MC0037 – Programação para Web

Aula 3: Introdução

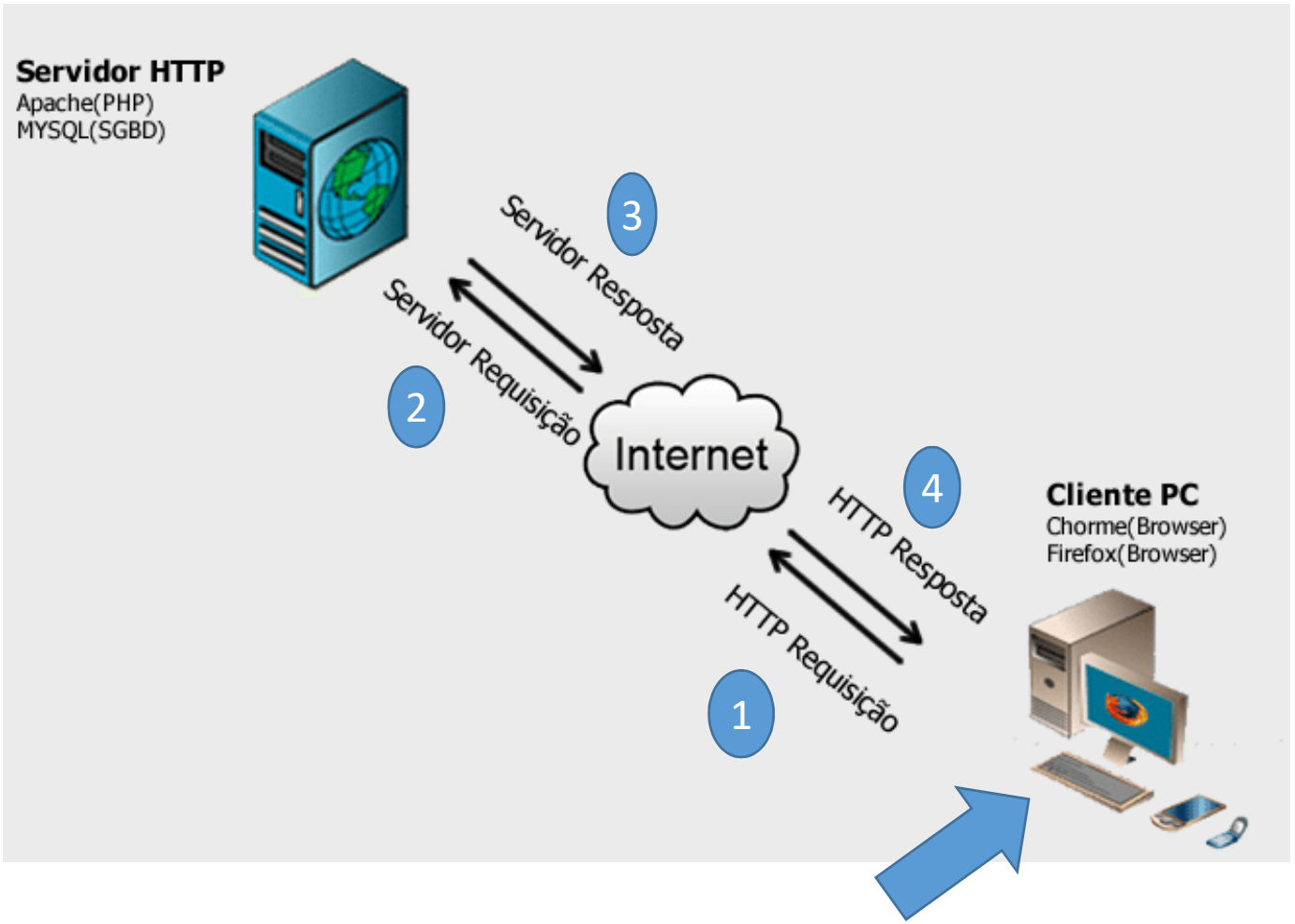
2 Q / 2018

Introdução

- Aulas passadas
  - POO
  - Banco de Dados
  - Hibernate
  - Não nos importamos com a estrutura do projeto, apesar de alguns conceitos terem sido introduzidos...
  - Diversas tarefas manuais...

Introdução

➢Arquitetura Cliente - Servidor



Introdução

- Hoje, foco na estrutura do projeto

Gerenciamento de Pacotes	Estrutura do projeto	Configuração do projeto

- Não iremos aprofundar nas interfaces hoje...próxima aula

MC0037 – Programação para Web

Aula 3: Gradle

2 Q / 2018

Introdução

- Hoje, foco na estrutura do projeto

- Não iremos aprofundar nas interfaces hoje...próxima aula

Motivação

- Dificuldades:
  - montar estruturas de projeto;
  - encontrar os pacotes/bibliotecas
  - identificar dependências entre pacotes
  - empacotar e distribuir as aplicações

- Build Tools: responsáveis por automatizar todas essas atividades e tarefas.

  - Exemplos:
    - Maven
    - Apache Ant
    - Etc

Gradle

➢ Gradle é um sistema de automação de compilação open source;

➢Baseado no Groovy;

➢Automatiza diversas tarefas importantes para a condução de projetos

Gradle

- Permite uma gestão mais eficiente de pacotes de interesse, bem como configuração dos mesmos no projeto;

- Orientado a tarefas;

- Definido por um “build” documento.

Gradle

- Definido por um “build” documento => build.gradle

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:2.0.3.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'org.springframework.boot'
apply plugin: 'io.spring.dependency-management‘
apply plugin: 'war'

war {
    baseName = 'gs-serving-web-content'
    version =  '0.1.0'
}
```

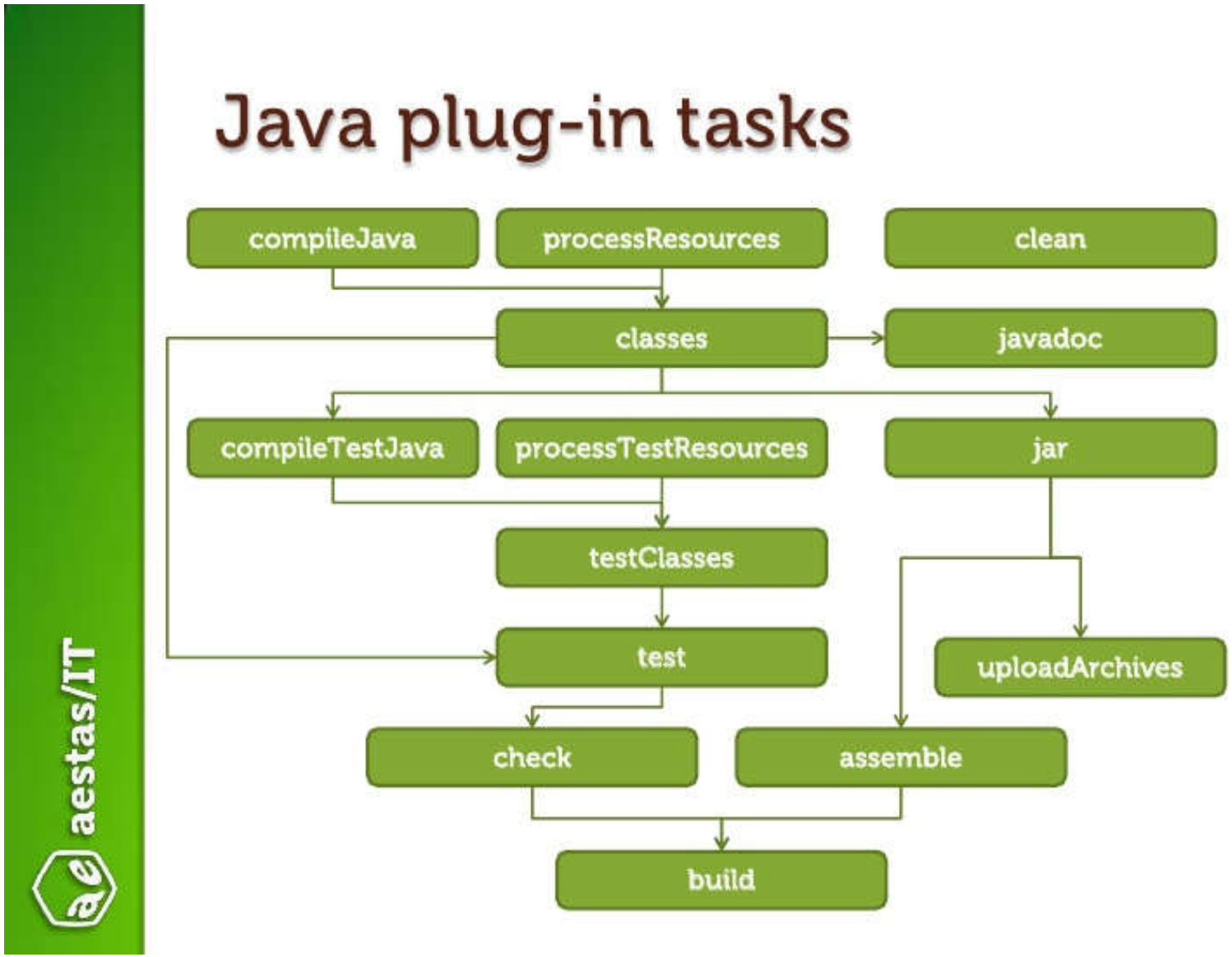
```
repositories {
    mavenCentral()
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    compile("org.springframework.boot:spring-boot-starter-thymeleaf")
    compile("org.springframework.boot:spring-boot-devtools")
    testCompile("junit:junit")
}
```

Gradle

- Operações



Gradle em projetos

- Arquivos
  - Configuração de projeto e dependencias de interesse no arquivo build.gradle

- Comandos
  - **Gradle init --type java-application:** cria um projeto javaapplication
  - **Gradle build:** “constroi” o projeto gradle
  - **Gradle tasks:** apresenta a lista de tarefas disponíveis **para o** gradle
  - **Gradle eclipse:** configura o projeto para o ambiente do eclipse
  - **Gradle run:** executa o arquivo gradle

Gradle em projetos

- Definido por um “build” documento => build.gradle

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:2.0.3.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'org.springframework.boot'
apply plugin: 'io.spring.dependency-management‘
apply plugin: 'war'

war {
    baseName = 'gs-serving-web-content'
    version =  '0.1.0'
}
```

```
repositories {
    mavenCentral()
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    compile("org.springframework.boot:spring-boot-starter-thymeleaf")
    compile("org.springframework.boot:spring-boot-devtools")
    testCompile("junit:junit")
}
```

