



# Groovy & Grails: Desarrollo rápido de aplicaciones

## Sesión 5: Introducción a Grails



# Introducción a Grails

- ¿Qué es?
- Arquitectura
- Instalación de Grails
- Scaffolding



# ¿Qué es?

- Características de Grails
- Software de código abierto



## ¿Qué es?

- “*Grails es un framework para el desarrollo de aplicaciones web basado en el lenguaje de programación Groovy, que a su vez se basa en la Plataforma Java*”
- Se basa en los paradigmas *convención sobre configuración* y *DRY (don't repeat yourself)*



# ¿Qué es?

- Basado en el patrón *Modelo Vista Controlador*
  - Modelo ↔ Clases de dominio
  - Controladores
  - Vista ↔ Páginas GSP (Groovy Server Pages)



## ¿Qué es?

- El programador se olvida de determinados aspectos de configuración
- Grails acorta los tiempos de desarrollo
- Grails es un framework muy ágil



## ¿Qué es?

- Grails no sólo es un framework de desarrollo web que sigue el patrón MVC, sino que es una plataforma completa de desarrollo
  - Contenedor web
  - Gestor de base de datos
  - Empaquetado de la aplicación
  - Realización de tests



# Características de Grails

- Convención sobre configuración

Se utilizan una serie de convenciones para evitar tener que escribir interminables archivos de configuración



# Características de Grails

- Tests
  - Tests unitarios (sin dependencias)
  - Tests de integración (entorno completo)
  - Tests funcionales (funcionalidad de la aplicación web)



# Características de Grails

- Scaffolding

*“Generación automática de código para las cuatro operaciones básicas de cualquier aplicación, que son la creación, lectura, edición y borrado”*



# Características de Grails

- Mapeo objeto-relacional
  - GORM (Grails Object Relational Mapping)
    - Uno a uno
    - Uno a muchos
    - Muchos a muchos



# Características de Grails

- Plugins
  - Seguridad
  - AJAX
  - Testeo
  - Búsqueda
  - Informes
  - Servicios web



# Software de código abierto

- Grails no sufre del síndrome *Not Invented Here* (*NIH*) e integra las mejores soluciones de software libre para conseguir un framework muy robusto



# Software de código abierto

- Groovy
  - Lenguaje dinámico
  - Potente y flexible
  - Sintaxis sencilla
  - Integración con Java



# Software de código abierto

- Framework Spring
  - Alto nivel de abstracción
  - Declaración de transacciones mediante POJOs



# Software de código abierto

- Hibernate
  - Framework de persistencia objeto-relacional
  - Es la base de GORM
  - Mapea clases de dominio contra las tablas de una base de datos



# Software de código abierto

- SiteMesh
  - Renderizado HTML
  - Patrón de diseño *Decorator*
  - Componentes: cabecera, pies de página y sistemas de navegación



# Software de código abierto

- Frameworks AJAX
  - Script.aculo.us
  - Rico
  - Prototype



# Software de código abierto

- Jetty y Tomcat
  - Contenedor y servidor web
  - No son los únicos sobre los que funciona Grails



