

Maven

O Maven foi criado pela fundação Apache e utiliza um arquivo XML (pom.xml) para descrever o projeto de software sendo construído, suas dependências sobre módulos e componentes externos, a ordem de compilação, diretórios e plug-ins necessários.

Maven

artifactId é o nome do jar sem versão. Se você o criou, pode escolher o nome que quiser com letras minúsculas e sem símbolos estranhos. Se for um jar de terceiros, você deve anotar o nome do jar conforme ele é distribuído. exemplo: maven, matematica, comum

groupId identificará seu projeto exclusivamente em todos os projetos, portanto, precisamos aplicar um esquema de nomenclatura. Ele deve seguir as regras de nome de pacote, o que significa que deve ser pelo menos um nome de domínio que você controla, e você pode criar quantos subgrupos desejar. Veja Mais informações sobre nomes de pacotes. por exemplo:
unc.com.br

br.com.unc.sistemaacademico

Maven

Exemplo de dependências para o Jasper

```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>8.0.21</version>  
</dependency>  
  
<dependency>  
  <groupId>net.sf.jasperreports</groupId>  
  <artifactId>jasperreports</artifactId>  
  <version>6.16.0</version>  
</dependency>
```

Jasper Studio

Jaspersoft Studio é o novo designer de relatórios baseado no Eclipse para desenvolvimento de relatórios JasperReports. É uma reescrita completa do iReport Designer, disponível como plug-in do Eclipse e como um aplicativo independente.

O Jaspersoft Studio permite criar layouts sofisticados contendo gráficos, imagens, sub-relatórios, crosstabs e muito mais. Você pode acessar seus dados por meio de JDBC, Coleções de JavaBeans, JSON File, XML, Hibernate, CSV e fontes personalizadas. Permite publicar os relatórios em saída como: PDF, RTF, XML, XLS, CSV, HTML, XHTML, texto, DOCX ou OpenOffice, entre outros.

Jasper Reports

Os relatórios JasperReports são definidos em um formato de arquivo XML, chamado JRXML, pode ser codificado manualmente, gerado ou projetado usando uma ferramenta.

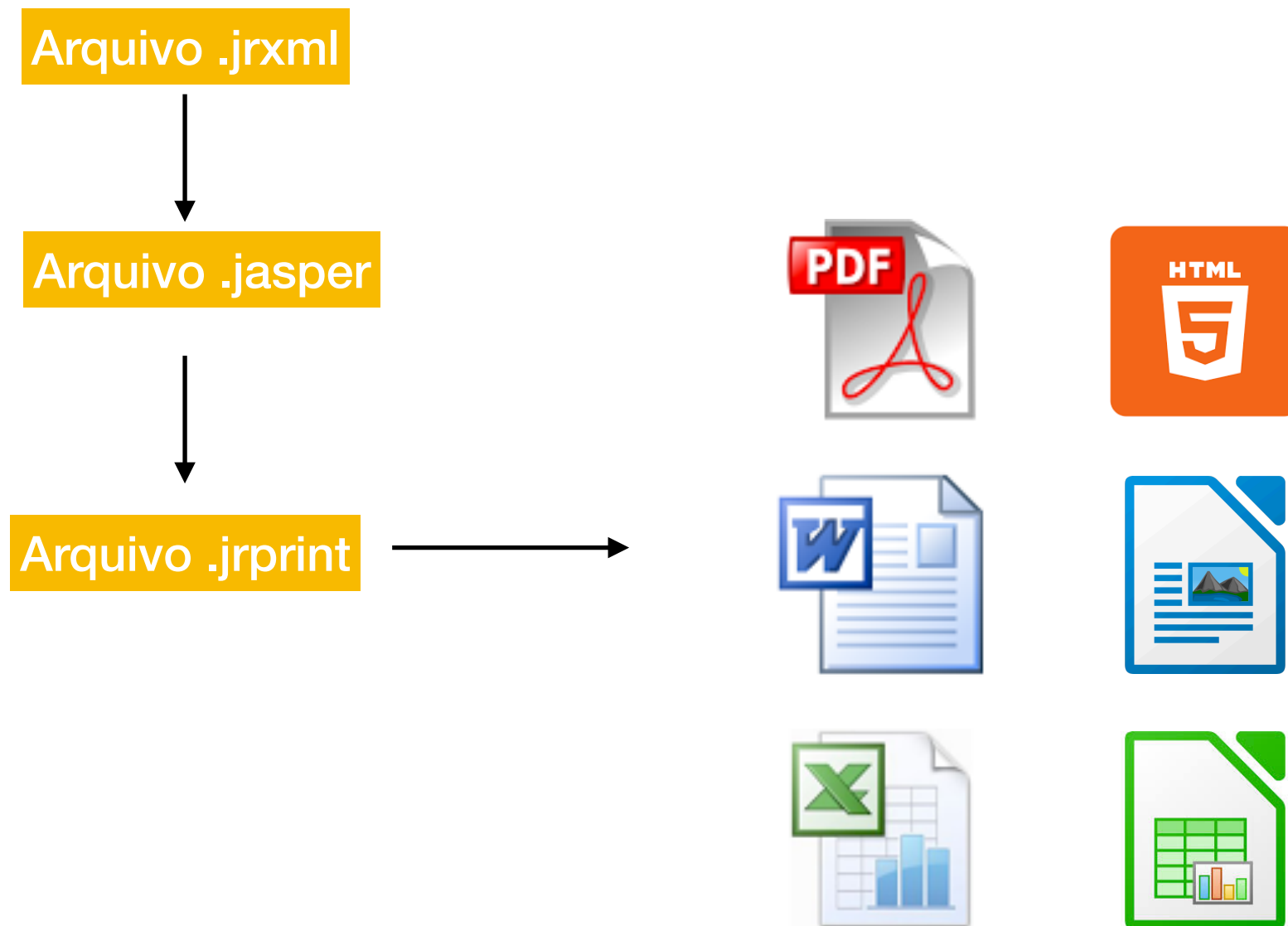
Um arquivo .jasper é uma versão compilada de um arquivo .jrxml. O iReport/Jasper Studio fazem a compilação em tempo real, mas a compilação também pode ser obtida no tempo de execução usando a classe JasperCompileManager.

