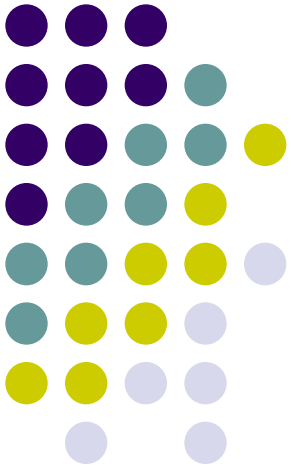
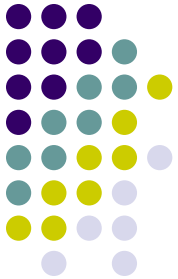
Spring

Intro Spring

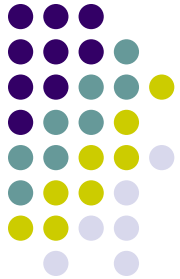


Contenidos



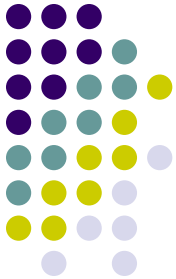
- Intro
- Spring Core
- Ejemplos

Spring

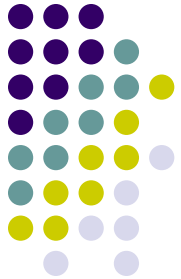


Spring es un framework de código abierto que permite desarrollar aplicaciones java muy bien documentadas, de forma fácil y rápida, utilizando las mejores practicas existentes en la industria. Es una excelente alternativa clara al desarrollo JEE.

Ecosistema Spring

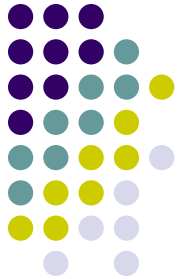


- **Web**: MVC, Rest
- **Data Access**: Hiberna, JDBC, NoSQL
- **Integracion**: WebService, JMS, Batch, Integration, etc
- **Mobile**: Android, IPhone, BB, REST API
- **Social**: Facebook, Twitter, Linkedin
- **Security**
- **Cloud**



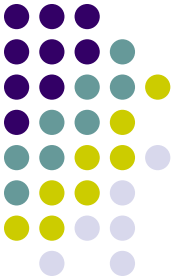
¿Qué ganamos utilizando Spring?

IOC



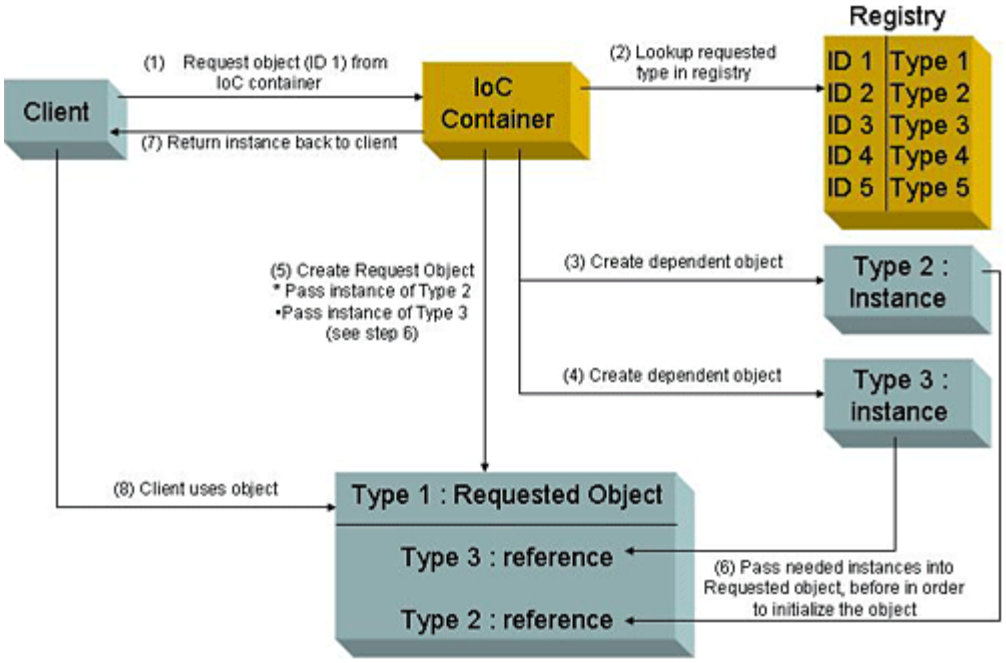
Un potente contenedor IoC (contexto) para gestionar el ciclo de vida de nuestros objetos utilizando los principios de Inversión de Control. Esto hace que el desarrollo y la configuración de nuestras aplicaciones sea rápida y sencilla.

IOC



En forma muy resumida, el objetivo del contenedor IoC es encargarse de instanciar los objetos de nuestro sistema, denominados beans, y asignarle sus dependencias. Para que el contenedor pueda llevar a cabo esta tarea, debemos, mediante información de configuración, indicarle dónde se encuentran dichos beans.