Gradle em projetos

- Seleciona a pasta do projeto
- Executa o comando gradle init
- Em seguida, define as dependencias de interesse no arquivo build.gradle
  - Hibernate
  - Postgresql
- Incluir arquivo persistence.xml no src/main/resources
- Executa no terminal o comando gradle build e o gradle eclipse
- Importa o projeto no eclipse
- Pronto!

Gradle em projetos

```
plugins {
    // Apply the java plugin to add support for Java
    id 'java'

    // Apply the application plugin to add support for building an application
    id 'application'

                        id 'eclipse‘
}

// Define the main class for the application
mainClassName = 'App'

dependencies {
    // This dependency is found on compile classpath of this component and consumers.
    compile 'com.google.guava:guava:23.0'

    // Use JUnit test framework
    testCompile 'junit:junit:4.12'

                        // https://mvnrepository.com/artifact/org.hibernate/hibernate-core
                        compile group: 'org.hibernate', name: 'hibernate-core', version: '5.0.0.Final'

                        // https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager
                        compile group: 'org.hibernate', name: 'hibernate-entitymanager', version: '5.0.0.Final'

                        // https://mvnrepository.com/artifact/org.postgresql/postgresql
                        compile group: 'org.postgresql', name: 'postgresql', version: '42.2.2'
}

// In this section you declare where to find the dependencies of your project
repositories {
    // Use jcenter for resolving your dependencies.
    // You can declare any Maven/Ivy/file repository here.
    jcenter()
}
```

Gradle em projetos

- Criar novas (um mundo!) tarefas
  - Subir o servidor de aplicação após compilação
  - Automatizar testes unitários
  - Por ex:

```
Id ‘base’

task zip(type: Zip, group: "Archive", description: "Archives sources in a zip file") {
    from "src"
}
```

- Para mais, verificar: https://plugins.gradle.org/search

MC0037 – Programação para Web

Aula 3: MVC: Model-View-Controller

2 Q / 2018

Introdução

- Hoje, foco na estrutura do projeto

- Não iremos aprofundar nas interfaces hoje...próxima aula

Introdução

- Hoje, foco na estrutura do projeto

- Não iremos aprofundar nas interfaces hoje...próxima aula


CODE CORRUPTED
I DON'T KNOW WHAT I DID TO BREAK THIS CODE
OH NOW IT'S WORKING
CLICK CLICK
OH, NO IT'S BROKE AGAIN
CLICK CLICK - CLICK -
NOW IT WORKS - WAIT, NO
HAH!
NOVEMBER 13, 2015

Introdução

Introdução

- Código mal escrito, não estruturado, difícil manutenção, complexo, conhecimento tácito. Resultado: preescrever!