# Software de código abierto

- HSQLDB
  - Gestor de base de datos
  - Almacenamiento en memoria o en disco

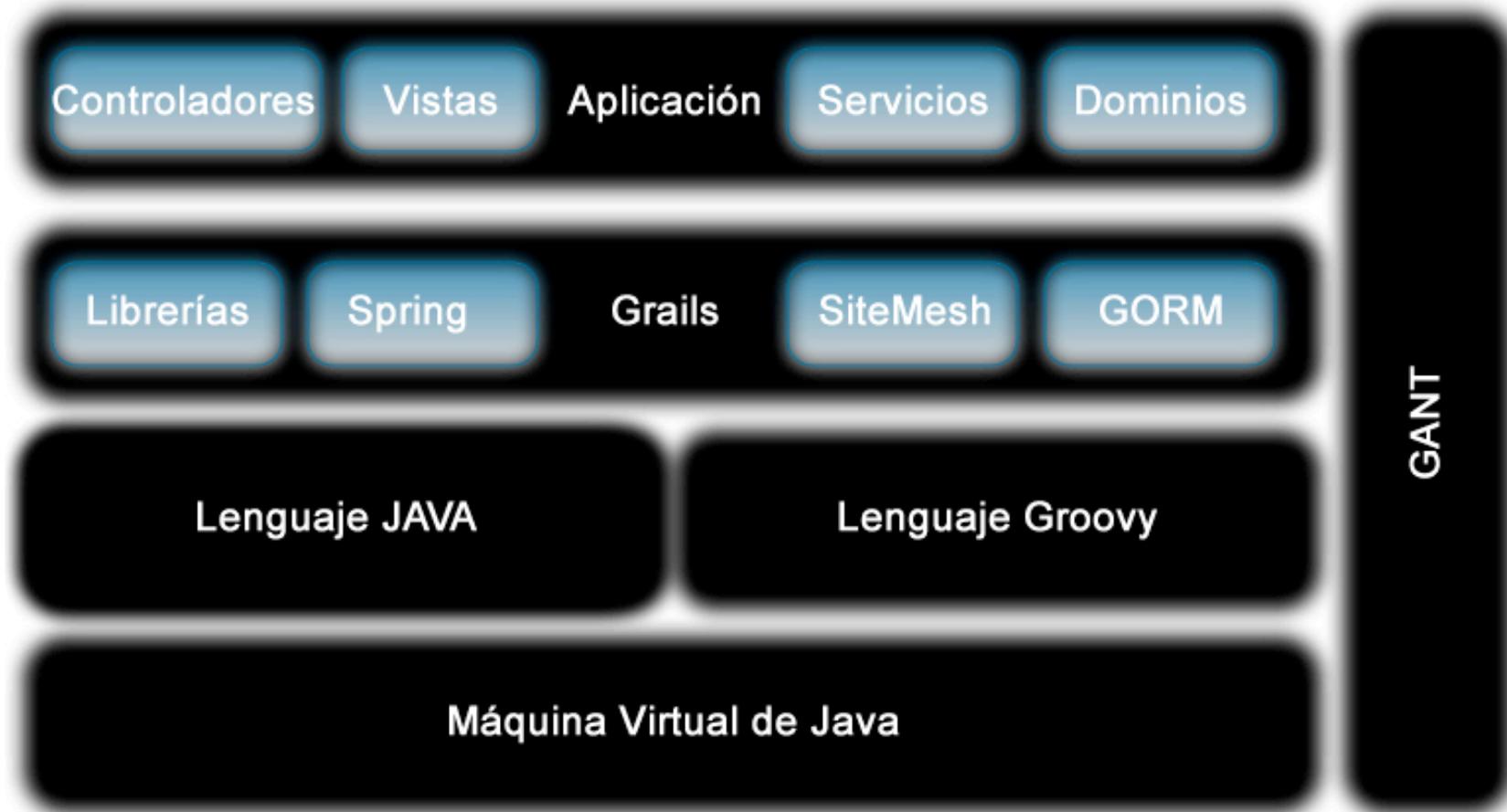


# Software de código abierto

- JUnit
  - Framework para la realización de tests unitarios
  - Muy extendido en Java



# Arquitectura





# Arquitectura

- Cuatro capas
  - 1ª: Máquina Virtual de Java
  - 2ª: Lenguajes Java y Groovy
  - 3ª: Grails y otros frameworks
  - 4ª: Aplicación siguiendo el MVC