Jasper Reports

<https://community.jaspersoft.com/project/jasperreports-library/releases>

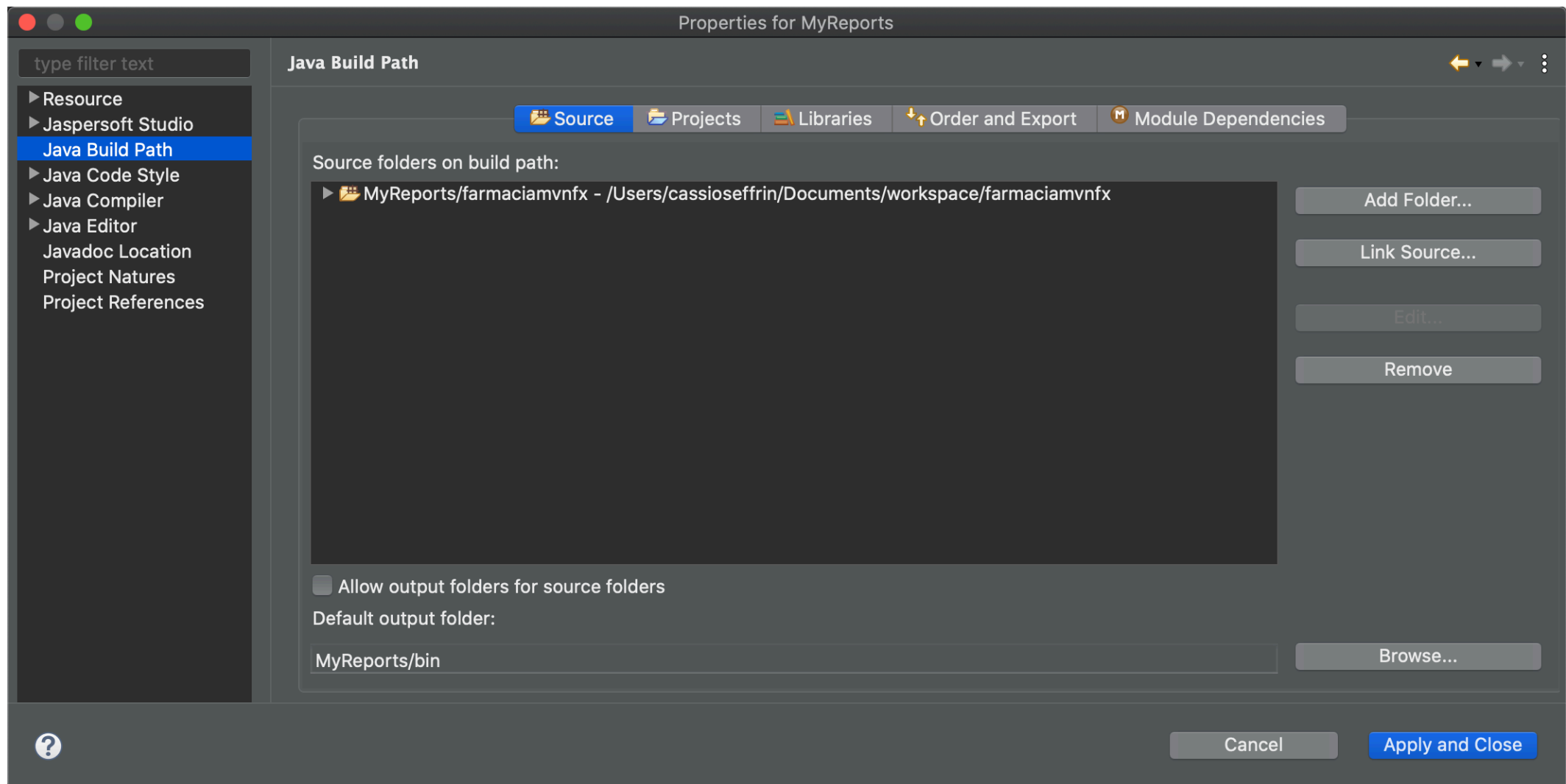
<https://sourceforge.net/projects/jasperreports/files/latest/download>