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>User Validation</title>
</head>
<body>
    <%
    response.setContentType("text/html");
    String un = request.getParameter("lgnuser");
    String pw = request.getParameter("lgnpass");
    try
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
        Statement st = con.createStatement();
        String sql = "select *from users";
        ResultSet rs = st.executeQuery(sql);
        boolean flag = false;
        while(rs.next())
        {
            if(un.equals(rs.getString(1)) && pw.equals(rs.getString(2)))
```

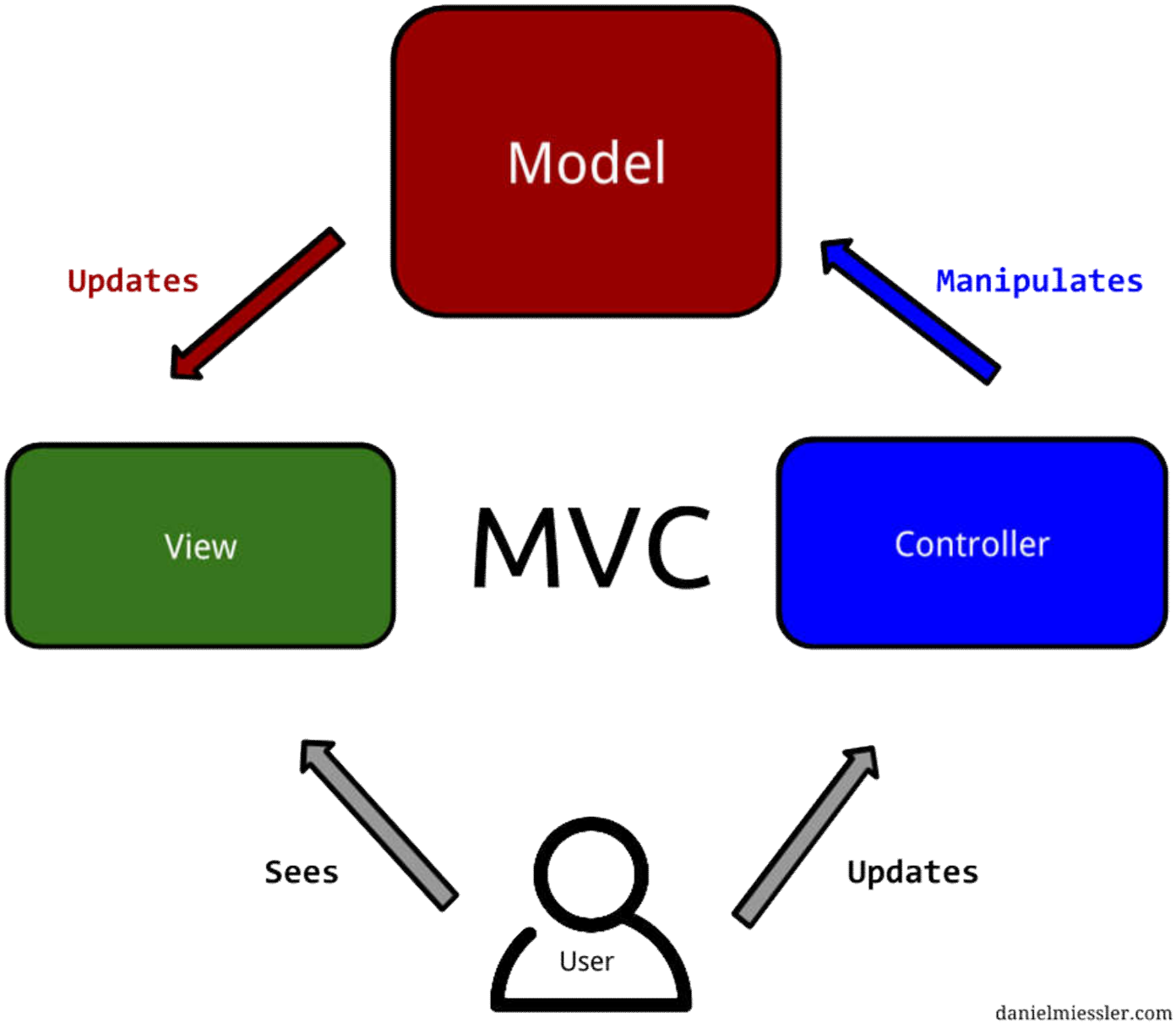
Motivação

- Separar as responsabilidades

- Um sistema de fácil manutenção é aquele em que as responsabilidades estão **separadas** e **implementadas em locais bem-definidos**
- Exemplos de responsabilidades:
  - Salvar um dado (ex.: aluno) no banco de dados
  - Decisão se um dado (ex.: aluno) pode ser removido ou não do banco de dados
  - Montagem da página de saída HTML para exibição dos dados

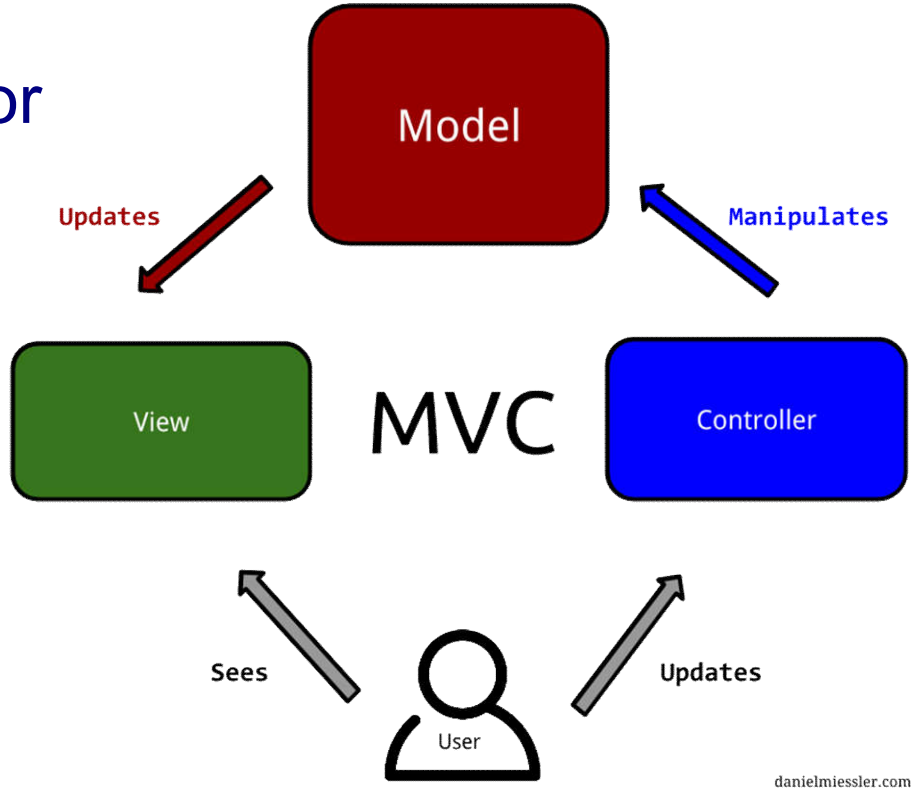
- Idealmente, essas responsabilidades devem estar separadas em **camadas**. Podemos distinguir 3 camadas:
  - Camada de acesso aos dados (ou camada de persistência) – Última aula.
  - Camada de apresentação
  - Camada das regras de negócio

Motivação


Model
Updates
Manipulates
View
MVC
Controller
Sees
User
Updates

MVC

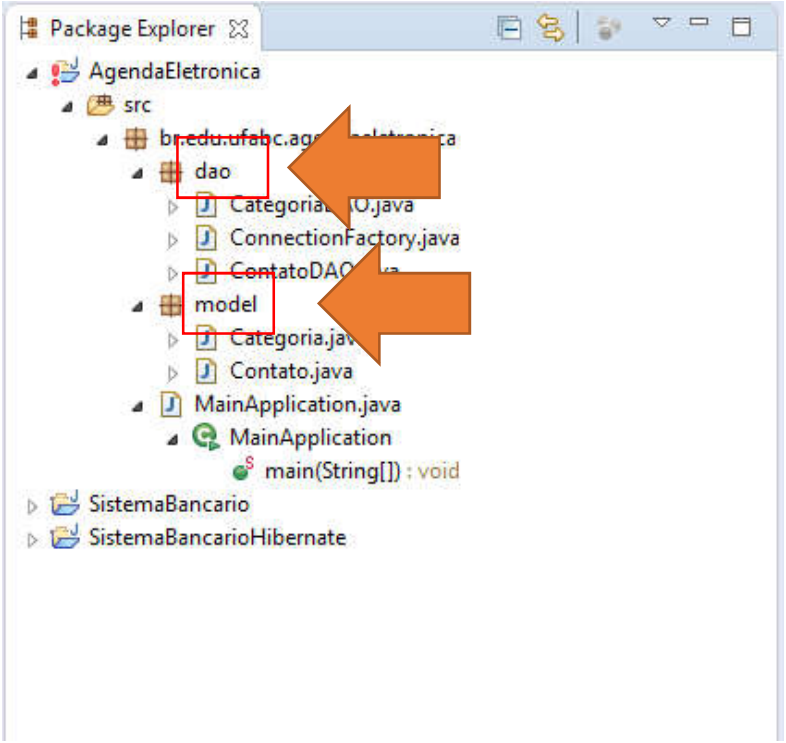
- O modelo de programação MVC foi postulado em 1979.
- O objetivo de organizar melhor uma aplicação, de modo que as funcionalidades dos componentes ficassem desmembradas.
- Os itens são agrupados conforme as semelhanças entre as ações desempenhadas.



MVC

- O modelo de programação MVC foi postulado em 1979.
