**IReport**

O **JasperReports** é um framework para a geração de relatórios. É uma ferramenta gratuita, e a mais utilizada com esse propósito atualmente. Entre as funcionalidades do JasperReports podemos destacar:

* É capaz de exportar relatórios para diversos formatos diferentes, tais como PDF, HTML, XML, XLS, etc.
* Aceita diversas formas de entrada de dados, tais como um arquivo XML ou CSV, conexão com o banco de dados, uma sessão do **Hibernate**, uma coleção de objetos em memória, etc.
* Permite o uso de diagramas, gráficos, e até códigos de barras, dentre outros componentes.

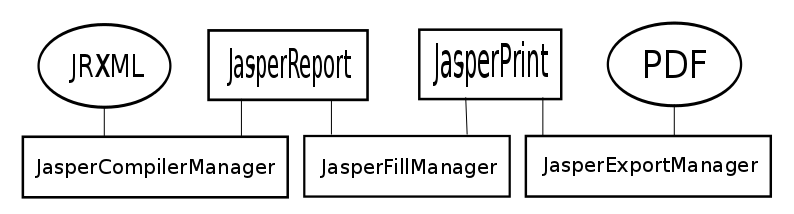
**Funcionamento**

Um aspecto importante do JasperReports é que o **layout** do relatório é definido em um **arquivo XML**, geralmente com a extensão **.jrxml**. Este XML possui todas as informações de formatação do relatório, e além disso, possui os campos que serão preenchidos posteriormente, de acordo com a fonte de dados utilizada **(data source**). Como dito anteriormente, a fonte de dados pode variar, e ser uma **tabela** em uma base de dados, ou ser um **arquivo CSV**, porém a formatação do relatório será a mesma em ambos os casos.

Os passos para gerar um relatório são bem simples:

* O primeiro passo é compilar o relatório em XML. Depois da compilação, o resultado é um objeto do tipo JasperReport.
* O próximo passo é preencher o relatório com os dados, e o resultado dessa etapa fica armazenado em um objeto do tipo JasperPrint. Esse objeto já representa o relatório finalizado, a partir dele podemos enviar para impressão diretamente, ou podemos exportar para um outro formato, tal como PDF por exemplo.

Esse processo é resumido na figura a seguir:



**Tipo de arquivos**

Arquivo gerados pelos IReport:

* **.jrxml** => arquivo do relatório em **xm**l
* **.jasper** => arquivo do relatório compilado

**Tipo de dados**

* **JasperReport**: Representa o arquivo compilado (.**jasper**)
* **JasperPrint**: Representa o relatório preenchido com a fonte de dados.

**Classes Gerenciadoras/Auxiliares**

**JasperCompileManager**: compila arquivos .jrxml e gera arquivos .jasper, devolve um objeto do tipo JasperReport (objeto do relatório em si)

**JasperFillManager**: preenche o objeto do relatório JasperReport a partir de uma fonte de dados (datasource), que poderá ser uma consulta SQL, coleção de objeto, arquivo XML, arquivo CVS, consulta HQL, dentre outros, e devolve um objeto do tipo JasperPrint para representar o relatório preenchido com os dados da fonte de dados.

**JasperExportManager:** exporta o relatório preenchido (JasperPrint) para os mais variados tipos de arquivos: PDF, XML, HTML.

**JasperPrintManager**: usado para enviar um relatório preenchido (JasperPrint) para a impressão.

**JasperViewer**: visualiza um relatório preenchido (JasperPrint) sem a necessidade da exportação.

**Fontes de dados**

Todas as fontes de dados implementam a interface **JRDataSource**, dentre as principais implementações temos:

**JREmptyDataSource**: representa uma fonte de dados vazia, geralmente utilizada para informar ao relatório que a seção "*No data*" deverá ser exibida.

**JRBeanCollectionDataSource**: representa uma fonte de dados criada a partir de uma coleção de objetos java.

**JRResultSetDataSource**: representa uma fonte de dados criada a partir de um **ResultSet**.

**Criando um design (modelo) com o IReport**

Ver referências

**Estrutura de um relatório no IReport**

O IReport divide o relatório em **seções**, cada seção é chamada de 'Banda' (**Band**, no inglês), e cada 'Band' possui um comportamento distinto na construção do relatório. São elas:

**Title**: onde fica o título do relatório;

**Page Header**: adiciona algum tipo de informação sobre o relatório (por exemplo, o ano vigente);

**Column Header**: onde ficam os cabeçalhos da tabela;

**Detail**: aqui ficam as “linhas” da tabela, a informação principal do relatório;

**Column Footer**: usado para o rodapé da tabela;

**Summary**: geralmente usado para uma conclusão, diagrama ou gráfico;

**Page Footer**: informações como data, página do relatório, etc.

**No data**: Esta seção aparecerá apenas quando a fonte de dados do relatório não retornar informação, ou for do tipo JREmptyDataSource

**Compilando e Exportando no Java**

De posse do design do relatório (arquivo **.jrxml**), são necessários os seguintes passos para exportar um relatório: compilar, preencher e exportar. **Ex.:**

Collection<Produto> produtos = new ArrayList<Produto>();

Produto p = null;

for (int i = 0; i < 10; i++) {

p = new Produto();

p.setDescricao("produto " + i);

p.setPrecoUnitario(Double.valueOf("12") + i);

produtos.add(p);