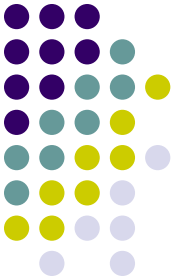
IOC


Registry
ID 1 Type 1
ID 2 Type 2
ID 3 Type 3
ID 4 Type 4
ID 5 Type 5
Client
IoC Container
(1) Request object (ID 1) from IoC container
(2) Lookup requested type in registry
(7) Return instance back to client
(3) Create dependent object
(4) Create dependent object
(5) Create Request Object
* Pass instance of Type 2
•Pass instance of Type 3
(see step 6)
Type 2 : Instance
Type 3 : instance
(8) Client uses object
Type 1 : Requested Object
Type 3 : reference
Type 2 : reference
(6) Pass needed instances into Requested object, before in order to initialize the object

Transacciones

Una capa genérica de abstracción para la gestión de transacciones; haciendo sencilla la demarcación de las mismas sin tratarlas a bajo nivel. El soporte de transacciones de *Spring* no está atado a entornos J2EE.

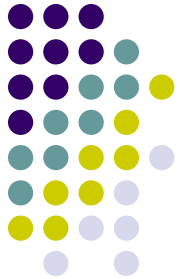
JDBC



Una capa de abstracción JDBC simple de utilizar que ofrece una significativa jerarquía de excepciones evitando la captura de los códigos de error SQL directamente.

Mediante la utilización de helpers también reduce considerablemente el tamaño de nuestro código al trabajar con JDBC.

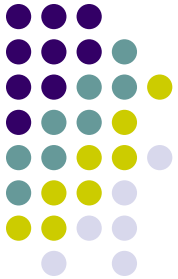
Hibernate



Integración con *Hibernate*, JDO e iBatis en términos de soporte (DAOs) y gestión de transaccione.

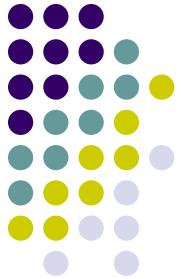
Especial soporte a *Hibernate* añadiendo convenientes características *IoC, configuración*

AOP



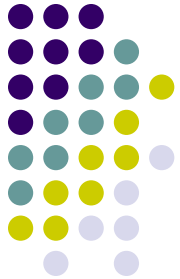
Funcionalidad AOP, totalmente integrada en la gestión de configuración de *Spring*. Se puede aplicar AOP a cualquier objeto gestionado por *Spring*, añadiendo aspectos como gestión de transacciones declarativa.

JMS



Integración con sistemas de mensajería mediante JMS. Tanto para el envió como la recepción de mensajes desde COLAS y/o TOPICOS.

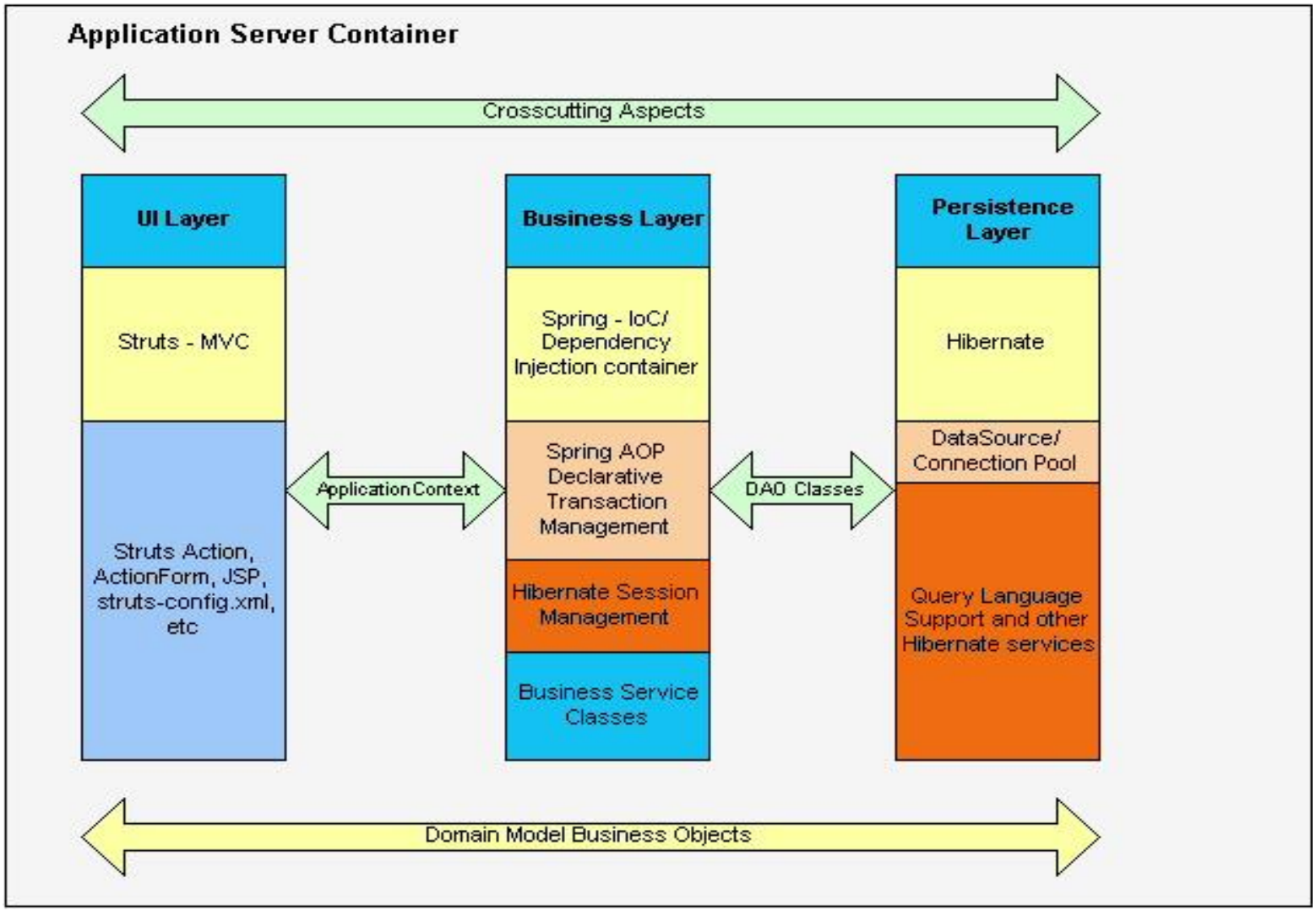
MVC

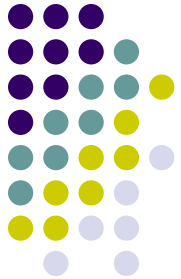


Un framework MVC + REST, construido sobre el núcleo de *Spring*. Este framework es altamente configurable y permite el

uso de múltiples tecnologías para las vistas (JSP, Velocity, Tiles, iText). Integracion con otras herramientas web como Tapestry y Struts 1 - 2.

Forma de trabajo



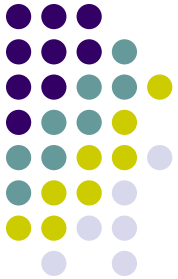


Spring con …..

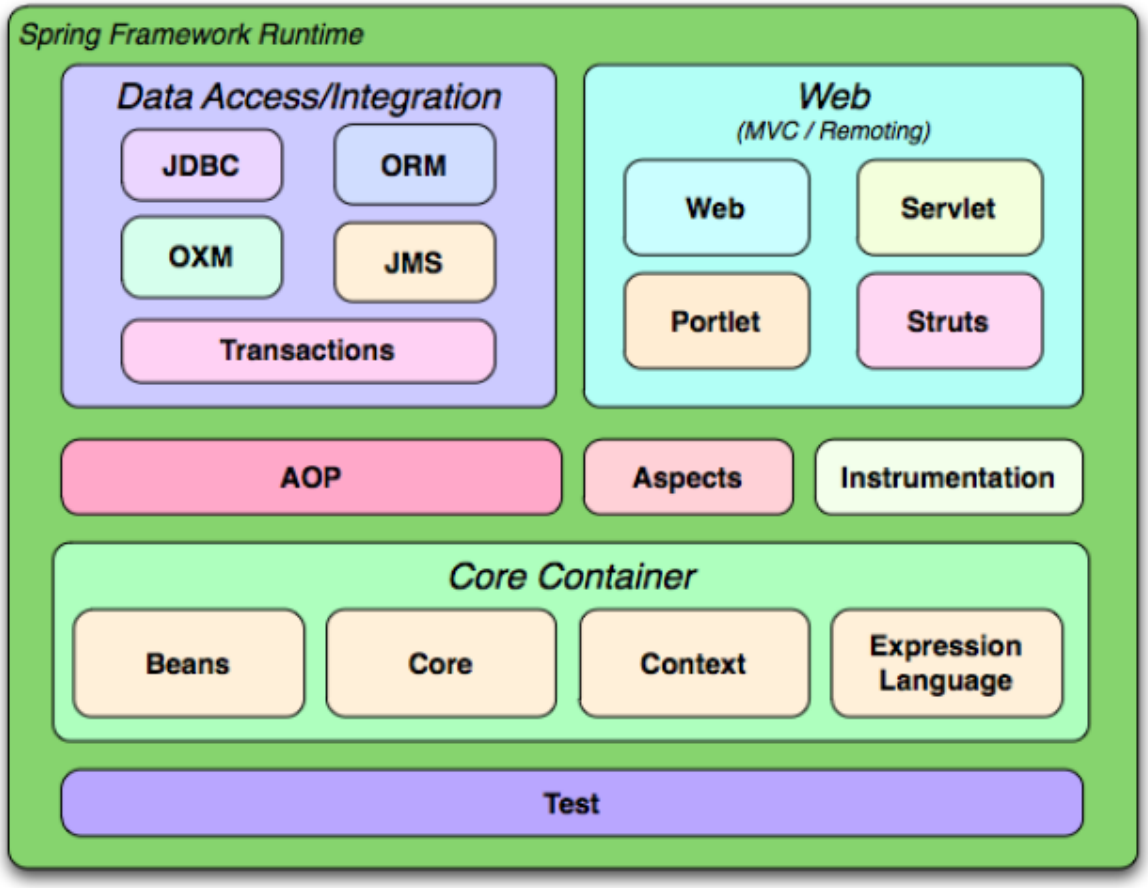
- Struts + Spring + Hibernate
- Struts + Spring + EJB
- JavaServer Faces + Spring + iBATIS
- Spring + Spring + JDO
- Flex + Spring + Hibernate
- Struts + Spring + JDBC
- Spring MVC + Spring + Hibernate
- Grails + Spring + ...

Otros

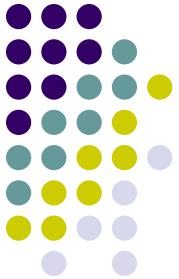
- Envió de mails
- Gestión de hilos async
- Scheduling
- Pool de objetos
- Integración con JMX
- Testing
- WebServices / RMI
- Validación de objetos de negocio
- Batch
- Social
- ……



Módulos de Spring

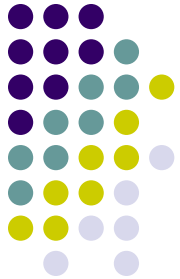


Configuración



- Bajar los .JARs según módulos.
- Seleccionar el método de configuración (xml, annotations, mixto)
- Definir beans
- Listo!!!

Ejemplo - XML



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

<bean id="..." class="...">…</bean>
<bean id="..." class="...">…</bean>

</ beans>
```

ApplicationContext context =
new ClassPathXmlApplicationContext(“xyz.xml");

Ejemplo - XML

….

<bean id="**accountDao**“ class=“com.xyz.SqlMapAccountDao">
</bean>

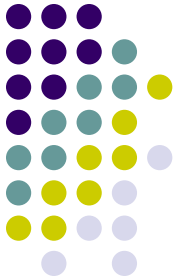
<bean id="**itemDao**" class="com.xyz..SqlMapItemDao">
</bean>

……..

Ejemplo - XML

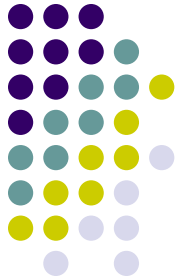
```
<beans>

   <import resource="services.xml"/>
   <import resource="daos.xml"/>
   <import resource="resources/messageSource.xml"/>

   <bean id="bean1" class="..."/>
   <bean id="bean2" class="..."/>

</beans>
```

Dependency Injection - Constructor



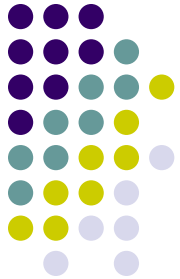
```
package x.y;
public class Foo {
  public Foo(Bar bar, Baz baz) {  // ...   }
}

<beans>
  <bean id="foo" class="x.y.Foo">
     <constructor-arg ref="bar"/>
     <constructor-arg ref="baz"/>
  </bean>

  <bean id="bar" class="x.y.Bar"/>
  <bean id="baz" class="x.y.Baz"/>

</beans>
```

Dependency Injection - Constructor



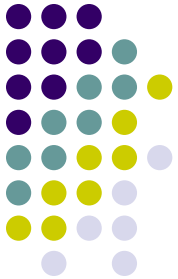
```
<bean id="exampleBean" class="examples.ExampleBean">
  <constructor-arg type="int" value="7500000"/>
  <constructor-arg type="java.lang.String" value="42"/>
</bean>

<bean id="exampleBean" class="examples.ExampleBean">
  <constructor-arg index="0" value="7500000"/>
  <constructor-arg index="1" value="42"/>
</bean>
```

Dependency Injection - Constructor



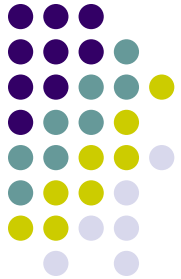
```
public class ExampleBean {
  private AnotherBean beanOne;
  private YetAnotherBean beanTwo;
  private int i;

  public ExampleBean(
    AnotherBean anotherBean, YetAnotherBean yetAnotherBean,
int i) {
    this.beanOne = anotherBean;
    this.beanTwo = yetAnotherBean;
    this.i = i;
  }
}
```

Dependency Injection - Constructor



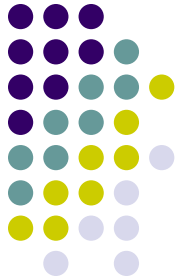
```
<bean id="exampleBean" class="examples.ExampleBean">
  <constructor-arg>
    <ref bean="anotherExampleBean"/>
  </constructor-arg>
  <constructor-arg ref="yetAnotherBean"/>
  <constructor-arg type="int" value="1"/>
</bean>

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

Dependency Injection - Setters



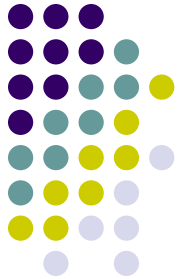
```
public class ExampleBean {
  private AnotherBean beanOne;
  private YetAnotherBean beanTwo;
  private int i;

  public void setBeanOne(AnotherBean beanOne) {
    this.beanOne = beanOne;
  }
  public void setBeanTwo(YetAnotherBean beanTwo) {
   this.beanTwo = beanTwo;
  }
  public void setIntegerProperty(int i) {  this.i = i;   }
}
```

Dependency Injection - Setters



```
<bean id="exampleBean" class="examples.ExampleBean">
   <property name="integerProperty" value="1"/>
   <property name="beanOne">
      <ref bean="anotherExampleBean"/>
  </property>
  <property name="beanTwo" ref="yetAnotherBean"/>
</bean>

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

Dependency Injection - Setters



```
<bean id="fileNameGenerator"
class="com.common.FileNameGenerator">
  <property name="name"><value>test</value></property>
  <property name="type"><value>txt</value></property>
</bean>

<bean id="FileNameGenerator"
class="com.common.FileNameGenerator">
  <property name="name" value="test" />
  <property name="type" value="txt" />
</bean>

<bean id="FileNameGenerator"
class="com.common.FileNameGenerator" p:name="test" p:type="txt" />
```

Inner Beans

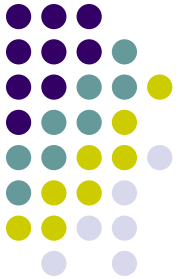
```
<bean id="outer" class="...">
  <property name="target">
    <bean class="com.example.Person">
      <property name="name" value="Fiona Apple"/>
      <property name="age" value="25"/>
    </bean>
 </property>
</bean>
```

Ojo que los inner beans son solo para usar dentro de una propiedad determinada y su scope es prototype

Collections

```
<bean id="moreComplexObject" class="example.ComplexObject">
  <property name="someList">
    <list>
      <value>Juan</value>
      <value>Pedro</value>
    </list>
    <property name="someSet">
      <set>
        <ref bean="myDataSource1" />
        <ref bean="myDataSource2" />
      </set>
</property>
</bean>
```

Lifecycle callbacks

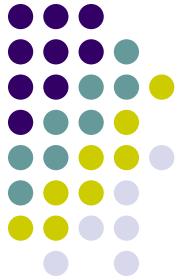


```
public class ExampleBean {
  public void init() {
     …
  }
}
```

```
<bean id="exampleInitBean" class="examples.ExampleBean" init-method="init"/>
```

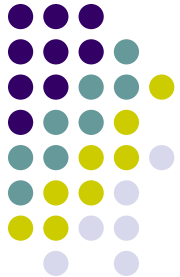
```
<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="closeX"/>
```

Bean scopes



- Singleton
- Prototype
- Request
- Session

Singleton

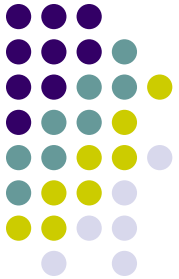


Por defecto todos los beans son **singleton** y se crean al iniciar el contexto

<bean id="accountService" class="com.foo.DefaultAccountService"/>

<bean id="accountService" class="com.foo.DefaultAccountService" scope="**singleton**"/>

<bean id="lazy" class="com.foo.ExpensiveToCreateBean" **lazy-init="true"**/>



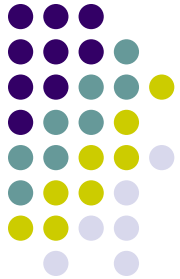
1 - Ejercicio

1. Crear una clase cuenta bancaria con numero, saldo y notas
2. Configurar una cuenta enviando la información mediante setters (probar recuperar un conjunto de cuentas como singleton y como prototype)
3. Modificar la cuenta bancaria para que los parámetros sean enviados mediante un constructor (probar recuperar cuentas)
4. Crear una clase banco que contenga nombre y una lista de cuentas (implementar métodos de creación y destrucción que impriman el estado general de banco, nombre, cantidad de cuentas y saldo total)
5. Configurar el banco para que tenga un nombre y un conjunto de cuentas (singleton + lazy)
6. Crear una interfaz para remplazar la cuenta bancaria
7. Crear una caja de ahorro y cuenta corriente que implementes la interfaz creada anteriormente (probar recuperar diferentes tipos utilizando la interfaz)
8. Modificar el banco para que tenga diferentes cuentes sin importar el tipo

Dates

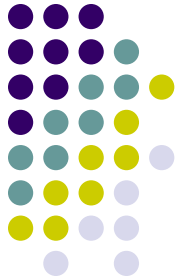
```
<bean id="dateFormat" class="java.text.SimpleDateFormat">
    <constructor-arg value="yyyy-MM-dd" />
</bean>

<bean id="customerUtils" class="com.common.CustomerUtils">
    <property name="date">
        <bean factory-bean="dateFormat" factory-method="parse">
            <constructor-arg value="2011-12-31" />
        </bean>
    </property>
</bean>
```

Static factory method



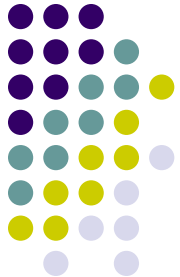
```
<bean id="clientService"
class="examples.ClientService"
factory-method="createInstance"/>

---------------------------------

public class ClientService {
  private static ClientService clientService = new ClientService();
  private ClientService() {}
  public static ClientService createInstance() {
    return clientService;
  }
}
```

Instance factory method



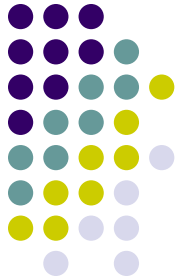
```
<bean id="serviceLocator" class="examples.DefaultServiceLocator">
</bean>
<bean id="clientService“ factory-bean="serviceLocator"
factory-method="createClientServiceInstance"/>

------------------------------

public class DefaultServiceLocator {
  private static ClientService clientService = new ClientServiceImpl();

  public ClientService createClientServiceInstance() {
    return clientService;
  }
….
```

Instance factory method



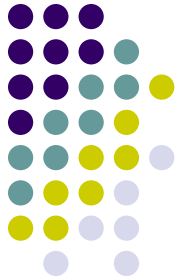
```
<bean id="serviceLocator" class="examples.DefaultServiceLocator">
</bean>

<bean id="clientService"
  factory-bean="serviceLocator"
  factory-method="createClientServiceInstance"/>

<bean id="accountService"
  factory-bean="serviceLocator"
  factory-method="createAccountServiceInstance"/>
```

Instance factory method



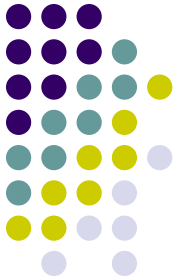
```
public class DefaultServiceLocator {
  private static ClientService clientService = new ClientServiceImpl();
  private static AccountService accountService = new
  AccountServiceImpl();

  public ClientService createClientServiceInstance() {
    return clientService;
  }

  public AccountService createAccountServiceInstance() {
    return accountService;
  }
}
```

Depends-on



```
<bean id="beanOne" class="ExampleBean" depends-on="manager"/>

<bean id="manager" class="ManagerBean" />

<bean id="beanOne" class="ExampleBean" depends-on="manager,
accountDao">
<property name="manager" ref="manager" />
</bean>

<bean id="manager" class="ManagerBean" />

<bean id="accountDao" class="x.y.jdbc.JdbcAccountDao" />
```

Method injection



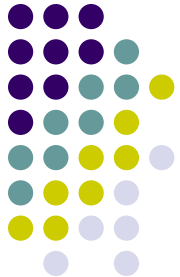
```
import org.springframework.beans.*;
import org.springframework.context.*;
public class CommandManager implements ApplicationContextAware {
  private ApplicationContext applicationContext;

  public Object process(Map commandState) {
    Command command = createCommand();
    command.setState(commandState);
    return command.execute();
  }

 protected Command createCommand() {
   return this.applicationContext.getBean("command", Command.class);
 }
 public void setApplicationContext(ApplicationContext applicationContext) {
   this.applicationContext = applicationContext;
 }
}
```

Lookup method injection

```
public abstract class CommandManager {
  public Object process(Object commandState) {
    Command command = createCommand();
    command.setState(commandState);
    return command.execute();
  }

  protected abstract Command createCommand();
}
```

```
<bean id="command" class="xxxx.AsyncCommand" scope="prototype">
</bean>

<bean id="commandManager" class="xxxxx.CommandManager">
<lookup-method name="createCommand" bean="command"/>
</bean>
```

PropertyPlaceholderConfigurer



```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
   <property name="locations" value="classpath:com/foo/jdbc.properties"/>
</bean>
<bean id="dataSource" destroy-method="close"
   class="org.apache.commons.dbcp.BasicDataSource">
  <property name="driverClassName" value="${jdbc.driverClassName}"/>
  <property name="url" value="${jdbc.url}"/>
  <property name="username" value="${jdbc.username}"/>
  <property name="password" value="${jdbc.password}"/>
</bean>
```

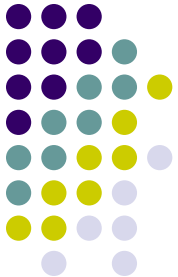
jdbc.driverClassName=org.hsqldb.jdbcDriver
jdbc.url=jdbc:hsqldb:hsql:*//production:9002*
jdbc.username=sa
jdbc.password=root

Spring autowired

```
package com.common;
public class Customer {
  private Person person;


  public void setPerson(Person person) {
    this.person = person;
  }

}
```

```
<bean id="customer" class="com.common.Customer">
  <property name="person" ref="person" />
</bean>

<bean id="person" class="com.common.Person" />
```

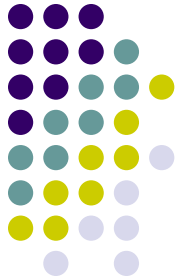
Spring autowired



<bean id="customer" class="com.common.Customer" autowire="byName" />
<bean id="person" class="com.common.Person" />

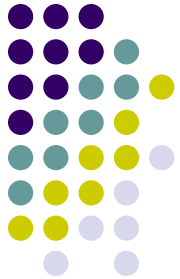
<bean id="customer" class="com.common.Customer" autowire="byType" />
<bean id="person" class="com.common.Person" />

2 - Ejercicio



1. Modificar el banco para que contenga métodos **factory** para poder crear cuentas del tipo CA y CC. (lugo modificar la clase para que mediante un solo método y un parámetro pueda realizar la misma tarea – idem service locator).
2. Crear un GerstorGeneralDeCreditos y mediante setters + autowired inyectarle un Banco (probar que la misma operación pero mediante un constructor).
3. Agregar un método al GerstorGeneralDeCreditos que calcule el monto dtotal a pagar por un préstamo de X $. (Leer la taza de interés mediante un PropertyPlaceholder)

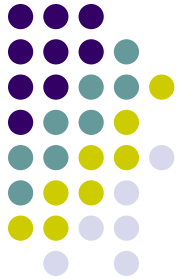
Spring annotations



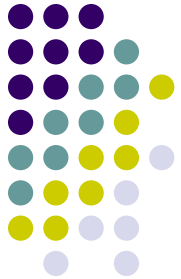
```
public class MovieRecommender {
  @Autowired
  @Qualifier(“mCatalog")
  private MovieCatalog movieCatalog;
  private CustomerPreferenceDao customerPreferenceDao;

 @Autowired
  public MovieRecommender(CustomerPreferenceDao customerPreferenceDao) {
   this.customerPreferenceDao = customerPreferenceDao;
  }
  private MovieFinder movieFinder;

  @Autowired(required=false)
  public void setMovieFinder(MovieFinder movieFinder) {
    this.movieFinder = movieFinder;
   }
 ...
}
```

Spring annotations

```
public class SimpleMovieLister {

   private MovieFinder movieFinder;

   @Resource
   public void setMovieFinder(MovieFinder movieFinder) {
     this.movieFinder = movieFinder;
   }

}
```



Spring annotations



```
public class CachingMovieLister {

  @PostConstruct
  public void populateMovieCache() {

  }

  @PreDestroy
  public void clearMovieCache() {
  }

}
```

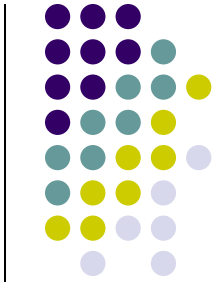
Spring annotations

@Component

@Repository

@Service

@Controller



Spring annotations

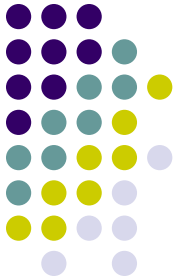
<context:component-scan base-package=*"edu.curso.java.spring" />*

<context:component-scan base-package="com.xxxx" **>**

<context:include-filter type="regex" expression="com.xxxx.dao.*DAO.*" **/>**

<context:exclude-filter type="regex" expression="com.xxx.service.*Ventas.*" **/>**

Spring annotations

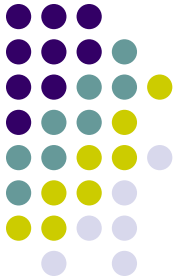


@Component
@Scope(“prototype”)
public class MyUtilXXX {
}

@Service("myMovieLister")
public class SimpleMovieLister {
}

@Repository
public class MovieFinderImpl implements MovieFinder {
}

Spring annotations

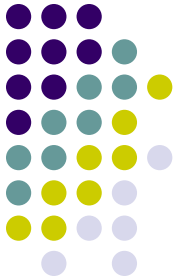


```
@Configuration
public class AppConfig {

    @Bean(name="myService")
    public MyService myService() {
        return new MyServiceImpl();
    }

}
```

<context:annotation-config/>

```
<beans>
<bean id="myService" class="com.acme.services.MyServiceImpl"/>
</beans>
```

Spring annotations



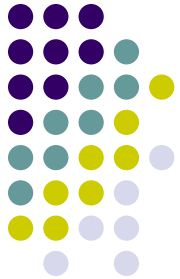
```
public static void main(String[] args) {

ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
MyService obj = (MyService ) context.getBean("myService ");
…..

}
```

Spring annotations

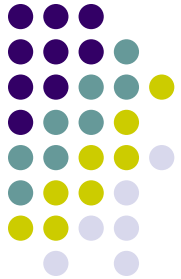
@Configuration

@Import({ CustomerConfig.class, SchedulerConfig.class })

public class AppConfig { …… }



3 - Ejercicios



Modificar el ejercicio 1 para que trabaje mediante annotations

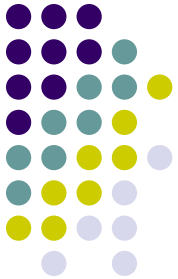
Spring EL

Spring Expression Language (SpEL) es un potente lenguaje de expresiones que permite manipular y realizar consultas sobre objetos en tiempo de ejecución. Su sintaxis es similar a EL de JSTL / JSF.

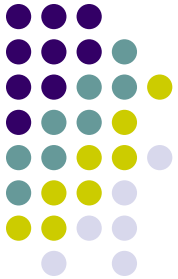
EL es utilizado en muchos otros frameworks de la familia de Spring. Ej Spring Security

Spring EL

```
public class Customer {
  private Item item;
  private String itemName;
...

public class Item {
  private String name;
  private int qty;
...

<bean id="itemBean" class="com.xxx.Item">
  <property name="name" value="itemA" />
  <property name="qty" value="10" />
</bean>

<bean id="customerBean" class="com.xxx.Customer">
  <property name="item" value="#{itemBean}" />
  <property name="itemName" value="#{itemBean.name}" />
</bean>
```

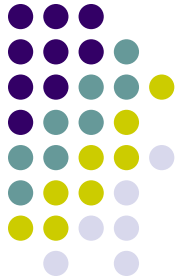


Spring EL



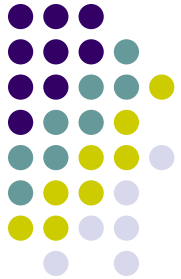
```
@Component("customerBean")
public class Customer {

    @Value("#{itemBean}")
    private Item item;

    @Value("#{itemBean.name}")
    private String itemName;

}
```

Spring EL



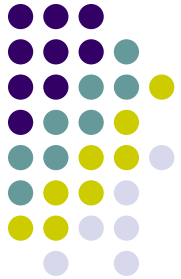
```
@Component("customerBean")
public class Customer {
  @Value("#{addressBean}")
  private Address address;

  @Value("#{addressBean.country}")
  private String country;

  @Value("#{addressBean.getFullAddress(‘general')}")
  private String fullAddress;
```

Spring EL



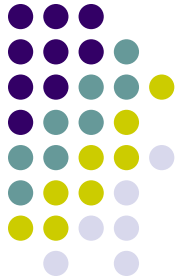
```
@Value("#{1 == 1}") //true
private boolean testEqual;

@Value("#{1 != 1}") //false
private boolean testNotEqual;

@Value("#{1 < 1}") //false
private boolean testLessThan;

@Value("#{1 <= 1}") //true
private boolean testLessThanOrEqual;

@Value("#{1 + 1}")
private double testAdd;
```

Spring EL



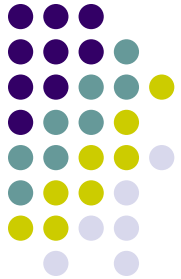
```
@Value("#{numberBean.no == 999 and numberBean.no < 900}") //false
private boolean testAnd;

@Value("#{itemBean.qtyOnHand < 100 ? true : false}")
private boolean warning;

@Value("#{testBean.map['MapA']}")
private String mapA;

@Value("#{testBean.list[0]}")
private String list;
```

Spring EL



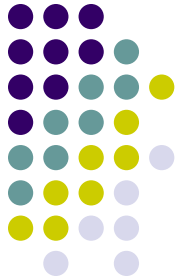
```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("new String('hello world').toUpperCase()");
String message = exp.getValue(String.class);

----------------------------------------------------------------------------------------------

boolean falseValue = parser.parseExpression("2 < -5.0").getValue(Boolean.class);

----------------------------------------------------------------------------------------------

String expression = "isMember('Nikola Tesla') and !isMember('Mihajlo Pupin')";
```

Spring EL



```
public class StringUtils {

  public static String reverseString(String input) {
    StringBuilder backwards = new StringBuilder();
    for (int i = 0; i < input.length(); i++)
      backwards.append(input.charAt(input.length() - 1 - i));
    }
    return backwards.toString();
  }
}
```

Spring EL



```
ExpressionParser parser = new SpelExpressionParser();
StandardEvaluationContext context = new StandardEvaluationContext();

context.registerFunction("reverseString",
                StringUtils.class.getDeclaredMethod("reverseString",
                                new Class[] { String.class }));

String helloWorldReversed =
        parser.parseExpression("#reverseString('hello')").getValue(context,
String.class);
```