- O objetivo de organizar melhor uma aplicação, de modo que as funcionalidades dos componentes ficassem desmembradas.
- Os itens são agrupados conforme as semelhanças entre as ações desempenhadas.

Durante exercícios passados....



MVC

- **Objetivos**
  - Desenvolvimento simultaneo
  - Reuso de código
- **Vantagens**
  - Desacoplamento
  - Fácil manutenibilidade
  - Multiplas visões para o mesmo modelo
- **Desvantagens**
  - Navegabilidade de código
  - Curva de aprendizado

MVC: Responsabilidade

- MVC (Model-View-Controller): é um padrão de arquitetura de software que visa a separar a lógica de negócio da lógica de apresentação

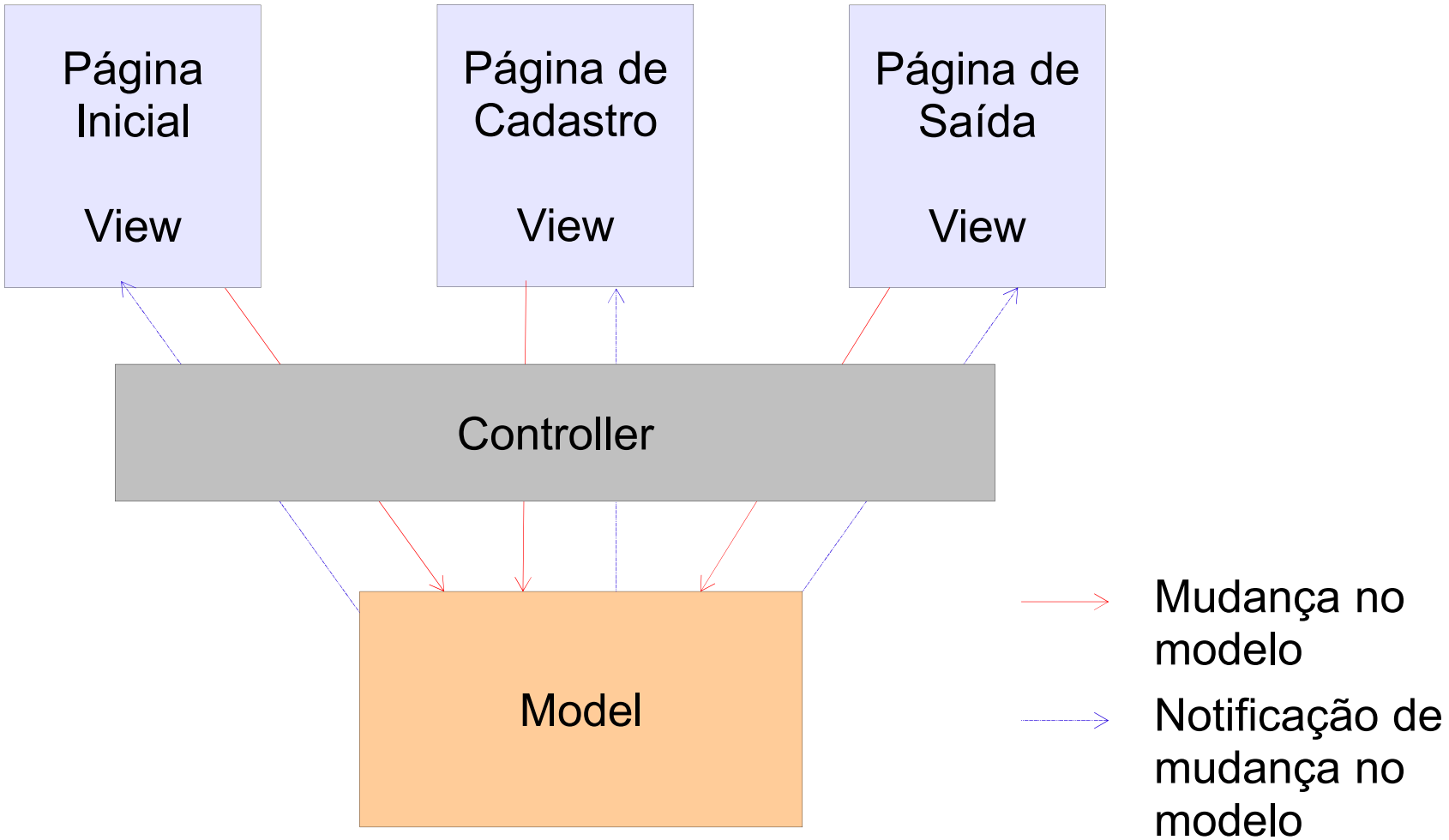
- Model (Modelo): acesso aos dados, contém as regras de negócio (responsável por manter o estado da aplicação)

- View (Visão): responsável pela apresentação (interfaces), o que é apresentado para o usuário

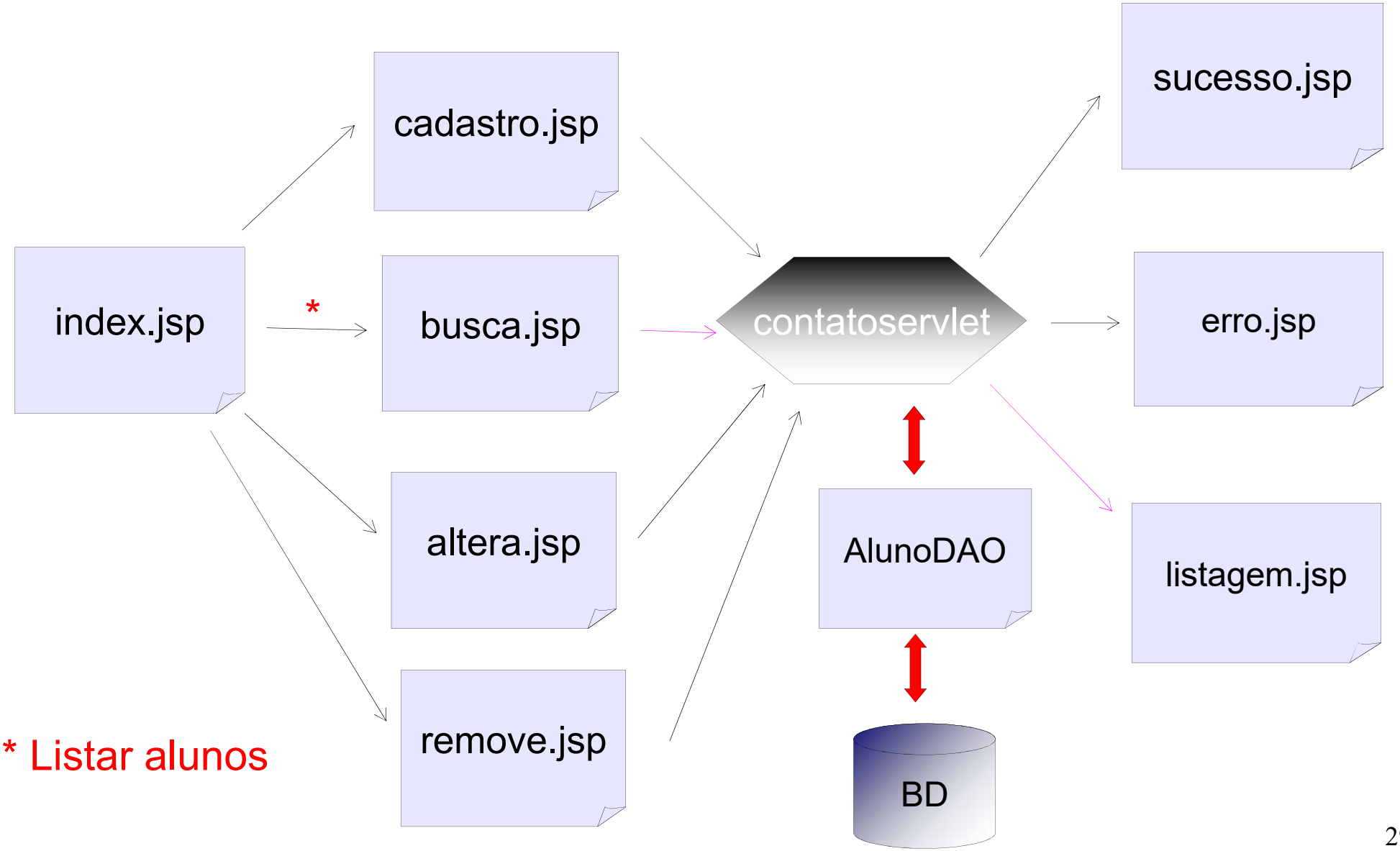
- Controller (Controlador): faz a intermediação entre Modelo e Visão, gerencia as interações com o usuário (através da view) e dispara atualizações para o modelo

MVC

- Interação entre View, Controller e Model

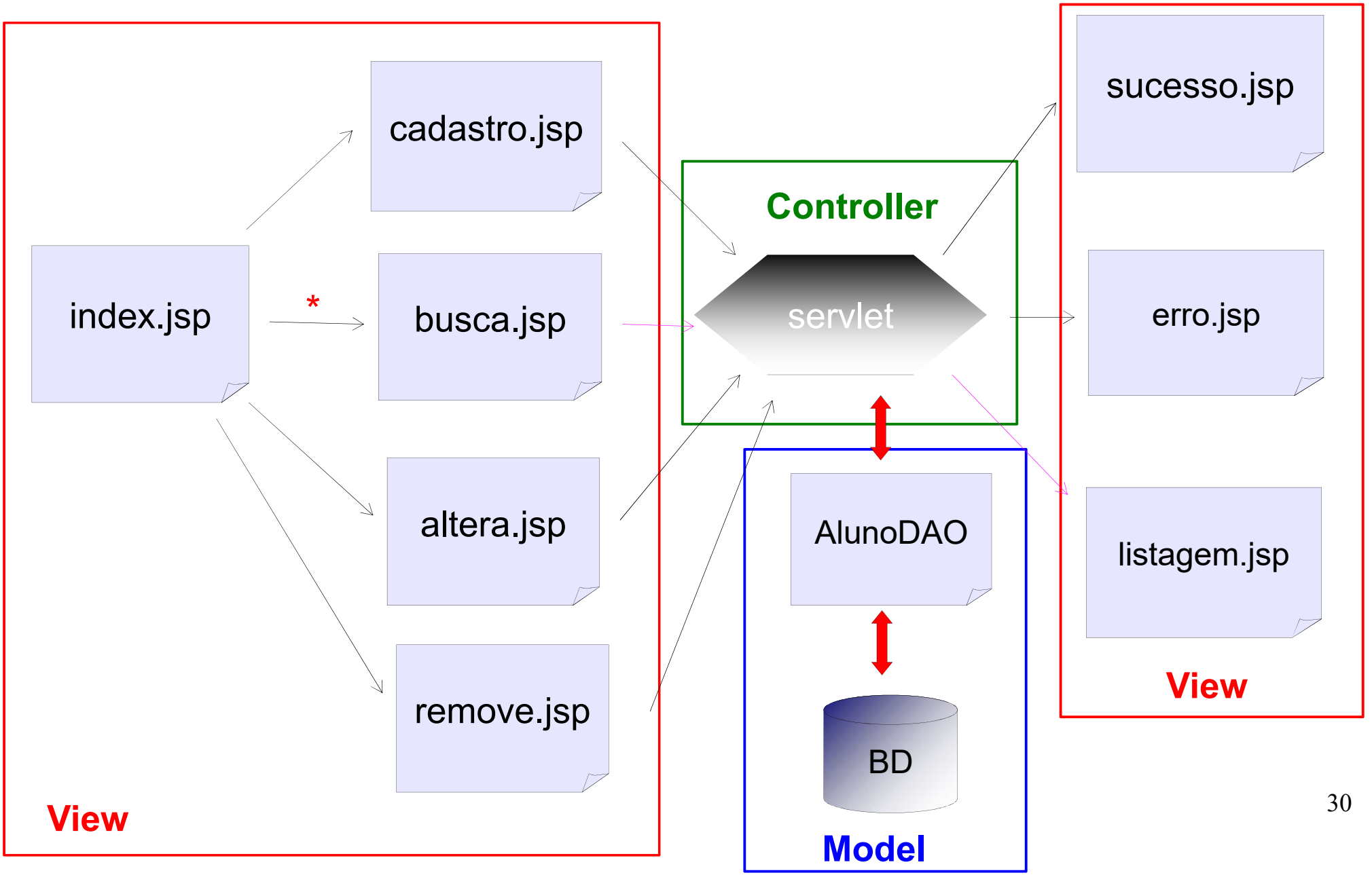


Exemplo: fluxo de navegação das páginas



* Listar alunos

MVC: separação de responsabilidades



MVC: separação de responsabilidades

O importante é você entender
a **motivação** para utilizar
esse modelo.

Model: DAO

- Na prática, são classes que tratam exclusivamente das operações do banco de dados
- DAO (Data Access Object) é um padrão de projeto do Java EE patterns
- A idéia básica do DAO é encapsular as operações básicas de acesso ao BD, conhecidas como CRUD (Create, Read, Update, Delete)
- Exemplos:
  - Salvar
  - Remover
  - Alterar
  - etc

Model: DAO