Fluxo



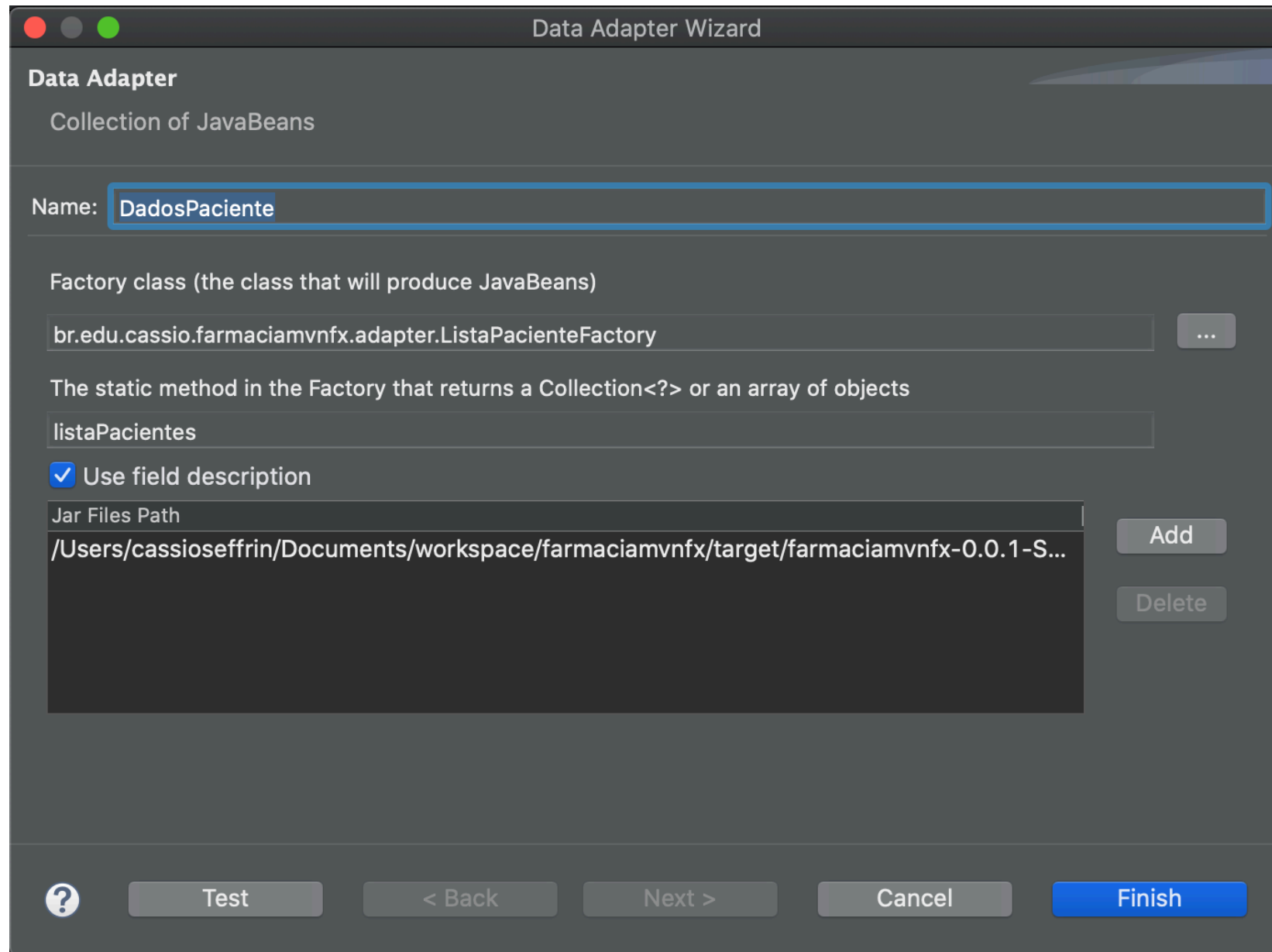
Jasper

Adicionando projeto ao source path,
Project -> Properties -> tab Source -> opção Java Build Path,
clicar o botão Link Source e adicionar a pasta do projeto.
Com isso será possível criar os relatórios diretamente no projeto
do Netbeans ou Eclipse.



Jasper

Tools -> Options -> Classpath



The screenshot shows the 'Data Adapter Wizard' dialog box. The title bar reads 'Data Adapter Wizard'. The main section is titled 'Data Adapter' with the subtitle 'Collection of JavaBeans'. The 'Name' field is filled with 'DadosPaciente'. The 'Factory class (the class that will produce JavaBeans)' field contains 'br.edu.cassio.farmaciamvnfx.adapter.ListaPacienteFactory'. The 'The static method in the Factory that returns a Collection<?> or an array of objects' field contains 'listaPacientes'. The 'Use field description' checkbox is checked. The 'Jar Files Path' field contains the path '/Users/cassioseffrin/Documents/workspace/farmaciamvnfx/target/farmaciamvnfx-0.0.1-S...'. There are 'Add' and 'Delete' buttons to the right of the 'Jar Files Path' field. At the bottom, there are buttons for '?', 'Test', '< Back', 'Next >', 'Cancel', and 'Finish'.

Data Adapter Wizard

Data Adapter
Collection of JavaBeans

Name:

Factory class (the class that will produce JavaBeans)
 ...

The static method in the Factory that returns a Collection<?> or an array of objects

