUMBER}.**

Inserimos dois **“fields”** no relatório. No primeiro, no campo “Evaluation time”, escolhemos o nível de execução **“Now”**, que considerará a página atual e para o segundo field, o nível **“Report",** que significa que a variável será inicializada somente uma vez, no começo do relatório, e que executa o cálculo especificado até que o fim do relatório seja alcançado, ou seja, a quantidade de total de páginas.



**9.** Para informarmos ao iReport em que formato desejamos visualizar o relatório, vamos ao menu “Build”, escolhendo uma das opões de visualização ali existentes.



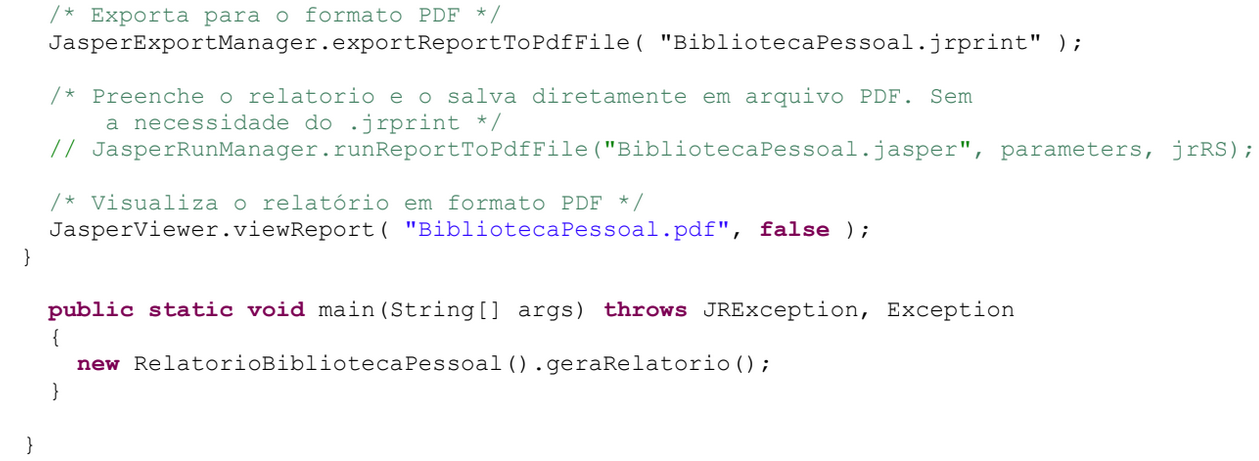
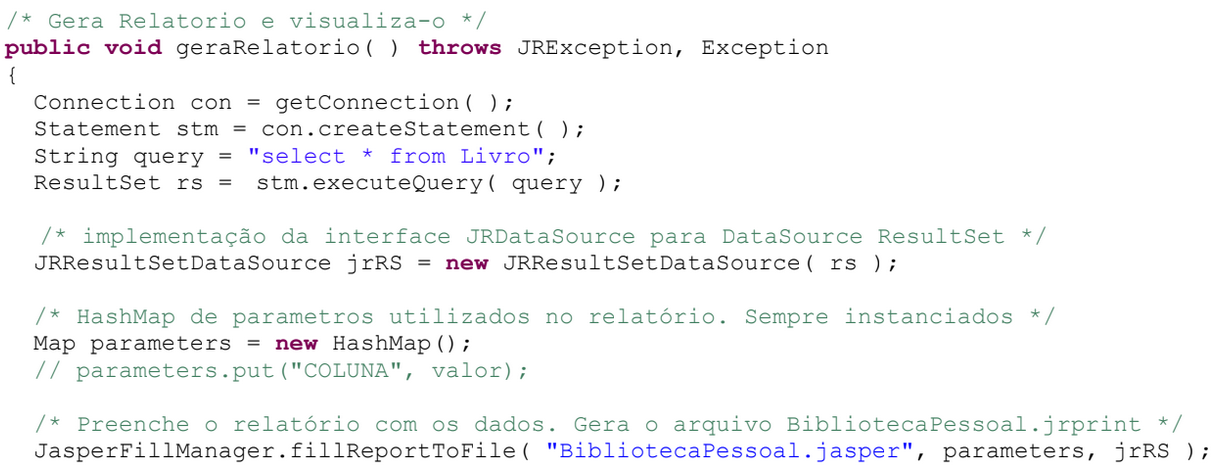
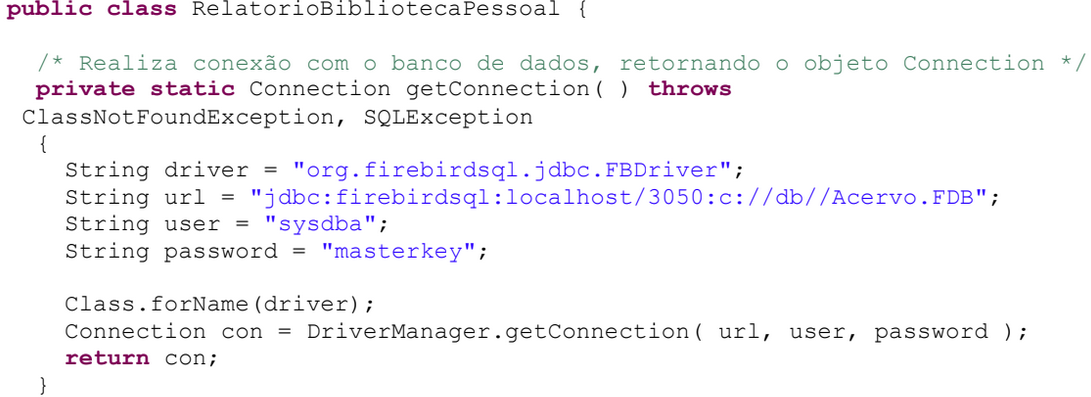
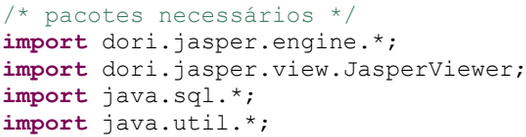


**Obs.:** Podemos agora realizar um teste, clicando no botão

# **Gerando relatório a partir de uma classe**

Vimos até o momento como configurar e utilizar o iReport para montar o layout de nosso relatório, e também visualizá-lo para fins de teste. Nesta seção, veremos na prática como gerar um relatório em formato **pdf** a partir de uma classe simples. A classe RelatorioBibliotecaPessoal será apresentada e descrita posteriormente.

Supõe-se que você já utilizou o iReport para criar o design do relatório e definir os campos a serem preenchidos, que estas informações foram salvas em um arquivo XML, e que este foi compilado neste mesmo ambiente, gerando um arquivo .jasper. Este último arquivo (.jasper) será utilizado pela classe a seguir:



Basicamente, a classe anterior realiza os seguintes passos:

**1.** O método geraRelatorio() chama o método getConnection() para realizar a conexão com o banco de dados e retornar um objeto Connection.

**2.** É realizada a consulta sql, gerando um objeto ResultSet com os dados referentes ao preenchimento do relatório. Este objeto é passado ao construtor da classe JRResultSetDataSource (implementação padrão da interface JRDataSource para **ResultSet**) do JasperReports, criando o objeto datasource (jrRS).

**3.** O objeto datasource (jrRS) é passado como parâmetro para o método estático fillReportToFile() da classe dori.jasper.engine.JasperFillManager para o preenchimento do relatório, assim como o XML compilado (BibliotecaPessoal.jasper) e possíveis parâmetros (parameters). Este procedimento cria o arquivo (BibliotecaPessoal.jrprint) referente ao relatório preenchido.

**Obs.:** O HashMap de parâmetros deve ser instanciado, mesmo não havendo a utilização destes no relatório. São exigidos por todos os métodos de preenchimento do JasperReports.

**4.** O arquivo **.jrprint** (BibliotecaPessoal.jrprint) é passado como parâmetro para o método exportReportToPdfFile() da classe dori.jasper.engine.JasperExportManager para exportar o relatório para o formato **pdf**.

**Obs.:** A classe dori.jasper.engine.JasperRunManager pode substituir a utilização das classes JasperFillManager e JasperExportManager nos passos 3 e 4. Seu método runReportToPdfFile(), por exemplo, recebe como parâmetros o arquivo .jasper referente ao relatório compilado, possíveis parâmetros (parameters) e o objeto JRResultSetDataSource.

**5.** Finalizando, é utilizada a classe

dori.jasper.viewer.JasperViewer, uma GUI para a visualização do relatório em **pdf**. Seu método viewReport() recebe como parâmetro o nome do arquivo de relatório e um boolean referente ao formato do arquivo de relatório, ou seja, **TRUE** para arquivos XML e false para outro.