

**FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA**

**MBA EM DESENVOLVIMENTO DE APLICAÇÕES JAVA (SOA)**

**Frameworks Utilitários (Jasper, Ant, Apache Commons, Maven) E Design Patterns**

LUIZ FERNANDO PEREIRA - RM: 46663

RODRIGO MAGALHÃES VASCO - RM: 46493

DOCUMENTO DE TRABALHO FINAL

Índice

[Resumo 3](#_Toc415247575)

[Idéia do Sistema 3](#_Toc415247576)

[Apresentação do Sistema 3](#_Toc415247577)

[Frameworks Utilizados 3](#_Toc415247578)

[Commons Lang 3](#_Toc415247579)

[Log4J 4](#_Toc415247580)

[Maven 4](#_Toc415247581)

[Design Patterns Utilizados 6](#_Toc415247582)

[Singleton 6](#_Toc415247583)

[Abstract Factory 7](#_Toc415247584)

[Framework de Pesquisa 9](#_Toc415247585)

[Spring Batch 9](#_Toc415247586)

[Exemplo de Utilização 9](#_Toc415247587)

[Instalando a Aplicação 10](#_Toc415247588)

[Instalação com Maven 10](#_Toc415247589)

[Testando a Aplicação 10](#_Toc415247590)

# Resumo

Esse documento tem o objetivo de explicar e demonstrar técnicas e métodos utilizados para a construção da aplicação de entrega final da matéria de Frameworks e Design Patterns, e está dividido conforme o que foi solicitado, como Instalação da Aplicação, Design Patterns, Frameworks utilizados pela aplicação e suas respectivas justificativas.

# Idéia do Sistema

## Apresentação do Sistema

A idéia do sistema é prover funcionalidades, onde uma imobiliária, prédio, administradoras, possam emitir boletos de suas locações, informando poucos dados para que a funciolidade seja realizada, possibilitando a emissão de boletos online, e um relatório completo de todos os boletos gerados.

Refente ao Boleto, como há muitas informações em um boleto, e não tendo a necessidade do usuário final informar dados que são padrões para todo os boletos, esses dados estão armazenados em um arquivo properties para que em tempo de runtime da aplicação, ele possa ser modificado alterando informações em tempo real.

A aplicação, não está utilizando nenhuma conexão com banco de dados, todas as funcionalidades estão sendo utilizadas na session, os dados adicionados no cadastro são armazenados na session, não havendo persistência dos dados.

# Frameworks Utilizados

## Commons Lang

É uma biblioteca Java padrão que fornece métodos para a manipulação de suas principais classes, o Commons Lang oferece uma série de serviços de ajuda para a API java.lang, nomeadamente os métodos de manipulação, métodos numéricos, objeto de reflection, criação e serialização de propriedades, além disso, ele contém melhorias básicas para java.util.Date e uma série de utilitários dedicados a ajudar outros métodos, tais como hashCode(), toString() e equals().

Na aplicação foi utilizado o Apache Commons Lang, para realizar concatenação de strings, para não necessitar utilizar o operador de soma "+" para a concatenação, segue o exemplo:

|  |
| --- |
| **Exemplo de Commons Lang** |
| /\*\*  \* Método responsável em criar o nome do arquivo seguindo o pattern do  \* negócio.  \*  \* Esse método Utiliza Apache Commons(vide comentário no trecho de código)  \*  \* **@return**  \*/  **private** String createNameFile() {  *logger*.info("Criando nome do arquivo de relatório.");  String nameFile = "";  ConfigProperties configProperties = ConfigProperties.*getInstance*();    /\*Na linha abaixo, é utilizado a classe StringUtils, do commons lang,  \* mais especificamente o método join(), que nos facilita a concatenação  \* de Strings sem a utilização do operador matemático "+", com isso deixando  \* o código mais limpo e performático.  \*/    nameFile = StringUtils.*join*(**new** Object[] {  "filename=\"",  configProperties.getProperty("export.file"),  **new** SimpleDateFormat("YYYY-MM-dd-HH-mm-ss").format(**new** Date()),  configProperties.getProperty("export.extension.pdf"),  "\""});    *logger*.info("Nome do arquivo gerado: " + nameFile);    *logger*.info("Nome do arquivo de relatório criado!");  **return** nameFile;  } |