☒ Use field description

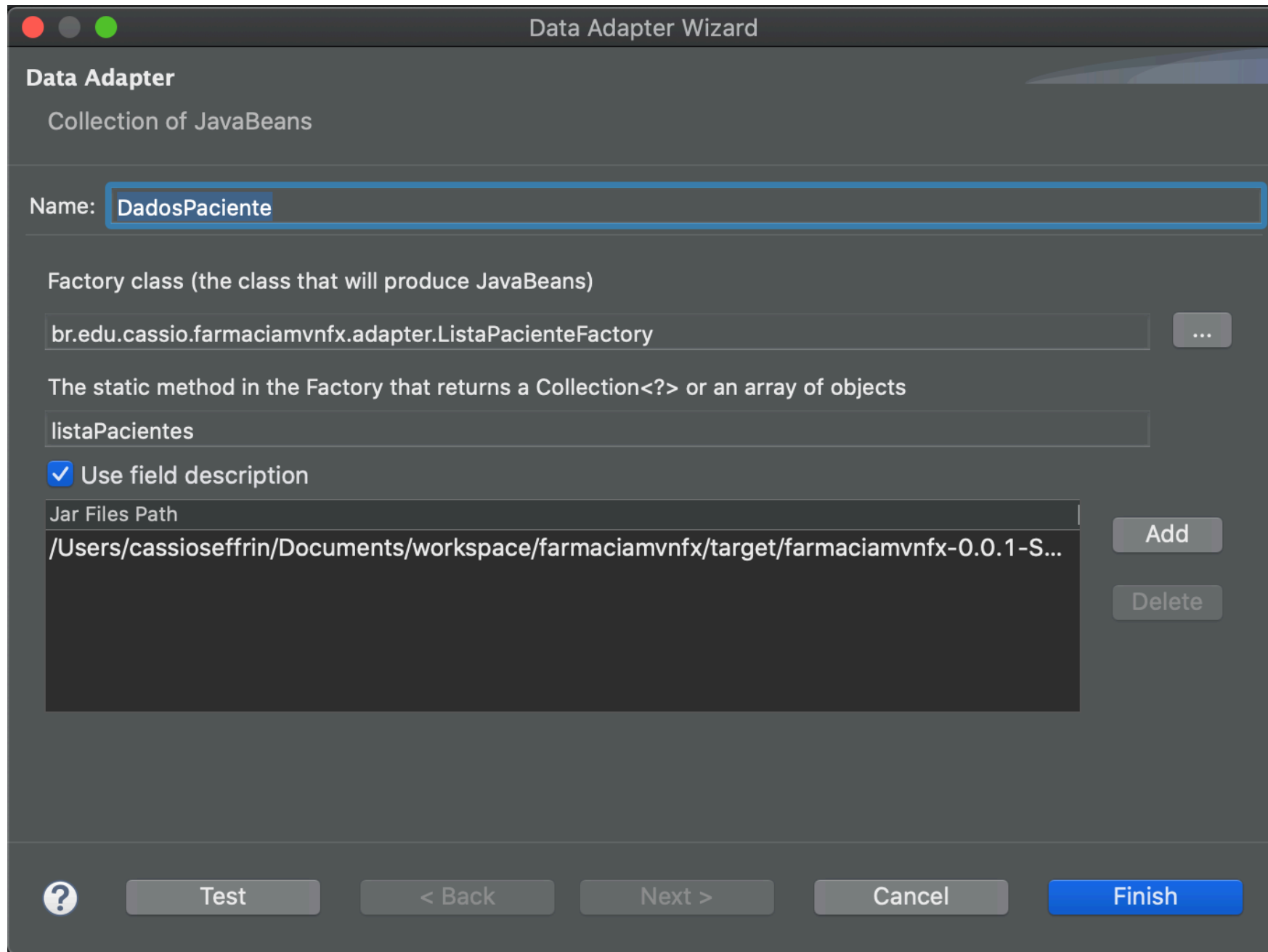
Jar Files Path

Add
Delete

? Test < Back Next > Cancel Finish

Jasper

Criando um data adapter -> Collection of JavaBeans



The screenshot shows the 'Data Adapter Wizard' dialog box in a dark-themed IDE. The title bar reads 'Data Adapter Wizard'. The main section is titled 'Data Adapter' with a subtitle 'Collection of JavaBeans'. The 'Name' field contains 'DadosPaciente'. The 'Factory class (the class that will produce JavaBeans)' field contains 'br.edu.cassio.farmaciamvnfx.adapter.ListaPacienteFactory'. The 'The static method in the Factory that returns a Collection<?> or an array of objects' field contains 'listaPacientes'. The 'Use field description' checkbox is checked. The 'Jar Files Path' field contains '/Users/cassioseffrin/Documents/workspace/farmaciamvnfx/target/farmaciamvnfx-0.0.1-S...'. There are 'Add' and 'Delete' buttons next to the Jar Files Path field. At the bottom, there are buttons for '?', 'Test', '< Back', 'Next >', 'Cancel', and 'Finish'.

Data Adapter Wizard

Data Adapter
Collection of JavaBeans

Name:

Factory class (the class that will produce JavaBeans)
 ...