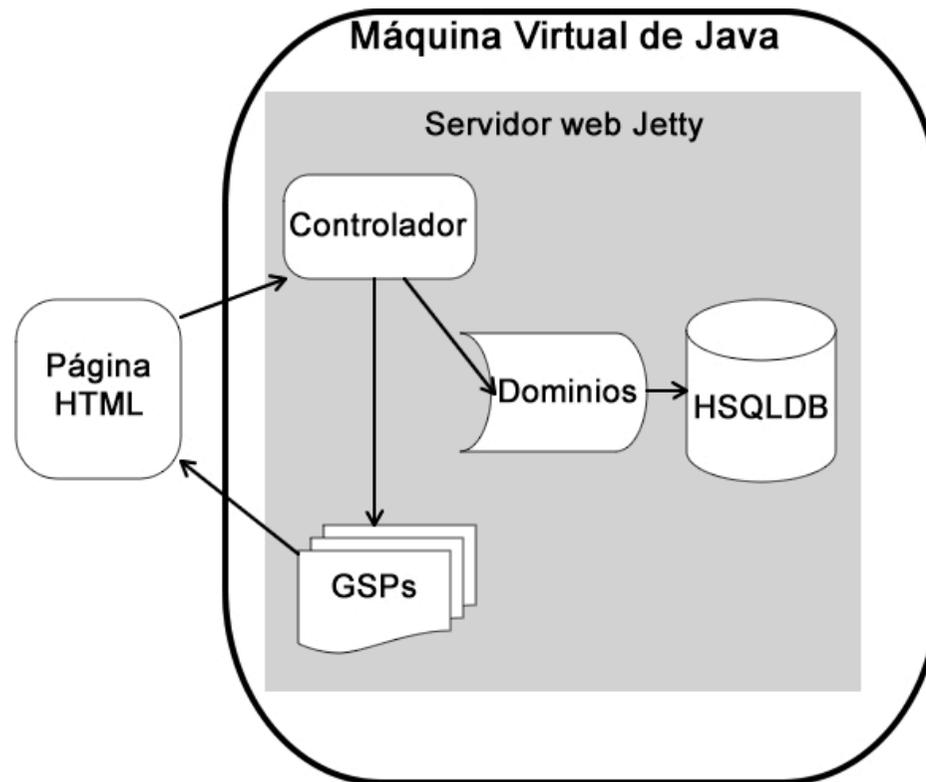
# Arquitectura

- Herramientas en línea de comandos
  - Facilita la escritura de código
  - Facilita la gestión de nuestros proyectos
  - Basado en Gant, un sistema de automatización de tareas basado en Apache Ant



# Arquitectura

- Ejecución de un proyecto Grails





# Instalación de Grails

1. Descargar la última versión de Grails
2. Descomprimir el archivo
3. Crear la variable de entorno *GRAILS\_HOME*
4. Agregar el directorio *GRAILS\_HOME/bin* al *PATH* del sistema



# Instalación de Grails

- Prerrequisito
  - Tener instalado al menos el JDK 1.4
  - Variable de entorno *JAVA\_HOME*



# Scaffolding

- “*Construcción automática de aplicaciones a partir del esquema de la base de datos*”
- Cuatro operaciones básicas (*CRUD*):
  - Creación (*Create*)
  - Lectura (*Read*)
  - Actualización (*Update*)
  - Borrado (*Delete*)



# Scaffolding

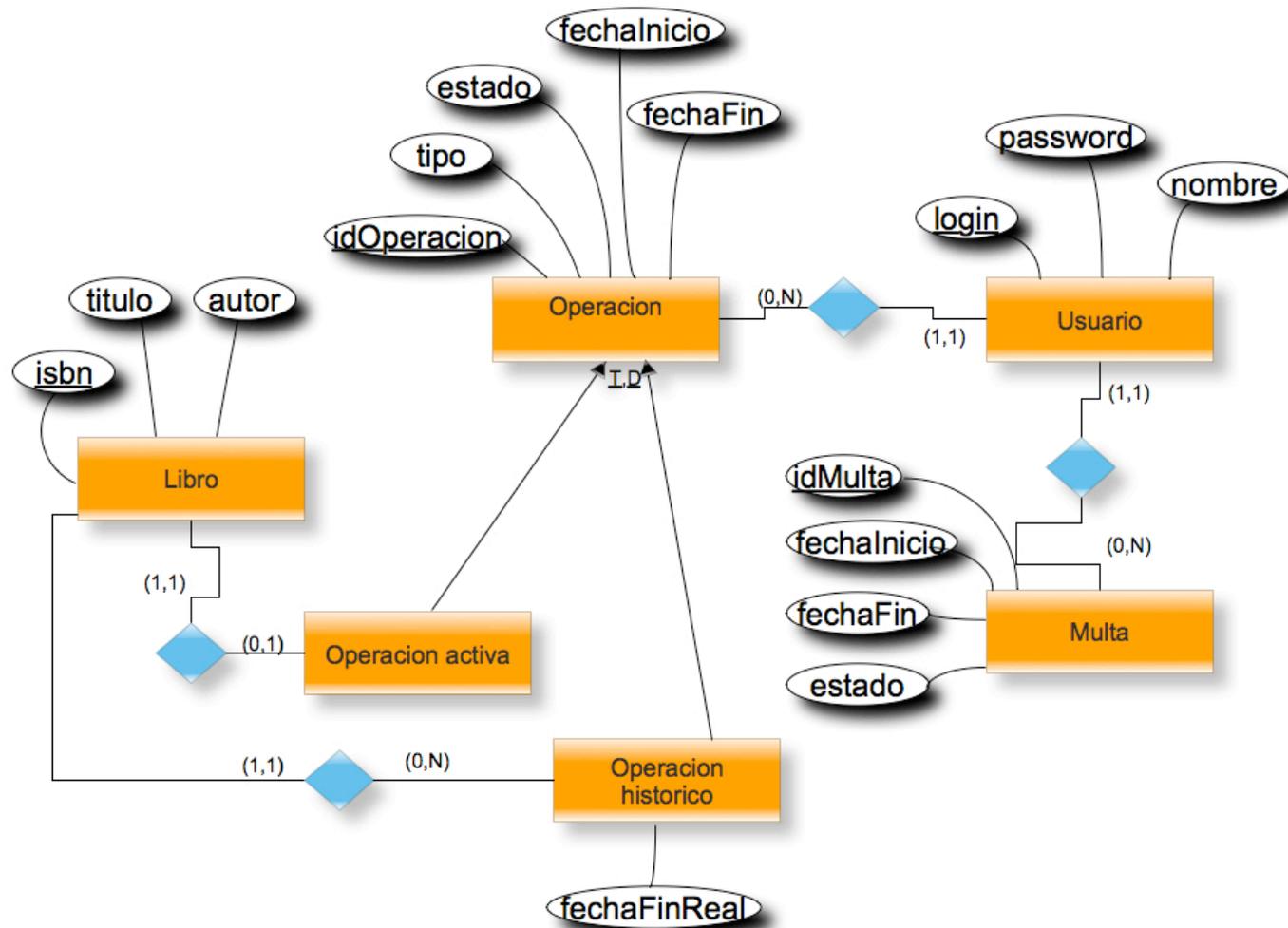
- Descripción de la aplicación ejemplo
- Creación del proyecto Grails
- Creación de las clases de dominio
- Creación de controladores