```java
package br.com.sistemabancario.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

import br.com.sistemabancario.entity.ContaCorrente;

public class ContaCorrenteDao {

    EntityManagerFactory factory = Persistence.createEntityManagerFactory("sistemabancario");
    EntityManager em = factory.createEntityManager();

    public void remove(ContaCorrente contaCorrente) {
        em.remove(contaCorrente);
    }

}
```

Model: Regras de Negócio

- Camada responsável por implementar as tomadas de decisão.
- Determina o que será gravado e o que será exibido.
- Existem Services sem regras específicas.
- Exemplos:
  - Validações
  - Condições
  - Formatações
  - etc

Model: DAO

```java
package br.com.sistemabancario.dao;

import br.com.sistemabancario.entity.ContaCorrente;

public class ContaCorrenteService {

    private ContaCorrenteDao ccDao;

    public ContaCorrenteService() {
        this.ccDao = new ContaCorrenteDao();
    }

    public void remove(ContaCorrente cc) {

        if (!cc.isAtiva()) {
            System.out.println("Correnta corrente já esta desativada!");
        } else {
            ccDao.remove(cc);
        }

    }

}

```

View

- Representada pelas telas do sistema, ou é uma camada de visualização e interação do sistema.

- Nunca acessa diretamente a camada de acesso ao banco de dados.

- Exemplos:
  - Telas em JSF
  - Telas em Javascript
  - Interfaces de comunicação com o sistema
  - etc

View

```
1  package br.com.sistemabancario.application;
2
3  import br.com.sistemabancario.dao.ContaCorrenteService;
4  import br.com.sistemabancario.entity.ContaCorrente;
5
6  public class Application {
7
8      public static void main(String[] args) {
9
10         ContaCorrenteService ccService = new ContaCorrenteService();
11         ContaCorrente cc = new ContaCorrente();
12         cc.setNumero("1234-5");
13
14         ccService.remove(cc);
15
16     }
17
18 }
19
```

View

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <div class="settings-view" id="settings-view">
        <form name="dsForm" class="lay-form" novalidate ng-submit="register()">
            <div id="legend">
                <legend class="">New Data Source</legend>
            </div>

            <md-input-container class="md-block" flex-gt-sm>
            <label>Title</label> <input md-maxlength="30" required name="dsTitle"
                ng-model="dataSource.title">

            <div ng-messages="dsForm.dsTitle.$error"
                ng-if="dsForm.dsTitle.$dirty">
                <div ng-message="required">Title is required.</div>
                <div ng-message="md-maxlength">The title has to be less than
                    30 characters long.</div>
            </div>
            </md-input-container>

            <md-input-container class="md-block" flex-gt-sm>
            <label>Description</label> <input ng-model="dataSource.description">
            </md-input-container>

            <br />

            <md-button class="md-raised" type="submit"
                ng-disabled="!dsForm.$valid">Register</md-button>
        </form>
    </div>
    </div>
</body>
</html>
```

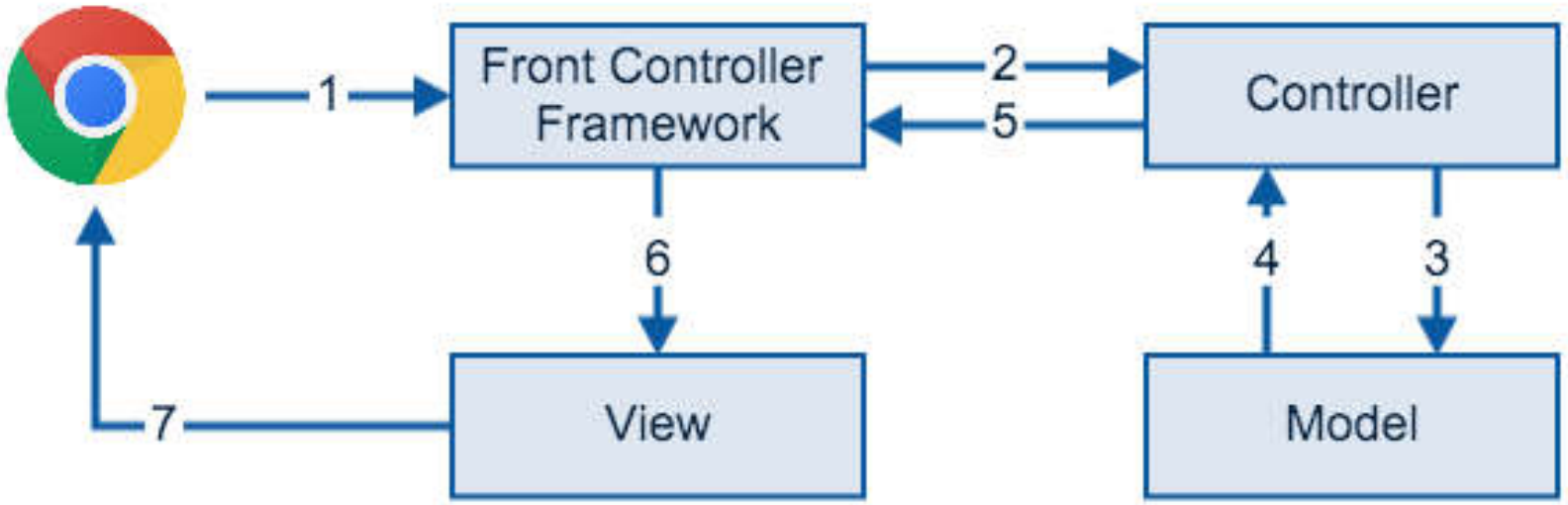
Controller

- Geralmente, “Porta de entrada” de um sistema;
- Middleware/Gateway para realizar a interoperabilidade entre view e models;
- Resolver as requisições e revolver para o requisitor;
- Desacoplamento entre as camadas de visão e modelo.

Controller

```
1  package br.com.sistemabancario.controller;
2
3  import br.com.sistemabancario.service.ContaCorrenteService;
4
5  public class ContaCorrenteController {
6
7      private ContaCorrenteService ccService;
8
9      public ContaCorrenteController() {
10         this.ccService = new ContaCorrenteService();
11     }
12
13     @RequestMapping(value = "/remove", method = RequestMethod.DELETE)
14     @ResponseBody
15     public void remove(@RequestBody String id) {
16         ccService.remove(id);
17     }
18
19 }
20
```

Fluxo de informações



Aplicando o padrão MVC

- Ao aplicar o padrão MVC em uma aplicação web Java, podemos ter o seguintes módulos:
- Componente View
  - HTML, JSP, imagens, scripts, taglibs, etc., ou seja, componentes com a finalidade de gerar a apresentação (interface) do usuário
- Componente Controller
  - Servlets que respondem às requisições HTTP
  - O controller trata cada ação originada da view ativando o componente apropriado do modelo para tratar a requisição
  - Seleciona a view de resposta para a ação
- Componente Model
  - JavaBeans e outras classes *helper* que constituem a parte da aplicação responsável em manter e manipular o estado da aplicação (lógica da aplicação)
  - Inclui também o banco de dados

Universidade Federal do ABC

MC0037 – Programação para Web

Aula 3: Spring MVC

2 Q / 2018

Introdução

- Hoje, foco na estrutura do projeto

- Não iremos aprofundar nas interfaces hoje...próxima aula

Spring Framework

➢O Spring Framework é um dos mais populares frameworks para o desenvolvimento de aplicações Java

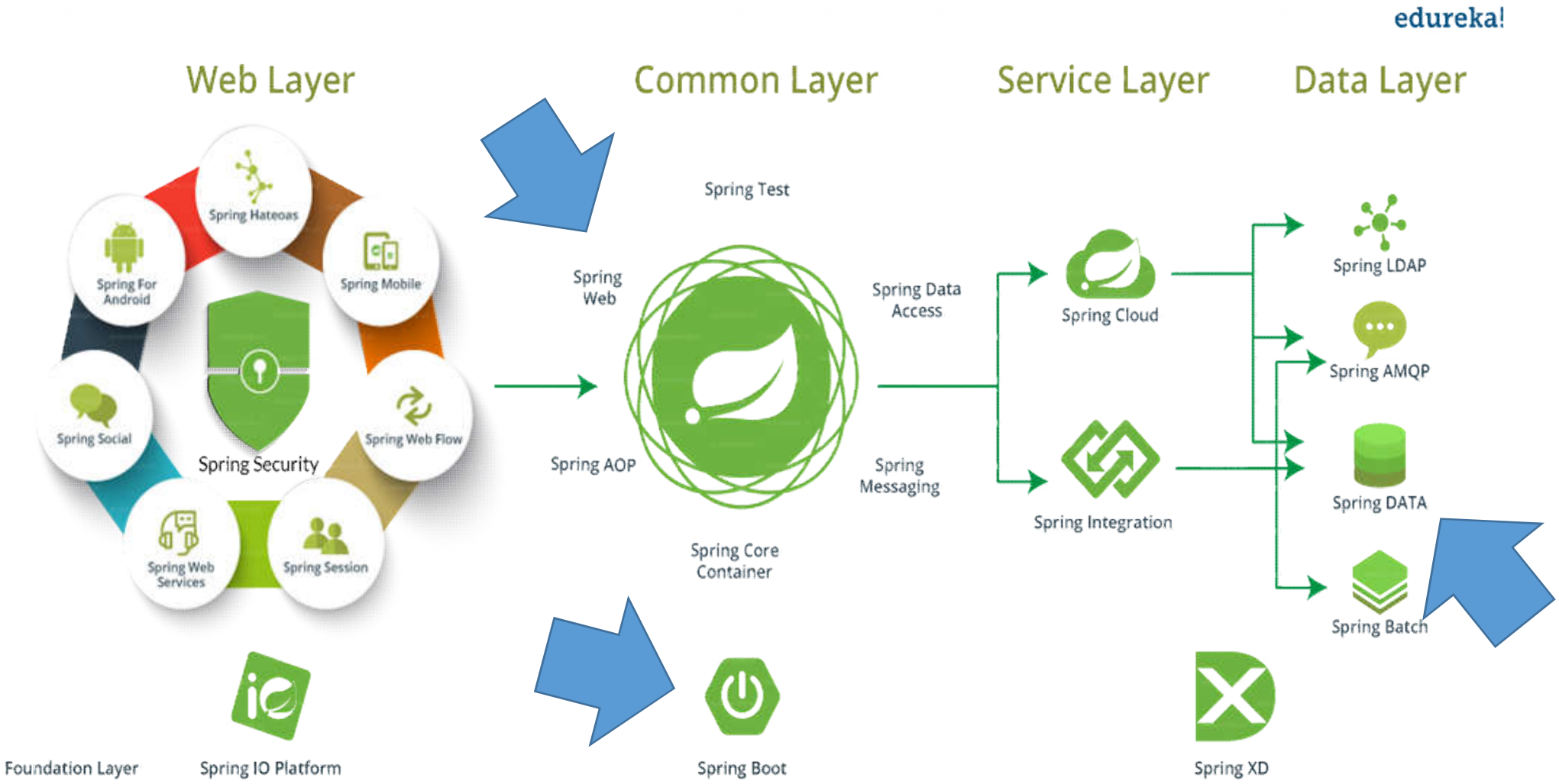
➢É open source, foi inicialmente escrito por Rod Johnson, lançado em junho de 2003