## Log4J

É um framework destinado a Logs também desenvolvimento pela Apache, ele é aplicado para registrar eventos do sistema, tanto eventos de informação, debug ou até mesmo de erro, muitas vezes em aplicações em ambiente de produção, precisamos depurar o código, para saber qual é o real problema que está acontecendo em determinada situação, é nesse momento que entra os logs, um log bem criado e configurado auxilia em uma boa análise, ainda mais quando não temos o acesso ao Aplicattion Server e muito menos a sua depuração em real time, por tanto, a utlização do Log4J é uma boa prática a se fazer no projetos, segue um exemplo de utilização na aplicação:

|  |
| --- |
| **Exemplo do Log4J (log4j-config.xml)** |
| Arquivo de configuração:  <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  <log4j:configuration debug=*"true"*  xmlns:log4j=*'http://jakarta.apache.org/log4j/'*>  <appender name=*"console"* class=*"org.apache.log4j.ConsoleAppender"*>  <layout class=*"org.apache.log4j.PatternLayout"*>  <param name=*"ConversionPattern"* value=*"%d{yyyy-MM-dd HH:mm:ss} %5p %c{1}:%L - %m%n"* />  </layout>  </appender>  <root>  <level value=*"DEBUG"* />  <appender-ref ref=*"console"* />  </root>  </log4j:configuration>  Arquivo Java:  **private** String createNameFile() {  // utilização do Log como INFO  *logger*.info("Criando nome do arquivo de relatório.");  String nameFile = "";  **return** nameFile;  } |

## Maven

O Apache Maven é um framework de gerenciamento de projetos de software e compreensão. Baseado no conceito de um modelo de objeto de projeto (POM), o Maven pode gerenciar um projeto de construção, elaboração de relatórios e documentação como uma ferramnenta central de informações.  
No projeto foi utilizado para gerenciar as depêndecias, instalação e deployment, segue abaixo o exemplo:

|  |
| --- |
| **Exemplo do Maven (pom.xml)** |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.fiap.exerciciofinal</groupId>  <artifactId>emissaoboleto</artifactId>  <packaging>war</packaging>  <version>0.0.1-SNAPSHOT</version>  <name>Emissaoboleto App</name>  <url>http://maven.apache.org</url>  <dependencies>  <!--+ Dependências J2EE para resolver HttpServlet, | HttpServletResquest,  HttpServletResponse etc.. + -->  <dependency>  <groupId>javax</groupId>  <artifactId>javaee-api</artifactId>  <version>6.0</version>  </dependency>  <!-- Dependencias para Log. -->  <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.17</version>  </dependency>  <!-- Dependencas para relatório. -->  <dependency>  <groupId>jasperreports</groupId>  <artifactId>jasperreports</artifactId>  <version>3.0.0</version>  </dependency>  <dependency>  <groupId>taglibs</groupId>  <artifactId>standard</artifactId>  <version>1.1.2</version>  </dependency>    <dependency>  <groupId>commons-lang</groupId>  <artifactId>commons-lang</artifactId>  <version>2.6</version>  </dependency>    </dependencies>  <build>  <finalName>emissaoboleto</finalName>  <!-- Plugin específico para instalação e deployment da aplicação no JBoss 7 -->  <plugins>  <plugin>  <groupId>org.jboss.as.plugins</groupId>  <artifactId>jboss-as-maven-plugin</artifactId>  <version>7.6.Final</version>  </plugin>  </plugins>  </build>  </project> |

# Design Patterns Utilizados

## Singleton

Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto.  
Na aplicação há uma necessidade de realizar a leitura de propriedades para algumas informações padrões para geração dos relatórios e com isso foi utilizado o Pattern Singleton, para que não seja criada várias instâncias do objeto de Properties toda vez que requisitado os properties, segue o exemplo do Singleton na aplicação:

|  |
| --- |
| **Exemplo de Singleton (ConfigProperties.java)** |
| /\*\*  \* Objeto Singleton responsável em obter as propriedades  \* padrões da aplicação  \*  \* **@author** Luiz  \*  \*/  **public** **class** ConfigProperties {  **private** **static** Properties *prop* = **null**;  **private** **static** ConfigProperties *instance* = **null**;    /\*\*  \* Singleton para manter a mesma instância do objeto em memória.  \*  \* **@return**  \*/  **public** **static** ConfigProperties getInstance() {  **if** (*instance* == **null**) {  *instance* = **new** ConfigProperties();  }  **return** *instance*;  }    /\*\*  \* Construtor default privado, para impedir a utilização da construção do  \* objeto sem ser pelo método estático getInstance()  \*/  **private** ConfigProperties() {  InputStream input = **null**;  **try** {  input = **new** FileInputStream("src/main/resources/config.properties");  // load a properties file  *prop* = **new** Properties();  *prop*.load(input);  } **catch** (IOException ex) {  ex.printStackTrace();  } **finally** {  **if** (input != **null**) {  **try** {  input.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  }    /\*\*  \* Método responsável em obter um property do arquivo.  \*  \* **@param** key  \* **@return**  \*/  **public** String getProperty(String key) {  **return** *prop*.getProperty(key);  }  } |