# Scaffolding

- Aplicación ejemplo: biblioteca de un instituto
  - Gestión de usuarios (administrador, bibliotecario, profesor o socio)
  - Administradores se encargan de la gestión de usuarios
  - Bibliotecarios de la gestión de libros y préstamos
  - Profesores y socios podrán realizar reservas
  - Imposición de multas por retraso en la devolución

# Scaffolding





# Scaffolding

- Esquema reducido
  - Operación
  - Usuario
  - Libro



# Creación del proyecto Grails

```
grails create-app biblioteca
```



# Creación del proyecto Grails

- Estructura de directorios

Directorio	Descripción
grails-app/conf	Ficheros de configuración de la aplicación
grails-app/conf/hibernate	Archivos de mapeado de Hibernate
grails-app/conf/spring	Archivos de mapeado de Spring
grails-app/controllers	Controladores de la aplicación que gestionan las peticiones
grails-app/domain	Clases de dominio del modelo
grails-app/i18n	Mensajes para la internacionalización de la aplicación
grails-app/services	Servicios
grails-app/taglib	Librerías de etiquetas dinámicas



# Creación del proyecto Grails

Directorio	Descripción
grails-app/views	Archivos GSP
grails-app/views/layout	Archivos de diseño de las páginas web
lib	Archivos JAR de terceras partes, tales como controladores de bases de datos
scripts	Scripts GANT para el automatizado de tareas
src/java	Archivos fuente adicionales en Java
src/groovy	Archivos fuente adicionales en Groovy
test/integration	Tests de integración
test/unit	Tests unitarios



# Creación del proyecto Grails

Directorio	Descripción
web-app	Artefactos web que finalmente serán comprimidos a un WAR (Web Application Archive)
web-app/css	Hojas de estilo
web-app/images	Imágenes de la aplicación
web-app/js	Javascript
web-app/WEB-INF	Archivos de configuración para Spring o SiteMesh



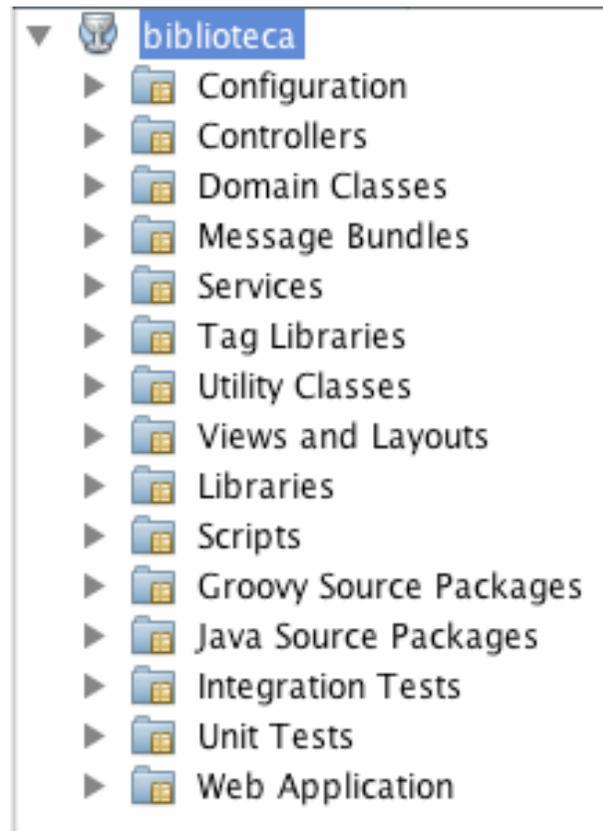
# Creación del proyecto Grails

- Archivos de configuración de proyecto para diferentes editores
  - Eclipse
  - Textmate
  - **NetBeans**



# Creación del proyecto Grails

- Proyecto con NetBeans





# Creación del proyecto Grails

```
grails run-app
```



# Creación del proyecto Grails

- `http://localhost:8080/biblioteca`
- Se crea una instancia del servidor web Tomcat



# Creación de clases de dominio

```
grails create-domain-class libro
```



# Creación de clases de dominio

- Se crea una clase de dominio llamada *Libro*

```
package biblioteca

class Libro {
    static constraints = {
    }
}
```



## Creación de clases de dominio

- Por cada clase de dominio creada, se crea un test unitario
- Debemos copiar este test unitario al directorio correspondiente a los tests de integración



## Creación de clases de dominio

```
package biblioteca
import grails.test.*

class LibroTests extends GrailsUnitTestCase {
    protected void setUp() {
        super.setUp()
    }

    protected void tearDown() {
        super.tearDown()
    }

    void testSomething() {

    }
}
```



## Creación de clases de dominio

- El método *setUp()* se ejecuta antes que cualquier otro método que empiece con la palabra *test*
- El método *tearDown()* se ejecuta una vez que se han ejecutado todos los tests de la clase
- El método *testSomething()* es un método ejemplo



# Creación de clases de dominio

```
package biblioteca
import grails.test.*

class LibroTests extends GrailsUnitTestCase {
    protected void setUp() {
        Libro.list()*.delete()
    }

    protected void tearDown() {
        super.tearDown()
    }
    ....
}
```



# Creación de clases de dominio

```
class LibroTests extends GrailsUnitTestCase {
    ....
    void testPersiste() {
        new Libro(titulo:'La colmena', anyo:1951, ...).save()
        new Libro(titulo:'La galatea', anyo:1585...).save()
        new Libro(titulo:'El ingenioso hidalgo ...', anyo:1605, ...).save()
        new Libro(titulo:'La dorotea', anyo:1632, ...).save()
        new Libro(titulo:'La dragontea', anyo:1602, ...).save()
        assert 5 == Libro.count()
    }
    void testToString(){
        def libro = new Libro(titulo:'Groovy in action', anyo: 2007, ...)
        assertToString(libro, 'Groovy in action')
    }
}
```



## Creación de clases de dominio

- El método *setUp()* deja vacía de datos la clase Libro con el método *delete()*
- El método *testPersiste()* inserta 5 libros en la base de datos con el método *save()*
- El método *testToString()* comprueba que de la clase Libro se devuelve el título del mismo



# Creación de clases de dominio

```
grails test-app
```



## Creación de clases de dominio

- Se crea un directorio llamado *test-reports* bajo el directorio *target*
- Ambos tests han fallado
- Debemos terminar de crear la clase de dominio Libro



# Creación de clases de dominio

```
package biblioteca

class Libro {
    String isbn
    String titulo
    String autor
    String editorial
    Integer anyo
    String descripcion
    Date fecha

    ....
}
```



# Creación de clases de dominio

```
class Libro {  
    ....  
  
    static constraints = {  
        isbn(blank:false)  
        titulo(blank:false)  
        autor(blank:false)  
        editorial()  
        anyo()  
        fecha(nullable:true)  
        descripcion(maxSize:1000,nullable:true)  
    }  
    ....  
}
```



# Creación de clases de dominio

```
class Libro {  
    ....  
    String toString(){  
        titulo  
    }  
}
```



## Creación de clases de dominio

- No es necesario especificar una clave primaria gracias a las propiedades *id* y *version*
- Con ellas se garantiza la integridad de los datos y el bloque de las tablas cuando sea necesario
- Las restricciones se añaden en la variable *constraints*