The static method in the Factory that returns a Collection<?> or an array of objects

☒ Use field description

Jar Files Path
 Add Delete

? Test < Back Next > Cancel Finish

Jasper Studio - JRBeanCollectionDataSource

```
private void handleRelatorio(ActionEvent event) throws JRException {  
    DatabaseMySQL db = new DatabaseMySQL();  
    Connection conexao = db.conectar();  
    MedicoDao pdao = new MedicoDao();  
    pdao.setConnection((Connection) conexao);  
    List<Medico> lst = pdao.listar();  
  
    URL url = getClass().getResource("/relatorios/medico.jasper");  
    JasperReport jasperReport = (JasperReport) JRLoader.loadObject(url);  
  
    JRBeanCollectionDataSource dsMedicos = new JRBeanCollectionDataSource(lst);  
    JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, dsMedicos);  
    JasperViewer jasperViewer = new JasperViewer(jasperPrint, false);  
    jasperViewer.setVisible(true);  
}
```

Jasper Studio - Spring 3,4

```
@RequestMapping(value = "/parcelas", method = RequestMethod.GET)  
public ModelAndView parcelas() {  
    ModelAndView modelAndView = new ModelAndView();  
    modelAndView.setViewName("/alunos");  
    return modelAndView;  
}
```

Jasper Studio - Spring 5

```
@RequestMapping(value = "alunos", method = RequestMethod.GET)
@ResponseBody
public void alunosInad(HttpServletResponse response) throws JRException, IOException {

    InputStream jasperStream = getClass().getResourceAsStream("/alunos.jasper");
    JasperReport jasperReport = (JasperReport) JRLoader.loadObject(jasperStream);
    JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, new
JREmptyDataSource());

    response.setContentType("application/x-pdf");
    response.setHeader("Content-disposition", "inline; filename=alunos.pdf");

    final OutputStream outputStream = response.getOutputStream();
    JasperExportManager.exportReportToPdfStream(jasperPrint, outputStream);
}
```

Jasper Studio - Spring 5

1. Criando relatório com DataSet do banco de dados
2. Collection de JavaBeans.
3. Montar o relatório consumindo uma REST API - <https://community.jaspersoft.com/wiki/how-create-report-uses-remote-json-data-source>

Jasper Studio - Spring 5

Configurando o Relatório Jasper no Jaspersoft Studio

No Jaspersoft Studio e crie um novo relatório em branco.
No painel de contorno esquerdo, adicione o seguinte

Parâmetros

título

nome

valor

Campos

nome

valor

Nota: Certifique-se de nomear esses campos exatamente como os seus campos json.

Na sequencia:

1. Arraste os parâmetros para o título
2. Arraste cada campo para a banda de detalhes.

Em seguida, basta clicar no botão construir tudo da barra de ferramentas e o jasper compilará seu relatório. O relatório compilado será nomeado terminando em .jasper.

Colocar esse arquivo em qualquer lugar de onde seu aplicativo tenha permissão para leitura.

Jasper Studio - Spring 5 relatório com JSON file

@FXML

private void handleRelatorio() **throws** JRException {

```
String rawData = "[{\"nome\":\"Nome\", \"valor\":\"Cassio\"},"  
    + "{\"nome\":\"Sexo\", \"valor\": \"Masculino\"},\" + \"{\"nome\":\"CPF\", \"valor\": \"3423423453\"}"  
    + "]";
```

```
URL url = getClass().getResource("/relatorios/jsonteste.jasper");  
JasperReport report = (JasperReport) JRLoader.loadObject(url);
```

// Converte a string json em uma matriz de bytes.

```
ByteArrayInputStream jsonDataStream = new ByteArrayInputStream(rawData.getBytes());
```

// Cria a fonte de dados json

```
JsonDataSource ds = new JsonDataSource(jsonDataStream);
```

```
Map parameters = new HashMap();
```

```
parameters.put("title", "Teste de relatorio com JSON");
```

```
JasperPrint jasperPrint = JasperFillManager.fillReport(report, parameters, ds);
```

// Abre o Jasper Viewer

```
JasperViewer jasperViewer = new JasperViewer(jasperPrint, false);
```

```
jasperViewer.setVisible(true);
```

```
}
```


Jasper Studio - Spring 5

consumido em MS para um Array de Bytes para o Jasper

```
@RequestMapping(value = "alunos", method = RequestMethod.GET)
@ResponseBody
public void alunosInad(HttpServletResponse response) throws JRException, IOException {
    List<ParcelaDTO> lstParcelas = financeiroProxy.todasParcelasInadimplentes();
    ArrayList<ClienteInadimplenteDTO> lstInadim = new ArrayList<>();
    for (ParcelaDTO p : lstParcelas) {
        AlunoDTO a = alunoProxy.pegarDados(p.getAlunoId());
        long diferencaData = diferencaData(p);
        ClienteInadimplenteDTO cli = new ClienteInadimplenteDTO(a.getNome(), a.getCpf(), p.getDataVencimento(),
            diferencaData, p.getDesconto(), p.getAcrescimo(), p.getNumero());
        lstInadim.add(cli);
    }
    ObjectMapper om = new ObjectMapper();
    byte[] valueAsBytes = om.writeValueAsBytes(lstInadim);
    URL url = getClass().getResource("/parcelasInadimplentes.jasper");
    JasperReport report = (JasperReport) JRLoader.loadObject(url);

    // String json em uma matriz de bytes.
    ByteArrayInputStream jsonDataStream = new ByteArrayInputStream(valueAsBytes);

    // Fonte de dados json
    JsonDataSource ds = new JsonDataSource(jsonDataStream);
    JasperPrint jasperPrint = JasperFillManager.fillReport(report, null, ds);

    response.setContentType("application/x-pdf");
    response.setHeader("Content-disposition", "inline; filename=alunos.pdf");
    final OutputStream outputStream = response.getOutputStream();
    JasperExportManager.exportReportToPdfStream(jasperPrint, outputStream);
}
```

GIT servidor/cliente

```
servidor git  
#mkdir repo  
#cd repo  
#git init --bare
```

```
cliente1 com git clone  
#mkdir cliente1  
#cd cliente1  
#git clone /Users/cassioseffrin/Desktop/repo  
#echo "teste" > teste.txt  
#git add teste.txt  
#git commit -m"primeiro commit"  
#git push origin master
```

```
cliente2 com projeto existente  
entrar no diretório  
git init  
git remote add origin /Users/cassioseffrin/Desktop/repo2  
git add .  
git commit -m"primeiro commit"  
git push origin master
```

```
cliente2 com git clone em outro host  
#mkdir cliente1  
#cd cliente1  
#git clone cassio@192.168.0.102:/Users/cassioseffrin/Desktop/repo  
#echo "teste" > teste.txt  
#git add teste.txt  
#git commit -m"primeiro commit"  
#git push origin master
```

Pull Request (PR) no GitHub

Clicar o botão Fork no canto superior direito. Isso cria uma nova cópia do meu repositório de demonstração em sua conta de usuário do GitHub com um URL como:

`https://github.com/<seunome>/farmaciaMvnFx`

A cópia inclui todo o código, branches e commits do repositório original.

Em seguida, clone o repo abrindo o terminal em seu computador e executando o comando:

`git clone https://github.com/<seunome>/farmaciaMvnFx`

Depois que o repo é clonado, você precisa fazer duas coisas:

Crie uma nova branch emitindo o comando:

`git checkout -b nova_branch`

Crie um novo controle remoto para o repositório upstream com o comando:

`git remote add upstream https://github.com/cassioseffrin/farmaciaMvnFx`

Nesse caso, "repo upstream" se refere ao repositório original a partir do qual você criou seu fork.

Agora você pode fazer alterações no código. O código a seguir cria uma nova branch, faz uma alteração arbitrária e o envia para nova_branch:

Pull Request (PR) no GitHub

```
$ git checkout -b nova_branch
Switched to a new branch 'nova_branch'
$ echo "texto qualquer" > teste.txt
$ git status
On branch nova_branch
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  teste.txt
nothing added to commit but untracked files present (use "git add" to track)
$ git add teste.txt
$ git commit -m "commit na nova_branch"
[nova_branch (root-commit) 4265ec8] Adding a test file to new_branch
1 file changed, 1 insertion(+)
create mode 100644 teste.txt
$ git push -u origin nova_branch
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 918 bytes | 918.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
Remote: Create a pull request for 'nova_branch' on GitHub by visiting:
Remote:  https://github.com/cassioseffrin/farmaciaMvnFx/pull/new/new_branch
Remote:
```

* [new branch] nova_branch -> nova_branch

Depois de fazer o push, voltar ao GitHub e verificar o botão verde Pull Request que irá aparecer.