## Abstract Factory

É padrão que permite a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem que a classe concreta seja especificada.  
Na aplicação temos a necessidade de gerar relatórios distintos porém utilizando a mesma funcionalidade, que é o JasperReport, com isso temos uma familia que é de relatórios, ou seja, utilizamos o Patterns também por manutenabilidade, pois, se há a necessidade de criar-se um novo relatório, o que deverá ser feito a nível de código é adicionar somente mais uma classe de implementação, a qual implementa o novo relatório, segue o exemplo:

|  |
| --- |
| **Exemplo de Abstract Factory** |
| /\*\*  \* Classe responsável em, ser o ponto de implementação  \* da criação dos objetos.  \*  \* **@author** Luiz  \*/  **public** **class** ReportFactory **implements** com.fiap.emissaoboleto.export.interfaces.ReportFactory {    **private** **final** **static** Logger *logger* = Logger.*getLogger*(EmitirBoletoServlet.**class**);    /\* (non-Javadoc)  \* @see com.fiap.emissaoboleto.export.interfaces.ReportFactory#createExportBoleto()  \*/  **public** ReportExportInterface createExportBoleto() {  *logger*.debug("criando objetos para export!");  ReportExportInterface reportExportInterface = **new** ReportExportBoleto();  **return** reportExportInterface;  }    /\* (non-Javadoc)  \* @see com.fiap.emissaoboleto.export.interfaces.ReportFactory#createExportRelatorioCompleto()  \*/  **public** ReportExportInterface createExportRelatorioCompleto() {  *logger*.debug("criando objetos para export!");  ReportExportInterface reportExportInterface = **new** ReportExportRelatorioCompleto();  **return** reportExportInterface;  }  }  ---------------------------------------------------------------------------------------  /\*\*  \* Classe responsável em gerar o relatório completo com jasperreport.  \*  \* **@author** Luiz  \*  \*/  **public** **class** ReportExportRelatorioCompleto **implements** ReportExportInterface {  **private** **final** **static** Logger *logger* = Logger.*getLogger*(ReportExportRelatorioCompleto.**class**);  **private** **static** **final** String *fileName* = "src/main/resources/jasper/relatorio\_completo.jasper";  /\*  \* (non-Javadoc)  \*  \* @see  \* com.fiap.emissaoboleto.export.interfaces.ReportExportInterface#export  \* (java.util.Map)  \*/  **public** **byte**[] export(Map<String, Object> parameters) {  *logger*.info("Export do relatório completo.");  **byte**[] bytes = **null**;  **try** {    ArrayList<Boleto> lista = (ArrayList<Boleto>) parameters.get("lista");    JRBeanCollectionDataSource beanColDataSource = **new** JRBeanCollectionDataSource(lista, **false**);    bytes = JasperRunManager.*runReportToPdf*(*fileName*, **null**, beanColDataSource);  } **catch** (JRException e) {  *logger*.error("Erro ao gerar o relatório", e);  } **catch** (Exception e) {  *logger*.error("Erro indefinido ao gerar o relatório", e);  }  *logger*.info("Fim do export do relatório completo.");  **return** bytes;  }  }  ---------------------------------------------------------------------------------------  **Utilização na Servlet:**  **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  List<Boleto> lista = (ArrayList<Boleto>) request.getSession().getAttribute("lista");    **try** {    HashMap<String, Object> map = **new** HashMap<String, Object>();  map.put("lista", lista);    // Realiza a chamada do relatório, fazendo a utilização do Abstract Factory.  ReportExportInterface export = **new** ReportFactory().createExportRelatorioCompleto();  **byte**[] bytes = export.export(map);    ServletOutputStream servletOutputStream = response.getOutputStream();  // adiciona ao header o tipo e o nome do arquivo exportado, isso  // faz com que o browser faça o download do arquivo.  response.setContentType("application/pdf");  response.setHeader("Content-disposition", createNameFile());  response.setContentLength(bytes.length);    servletOutputStream.write(bytes, 0, bytes.length);  servletOutputStream.flush();  servletOutputStream.close();    } **catch** (Exception e) {  *logger*.error("Erro ao gerar relatório!");  }  } |

# Framework de Pesquisa

## Spring Batch

O Spring Batch, é um dos muitos framework da Spring Foundantion, ele é específico para criação de Batchs ou rotinas, fornece funções reutilizáveis que são essenciais para o processamento de grandes volumes de registros, inclusive de exploração/rastreamento, gerenciamento de transações, as estatísticas de processamento do job, reiniciar job, pular e gestão de recursos. Ele também fornece serviços técnicos mais avançados e recursos que permitirão trabalhos em lote de alto desempenho através de técnicas de otimização e particionamento.   
Simples, bem como trabalhos em lote complexos, de alto volume pode alavancar a estrutura de uma maneira altamente escalável para processar volumes significativos de informações.  
Ele também dispôe de funcionlidades de gerenciamento do Batch, ou seja, em uma tabela especifica do seu banco de dados o Spring Batch grava o status, inicio, finalização do Job, para que essa tabela seja uma referencia das execuções.