# Creación de controladores

```
grails create-controller libro
```



# Creación de controladores

- Los controladores se encargan de gestionar las peticiones que llegan a cada clase de dominio
- Se encarga también de gestionar la interacción entre la vista y las clases de dominio



## Creación de controladores

- Con el comando *grails create-controller libro* se crea un nuevo controlador en *grails-app/controller* llamado *LibroController.groovy*

```
package biblioteca

class LibroController {
    def index = { }
}
```



# Creación de controladores

- Definiendo el scaffolding sobre la clase *Libro*

```
package biblioteca

class LibroController {
    def scaffold = Libro
}
```



# Creación de controladores

Home Libro List

## Create Libro

Isbn:

Titulo:

Autor:

Editorial:

Anyo:

Fecha:     :

Descripcion:



## Creación de controladores

- El orden de los elementos del formulario coincide con el orden en el que se indicaron las restricciones
- Al especificar un tamaño máximo en la propiedad *descripción* se crea un *textarea* en lugar de un *text*
- Si se incumple alguna restricción, se muestra el error correspondiente

❗ La propiedad [isbn] de la clase [class Libro] no puede ser vacía



# Creación de controladores

```
grails test-app
```



# Creación de controladores

## Unit Test Results.

Designed for use with [JUnit](#) and [Ant](#).

### Class biblioteca.LibroTests

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
<a href="#">LibroTests</a>	<u>2</u>	0	0	1.100	2010-06-14T18:46:05	frangarciamac.local

### Tests

Name	Status	Type	Time(s)
testPersiste	Success		1.088
testToString	Success		0.008

[Properties »](#)  
[System.out »](#)  
[System.err »](#)



# Creación de clases de dominio

```
grails create-domain-class usuario
```

```
grails create-domain-class operacion
```



# Creación de clases de dominio

```
package biblioteca

class Usuario {
    String login
    String password
    String nombre
    String apellidos
    String tipo

    static hasMany = [operaciones:Operacion]

    ....
}
```



# Creación de clases de dominio

```
class Usuario {
    ....
    static constraints = {
        login(size:6..20, blank:false, unique:true)
        password(size:6..20, blank:false,password:true)
        nombre(blank:false)
        apellidos(blank:false)
        tipo(inList:["administrador", "bibliotecario", "profesor", "socio"])
    }

    String toString(){
        "$nombre $apellidos"
    }
}
```



# Creación de clases de dominio

```
package biblioteca

class Operacion {
    String tipo
    Boolean estado
    Date fechaInicio
    Date fechaFin
    Usuario usuario
    Libro libro

    static belongsTo = [Usuario,Libro]

    ....
}
```



# Creación de clases de dominio

```
class Operacion {  
    ....  
  
    static constraints = {  
        tipo(inList:["prestamo", "reserva"])  
        estado()  
        fechaInicio(nullable:false)  
        fechaFin(nullable:false)  
    }  
  
    String toString() {  
        "$tipo ($estado) [$fechaInicio - $fechaFin]"  
    }  
}
```



# Creación de clases de dominio

```
package biblioteca

class Libro {
    ....

    static hasMany = [operaciones:Operacion]

    ....
}
```



# Creación de controladores

```
grails create-controller usuario
```

```
grails create-controller operacion
```



# Creación de controladores

```
package biblioteca

class UsuarioController {

    def scaffold = Usuario
}
```

```
package biblioteca

class OperacionController {

    def scaffold = Operacion
}
```



# Inserción de datos con BootStrap.groovy

- El archivo *BootStrap.groovy* servirá para realizar acciones al arrancar y al finalizar la aplicación
- Utilizaremos este fichero para insertar datos de ejemplo en la aplicación



# Inserción de datos con BootStrap.groovy

```
class BootStrap {
    def init = { servletContext ->
        new Usuario(login:'frangarcia',password:'mipassword',...).save()
        new Usuario(login:'pablomarmol',password:'marmol',...).save()
        new Usuario(login:'pedropp',password:'picapiedra',...).save()
        new Usuario(login:'wilmapp',password:'picapiedra2',...).save()
        new Usuario(login:'bettymarmol',password:'marmol2',...).save()
        new Libro(titulo:'La colmena', anyo:1951,...).save()
        new Libro(titulo:'La galatea', anyo:1585,...).save()
        new Libro(titulo:'El ingenioso hidalgo ...', anyo:1605,...).save()
        new Libro(titulo:'La dorotea', anyo:1632,...).save()
        new Libro(titulo:'La dragontea', anyo:1602,...).save()
    }
    def destroy = { }
}
```