Modulo II
Spring Framework
IoC Container

Prof. Ismael H F Santos

Bibliografia

- Spring in Action
  - Craig Walls and Ryan Breidenbach
- Professional Java Development with Spring
  - Rod Johnson, Juergen Hoeller and Team
- Spring – www.springframework.org
- J2EE without EJB – Rod Johnson/ Jurgen Hoeller
- Better, Faster, Lighter Java – Bruce Tate
- Wiring your Web Application with Open Source Java http://www.onjava.com/pub/a/onjava/2004/04/07/wiringwebapps.html

Spring Related Tools and Add-Ons

- ACEGI Security - comprehensive security services for the Spring Framework
- Spring IDE - graphical user interface for the configuration files used by the Spring Framework
- Spring BeanDoc - tool that facilitates documentation and graphing of Spring bean factories and application context files
- XDoclet Spring Tags - support for generating Spring XML config files from annotations in Java classes (you could also use JDK1.5 annotations to achieve this)
- Spring Web Flow - for web applications with demanding page flow requirements
- AppFuse Not really a tool or add-on, but AppFuse is Matt Raible's project to jumpstart your Java web projects. It uses Spring at it's core and studying it is a great way to learn about Spring.
- Spring Framework .NET – Spring Clone for the Dark Side ☺

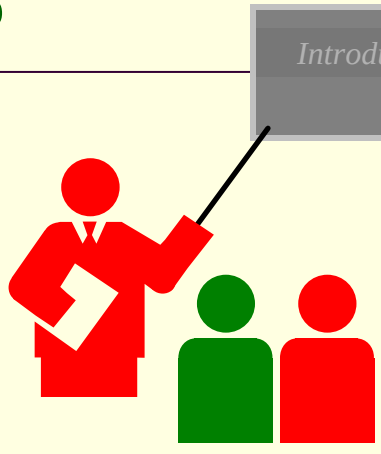
Spring Framework / Spring Related References

- **The Official Spring Reference Manual**
  http://www.springframework.org/docs/reference/
- **Introduction to Spring by Rod Johnson**
- http://www.theserverside.com/articles/article.tss?l=SpringFramework
- **Spring in Action** by Craig Walls and Ryan Breidenbach
- **Pro Spring** by Rob Harrop and Jan Machacek
- **J2EE Without EJB** by Rod Johnson and Juergen Holler
- **Expert One-on-One J2EE Design and Development** by Rod Johnson
- **Spring Developers Notebook** by Bruce Tate and Justin Gehtland
- **Better, Faster, Lighter Java** by Bruce Tate and Justin Gehtland
- **Spring Live** by Matt Raible
- **Professional Java Development with the Spring Framework**
- by many of the core Spring developers: **Coming in July 2005**

Ementa

- Introdução ao Framework Spring
- Spring IOC

WebApp



Background to Spring 2.5

- Spring has become de facto standard component model for enterprise Java
  - Gartner:
    - 75% of middleware vendors will provide Spring integration by 2009
  - Forrester
    - "Most enterprise Java users reported using Spring"
  - BEA
    - Most respondents to Dev2Dev survey use Spring
  - Job listings
    - Spring leads among Java component model technologies in worldwide job requirements

Spring

Spring Framework History

- Started 2002/2003 by Rod Johnson and Juergen Holler
- Started as a framework developed around Rod Johnson's book Expert One-on-One J2EE Design and Development
- Spring 1.0 Released March 2004
- 2004/2005 Spring is emerging as a leading full-stack Java/J2EE application framework
- **Where is Spring Today ?**
  - The favourite and **most trusted** Java application framework
  - Most powerful technology for enabling **POJO-based** application development;
  - **Widely adopted** across most industries and proven in many demanding applications

Spring

Spring

What is the Spring Framework?

- Spring is a Lightweight ***Application*** Framework
- Where **Struts**, **WebWork** and others can be considered ***Web*** frameworks, **Spring** addresses all tiers of an application
- **Spring** provides the plumbing so that you don't have to!
- ***The Spring Framework is a layered Java/J2EE application framework based on code published in Expert One-on-One J2EE Design and Development. The Spring Framework provides a simple approach to development that does away with numerous properties files and helper classes littering the codebase.*** ***-> en.wikipedia.org***

Spring

Spring Framework

- A lightweight framework that addresses each tier in a Web application.
  - **Presentation layer** – An MVC framework that is most similar to Struts but is more powerful and easy to use.
  - **Business layer** – Lightweight IoC container and AOP support (including built in aspects)
  - **Persistence layer** – DAO template support for popular ORMs and JDBC
    - Simplifies persistence frameworks and JDBC
    - Complimentary: Not a replacement for a persistence framework
- Helps organize your middle tier and handle typical J2EE plumbing problems.
- Reduces code and speeds up development
- Promotes decoupling and reusability
- POJO Based
- Built in aspects such as transaction management

Spring

Spring Framework (continued)

- Allows developers to focus more on reused business logic and less on plumbing problems.
- Reduces or alleviates code littering, ad hoc singletons, factories, service locators and multiple configuration files.
- Removes common code issues like leaking connections and more.
- Most business objects in Spring apps do not depend on the Spring framework.
- Do I have to use all components of Spring?
  - Spring is a non-invasive and portable framework that allows you to introduce as much or as little as you want to your application.

Spring is Non-Invasive

- What does that mean?
  - You are not forced to import or extend any Spring APIs
- An invasive API takes over your code.
- Anti-patterns:
  - EJB 2.1 forces you to use JNDI
  - Struts forces you to extend **Action**
- Invasive frameworks are inherently difficult to test. You have to stub the *runtime* that is supplied by the application server

Spring Framework (continued)

- **Modules of the Framework**
  - **Inversion of Control (IoC) Container**
    - Also known as Dependency Injection (Fowler's term)
  - **Aspect-Oriented Programming Framework (AOP)**
    - Spring provides a proxy-based AOP framework
    - You can alternatively integrate with AspectJ or AspectWerkz
  - **Data access abstraction and JDBC** simplifications
  - **Transaction Management**
  - **MVC web framework**
  - **A Service Abstraction Layer**
    - Simplification for working with J2EE APIs JNDI, JTA, etc.
  - **Lightweight remoting, JMS support, JMX support**
  - **Support for a comprehensive testing strategy** for application developers

Spring Mission Statement

- **J2EE should be easier to use**
  - It's best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero.
  - JavaBeans offer a great way of configuring applications – **POJO programming model**
  - OO design is more important than any implementation technology such as J2EE.
  - Checked exceptions are overused in Java. A framework shouldn't force you to catch exceptions you're unlikely to be able to recover from.
  - Testability is essential, and a framework such as Spring should help make your code easier to test.

www.springframework.org

Spring Framework Mission Statement (continued)

- **The authors of Spring aim that:**
  - Spring should be a pleasure to use
  - Your application code should **not** depend on Spring APIs
  - Spring should not compete with good existing solutions, but should foster integration. (For example, JDO and Hibernate are great O/R mapping solutions. We don't need to develop another one.)
- **Spring == J2EE Application Server?**
  - Spring is NOT a J2EE application server
  - Spring can integrate nicely with J2EE application servers (or any Java environment)
  - Spring can, in many cases, elegantly replace services traditionally provided by J2EE application servers

Spring x J2EE

Spring Framework vs. J2EE

Summation of Persistence Comparison Between Spring and EJB 3.0.

Feature	Spring	EJB 3.0
Simple ORM Persistence	√	√
Implementation	Hibernate, JPA, JDO, TopLink, IBatis	JPA (providers include Hibernate, Kodo and Toplink)
JDBC Support	√	--
Mapping	XML, Annotations	Annotations, XML
Cache Propagation	Thread local	Transaction
Extended cache scoping	Open session in view	Extended persistence context
Standard	√ (If using JPA)	√

http://www.[illegible].com/Java/Article/32314/0/page[illegible]1

Spring x J2EE

Spring Framework vs. J2EE

Equivalent Transaction Management Functionality in Both Spring and EJB 3.0.

Feature	Spring	EJB 3.0
Declarative Transactions	√	√
Programmatic Transactions	√	√
Demarcation	AOP	Session bean methods
Supported Transaction Types	JDBC, Hibernate, JTA	JTA
Support for Distributed Transactions	√ (With JTA)	√
Configuration	XML	Transactional by default, override with annotations or XML
Standard	√ (With JTA)	√

http://www.devx.com/Java/Article/32314/0/page/1

Spring x J2EE

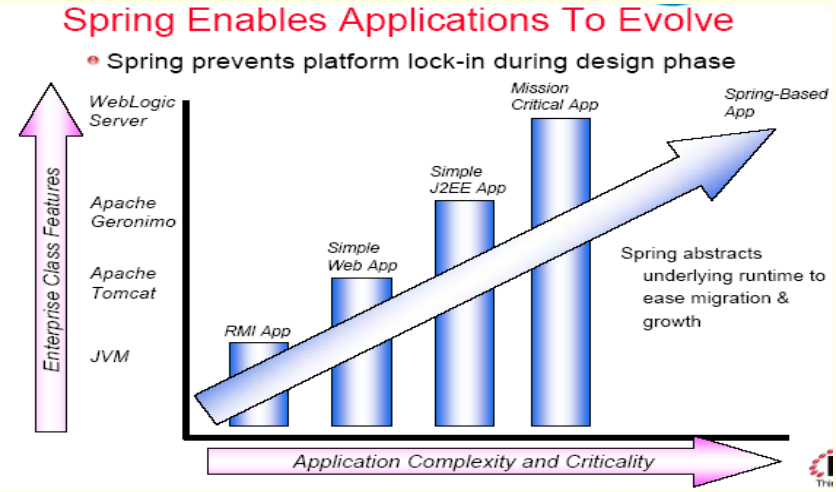
Spring Framework vs. J2EE

Differences in State-handling Between Spring and EJB 3.0.

Feature	Spring	EJB 3.0
Stateful construct	Prototype (instance) beans	Stateful Session Bean
Instance Management	Singleton, prototype, request, session, global session	Instance per lookup
Lifecycle Management	√ (Initialization/Destruction)	√ (Creation/Destruction, Activation/Passivation)
Transaction awareness	--	√ (SessionSynchronization interface)
Standard	N/A	√

http://www.devx.com/Java/Article/32314/0/page/1

Usage of Spring



Lessons Learned from Struts

- Before Struts, everyone wrote their own front controllers (or worst, put their controller logic in JSP)
- After Struts, the custom front controllers could be thrown out
  - Developers focus on solving business problems
  - Productivity Gain!
- But with Struts (and most of the other web frameworks) you still have to write your own business delegates or service layers…

Spring Can Help!

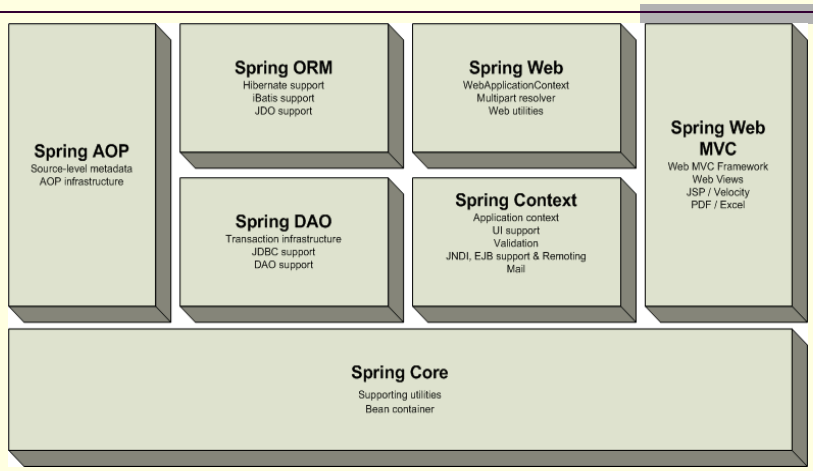
- Spring brings a consistent structure to your entire application.
  - Organizes middle tier objects, takes care of plumbing, eliminates the proliferation of Singletons. Beans are defined in a centralized configuration file
- Spring provides a consistent way to glue your whole application together. Provides a loosely couple business logic in a POJO fashion. Allows to build portable applications that provided clearer separation of presentation, business, and persistence logic.
  - Applications depend on as few of its APIs as possible.
  - Applications are easy to unit test.
  - Provides a consistent framework for data access
  - Enables you to stop polluting code. No more custom factory object to build and/or locate other objects

Spring

Benefits of Spring?

- Spring can provide a way to simplify DAO objects and provide declarative transaction support to our non-EJB applications.
  - Consistent CRUD and Data access templates
  - No more copy-paste try/catch/finally blocks
  - No more passing Connection objects between methods
  - No more leaked connections
- Spring provides elegant integration points with standard and de facto-standard interfaces: JPA, Hibernate, JDO, TopLink, EJB, RMI, JNDI, JMS, Web Services, Struts, etc.
- Spring has a nice balance between constraint and freedom. A good framework should provide guidance with respect to good practice, making the right think easy to do, but should not be overly restrictive placing requirements on code using it causing lock in and constraining developers in inappropriate ways.

Spring Overview

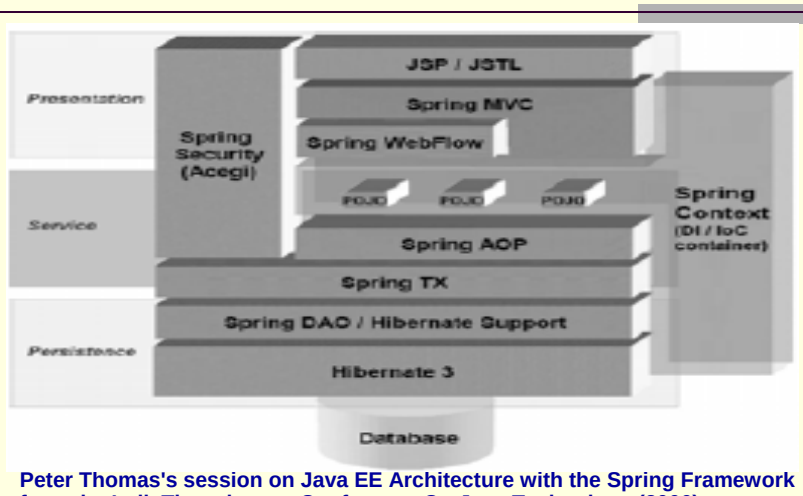


Note: Spring distribution comes as one big jar file and alternatively as a series of smaller jars broken out along the above lines (so you can include only what you need)

Spring Overview

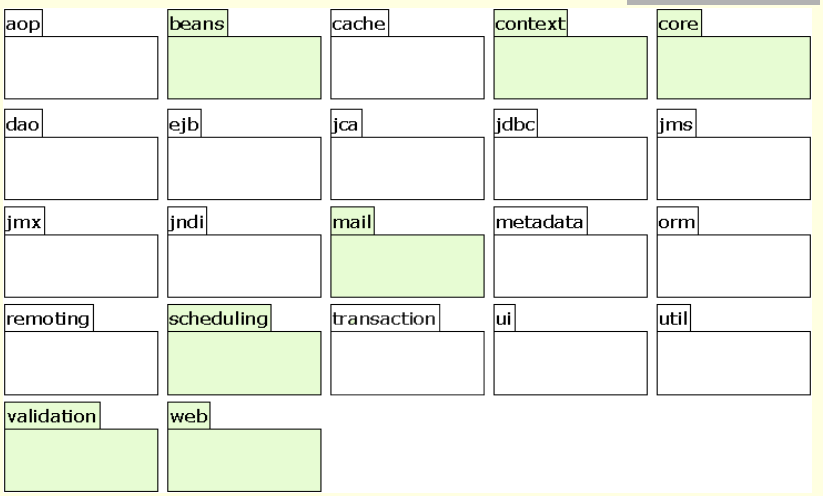
- O módulo Spring Core contém as principais funcionalidades do Spring, no qual o principal elemento é o BeanFactory. Trata-se de uma implementação do padrão *Factory*, é responsável pela criação dos objetos da aplicacao (*Singletons ou Prototypes)* isolando a especificação de dependências da lógica de programação aplicacão.
- O módulo Spring DAO provê uma camada de abstração para JDBC. Enquanto o módulo ORM, provê integração do Spring com outros frameworks para persistência de objetos: JPA, Hibernate e iBatis.
- O módulo Spring AOP provê uma implementação de Orientação a Aspectos que permite a definição de *pointcuts* e *methods interceptors*.
- Spring Web disponibiliza funcionalidades específicas para projetos Web. São funcionalidades como componentes para *upload* de arquivos e suporte para utilização de Inversão de Controle neste tipo de aplicação. O módulo Spring MVC, fornece uma implementação de framework Web, similar ao Struts.

Spring Overview



- **Peter Thomas's session on Java EE Architecture with the Spring Framework from the IndicThreads.com Conference On Java Technology (2006)**

Spring Java Packaging - org.springframework.*



Some Spring Features

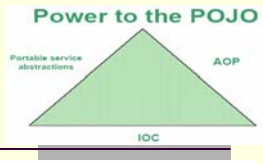
- JPA(Java Persistence Architecture) integration
- JdbcTemplate simplification for Java 5
- Ability to define any named bean in scripting language such as Groovy or JRuby
  - Named bean conceals both configuration and implementation language
  - Allows for DI, AOP and dynamic reloading
- MVC Simplification: Intelligent defaulting, JSP form tags
- Spring Portlet MVC, an MVC framework for JSR-168 Portlets
- Asynchronous JMS facilities enabling message-driven POJOs

But really, what *IS* Spring?

At it's core, Spring provides:

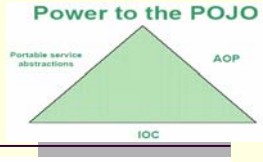
- An **Inversion of Control Container**
  - Also known as Dependency Injection (Fowler's term)
- An **AOP Programming Framework**
  - Spring provides a proxy-based AOP framework
  - You can alternatively integrate with AspectJ or AspectWerkz
- A **Service Abstraction Layer**
  - Consistent integration with various standard and 3rd party APIs
- These together enable you to write powerful, scalable applications using POJOs.

The Spring Triangle



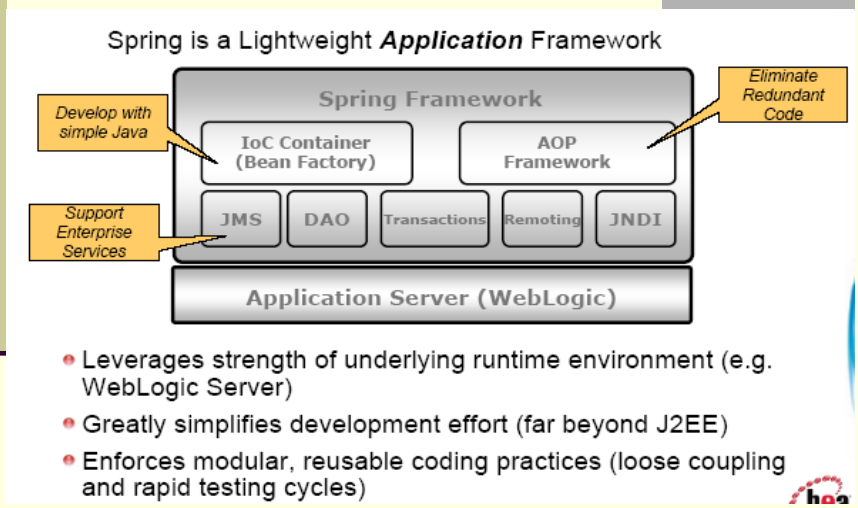
- **POJO** stands for **Plain Old Java Object**
- A POJO is not bound to any environment
  - No imports of classes that bind it to a specific environment
  - Only uses framework classes that offer good abstraction like Spring's persistence framework
  - Not dependent on a particular lookup mechanism
    - Collaborating instances are injected using plain Java constructors or setter methods
  - True POJOs are testable in isolation

The Spring Triangle

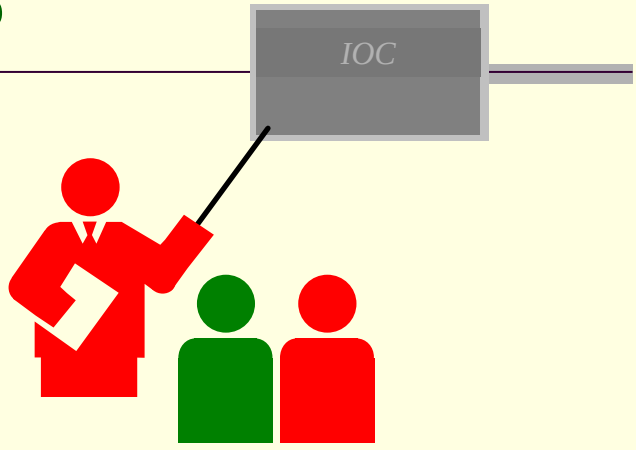


- Decouples your application objects from their environment
  - Brings leverage, enables reuse
- Actually *more* powerful than traditional *invasive* component models
  - Lets you scale up or down without rewriting application code
- Examples
  - Switch to global transactions over JTA
  - Export your business objects in different environments
    - Switch between SLSB, web service, write/take from JavaSpace etc.

What is the Spring Container?



WebApp



Inversion of Control (Dependency Injection)

Spring

- **Dependency injection (Martin Fowler – 2004)**
  - Beans define their dependencies through constructor arguments or properties
  - Dependencies used from within a bean aren't asked for outwardly, but are injected into the bean by the container, ie the container provides the injection at runtime
- "Don't talk to strangers", also known as the **Hollywood principle** – "don't call me I will call you"
  - Decouples **object creators and locators** from **application logic**. Easy to maintain and reuse
- Testing is easier

Inversion of Control (Dependency Injection)

Spring

- **IoC** can be thought of in terms of what distinguishes a framework from library.
  - A Library performs some work when called and returns to caller. Framework encapsulates some abstract design incorporated with behavior but to use you must incorporate your unique behavior via callbacks or sub-classing.
  - IoC is a principle that is used to wire an application together, how dependencies or object graphs are created.
- In Spring, the **IoC** "flavor" is referred to as **Dependency Injection - DI**.

Inversion of Control (Dependency Injection)

- Eliminates lookup code from within your application
- Allows for pluggablity and hot swapping
- Promotes good OO design
- Enables reuse of existing code
- Makes your application extremely testable
- **IoC container**
  - Setter based and constructor based dependency injection
  - Portable across application servers
  - Promotes good use of OO practices such as programming to interfaces.
  - Beans managed by an IoC container are reusable and decoupled from business logic

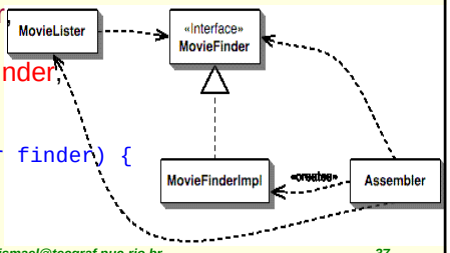
Inversion of Control (Dependency Injection)

- Martin Fowler exemplifica a inversão de dependência *(Inversion of Control)* através de interfaces de interação com o usuário (GUI).
  - Em Swing, definimos os tratadores de eventos para os vários campos da tela, enquanto o framework (Swing) contém o loop principal da aplicação.
- O padrão *Dependency Injection*, idealizado por Martin Fowler, pode sere visto como uma especialização do padrão *Inversion of Control*.
- Spring e PicoContainer, denominados de *lightweight containers*, adotam a inversão de controle, entretanto, todo framework utiliza-se de inversão de controle.
- Que tipo de inversão de controle o Spring realiza? Observe que qualquer framework aplica este padrão !!!

Inversion of Control (Dependency Injection)

- Exemplo: Gerência de cadastro de filmes
  - *interface* MovieFinder defini o comportamento padrão para classes que gerenciam um cadastro de filmes.
  - MovieFinderImpl define uma implementação concreta da *interface*
  - MovieLister utiliza uma implementação da *interface* MovieFinder para realizar a busca de filmes para apresentar em um tocador (*player*) de vídeo.
  - A dependência existente entre MovieLister e MovieFinder é resolvida pela classe Assembler, a qual gerencia a "injeção" de uma implementação de MovieFinder.

```
public class MovieLister {
  public MovieLister(MovieFinder finder) {
    this.finder = finder;
  }
}
```



Inversion of Control (Dependency Injection)

- Exemplo: Gerência de cadastro de filmes (cont.)
  - Caso a classe MovieLister instanciasse diretamente (através da chamada de new MovieFinderImpl) um objeto do tipo MovieFinder, perderíamos a capacidade de tornar MovieFinder "plugável".
  - A interface MovieFinder é, portanto, um contrato ou um padrão a ser seguido por quem deseja criar "Buscadores" de filmes, de forma que um módulo separado, o Assembler, possa injetar esta implementação em MovieLister. Podemos, desta forma, criar programas no qual as partes que o compõem são plugins gerenciados pelo Assembler.
  - Basicamente, existem dois tipos de injeção de dependência: Constructor Injection e Setter Injection. No primeiro tipo, Constructor Injection, a dependência é resolvida através de um construtor do objeto a receber o objeto dependente (veja slide anterior)

Inversion of Control (Dependency Injection)

- Exemplo: Gerência de cadastro de filmes (cont.)
  - Na dependência Constructor Injection, o objeto Assembler resolverá a dependência entre os dois objetos passando para MovieLister uma implementação concreta de MovieFinder através do seu construtor.
  - Na caso de Setter Injection a dependência entre os objetos é resolvida pelo Assembler através de um método *Setter* no objeto MovieFinder.

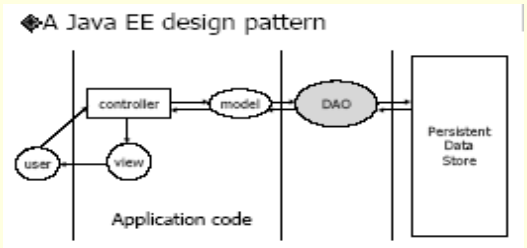
```
public class MovieLister {
  private MovieFinder finder;
  public void setFinder (MovieFinder finder) {
         this.finder = finder;
  }
}
```


Spring

DAO Example – The need for IoC

```
public interface UserDao {
  public User getUserById( Integer id );
  public List getUsersById( Integer ids[] );
  public List getUsersByRoleName( String roleName );
  public User getUserByCin( String cin );
  public User getUserByName( String username );
  public User getUserByEmail( String email );
  public void saveUser( User user );
}
```

Data Access Object (DAO)



Spring

DAO Example – The need for IoC

- Database access through Hibernate

```
public class UserDaoImpl extends HibernateDaoSupport
                         implements UserDao {
  public User getUserById( Integer id ) {
    HibernateTemplate hib = getHibernateTemplate();
    return (User)hib.get(User.class, id);
  }
  ... ...
}
```

- Used in more than twenty controllers, validators, and access decision voters
  - Add instructor/student to class sections
  - Validate whether a username is already used
  - Check whether a user can access certain assignment or grade
  - ...

Spring

UserDao – Usage in Application Code

```
User instructor = userDao.getUserById( instructorId ) );
Section section = sectionDao.getSectionById( sectionId );
section.addInstructor( instructor );
sectionDao.saveSection( section );
```

- Advantages of DAO
  - Provide a data access API that is
    - Independent of persistent storage types, e.g. relational DB, OODB, XML flat files etc.
    - Independent of persistent storage implementations, e.g. MySQL, PostgreSQL, Oracle etc.
    - Independent of data access
    - implementations, e.g. JDBC, Hibernate, JDO, etc.

Instantiate a UserDao Object in Application Code

Spring

```
UserDaoHibernateImpl userDao = new UserDaoHibernateImpl();

UserDao userDao = new UserDaoHibernateImpl();
```

Which one is better??

- What if we decide to use JDBC instead of Hibernate, i.e. replace UserDaoHibernateImpl with UserDaoJdbcImpl
  - The application is not really independent of the data access method
  - Switching to a different UserDao implementation affects all the code that uses UserDao

Instantiate a UserDao Object in Application Code

Spring

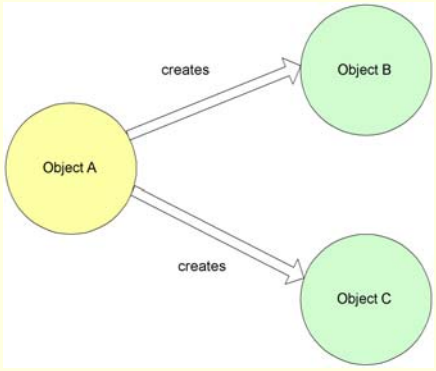
```
UserDao userDao;
...
public void setUserDao( UserDao userDao) {
  this.userDao = userDao;
}
```

- No more dependency on a specific implementation of the DAO. But who will call the setter?

- Inversion of Control (IoC)
  - The application code is no longer responsible for instantiate an interface with a specific implementation, A.K.A. Dependency Injection

Non-IoC / Dependency Injection

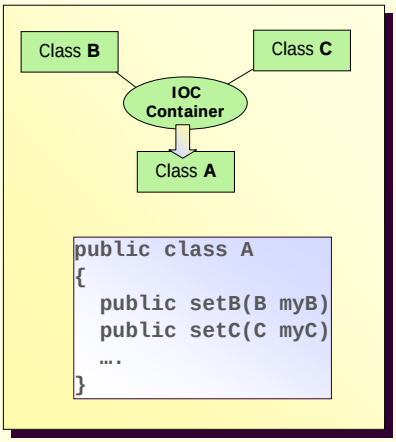
```
public class A
{
   B myB = new B();
   C myC = new C();
}
```

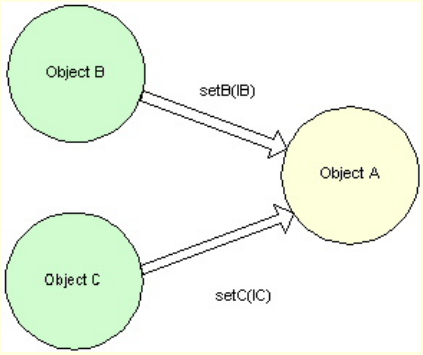


Non-IoC Service Object

```
public class OrderServiceImpl implements IOrderService {
public Order saveOrder(Order order) throws OrderException{
   try{
   // 1. Create a Session/Connection object
   // 2. Start a transaction
   // 3. Lookup and invoke one of the methods in a
  // DAO and pass the Session/Connection object.
   // 4. Commit transaction
   }catch(Exception e){
   // handle e, rollback transaction, //cleanup, // throw e
   }finally{
   //Release resources and handle more exceptions
   }
}
```

IoC / Dependency Injection





IoC Service Object – another example

```
public class OrderSpringService implements IOrderService {
  IOrderDAO orderDAO;

  public Order saveOrder(Order order) throws OrderException{
     // perform some business logic…
     return orderDAO.saveNewOrder(order);
  }

  public void setOrderDAO(IOrderDAO orderDAO) {
    this.orderDAO = orderDAO;
  }
}
```

- Program to interfaces for your bean dependencies!

Simple Spring Bean – another example

```
<bean id="orderBean" class="example.OrderBean"
      init-method="init">
  <property name="minimumAmountToProcess">
      <value>10</value>
  </property>
  <property name="orderDAO">
      <ref bean="orderDAOBean"/>
  </property>
</bean>
```

```
public class OrderBean implements IOrderBean{      …
    public void setMinimumAmountToProcess(double d){
      this.minimumAmountToProcess = d;
    }
    public void setOrderDAO(IOrderDAO odao){
      this.orderDAO = odao;
    }
}
```


BeanFactories

- BeanFactories are the heart of Spring
  - Lightweight container that loads bean definitions and manages your beans. Knows how to serve and manage a singleton or prototype defined bean
  - Responsible for lifecycle methods.
  - Injects dependencies into defined beans when served
  - The **Factory Pattern**: one object is responsible for creating and maintaining the lifecycle of another object.
- A BeanFactory is typically configured declaratively in an XML file, or files, with the root element: <beans>. That file determine how beans can be referenced and wired together and contains one or more <bean> elements
  - id (or name) attribute to identify the bean
  - class attribute to specify the fully qualified class
- By default, beans are treated as singletons, but can also be prototypes

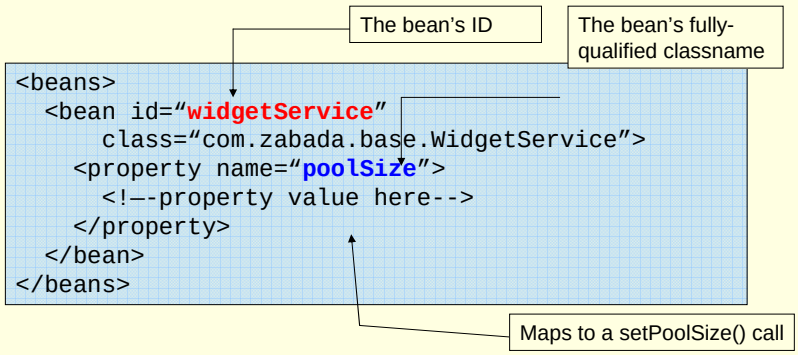
Spring

Spring Bean Definition

- Bean examples – DAO, DataSource, Transaction Manager, Persistence Managers, Service objects, etc
- Spring config contains implementation classes while your code should program to interfaces.
- Bean behaviors include:
  - Singleton or prototype
  - **Autowiring**: byName, byType, constructor, autodetect
  - Initialization and destruction methods
    - init-method
    - destroy-method
- Beans can be configured to have property values set.
  - Can read simple values, collections, maps, references to other beans, etc.

BeanFactories

Here is an example:



Property Values for BeanFactories

- Strings and Numbers

```
<property name="size"><value>42</value></property>
```

```
<property name="name"><value>Jim</value></property>
```

- Arrays and Collections

```
<property name="hobbies">
  <list>
    <value>Basket Weaving</value>
    <value>Break Dancing</value>
  </list>
</property>
```


Property Values for BeanFactories (continued)

- The real magic comes in when you can set a property on a bean that refers to another bean in the configuration:

```
<bean name="widgetService"
  class="com.zabada.base.WidgetServiceImpl">
  <property name="widgetDAO">
    <ref bean="myWidgetDAO"/>
  </property>
</bean>
```

calls **setWidgetDAO(myWidgetDAO)** where **myWidgetDAO** is another bean defined in the configuration

- *This is the basic concept of Inversion of Control*

Types of Dependency Injections

- Type 1 IOC also called Interface Injection
  - In Type 1 IOC the injection is done though an interface. The interface will define the injection method and the implementation class has to implement this interface and provide concrete implementation for the injection method.
- Type 2 IOC also called Setter Injection
  - In Type 2 IOC the injection is done via a setter method. Type 2 IOC uses setter methods to get the dependent classes it needs.
- Type 3 IOC also called Constructor Injection.
  - In Type 3 IOC implementing class defines a constructor to get all its dependents. The dependent classes are defined in the constructor arguments.

Spring

Dependency Injection

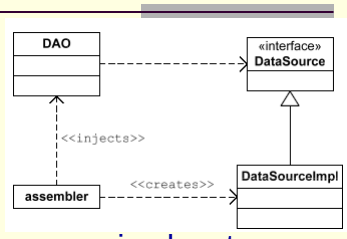
- Methods of injection
  - via Setters
  - via Constructors

Bean	<bean>, "id", "class"
Simple type property	<property>, "name", "value"
Class type property	<property>, "name", "ref" (to another <bean>)
Collection type property	<list>/<set>/<map>/<props>, <value>/<ref>/<entry>/<prop>
Constructor arguments	<constructor-arg>, "index", same as other properties

- Objects that can be injected
  - Simple types: strings and numbers
  - Collection types: list, set, and maps
  - Other beans

Dependency Injection Example

```
public class DAO {
  private DataSource datasource;
  public DAO(DataSource ds) {
    datasource = ds;
  }
  private Connection getConnection() {
    return datasource.getConnection();
  }…
}
```



- With Dependency Injection, unlike a service locator pattern (JNDI), there is no dependency on the service locator mechanism. Dependencies are more apparent with Injection.
- 3 Types of Dependency Injection:
  - Constructor injection: previous example
  - Property (setter injection);
  - Lookup-method injection.

Spring Dependency Injection

- To use Spring Dependency Injection all you need is…
  - POJO with correct constructor (or setter)
  - Spring bean defined in spring-config.xml
  - Access the Bean through the Spring context bean factory.
- POJOs
  - Constructor
  - Setter
- Declarative Dependency Injection with Spring Beans
  - Constructor Injection
  - Setter Injection
  - Lookup Method Injection

Spring Dependency Injection

- Declarative Dependency Injection with Spring Beans
  - Constructor Injection
  - Setter Injection
  - Lookup Method Injection
- Bean Factory Lookup
  - Provê suporte básico para a injeção de dependência
  - Gerencia configs e ciclo de vida de beans

```
SomeClass instance =(SomeClass)context.getBean("beanName");

Where context is an implementation of
org.springframework.beans.factory.BeanFactory
```


Spring ApplicationContext

- An **ApplicationContext** is a **BeanFactory**, but adds "framework" features such as:
  - i18n messages (internationalization)
  - Event notifications and treatment
- **ApplicationContext** extends **BeanFactory**
  - Adds services such as international messaging capabilities.
  - Add the ability to load file resources in a generic fashion
- A Spring **ApplicationContext** allows you to get access to the objects that are configured in a BeanFactory in a framework manner. There are Several ways to configure a context:
  - **XMLWebApplicationContext** – Configuration for a web application.
  - **ClassPathXMLApplicationContext** – standalone XML application context
  - **FileSystemXmlApplicationContext**
- Allows you to avoid writing Service Locators

Configuring an XMLWebApplicationContext

- Application context located in the war file
  - Single root context per application
  - Default: /WEB-INF/applicationContext.xml
- Context is loaded by
  - ContextLoaderListener (Servlet 2.4)
  - ContextLoaderServlet (Servlet 2.3)

Load root application context from `/WEB-INF/applicationContext.xml`

1- web.xml

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>
  <listener-class> org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

2- Inside a Servlet

```
WebApplicationContextUtils.getWebApplicationContext(ServletContext);
```


Configuring an XMLWebApplicationContext

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
      /WEB-INF/applicationContext.xml
  </param-value>
</context-param>

<servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>
  org.springframework.web.context.ContextLoaderServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Spring

Locating a Bean ….

```
public abstract class BaseAction extends ActionSupport
{
  protected IOrderService getOrderService() {
  return (IOrderService)getWebApplicationContext()
            .getBean("orderService");
  }
}

<bean id="orderService"
  class="com.meagle.service.spring.OrderServiceImpl">
```


Spring

FileSystemXMLApplicationContext

- Usually used with a standalone application to obtain a context from files in the FileSystem

▶ Example

```
ApplicationContext ctx =
       new FileSystemXmlApplicationContext("c:/beans.xml");

ExampleBean eb = (ExampleBean)ctx.getBean("exampleBean");
```

▶ ApplicationContext can be read from many files

```
String[] ctxs = new String[]{"ctx1.xml", "ctx2.xml"};

ApplicationContext ctx = new FileSystemXmlApplicationContext(ctxs);
```

Spring

ClassPathXMLApplicationContext

- Usually used with a standalone application to obtain a context
  - Ex: **Swing application**
- Useful when performing integration testing
  - In-container testing.
  - Define a separate application context XML file that contains mappings to your mock objects.

Example-2a

- Example without DI

```
public class WeatherService {
  private WeatherDao weatherDao;
  public Double getHistoricalHigh(Date date) {
    WeatherData wd = weatherDao.find(date);   .....
  }
}
public class StaticDataWeatherDaoImpl implements WeatherDaov{
  public WeatherData find(Date date) { .... }
}
```

- Client

```
WeatherService ws = new WeatherService();
Double high = ws.getHistoricalHigh(
new GregorianCalendar(2004, 0, 1).getTime());
System.out.println("High was: " + high);
```

Example-2b

- Example with DI

```
public class WeatherServiceImpl implements WeatherService {
  private WeatherDao weatherDao;
  public void setWeatherDao (WeatherDao weatherDao) {
    this.weatherDao = weatherDao;
  }
}
```

- XML descriptor file

```
<bean id="weatherService"
   class="swe645.ioc.WeatherServiceImpl">
  <property name="weatherDao">
    <ref local="weatherDao"/>
  </property>
</bean>
<bean id="weatherDao"
  class="swe645.ioc.StaticDataWeatherDaoImpl">
</bean>
```


Example-2b (cont.)

- Client (Stand-alone)

```
ApplicationContext ctx = new
              ClassPathXmlApplicationContext(
                        "ioc/applicationContext.xml");
 WeatherService ws = (WeatherService)ctx.getBean(
                      "weatherService");
 Double high = ws.getHistoricalHigh(new
                GregorianCalendar(2004, 0,1).getTime());
 System.out.println("High was test: " + high);
```

- DI
  1. **Load context**
  2. **Instantiates beans**
  3. **Configures simple properties**
  4. **Resolves dependencies**

Example-4, XmlBeanFactory

```
<beans>
    <bean id="exampleBean" class="eg.ExampleBean"/>
    <bean id="anotherExample" class="eg.ExampleBeanTwo"/>
</beans>
```

- Usage example

```
InputStream input = new FileInputStream("beans.xml");
BeanFactory factory = new XmlBeanFactory(input);

ExampleBean eb =
        (ExampleBean)factory.getBean("exampleBean");

ExampleBeanTwo eb2 =
        (ExampleBeanTwo)factory.getBean("anotherExample");
```

Can throw **NoSuchBeanDefinitionException**

```
ExampleBean eb =
  (ExampleBean)factory.getBean("exampleBean", ExampleBean.class);
```

Can throw **BeanNotOfRequiredTypeException**

Example-4

- Bean Collaborators
  - Other beans your bean needs to do its work

```
package eg;
public class ExampleBean {

    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;

    public void setBeanOne(AnotherBean b) { beanOne = b; }
    public void setBeanTwo(YetAnotherBean b) { beanTwo = b; }
}
```

```
<bean id="exampleBean" class="eg.ExampleBean">
    <property name="beanOne"><ref bean="anotherExampleBean"/></pro
    <property name="beanTwo"><ref bean="yetAnotherBean"/></propert
</bean>

<bean id="anotherExampleBean" class="eg.AnotherBean"/>
<bean id="yetAnotherBean" class="eg.YetAnotherBean"/>
```

Example-4

- Setting Bean properties

```
package eg;
public class ExampleBean {

    private String s;
    private int i;

    public void setStringProperty(String s) { this.s = s; }
    public void setIntegerProperty(int i) { this.i = i; }
}
```

```
<bean id="exampleBean" class="eg.ExampleBean">
    <property name="stringProperty"><value>Hi!</value></property>
    <property name="integerProperty"><value>1</value></property>
</bean>
```


Example-4

- Constructor example

```
public class ExampleBean {
    private AnotherBean beanOne;
    private YetAnotherBean beanTwo;
    private int i;
    public ExampleBean(AnotherBean b1, YetAnotherBean b2, int i) {
        this.beanOne = b1;
        this.beanTwo = b2;
        this.i = i;
    }
}

<bean id="exampleBean" class="eg.ExampleBean">
    <constructor-arg><ref bean="anotherExampleBean"/></constructor-arg>
    <constructor-arg><ref bean="yetAnotherBean"/></constructor-arg>
    <constructor-arg><value>1</value></constructor-arg>
</bean>

<bean id="anotherExampleBean" class="eg.AnotherBean"/>
<bean id="yetAnotherBean" class="eg.YetAnotherBean"/>
```

Example-5

Bean Lifecycle

- Beans can be initialized by the factory before its first use

```
public class ExampleBean {
    public void init() {
        // do some initialization work
    }
}

<bean id="exampleBean" class="eg.ExampleBean"
      init-method="init"/>
```

- Beans can be cleaned up when not used anymore

```
public class ExampleBean {
    public void cleanup() {
        // do some destruction work
    }
}

<bean id="exampleBean" class="eg.ExampleBean"
      destroy-method="cleanup"/>
```


Internationalization - i18n

- ApplicationContext method
  - String getMessage (String code, Object[] args, String default, Locale loc)

Delegated to a "**messageSource**" bean

- ApplicationContext searches for the "messageSource" bean
  - Must implement MessageSource interface
- Example
  - definition of two resource bundles in classpath: messages and errors

```
<bean id="messageSource" class="...ResourceBundleMessageSource">
    <property name="basenames">
        <value>messages,errors</value>
    </property>
</bean>
```

Search in classpath:

messages_pt_BR.properties	errors_pt_BR.properties
messages_pt.properties	errors_pt.properties
messages.properties	errors.properties

The Inversion of Control (IOC) container

- Annotation-driven configuration
  - Support for a complete set of configuration annotations: @Autowired in combination with support for the JSR-250 annotations @Resource, @PostConstruct and @PreDestroy
- Autodetecting components in the classpath
  - Introduces support component scanning: autodetecting annotated components in the classpath. Typically, such component classes will be annotated with stereotypes such as @Component, @Repository, @Service, @Controller. Depending on the application context configuration, such component classes will be autodetected and turned into Spring bean definitions, not requiring explicit configuration for each such bean.
- Support for bean name pointcut element
  - introduces support for the bean(...) pointcut element, matching specific named beans according to Spring-defined bean names.

Resolving Dependencies: @Autowired

- Injection at constructor/field/method level
- Supports multi argument methods
  - Concise
- Default behavior is Spring's traditional *autowire by type*
- Annotations make autowiring much more useful

```
@Autowired
public void createTemplates(DataSource ds,
                            ConnectionFactory cf) {
  this.jdbcTemplate = new JdbcTemplate(ds);
  this.jmsTemplate = new JmsTemplate(cf);
}
```

Out-of-the-box *stereotype* annotations

- @Service
  - Identifies a stateless service
- @Repository
  - Identifies a repository (DAO)
- @Aspect
  - @AspectJ aspect
- @Controller
  - Spring MVC controller

- Can define your own…
- @Component
  - *Meta-annotation*
    - Annotate your own annotation with @Component and your classes get picked up by scanning

Component Scanning

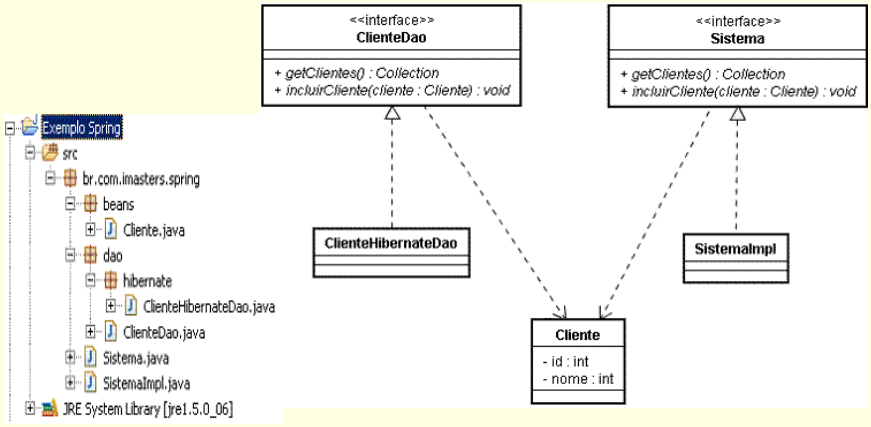
- Scans the classpath for annotated classes
- Removes the need for XML definitions unless you need to do something you can't do in annotations

```
@Service
public class DefaultAccountService { ...
```

```
<bean id="defaultAccountService"
                    class="DefaultAccountService"/>
```

Exercice

- Cadastrar e listar clientes de uma empresa fictícia



Exercice

- Cadastrar e listar clientes de uma empresa fictícia (cont.)
  - A classe SistemaImpl funciona conforme o padrão Facade, fornecendo um único caminho de entrada para o sistema, evitando que o usuário se perca na complexidade do sistema.

```
public class SistemaImpl implements Sistema {
  private ClienteDao daoCliente;
  public Collection getClientes() {
    return this.daoCliente.getClientes() ;
  }
  public void incluirCliente(Cliente cliente) {
    this.daoCliente.incluirCliente(cliente);
  }
  public ClienteDao getDaoCliente() {
    return daoCliente;
  }
  public void setDaoCliente(ClienteDao daoCliente) {
    this.daoCliente = daoCliente;
  }
}
```

Exercice

- Cadastrar e listar clientes de uma empresa fictícia (cont.)
  - O arquivo applicationContext.xml colocado no classpath da aplicação defini a configuração de dependência entre os objetos da aplicação

```
<?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
   "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
   <bean id="ClienteDao“
         class="br.com.imasters.spring.dao.hibernate.ClienteHibernateDao"/>
   <bean id="Sistema" class="br.com.imasters.spring.SistemaImpl">
       <property name="clienteDao" ref="ClienteDao"/>
   </bean>
</beans>
```


Exercice

- Cadastrar e listar clientes de uma empresa fictícia (cont.)
  - A **SistemaImpl** possui uma propriedade clienteDao que deve ser injetada com um objeto definido no bean ClienteDao

```
public class ClienteHibernateDao implements ClienteDao {
  public Collection getClientes() {
    System.out.println( "Listagem" ) ;
    Collection list = new ArrayList() ;
    Cliente c = new Cliente();
    Cliente c2 = new Cliente();
    Cliente c3 = new Cliente();
    list.add(c); list.add(c2); list.add(c3);

    return list;
  }
  public void incluirCliente(Cliente cliente) {
    System.out.println ( "Cliente Incluído" ) ;
  }
}
```

Exercice

- Cadastrar e listar clientes de uma empresa fictícia (cont.)
  - A aplicação principal ativa o container IoC para que as dependências entre os objetos possam ser resolvidas.

```
public class Aplicacao {
  public static void main ( String[] args ) {

    // XmlBeanFactory factory = new XmlBeanFactory(
    //          new FileSystemResource("applicationContext.xml"));
    //
    ApplicationContext factory = new
                ClassPathXmlApplicationContext("applicationContext.xml");

    Sistema sistema = (Sistema)factory.getBean("Sistema") ;
    sistema.incluirCliente(new Cliente());
  }
}
```