## Exemplo de Utilização

Abaixo observamos uma configuração e uma execução simples da utilização do Spring Batch:

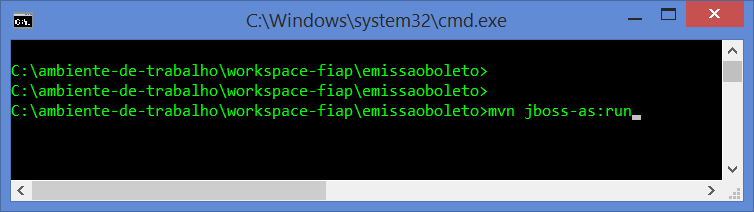
|  |
| --- |
| **Dependência** |
| <dependencies>  <dependency>  <groupId>org.springframework.batch</groupId>  <artifactId>spring-batch-core</artifactId>  <version>3.0.3.RELEASE</version>  </dependency>  </dependencies> |
| **Configuração do Job** |
| **@Configuration**  **@EnableBatchProcessing**  **@EnableAutoConfiguration**  **public** **class** **BatchConfiguration** **{**  **@Autowired**  **private** JobBuilderFactory jobBuilderFactory**;**  **@Autowired**  **private** StepBuilderFactory stepBuilderFactory**;**  **@Bean**  **public** Step **step1()** **{**  **return** stepBuilderFactory**.**get**(**"step1"**).**tasklet**(new** **Tasklet()** **{**  **public** RepeatStatus **execute(**StepContribution contribution**,** ChunkContext chunkContext**)**  **return** **null;**  **}**  **)**  **.**build**();**  **}**  **@Bean**  **public** Job **job(**Step step1**)** **throws** Exception **{**  **return** jobBuilderFactory**.**get**(**"job1"**)**  **.**incrementer**(new** **RunIdIncrementer())**  **.**start**(**step1**)**  **.**build**();**  **}**  **}** |
| **Execução para Teste** |
| **public** **class** **Main** **{**  **public** **static** **void** **main(**String **[]** args**)** **{**  System**.**exit**(**SpringApplication**.**exit**(**SpringApplication**.**run**(**  BatchConfiguration**.**class**,** args**)));**  **}**  **}** |

# Instalando a Aplicação

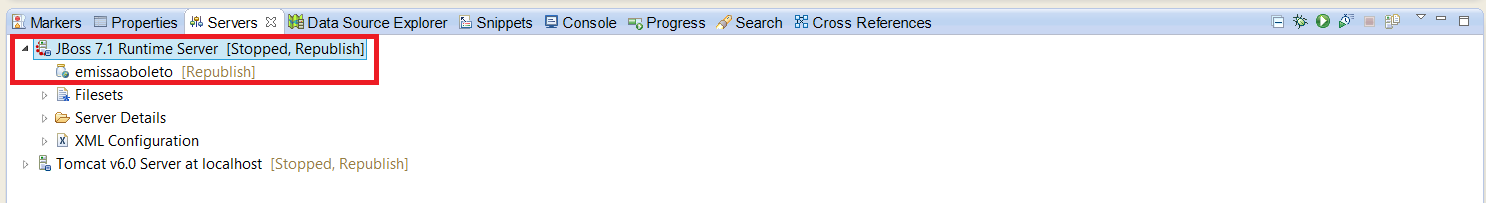
## Instalação com Maven

Para compilar, instalar e executar a aplicação é sim, é preciso somente um comando, na pasta raíz do projeto onde encontra-se o pom.xml, execute a seguinte linha:

**mvn jboss-as:run**

****

Em caso do comando não funcionar por conta do proxy, é possível importar e executar o projeto da forma padrão, como na imagem abaixo:

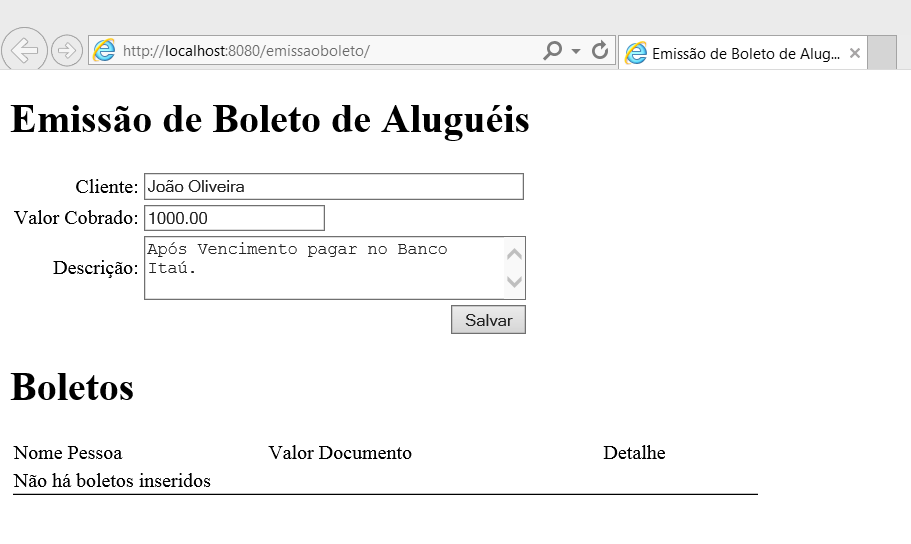
****

# Testando a Aplicação

Executar a aplicação que ficará disponível no link:

**http://localhost:8080/emissaoboleto/**

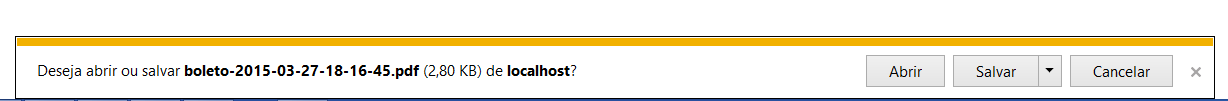
Primeiramente, é necessário adicionar boletos na aplicação, como no exemplo:



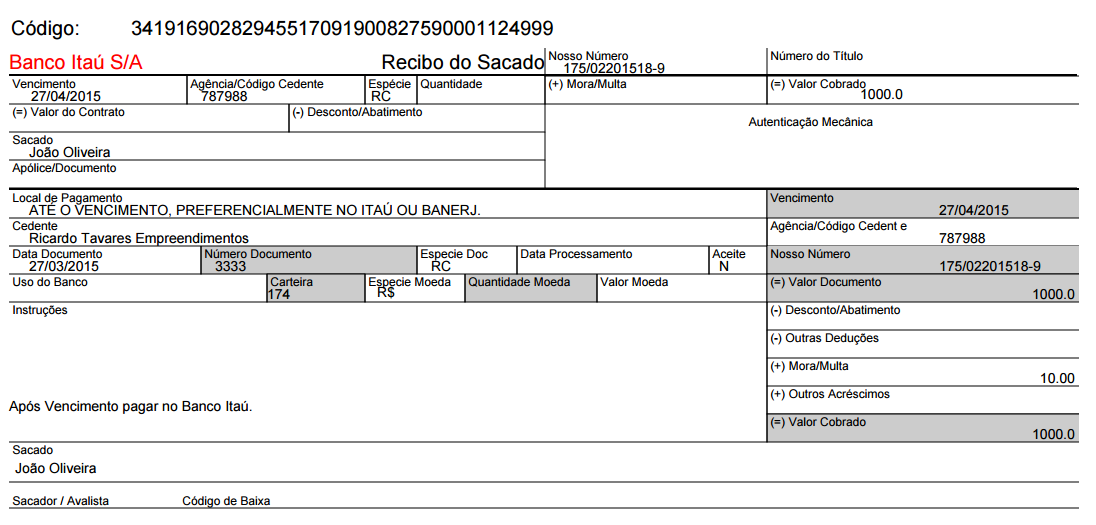
Após clicar em Salvar, será adicionado na lista o boleto, e será habilitada as funcionalidades de "**Emissão do Boleto**" e "**Gerar Relatório**" como na imagem abaixo:



Com isso é possível emitir o boleto clicando, no link "**Emitir Boleto**", clicando no link irá gerar o boleto e realizar o download, o browser irá pedir para realizar o download, salve ou abra-o:



E o seguinte boleto irá aparecer:



E para gerar o **Relatório Geral,** pode seradicionado mais boletos, ou somente clicar no botão "**Gerar Relatório**", o cadastro com mais boletos adicionados ficará assim:



Ao clicar no botão "**Gerar Relatório**", irá gerar e realizar o download do seguinte relatório:

