



FORMACIÓN JAVA

SPRING, HIBERNATE Y JUNIT

Índice

Módulo 1 - Introducción a Spring

Módulo 2 - Configuración de Spring

Módulo 3 - Inyección de Dependencias

Módulo 4 - POA

Módulo 5 - Spring JDBC

Módulo 6 - Introducción a Hibernate

Módulo 7 - Introducción a test unitarios

</ Índice

Módulo 1 - Introducción a Spring

Introducción a Spring framework

Conceptos Básicos en Spring

Módulos en Spring

Arquitectura Multicapas

Instalación de Spring

</ INTRODUCCIÓN A SPRING

¿Qué es un *framework*?

Es un programa “hecho a medias”, al cual yo le puedo añadir mis piezas, adecuándome al estilo que el programa base espera y construir así mi aplicación, ajustada a mis necesidades



</ INTRODUCCIÓN A SPRING

¿Por qué usar *framework*?

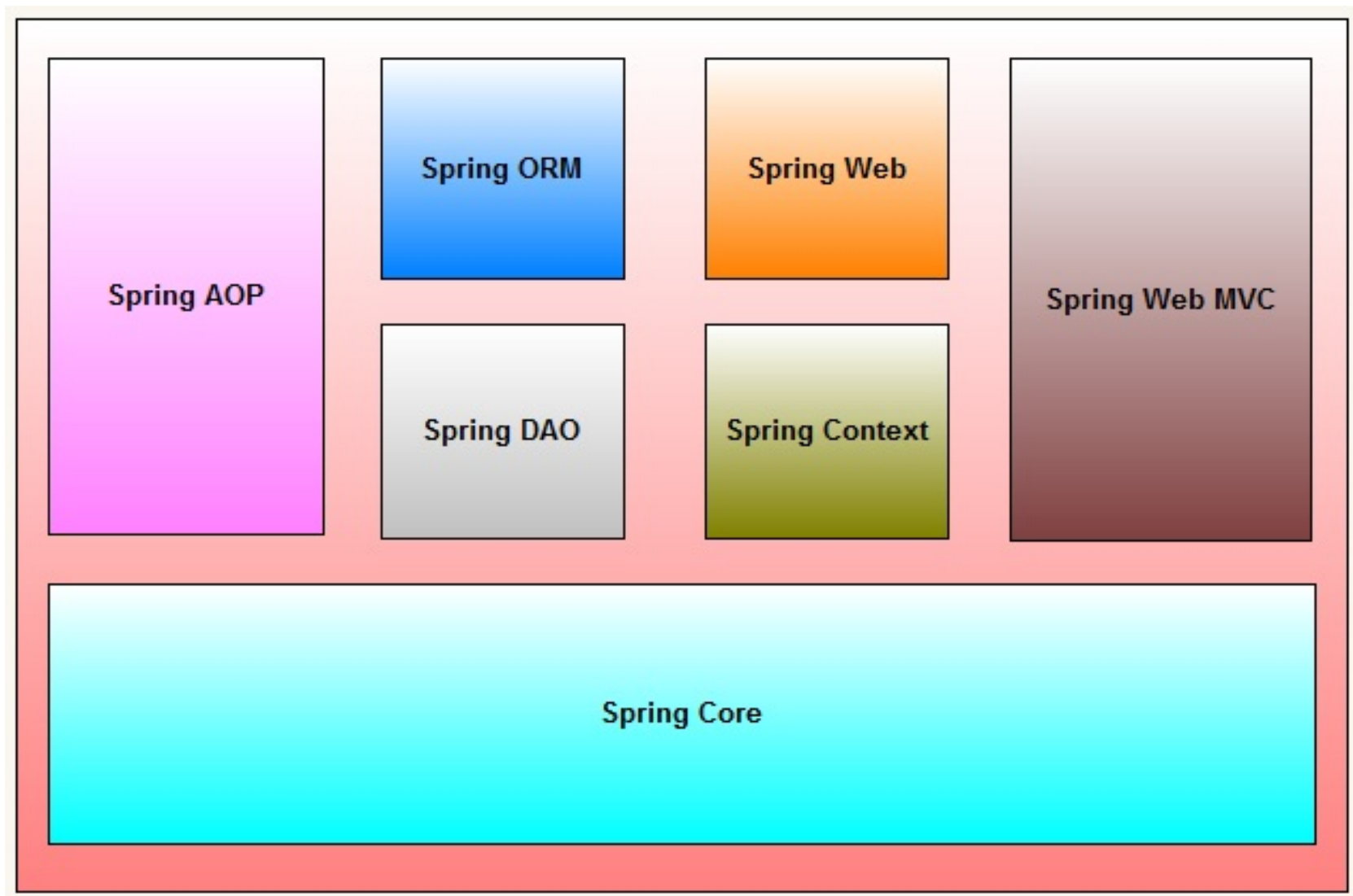
- Soluciones a problemas comunes
- Rapidez en el desarrollo
- Robustez : tecnología testada
- Productividad

</ MÓDULOS EN SPRING

Cada módulo de Spring agrupa una serie de librerías enfocadas a ofrecer soluciones sobre un área común.

La analogía al concepto de módulo manejado por Spring sería al de paquete en nuestra aplicación (seguridad, web, acceso a datos, trabajos por lotes, etc...)

</ MÓDULOS EN SPRING



</ CONCEPTOS BÁSICOS

Los conceptos que subyacen al uso de Spring, con independencia del módulo a emplear, son:

- CONTEXTO
- BEAN
- INVERSIÓN DE CONTROL
- INYECCIÓN DE DEPENDENCIAS

</ VERSIONES Y FUENTES

En la práctica, Spring está constituido por cientos de bibliotecas java (JAR's), disponibles en **MAVEN**

La última versión estable (GA) es la 5.08. siendo la anterior la **4.3.18** y esta su **API**

</ VERSIONES Y FUENTES

Una característica estrella de Spring es que dada la sencillez de sus principios, es una herramienta que puede emplearse en **CUALQUIER ENTORNO JAVA**, ya sea web, standalone, batch, microservicios, etc..

</ ARQUITECTURA MULTI-CAPA

Spring cuenta con soporte para clasificar nuestras clases según su funcionalidad dentro del proyecto.

De cualquier modo, es responsabilidad del diseñador implementar su solución correctamente.

</ Índice

Módulo 2 - Configuración de Spring

Concepto de Beans

Configuración de Beans en Spring

Instanciar un Bean con Spring

</ ARRANQUE Y CONFIGURACIÓN DE SPRING

Spring se gestiona siempre dentro de un proyecto MAVEN, bien partiendo de algún arquetipo o declarando y añadiendo las dependencias en el POM según sea necesario.

</ ARRANQUE Y CONFIGURACIÓN DE SPRING

XML VS JCONFIG

La configuración de Spring es un aspecto esencial en su funcionamiento. Ésta, se puede detallar mediante un XML externo o bien mediante **anotaciones** (@) en las propias clases

</ ARRANQUE Y CONFIGURACIÓN DE SPRING

La información declarada en la configuración, será procesada al inicio y sirve para construir la instancia de Spring sobre la que se desplegará nuestra aplicación.

Se puede cargar de forma explícita o declarativa.

</ ARRANQUE Y CONFIGURACIÓN DE SPRING

Todas la clases referidas en la configuración que Spring se encarga de instanciar son llamados Beans

```
<bean  
id="persona" class="edu.val.course.Persona">  
  <property  
name="nombre" value="Ramón" />  
</bean>
```


</ ARRANQUE Y CONFIGURACIÓN DE SPRING

Todos los beans instanciados y cargados en memoria constituyen el contexto de Spring.



</ ANOTACIONES PARA BEANS JCONFIG

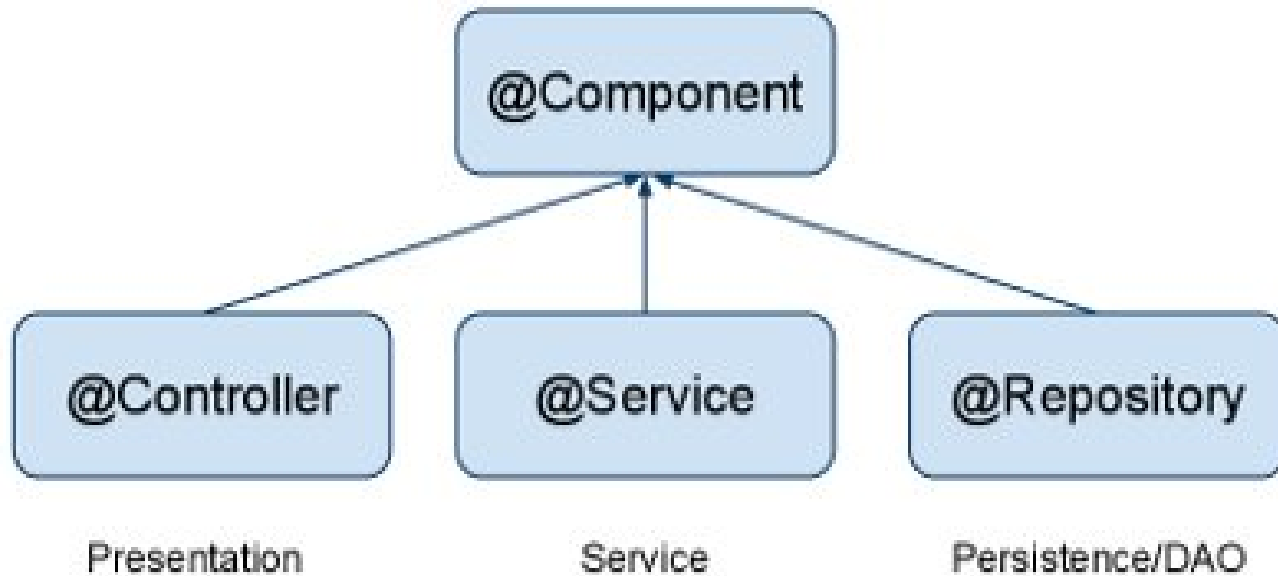
@Component

@Controller

@Service

@Repository

</ ANOTACIONES PARA BEANS JCONFIG



</ Índice

Módulo 3 - Inyección de Dependencias

Inversion of Control (IoC)

Inyecciones de Dependencias con Spring

Tipos de Inyecciones de Dependencias

Autodetección de Dependencias (autowired)

Ejemplos de Inyecciones de Dependencias

</ LA INVERSIÓN DE CONTROL (IOC)

Mediante la inversión de control, las clases declaradas en la configuración son instanciadas por el propio Spring.

No seré yo por tanto el que haga `new Clase ()` invocando al constructor explícitamente. Si no que será Spring el que instancie todos los beans por mí.

</ DI CONCEPTO DE DEPENDENCIA

Cuando una clase A emplea a otra clase B para acometer una tarea, decimos que A depende de B.

Este diseño se modelaría en JAVA por **composición**:

```
public class A  
    private B object
```



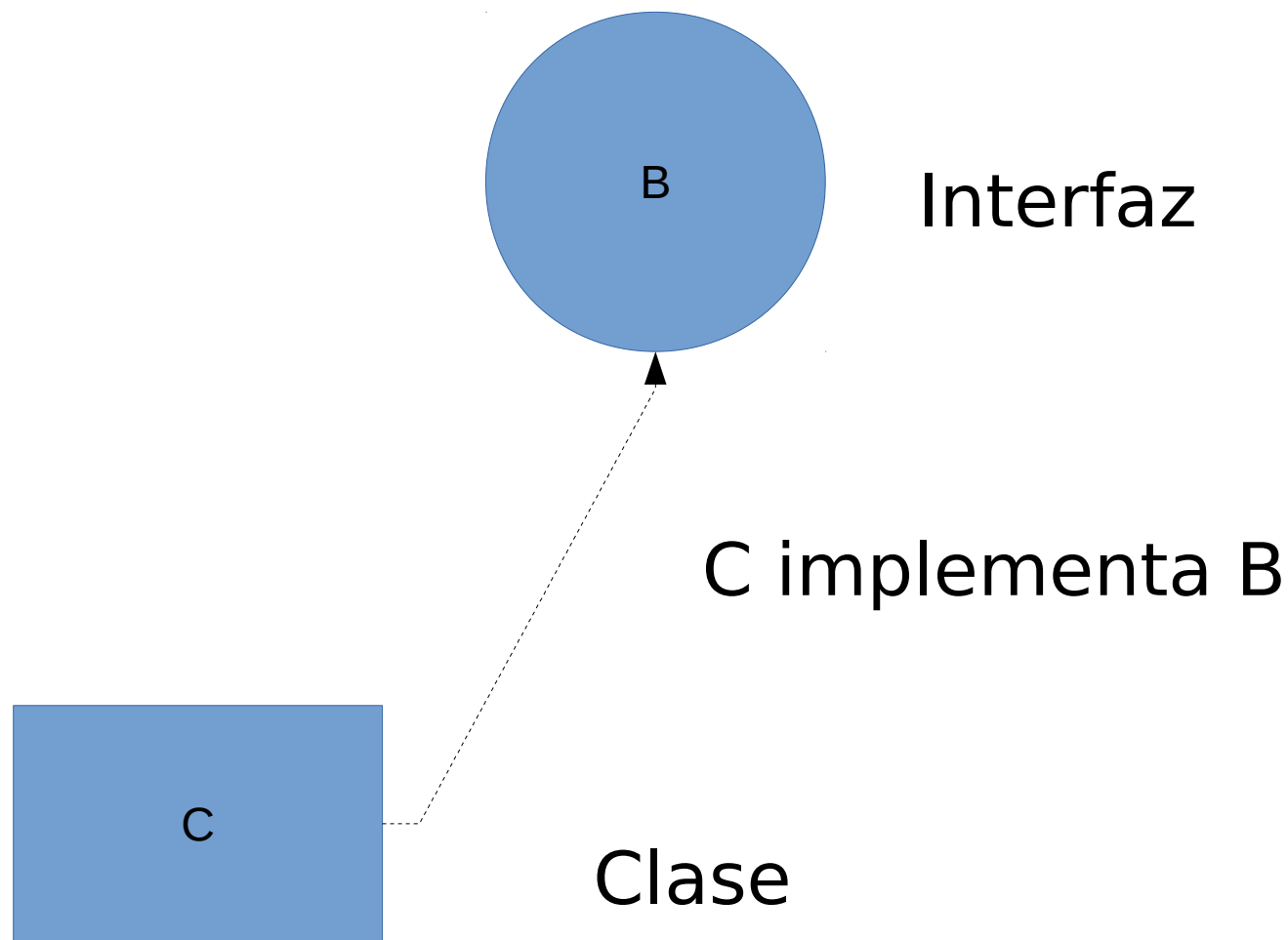
LA INTERFAZ COMO CLAVE EN LA DI

Al buen programador, le interesa crear clases independientes entre si, de modo que puedan reutilizarse y ser flexibles ante cambios.

Si B hace algo (Servicio) lo más apropiado sería modelar a B como una interfaz y dar una implementación, separando así el QUÉ del CÓMO

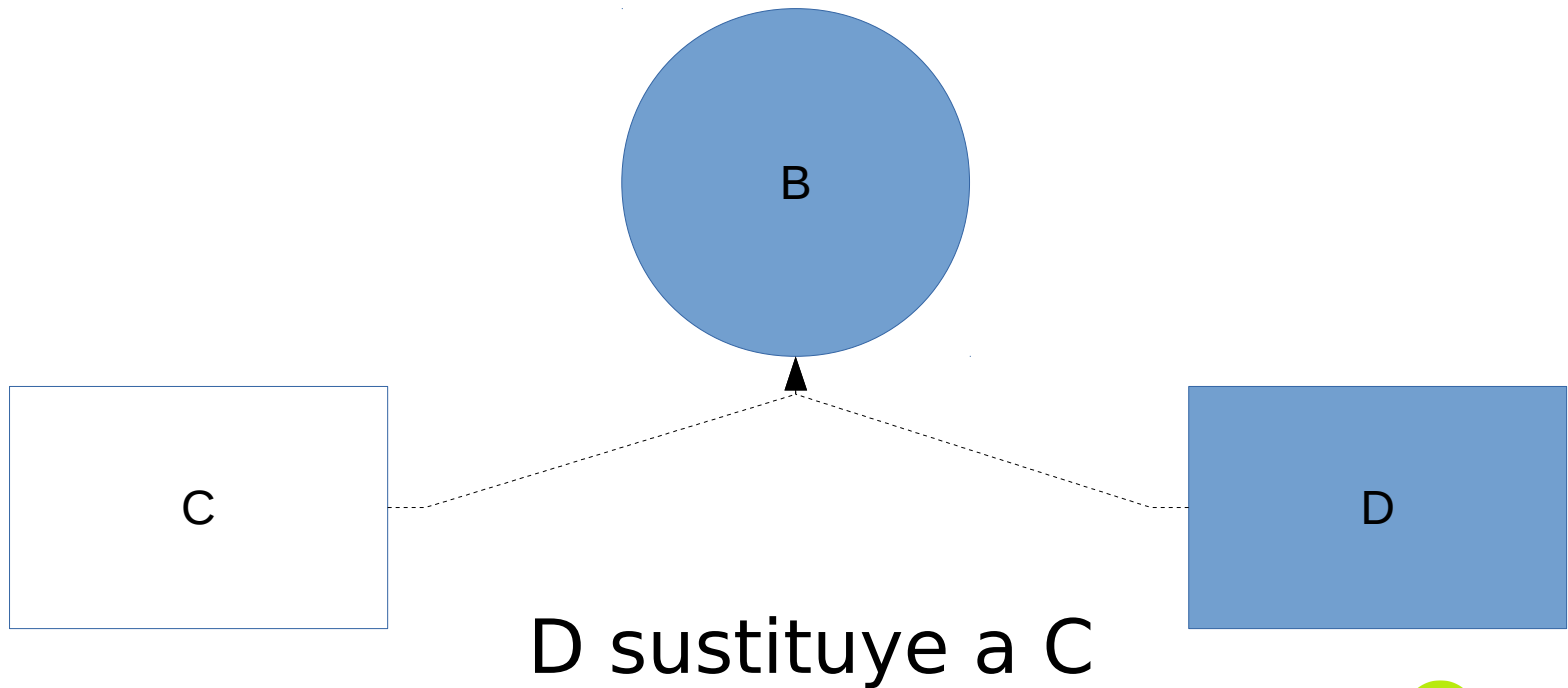


LA INTERFAZ COMO CLAVE EN LA DI



</ LA INTERFAZ COMO CLAVE EN LA DI

Si mañana cambian los requisitos, la implementación de B puede cambiar, por lo que habría que realizar otra clase, D.





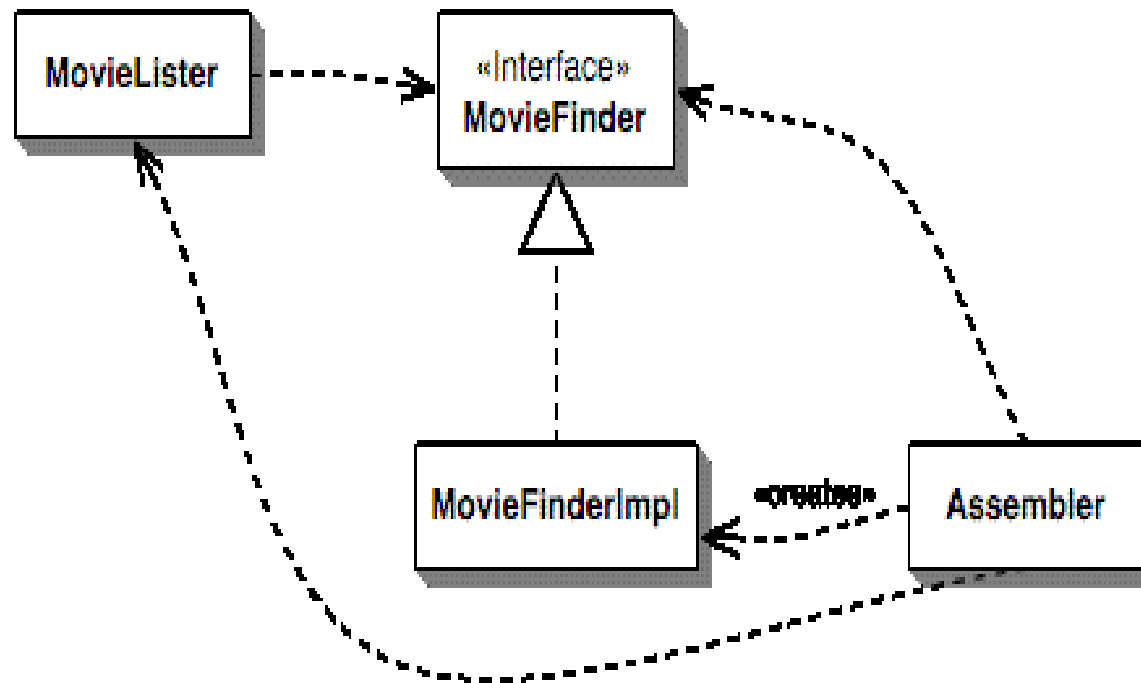
LA INYECCIÓN DE DEPENDENCIAS

Y aquí interviene la DI. Cuando Spring instancie a A, le asignará al servicio B el valor concreto de C o D, según quede definido en la configuración. Se inyectará dinámicamente un valor u otro.

Consigo así modificar el Sistema con el mínimo impacto

</ LA INYECCIÓN DE DEPENDENCIAS

Ejemplo buscador de películas





TIPOS DE DI

Según la configuración

- Anotaciones
- XML

Según el método

- Vía constructor
- Vía *setter*



Configuración

```
<context:component-scan  
base-package="edu.curso.clases" />
```

@Autowired

@Inject

</ Índice

Módulo 4 - Programación Orientada a Aspectos (AOP)

Concepto de AOP

Concepto Proxy Dinámico y otros conceptos

Usos de AOP

Ejemplos de AOP

</ POA

La programación orientada a aspectos es una de las características clave de Spring.

Aplicándola, puedo conseguir diseños más optimizados: menos código y mejor encapsulación de mi sistema

Es un gran desconocido y está infrautilizada

Es un paradigma construido sobre el de POO



POA USOS

Cuando hay una funcionalidad transversal, el uso de la POA se hace idóneo.

LOG, GESTION EXCEPCIONES

También cuando hay una tarea que de forma repetida se hace antes y/o después de otra

TRANSACCIONES EN BASE DE DATOS



POA Conceptos Clave

ASPECTO: La clase que estará relacionada con muchas otras y que representa un “aspecto” de las demás.

JOINPOINT: El punto de nuestro código fuente donde es posible la ejecución del aspecto



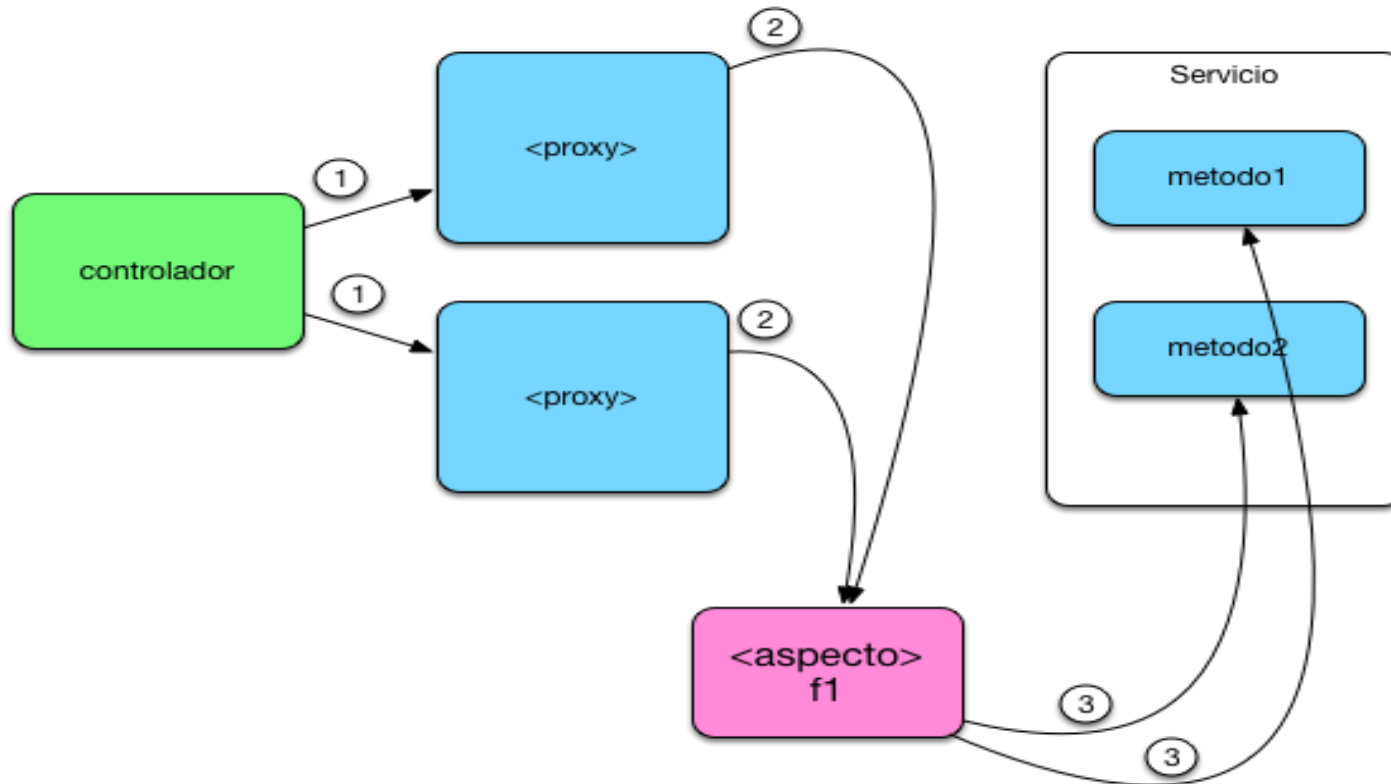
POA Conceptos Clave

POINTCUT: Define la intersección del Aspecto con un JOINPOINT

ADVICE: Acciones que se ejecutan alrededor de un POINTCUT y que amplían la lógica de negocio

</ POA Proxy

El proxy es el objeto que crea Spring que intercepta e invoca al aspecto en la ejecución “el envoltorio”





POA Anotaciones

@Aspect

@PointCut

@Before

@After

@AfterThrowing

@AfterReturning

@Around



POA Configuración

```
<!--Maven-->
```

```
<dependency>
```

```
  <groupId>org.springframework</groupId>
```

```
  <artifactId>spring-aop</artifactId>
```

```
  <version>${versionSpring}</version>
```

```
</dependency>
```

```
<!--Contexto-->
```

```
<aop:aspectj-autoproxy />
```

</ Índice

Módulo 5 - Spring JDBC

Patrones de Diseño en la capa de Datos

Uso de JDBC Template para consultas

Uso de JDBC Template para modificación

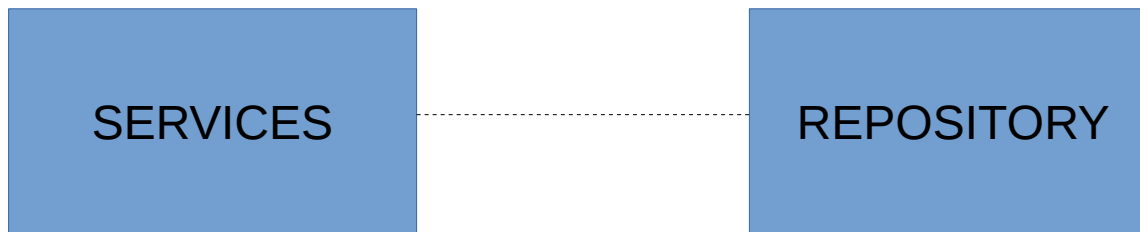
Manejo de Excepciones con Spring JDBC

Ejemplos con Spring JDBC



Capa de Persistencia

Toda la acción de la capa de persistencia se lleva en clases anotadas con `@Repository`, que a su vez, serán inyectadas en las clases `@Service`





Capa de Persistencia

Una buena organización podría reflejar la siguientes distribución de paquetes:

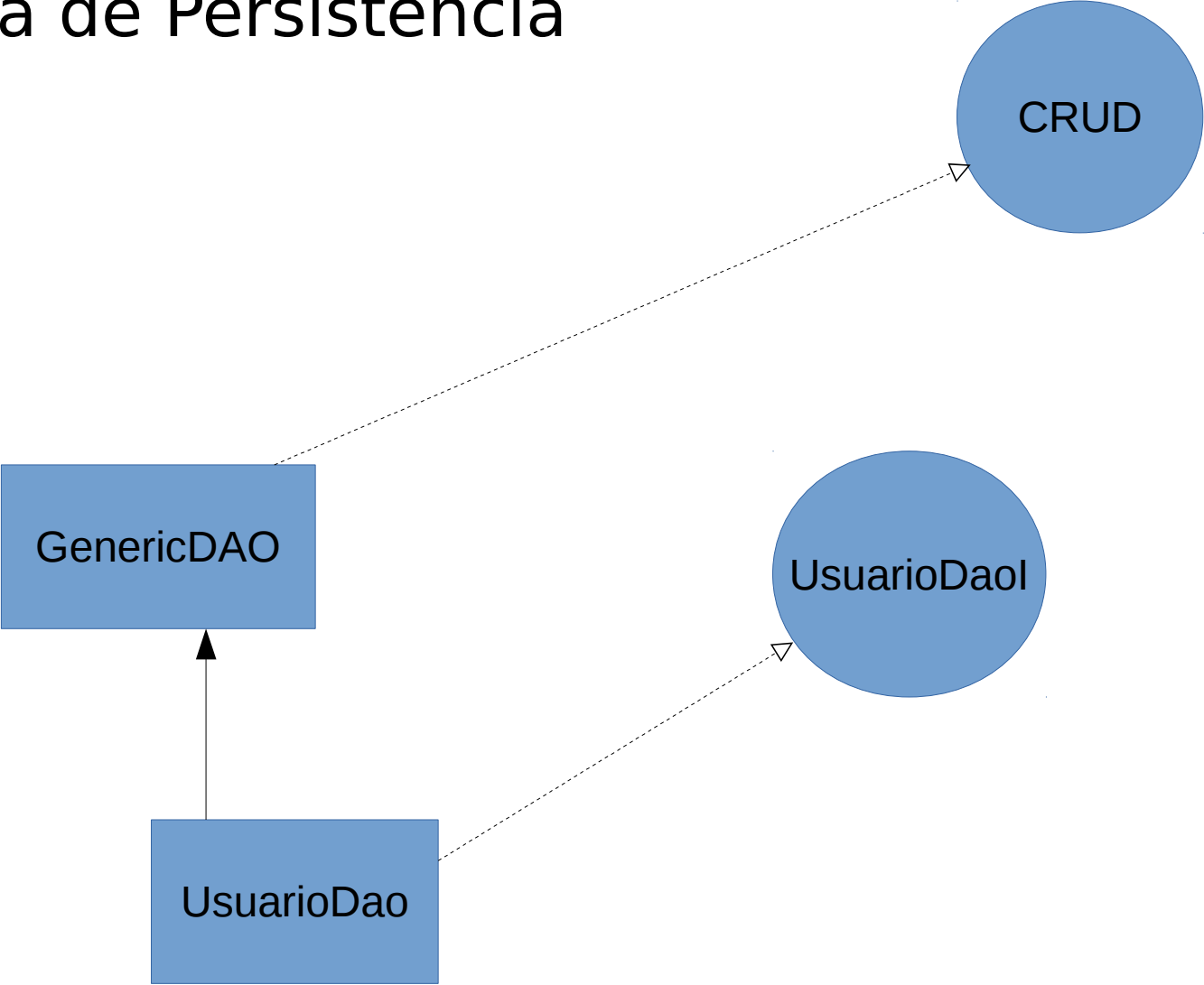
persistence.dao: Clases @Repository

persistence.interfaces: Interfaces

persistence.entity: Clases de persistencia



Capa de Persistencia





DataSource

Es la clase que nos da acceso a la base de datos, en la que se configura la cadena de conexión y el driver

```
<bean id="dataSource"  
class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />  
  <property name="url" value="jdbc:mysql://127.10.7.2:3306/mibasedatos" />  
  <property name="username" value="admin" />  
  <property name="password" value="admin1234" />  
</bean>
```

</ JDBCTemplate

Es una clase auxiliar que encapsula la ejecución de sentencias, encargándose de obtener y liberar recursos, liberando de ello al programador

```
private JdbcTemplate jdbcTemplate;
```

```
...
```

```
jdbcTemplate = new JdbcTemplate  
(dataSource);
```



RowMapper

Es una interfaz que prevé el tratamiento de cada registro transformado en objeto como resultado de una consulta

```
public class PersonMapper implements  
RowMapper<Person> {
```

```
public Person mapRow(ResultSet resultSet,  
int i) throws SQLException {
```

</ Gestión de Transacciones

@Transactional Anotación realizada en la capa de servicio para delegar la gestión de las transacciones a base de datos. Básicamente el commit y el rollback los hace Spring por mí.

RECUERDA: Una transacción es un conjunto de operaciones sobre la base de datos lógicamente agrupadas, de forma que es deseable que se hagan todas o ninguna

</ Gestión de Transacciones

Debo por configuración instanciar el bean que se encarga de las transacciones y asociarle a él esa responsabilidad

Se hace por configuración de este modo

```
<bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
  <property name="dataSource" ref="dataSource" />  
</bean>
```

```
<tx:annotation-driven transaction-manager="transactionManager" />
```



Gestión de Errores

La gestión de errores conviene centralizarla por claridad en el código y por motivos de seguridad

Con la anotación `@ControllerAdvice` puedo indicar en un ecosistema Spring que la clase anotada así, se encarga de recibir excepciones



Gestión de Errores

Además con `@ExceptionHandler` puedo indicar los métodos que hace “match” con cada tipo de excepción prevista

```
@ControllerAdvice(basePackages =  
{"org.springframework."} )  
public class GestionaErrores {  
    @ExceptionHandler(Throwable.class)  
    public String errores (Exception e)  
    { return "error";  
    }  
}
```


</ Índice

Módulo 6 - Introducción a Hibernate

Instalación de Hibernate

Operaciones Básicas con Hibernate

Patrones de Diseño

Integración con una Aplicación Web

Asociaciones con Hibernate

Entidades y ciclo de vida

Persistencia en Cascada

Consultas con HQL, Join y Fetch

API de Criterias

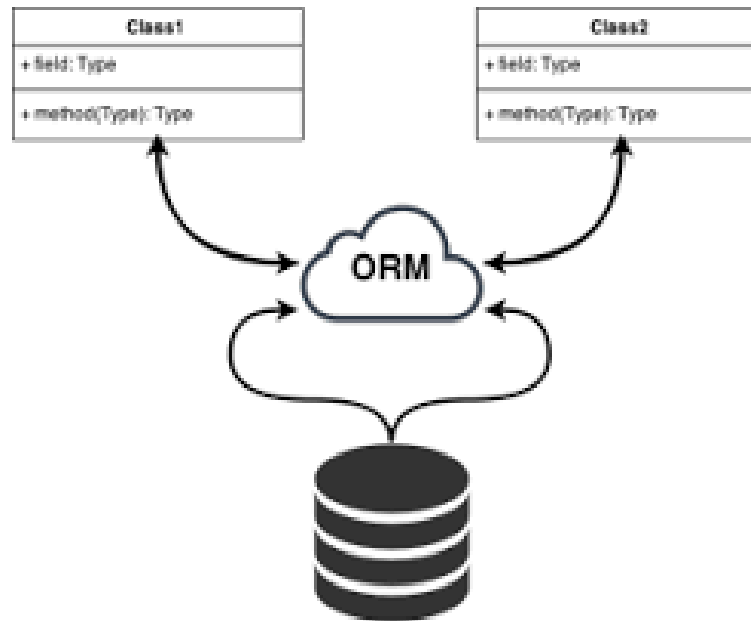
</ Introducción a Hibernate

En su versión inicial, Hibernate eran unas bibliotecas Java para ayudar al programador con la capa de persistencia.

Es de los conocidos como ORM -modelo objeto relacional -

</ Introducción a Hibernate

Una clase equivale a una tabla. Un objeto por tanto, será un registro.



</ Introducción a Hibernate

Esta correspondencia se hacía mediante un xml. Posteriormente, ha sido sustituida por anotaciones en la propia clase.

Se puede hacer manualmente, pero hay herramientas que ayudan y vienen integradas con eclipse


</> Introducción a Hibernate

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.


Search Recent Popular Installed

Find: All Markets All Categories Go

JBoss Tools 4.3.1.Final

 JBoss Tools is an umbrella project for a set of Eclipse plugins that includes support for JBoss and related technologies, such as Hibernate, JBoss AS / WildFly, ...
[more info](#)
by [Red Hat, Inc.](#), EPL
[openshift](#) [WildFly](#) [jbosstools](#) [maven](#) [hibernate](#) [Mobile JSF](#) [cdi](#) [visual editor](#) [jboss](#) [richfaces](#) [jbds](#) [camel](#) [apache camel](#) [fuse](#) [fuse tooling](#) [fuse tools](#) [esb](#) [integration](#) [eip](#) [fileExtension_htm](#) [fileExtension_html](#) [fileExtension_xhtm](#) [fileExtension_xhtml](#) [fileExtension_ftl](#) [fileExtension_ftlx](#) [fileExtension_ftlh](#) [fileExtension_jsp](#) [fileExtension_jsp](#) [fileExtension_jsf](#) [fileExtension_jsfx](#) [fileExtension_hbm](#) [fileExtension_is](#) [fileExtension_ison](#) [fileExtension_java](#)

Marketplaces





Hibernate y JPA

Aunque Hibernate es previo a JPA, conviene usar siempre que se pueda la versión y bibliotecas del estándar (javax.persisntece)





Hibernate + Spring

La clase más importante es el EntityManager. A través de él, se gestionarán todos los accesos a base de datos.

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
">
<property name="persistenceXmlLocation" value="classpath:persistence.xml"
/>
<property name="persistenceUnitName" value="miPersistenceUnit" />
<property name="dataSource" ref="dataSource" />
<property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
<property name="jpaDialect" ref="jpaDialect" />
</bean>
```



EntityManager

Métodos destacados de EntityManager

`find (class, pk) //leo`

`merge (entity) //actualizo`

`persist (entity) //inserto`

`remove (entity) //elimino`

</ Estados de una Entidad

Un objeto de una entidad puede encontrarse en los siguientes estados

TRANSIENT

PERSISTENT

REMOVED

DETACHED

</ Estados de una Entidad

TRANSIENT

```
Usuario u = new Usuario();
```

Hibernate, no es consciente de este objeto, no tiene relación con la base de datos, no representa un registro

</ Estados de una Entidad

PERSISTENT

Usuario u=em.find(Usuario.class, 5);

Hibernate, SÍ es consciente de este objeto u, cualquier modificación en su estado, se efectúa automáticamente en la base de datos

</ Estados de una Entidad

REMOVED

```
em.remove(usuario);
```

Después de esta operación, este objeto no existe para Hibernate (no está asociado a ninguna fila de la base de datos)

</ Estados de una Entidad

DETACHED

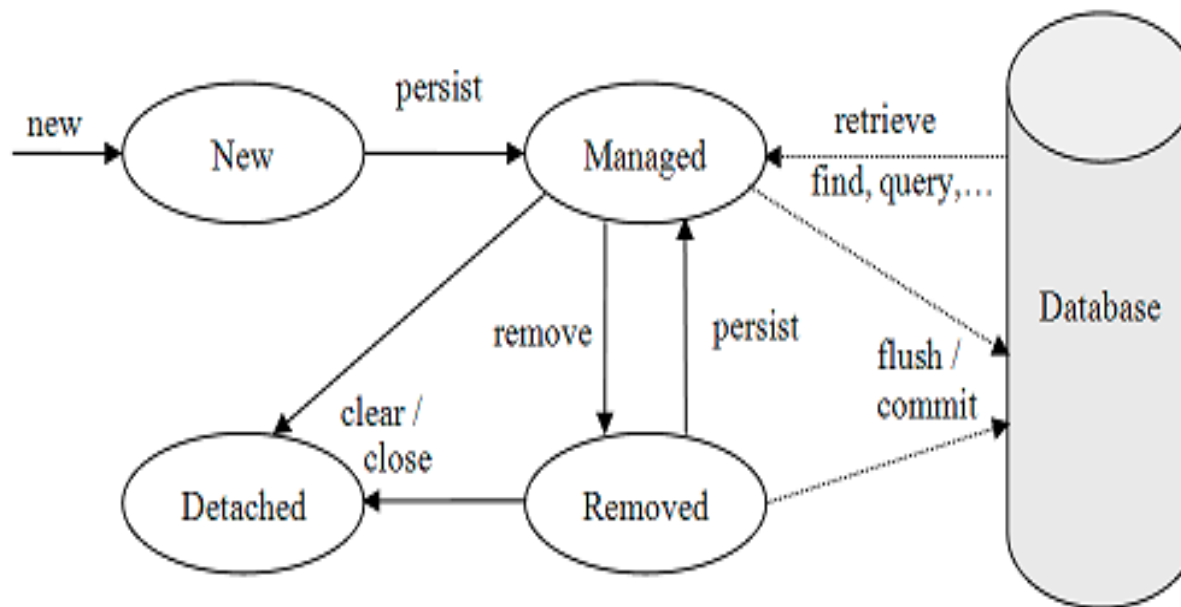
```
Usuario u=em.find(Usuario.class, 5);
```

```
//fuera de la transacción
```

```
u.setNombre ("Alex");
```

**El objeto sí existe para Hibernate,
representa una fila, aunque
cualquier modificación en su
estado NO tiene efecto**

</ Estados de una Entidad



</ Gestión de transacciones

Las necesidades normales de una aplicación llevan a la gestión de *transaction per request*, con lo que los estados más comunes son el de transient y persistent



Hibernate + Spring

Además de las entidades (clases asociadas a la tabla) necesito un archivo llamado persistence.xml que describa cuál es mi unidad de persistencia.

```
<persistence-unit name="miPersistenceUnit"
transaction-type="RESOURCE_LOCAL">
  <class>entity.Rol</class>
  <class>entity.Usuario</class>
  <class>entity.UsuarioRol</class>
  <properties>
    <property name = "hibernate.show_sql" value = "true" />
  </properties>
</persistence-unit>
```




Hibernate + Spring

Se inyectará en las clases @Repository, el EntityManager del siguiente modo

@Repository

```
public class UsuarioRepository {
```

@PersistenceContext

```
private EntityManager entityManager;
```

</ Integración WebServices Rest

En un sistema distribuido, es más común cada día que la ejecución de un servicio no devuelva un web.

Sino simplemente envíe y recibe texto en formato JSON

</ Integración WebServices Rest

Los servicios Rest aprovechan la semántica de los métodos HTTP (Get, Post, Put y Delete) para corresponderlos con operaciones de consulta, inserción, modificación y borrado respectivamente.

Dada sus sencillez y las ventajas de diseño que poseen, son un estándar en la industria de hoy.

</ Integración WebServices Rest

ResponseEntity.- Permite manejar el cuerpo y la cabecera del mensaje HTTP de respuesta con suma facilidad.

@RestController.- Indicamos que es que un Controller, que gestiona peticiones Rest

Integración WebServices Rest

ResponseEntity

@RestController

@PathVariable

@JsonIgnore



EXTRA

@CrossOrigin

@Valid

DTO vs Entity

CASCADE {ALL}

FECTH {Lazy, Eager}

</ EXTRA JPA CascadeType

ALL.- en todo caso

DETACH.- descx manual en cascada

MERGE.- merge() // al hacer update

REFRESH.- refresh () //al recargar

REMOVE.- delete() //al borrar

PERSIST.- save() persist() //al crear

Hibernate tiene algunos más

</ EXTRA ID GenerationType

AUTO.- en f() de la BD elige

IDENTITY.- AUTOINC MySql

SEQUENCE.- SEQ Oracle o propia

TABLE.- No usar (retardo Update SEQ)

</ HQL

Hibernate Query Language es una forma muy similar al SQL que pretende lograr independencia de la BD subyacente

//para referirnos a las entidades
`@Entity(name = "usuario")`

</ HQL

```
String hql = "FROM Employee AS E";  
Query query = session.createQuery (hql);  
List results = query.list();
```

Ejemplo de consulta HQL. Omitiendo SELECT, estamos diciéndole a Hibernate que quiero todos los atributos de la tabla/entidad



HQL SELECT

```
String hql = "SELECT E.firstName FROM Employee E";
```

```
Query query = session.createQuery(hql);
```

```
List results = query.list();
```

Al incluirlo, estamos diciéndole a Hibernate que quiero sólo algunos los atributos de la tabla/entidad.

Las palabras reservadas (WHERE, SELECT, pueden usarse en indistintamente en mayúsculas o minúsculas)



HQL WHERE

```
String hql = "FROM Employee E WHERE E.id = :employee_id";  
Query query = session.createQuery(hql);  
query.setParameter("employee_id",10);  
List results = query.list();
```

Puedo usar la cláusula WHERE como en SQL y hacer consultas con parámetros por nombre

</ HQL JPA NamedQuery

```
@NamedQueries({  
    @NamedQuery(name="Departments.todos",  
        query="SELECT d FROM departments d"),  
    @NamedQuery(name="Departments.pornombre",  
        query="SELECT d FROM departments d WHERE  
d.departmentName = :name"),  
})
```

Puedo definir consultas en cada entidad y luego ejecutarlas por su nombre

</ HQL JPA NamedQuery

```
TypedQuery<Departments> query =  
em.createNamedQuery("Departments.todos",  
Departments.class);  
List<Departments> results = query.getResultList();
```

Se emplea TypedQuery para armar la consulta por nombre

</ EXTRA

Cambio API JPA-HB (Filter)

Stateless para batch

Criteria API

JPQL

HQL referencia

</ Índice

Módulo 7 - Introducción a test unitarios

Introducción a los test unitarios

Descripción general de JUnit

JUnit API

Importar paquetes

Composición de pruebas con TestSuite

Mocks

</ TDD

El desarrollo dirigido por tests también conocido por su acrónimo inglés TDD se ha convertido casi en un mantra metodológico en nuestros días.

Cada pieza, es probada de forma individual antes de ser integrada y ser sometida a los test integrales

</ TDD

Por cada clase/servicio que deseemos testar dentro de src/main, se crea una clase análoga en src/test

SRC/MAIN/JAVA



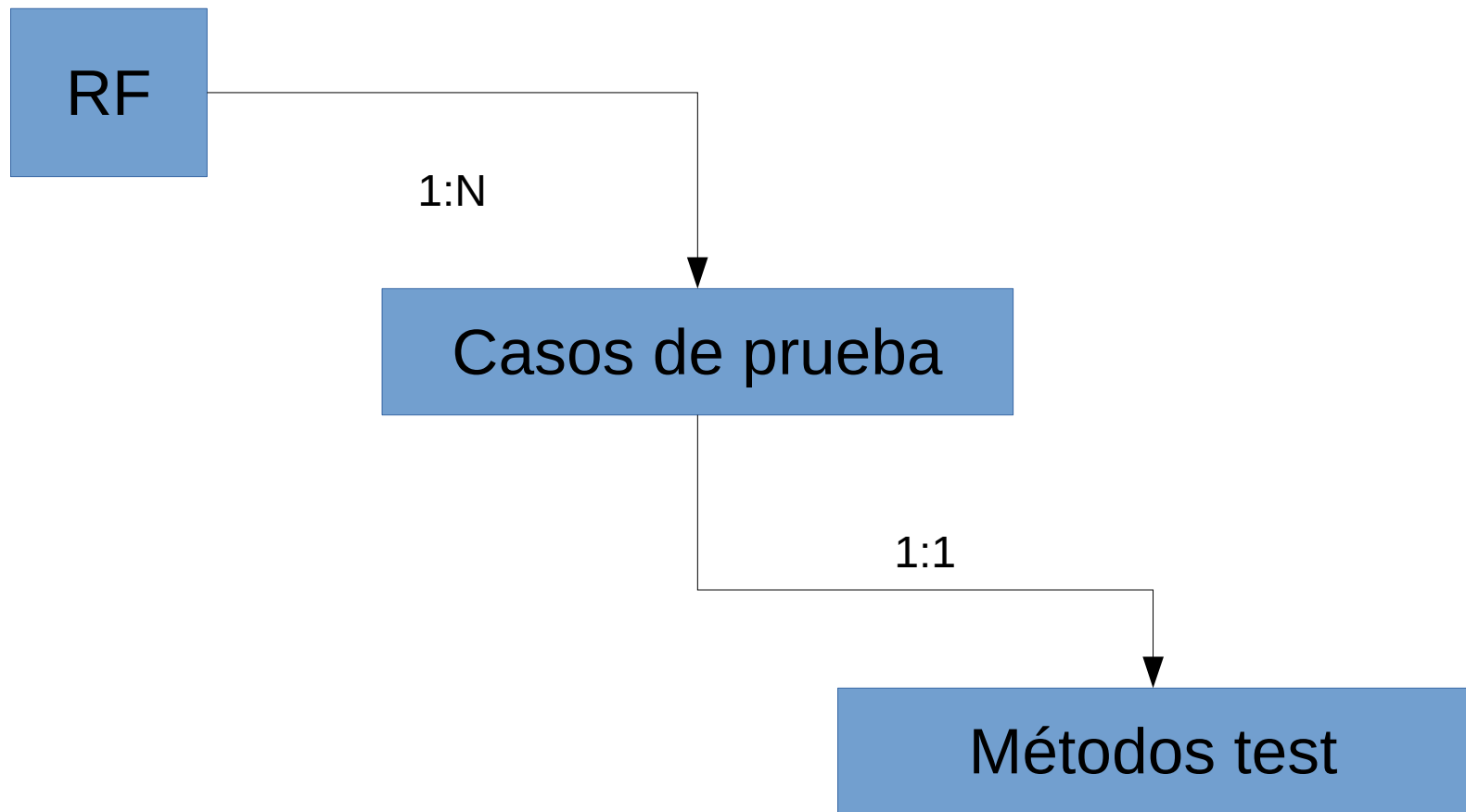
SRC/TEST/JAVA

</ TDD

Los requisitos funcionales (RF) han debido ser definidos e identificados claramente en la fase de análisis.

La satisfacción de cada RF llevará a definir varios casos de prueba, lo que significarán tantos métodos test

</ TDD



</ TDD

De forma idónea, los RF y los casos de prueba, así como la escritura de los tests, deberían definirse por alguien ajeno al programador cuyo código quiere testarse.

Un diagrama de clases detallado y una buena capa de test, son las mejores herramientas de un arquitecto.

</ Junit

```
<dependency>  
<groupId>junit</groupId>  
<artifactId>junit</artifactId>  
<version>4.9</version>  
</dependency>
```

Junit se ha convertido en la herramienta Java de referencia para el desarrollo de test unitarios

</ Junit

@Test

@Before

@BeforeClass

@After

@AfterClass

@Test

(expected=RuntimeException.class)



Junit - Aserciones

La clase *org.junit.Assert* engloba una serie de método estáticos que permiten expresar las condiciones que debe satisfacer un método `@Test`

`Assert.assertTrue(condición);`

`Assert.assertEquals(expected, got);`

`Assert.assertNull(objeto)`



Junit - Aserciones

La clase *org.junit.Assert* engloba una serie de método estáticos que permiten expresar las condiciones que debe satisfacer un método `@Test`

`Assert.assertTrue(condición);`

`Assert.assertEquals(expected, got);`

`Assert.assertNull(objeto)`



Junit - Ejecución

Botón derecho a la clase que alberga los test Run → Run As → Junit Test y se muestran los resultados

Markers Properties Servers Data Source Explorer Snippets Problems Console Search JUnit

Finished after 0,223 seconds

Runs: 5/5 Errors: 0 Failures: 0

edu.val.spring.tdd.testingbasico.StringCalculatorTest [Runner: JUnit 4] (0,003 s) Failure Trace

- sumaOK (0,002 s)
- dosNumerosCasoCorrecto (0,000 s)
- casoDeNumeroYNoNumeroEsExcepcion (0,000 s)
- casoParametroNull (0,000 s)
- masDeDosNumerosEsExcepccion (0,000 s)



Junit - Hamcrest

Hamcrest es un comparador o *Matcher* que se descarga al hacerlo Junit y nos facilita hacer predicados más ricos.

```
assertThat(list, hasSize(3));
```

```
assertThat(objeto,  
hasProperty("nombre"  
equalTo("Alex")));
```



Junit - Suite

Puedo definir un conjunto de Test para que se ejecuten de forma agrupada mediante de esta forma:

```
@RunWith(Suite.class)
@SuiteClasses({
    Test1.class,
    Test2.class})
public class TestSuite{ }
```

</ Junit

Antes de la aparición de anotaciones en la versión Junit4, los test eran los métodos cuyo nombre empezaba por test y estaban en una clase que heredaba de TestCase

Otra característica que puede emplearse en Junit son los Listeners



Mocks

¿Qué pasa si desde mi entorno de pruebas no puedo contar con el servidor de aplicaciones o no tengo acceso a la base de datos?

Los MOCK son objetos sustitutivos de un entorno real. Spring incluye algunos y Mockito añade más



Spring test

```
<dependency>  
<groupId>org.springframework</groupId>  
<artifactId>spring-test</artifactId>  
<version>${versionSpring}</version>  
<scope>test</scope>  
</dependency>
```

</ org.springframework.test

MockHttpServletRequest

MockHttpServletResponse

MockMvc

MockMvcRequestBuilders

MockMvcResultMatchers

MockMvcBuilders

SpringJUnit4ClassRunner



Test de Integración

Al estar probando nuestras clases en un entorno integrado con las de Spring, podemos hablar de test de integración y de test unitario.

Pese a levantar el contexto, no es necesario ejecutar el servidor. Basta añadir las anotaciones oportunas `@WebAppConfiguration`



Mocks Mockito

```
<dependency>  
<groupId>org.mockito</groupId>  
<artifactId>mockito-all</artifactId>  
<version>2.0.2-beta</version>  
</dependency>
```





@RunWith - Junit

Si omito la anotación, es una clase de Junit quién gestiona y lanza la ejecución de test

//integración blanda

```
@RunWith(MockitoJUnitRunner.class)
```

//integración dura

```
@RunWith(SpringJUnit4ClassRunner.class)
```



Mockito Anotaciones y métodos

//marca el mock a crear

@Mock

//indicamos la clase donde se inyectan

@InjectMocks

//programamos la simulación

when(method).thenReturn(obj)

//controlamos invocación

verify(mock, n*).method()



VALERIANO MORENO

valerianomoreno@gmail.com

info@devacademy.es

634504842