➢Foi concebido para facilitar o desenvolvimento de aplicações Java, em particular o gerenciamento de objetos e de transações, oferece facilidades para testes, etc. e possibilita a criação de código flexível, reutilizável e fracamente acoplado

➢

➢Posteriormente foram lançadas extensões para a construção de aplicações web (Spring MVC)

Spring Framework



Spring MVC

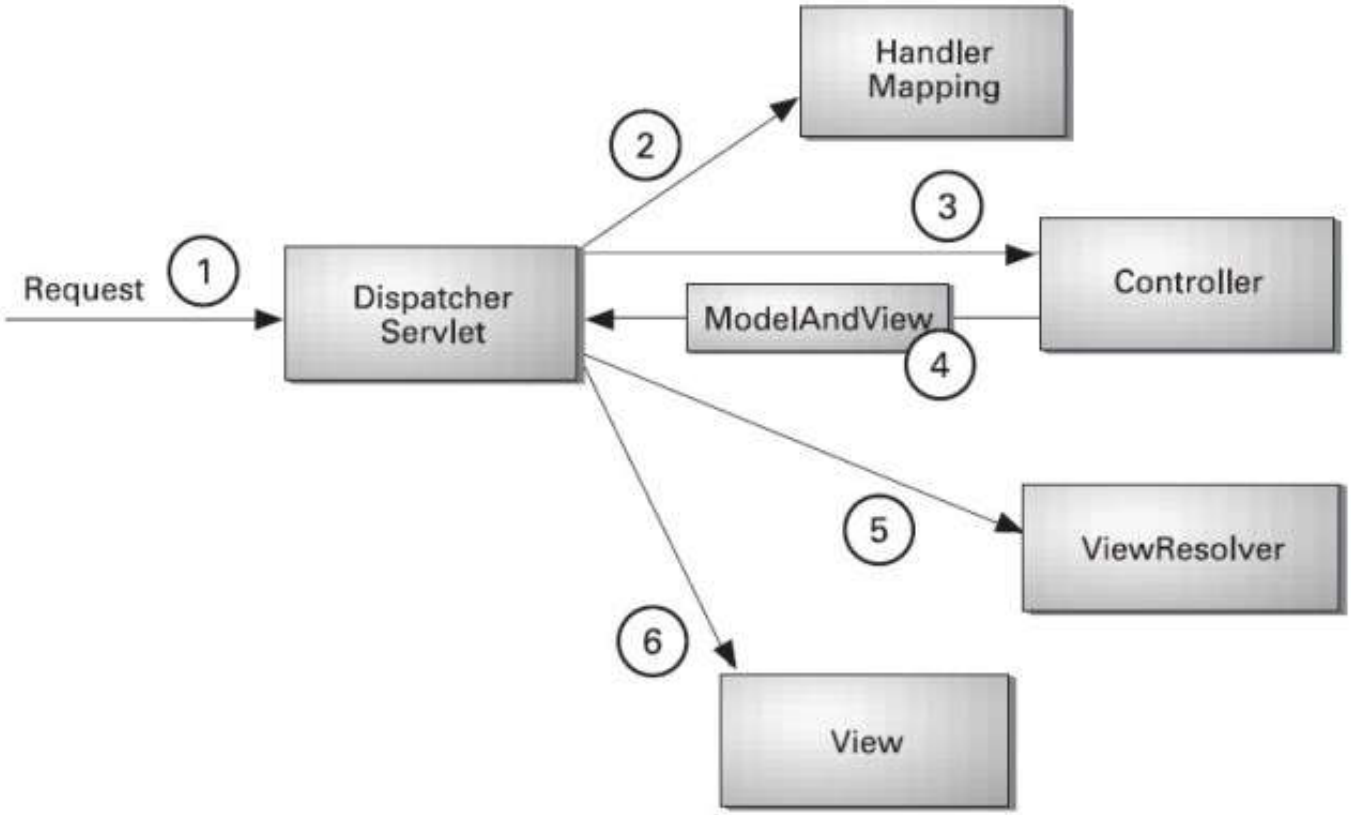
➢Conforme visto anteriormente, o padrão MVC permite a separação dos diferentes aspectos da aplicação permitindo um desacoplamento entre elementos

➢Surgiram alguns frameworks MVC como o Structs, um dos mais antigos e que se tornou o mais utilizado por anos

➢O Spring MVC foi desenvolvido aproveitando-se os benefícios do framework Spring, é leve e oferece componentes para facilitar o desenvolvimento de aplicações web, permitindo que os desenvolvedores se concentrem mais na lógica de negócios

Spring MVC

➢O Spring MVC, assim como outros frameworks MVC, é baseado no padrão *Front-Controller*, um servlet chamado de DispatcherServlet trata todas as requisições e respostas HTTP



Spring MVC

➢ Assim como o Hibernate, o Spring é baseado no conceito de *annotations*, utilizadas para atribuir um comportamento ou interagir com o atributo;

Spring MVC

- @Controller
  - Anotação utilizada para definir uma classe Controller

- @RequestMapping(value="/<ref_url>", method = RequestMethod.POST)
  - Anotação para indicar qual método será executando quando uma URL for solicitada. No caso, a URL será definida no atributo VALUE. O outro atributo method indica o método empregado para passagem de parâmetros (GET ou POST).

- @Repository
  - Anotação utilizada para definir uma classe Model

- @Service
  - Responsável pelas regras de negócio.

Criando um Controller

➢Controller: recebe requisições do servlet (DispatcherServlet) e invoca o método apropriado para tratá-la

➢É preciso que a classe Controller esteja dentro do pacote da aplicação web, definido no arquivo spring-context.xml (br.edu.ufabc.testespring) e deve ter a anotação @Controller

```
@Controller
public class TesteSpringController {
        @RequestMapping("/teste")
        public String execute()      {
                System.out.println("Teste com Spring MVC");
                return "resposta";
        }
}
```

➢A anotação @RequestMapping é usada para mapear uma URL ao método que irá tratar a requisição, usando o atributo “value”, que pode ser omitido

➢O método retorna uma String, que é o nome do arquivo JSP que

Criando um Controller

- @Repository
  - Anotação utilizada para definir uma classe Model

```
import org.springframework.stereotype.Repository;
// omitindo os outros imports

@Repository
public class Convidados {

    // omitindo o código

}
```

- @Service
  - Responsável pelas regras de negócio.

```
@Service
public class DataSourceService {

    @Autowired
    private DataSourceRepository dsRepository;

    public List<DataSource> listAll() {
        return dsRepository.findAll();
    }

    public DataSource get(String id) {
        return dsRepository.findOne(id);
    }
```

Criando um Controller

- @Repository
  - Anotação utilizada para definir uma classe Model

```
12 @Repository
13 public interface ContaCorrenteRepository extends JpaRepository<ContaCorrente, Long> {
14
15     //Spring Data JPA
16     public List<ContaCorrente> findByNumero(String numero);
17
18     @Query("select c from ContaCorrente c where c.numero LIKE (%:numero%)")
19     public List<ContaCorrente> findByNumeroJSQL(@Param("numero") String numero);
20
21     @Query(value = "select * from conta_corrente where numero LIKE %?1%", nativeQuery = true)
22     public List<ContaCorrente> findByNumeroNativo(String numero);
23
24
25 }
26
```