}

JRDataSource datasource = new JRBeanCollectionDataSource(produtos); // definindo a fonte de dados através de uma coleção de produtos

try {

// parâmetros que serão enviados para o relatório

Map<String, Object> params = new HashMap<String, Object>();

// buscando o .jrxml do template do relatório

InputStream is = TestReport.class.getResourceAsStream("/report1.jrxml");

String arquivoPDF = "arquivo.pdf"; // nome do arquivo que será gerado

// compilando o arquivo .jrxml e recebendo uma instancia do relatório

JasperReport jasperReport = JasperCompileManager.compileReport(is);

// preenchendo a instancia do relatório com a fonte de dados e recebendo uma instancia do relatório preenchida

JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, params, datasource);

// visualizando a instância do relatório preenchida

JasperExportManager.exportReportToPdfFile(jasperPrint,arquivoPDF); JasperViewer.viewReport(jasperPrint);

} catch (JRException e) {

e.printStackTrace();

}

**DynamicJasper**

**Configuração**

Para ter acesso às funcionalidades do **DynamicJasper** é necessário adicionar a dependência no **pom.xml**. O detalhe é que esta dependência **não está localizada no repositório central no maven**, somente está disponível do **repositório próprio da dynamicjasper**, por conta disso nosso arquivo pom.xml deverá fazer referência a outro repositório diferente do padrão do maven. Para fazer isso basta apenas adicionar a tag **<repositories>** para listar o(s) novo(s) repositorio(s).

<repositories>

<repository>

<id>fdvsolution.public</id>

<!-- repositorio do dynamicjasper -->

<url>http://archiva.fdvs.com.ar/repository/public1/</url>

</repository>

</repositories>

Após feito essa configuração você poderá adicionar normalmente a dependência para a biblioteca.

<!-- dependencia do DynamicJasper -->

<dependency>

<groupId>ar.com.fdvs</groupId>

<artifactId>DynamicJasper</artifactId>

<version>5.0.0</version>

</dependency>

**Utilização**

Para criação de relatórios dinâmicos apenas necessitamos de uma instância da classe "**FastReportBuilder**", através dela faremos as principais configurações:

// coluna referente ao atributo "descricao" da entidade produto

AbstractColumn colunaDescricao = ColumnBuilder.getNew()

// definindo cabeçalho da coluna

.setTitle("Descrição")

// definindo as propriedades da coluna (nome e tipo), o nome deverá ser igual ao atributo na classe produto

.setColumnProperty("descricao", String.class)

// construindo a coluna

.build();

// coluna referente ao atributo "precoUnitario" da entidade produto

AbstractColumn colunaPreco = ColumnBuilder.getNew()

// definindo cabeçalho da coluna

.setTitle("Preço")

// propriedades da coluna

.setColumnProperty("precoUnitario", Double.class)

// construindo a coluna

.build();

Depois de configurar todas as propriedades desejadas é necessário chamar o método "**build()**" para ele retornar uma instância de uma coluna.

// objeto que define o construtor do relatório dinâmico

// definindo o titulo do relatório

FastReportBuilder frb = new FastReportBuilder();

// adicionando uma coluna previamente criada

frb.setTitle("Relatório de Produtos");

// adicionando colunas previamente criadas

frb.addColumn(colunaDescricao);

frb.addColumn(colunaPreco);

// define cores em forma de zebra no relatório

frb.setPrintBackgroundOnOddRows(true);

// relatório passará a utilizar 100% da largura da página

frb.setUseFullPageWidth(true);

// constrói o relatório dinâmico

DynamicReport dr = frb.build();

// mapa para armazenar parâmetros para o relatório

Map params = new HashMap();

try {

// construção do relatório

JasperReport jr = DynamicJasperHelper.generateJasperReport(dr, new ClassicLayoutManager(), params);

// construção do relatório preenchido

JasperPrint jasperPrint = JasperFillManager.fillReport(jr, params, source);

// obtém o caminho do diretório temporário do SO

String tmp = System.getProperty("java.io.tmpdir");

// exporta o relatório em PDF para o diretório temporário

JasperExportManager.exportReportToPdfFile(jasperPrint, tmp + "/Relatorio.pdf");

} catch (JRException

**Fazendo download do relatório gerado**

**ManagedBean**

Deverá retornar um objeto do tipo **StreamedContent**

// método no ManagedBean

public StreamedContent getDownload() {

try {

// método que gera o relatório no diretório temporário

this.produtoEJB.gerarRelatorio();

// referencia para o caminho do arquivo gerado

String path = System.getProperty("java.io.tmpdir") + "/Relatorio.pdf";

// referência para o arquivo gerado

java.io.File f = new java.io.File(path);

// referência para a stream do arquivo gerado

InputStream stream = new java.io.FileInputStream(f);

// parametros: stream do arquivo, tipo do arquivo (mimetype), nome do arquivo de download

return new DefaultStreamedContent(stream, "application/pdf", "Relatório.pdf");

} catch (FileNotFoundException e) {

e.printStackTrace();

}

return null;

}

**Propriedades do botão na página .xhtml**

icon="ui-icon-arrowthick-1-s" immediate="true">

**Componentes**

**fileDownload** => faz download de arquivos

**media** => exibe determinados tipos de arquivos

**Referências**

**Livro de apoio**: <http://alga.works/livro-javaee/>

**iReport**: <https://community.jaspersoft.com/wiki/ireport-designer-getting-started>

**Documentação Oficial**: <http://community.jaspersoft.com/documentation?version=13758>

**DynamicJasper**: <http://dynamicjasper.com/>