Criando um Repository

- @Repository
  - Anotação utilizada para definir uma classe Model

```
@Repository
public interface ContaCorrenteRepository extends JpaRepository<ContaCorrente, Long> {

    //Spring Data JPA
    public List<ContaCorrente> findByNumero(String numero);

    @Query("select c from ContaCorrente c where c.numero LIKE (%:numero%)")
    public List<ContaCorrente> findByNumeroJSQL(@Param("numero") String numero);

    @Query(value = "select * from conta_corrente where numero LIKE %?1%", nativeQuery = true)
    public List<ContaCorrente> findByNumeroNativo(String numero);


}
```

Spring MVC em Projetos: Gradle

```
buildscript {
   repositories {
      mavenCentral()
   }
   dependencies {
      classpath("org.springframework.boot:spring-boot-gradle-plugin:2.0.3.RELEASE")
   }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'org.springframework.boot'
apply plugin: 'io.spring.dependency-management'

bootJar {
   baseName = 'gs-spring-boot'
   version =  '0.1.0'
}

repositories {
   mavenCentral()
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

dependencies {
   compile("org.springframework.boot:spring-boot-starter-web")
   testCompile("junit:junit")
}
```

Spring MVC em Projetos: Controller

```
@Controller
public class AppController {

    @RequestMapping(value = "/")
    @ResponseBody
    public void helloWorld() {
        System.out.println("Oi Mundo!");
    }

}
```

Spring MVC em Projetos: Main

```
@SpringBootApplication
public class App {
    public static void main(String[] args) {
    SpringApplication.run(App.class, args);
    }
}
```

Spring MVC

- Inversão de Controle (IoC) e Injeção de Dependências
  - Classes acopladas e dependentes entre si;

Spring MVC

➢Inversão de Controle (IoC) e Injeção de Dependências

- ➢Classes acopladas e dependentes entre si;
- ➢Por ex:, factory...

```
public class ContatoDAO {

    private Connection connection;

    public ContatoDAO() {
        this.connection = new ConnectionFactory().getConnection();
    }
```

- ➢... Herança...

Spring MVC

➢Inversão de Controle (IoC) e Injeção de Dependências

- ➢Classes acopladas e dependentes entre si;
- ➢Por ex:, classe estática...

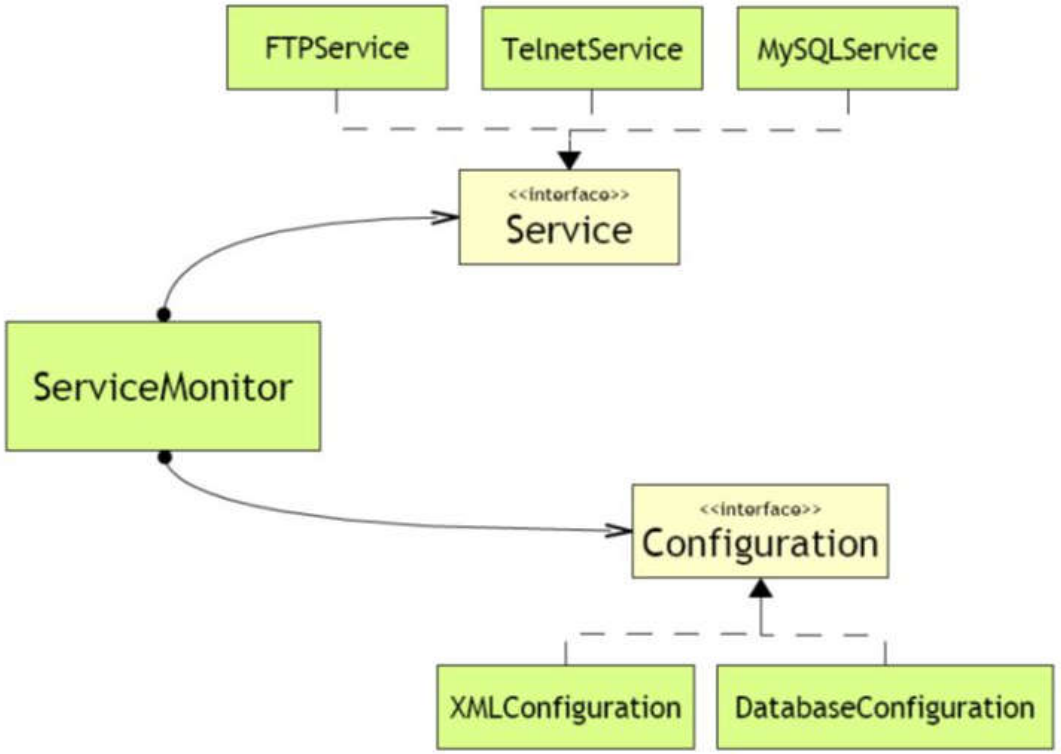
```
public void save(ContaCorrente cc) {
    EntityManagerFactory factory = Persistence.createEntityManagerFactory("sistemabancario");
    EntityManager em = factory.createEntityManager();
```

- ➢... instanciação

Spring MVC

➢Inversão de Controle (IoC) e Injeção de Dependências

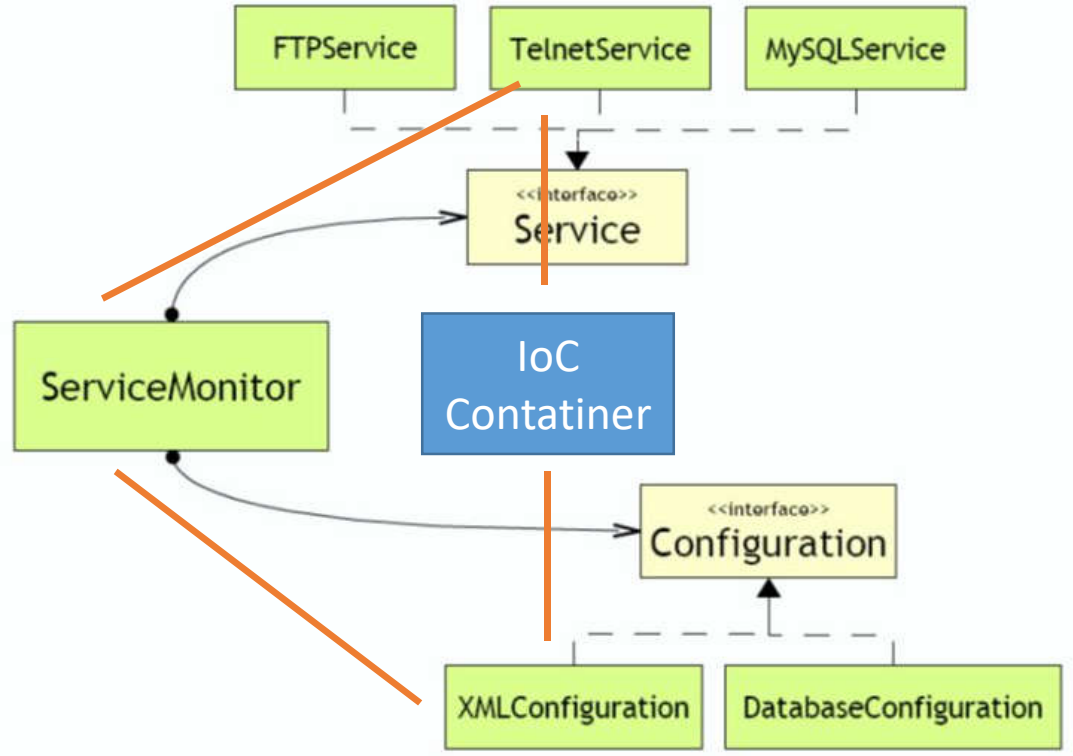
- ➢O programador não cria as instancias diretamente, mas sim “delega” para um terceiro”.
- ➢Container IoC



Spring MVC

➢Inversão de Controle (IoC) e Injeção de Dependências

- ➢O programador não cria as instancias diretamente, mas sim “delega” para um terceiro”.
- ➢Container IoC



Spring MVC

➢Inversão de Controle (IoC) e Injeção de Dependências

- ➢O programador não cria as instancias diretamente, mas sim “delega” para um terceiro”.
- ➢Container IoC
- ➢ “Don’t call me, I’ll call you” (Hollywood Principles)
- ➢ Desacoplamento
- ➢ Código mais legível

Spring MVC

➢Inversão de Controle (IoC) e Injeção de Dependências

```
class MyLogic {
    private BusinessServiceInterface businessService = null;
    protected final BusinessServiceInterface getBusinessService() {
        if (businessService == null) {
            // retrieve JNDI InitialContext
            Context ctx = new InitialContext();
            Context env = (Context) ctx.lookup("java:comp/env");
            Object obj = env.lookup("ejb/BusinessServiceHome");
            // retrieve EJB stub
            BusinessServiceHome businessServiceHome =
                (BusinessServiceHome) PortableRemoteObject.narrow(
                                env, BusinessServiceHome.class);
            businessService = businessServiceHome.create();
        }
        return businessService;
    }
    public void doYourThing() {
        getBusinessService().doYourBusiness();
    }
}
```

Sem IoC

```
class MyLogic {
    // businessService will be set by the IoC container:
    private BusinessServiceInterface businessService;
    // we'll use setter-based injection (explained later):
    public void setBusinessService(BusinessServiceInterface businessService) {
        this.businessService = businessService;
    }
    public void doYourThing() {
        // perform call on businessService
        businessService.doYourBusiness();
    }
}
```

IoC

Spring MVC

- Inversão de Controle (IoC) e Injeção de Dependências

- @Autowired
  - Anotação empregada para definir que uma classe será injetada.

```
@Service
public class DataSourceService {

    @Autowired
    private DataSourceRepository dsRepository;

    public List<DataSource> listAll() {
        return dsRepository.findAll();
    }

    public DataSource get(String id) {
        return dsRepository.findOne(id);
    }
```

Configuração do Spring MVC

Configuração do Spring MVC

➢É preciso declarar o servlet (DispatcherServlet) e mapear as requisições, da mesma forma como já fizemos antes (web.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         id="WebApp_ID" version="3.1">
         <display-name>progradwebspring</display-name>
         <servlet>
                  <servlet-name>springmvc</servlet-name>
                  <servlet-class>org.springframework.web.servlet.DispatcherServlet
                  </servlet-class>
                  <init-param>
                           <param-name>contextConfigLocation</param-name>
                           <param-value>/WEB-INF/spring-context.xml</param-value>
                  </init-param>
                  <load-on-startup>1</load-on-star
         </servlet>
         <servlet-mapping>
                  <servlet-name>springmvc</servlet-name>
         <url-pattern>/</url-pattern>
         </servlet-mapping>
</web-app>
```

Arquivo de configuração do Spring

Aquivo de configuração do Spring

- Arquivo spring-context.xml (em WEB-INF)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

        <context:component-scan base-package="br.edu.ufabc.testespring" />
        <mvc:annotation-driven />
        <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                <property name="prefix" value="/WEB-INF/views/" />
                <property name="suffix" value=".jsp" />
        </bean>
</beans>
```

Pacote padrão da aplicação.

Pasta onde estão os Arquivos JSP.

Spring MVC

MC0037 – Programação para Web

Aula 3: Spring Boot

2 Q / 2018

Introdução

- Hoje, foco na estrutura do projeto

- Não iremos aprofundar nas interfaces hoje...próxima aula

Spring Boot: Motivação

➢Complexidade na configuração de aplicações em Java;

➢Diversos arquivos XML;

➢Foco excessivo e "perda" de tempo para tratar apenas de por menos na configuração de projetos Java;

➢Menos tempo para outras partes de interesse: negócio, visualização, acesso ao banco, etc.

Spring Boot

- Componente do projeto Spring

- Busca facilitar o processo de configuração e publicação de nossas aplicações: adoção de convenções para configuração

  - Spring Boot é tudo acerca de opinião, ou seja, a menos que especifiquemos explicitamente, ele irá trazer dependências predefinidas

- Baseado em anotações, assim como os demais projetos apresentados anteriormente, SpringMVC e Hibernate

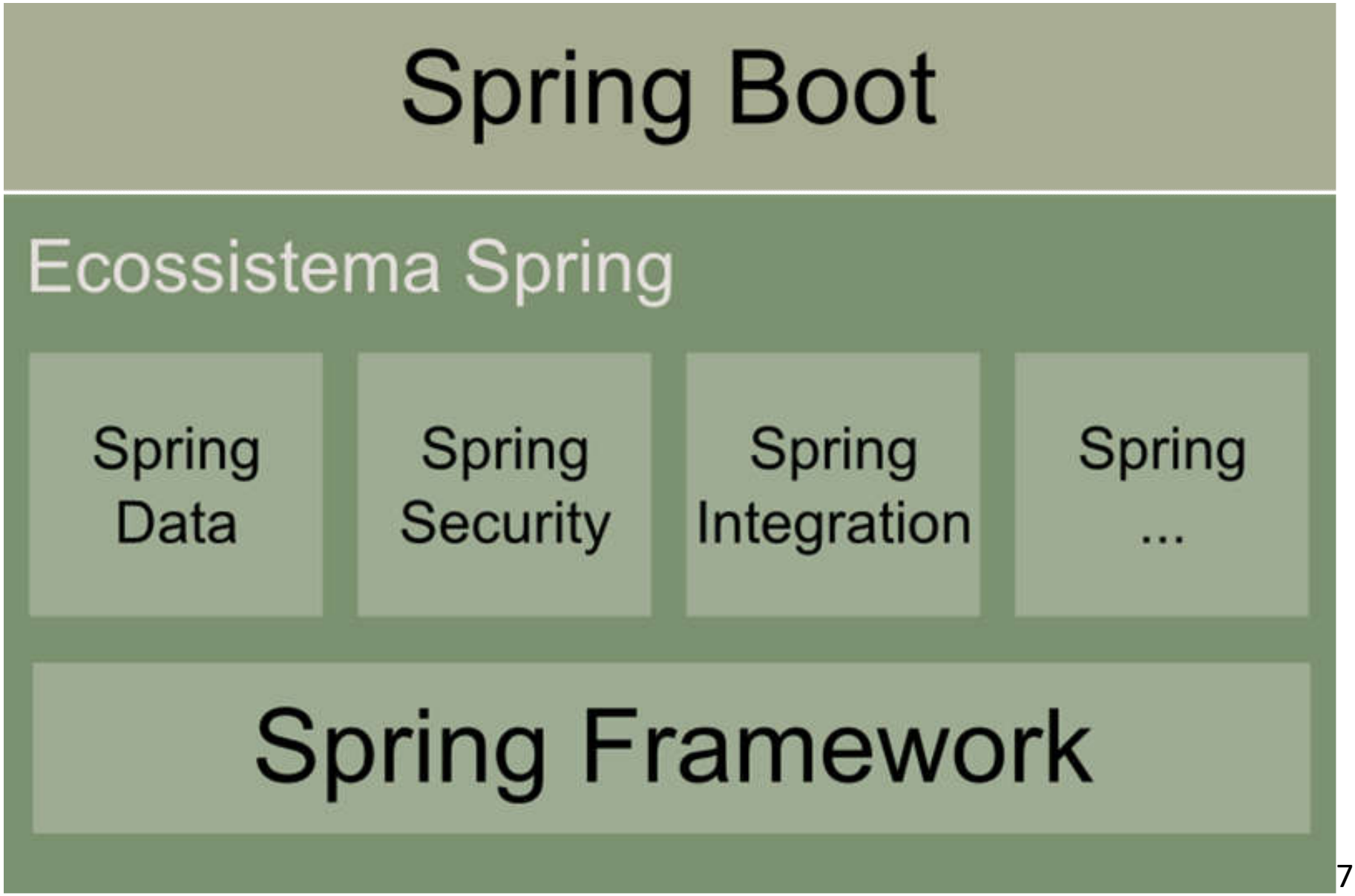
Spring Boot

➢Auxilia na gestão das dependências de pacotes (Gradle e Maven)

➢Também é flexível para permitir que você estenda as definições e possa adequá-las as suas necessidades

➢Servidor embarcado (Tomcat)

Spring Boot


Spring Boot
Ecossistema Spring
Spring Data
Spring Security
Spring Integration
Spring ...
Spring Framework

Spring Boot

- Gerador de projetos online: https://start.spring.io/

- Generate a Gradle Project
- With Java
- And Spring Boot 2.0.3

- Project Metadata
  - Group: refere-se ao pacote organizador do nosso projeto; sem o nome do projeto em si (por ex., br.edu.ufabc ou br.com.bb)
  - Artifact: nome do nosso projeto.

- Dependencies
  - JPA: inclui as dependências do hibernate
  - Web: inclui as dependências do SpringMVC
  - DevTools: Inclui as dependências do SpringBoot
  - Postgres: Inclui os drivers do Postgres

Spring Boot

➢Configuração do banco de dados em Resources no arquivo application.properties

```
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/SistemaBancario
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.jpa.properties.hibernate.default_schema=public
```

Spring Boot

➢Elementos de interface localizados na pasta: resources/templates/

```
<table>
    <thead>
      <tr>
        <th>número</th>
      </tr>
    </thead>
    <tbody>
<tr th:each="contaCorrente : ${ccs}">
<td th:text="${contaCorrente.numero}"></td>
</tr>
</tbody>
```

Spring Boot

➢Elementos de interface localizados na pasta: resources/templates/

```
<form action="save" method="post">
<div class="form-group">
<label for="nome">Numero</label> <input type="text"
class="form-control" id="numero" name="numero" placeholder="numero" />
</div>
<div class="form-group">
<label for="email">Descricao</label> <input type="text"
class="form-control" id="descricao" name="descricao" placeholder="descricao" />
</div>
<div class="form-group">
<label for="email">Ativa</label>
<input type="radio" name="ativa" value="true"> Ativa<br>
  <input type="radio" name="ativa" value="false"> Desativa<br>
</div>
<div class="form-group">
<label for="telefone">Variacao</label> <input type="text"
class="form-control" id="variacao" name="variacao"
placeholder="variacao" />
</div>
<button type="submit" class="btn btn-success">Salvar</button>
</form>
```