

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

Antonio Santos Ramos

[antoniosantosramos@gmail.com](mailto:antoniosantosramos@gmail.com)

Curso de Programación para Internet con Java EE

(29 Octubre 2018 – 14 Diciembre 2018)<sub>v14</sub>

Java (Lucatc) (8h-17h) Puente 2/11, 9/11 y 07/12

© Antonio Santos Ramos 2018

1

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

T0 - Índice

© Antonio Santos Ramos 2018

2

# Contenido

## Curso de Programación para Internet con Java EE (Parte V - Proyectos)

### Temario

- T01 – SDLC: Software Development Life Cycle
- T02 – UML Modeling
- T03 – Metodologías Ágiles: SCRUM
- T04 – Introducción a DevOps
- T05 – Pruebas
- T06 – Best Practices

© Antonio Santos Ramos 2018

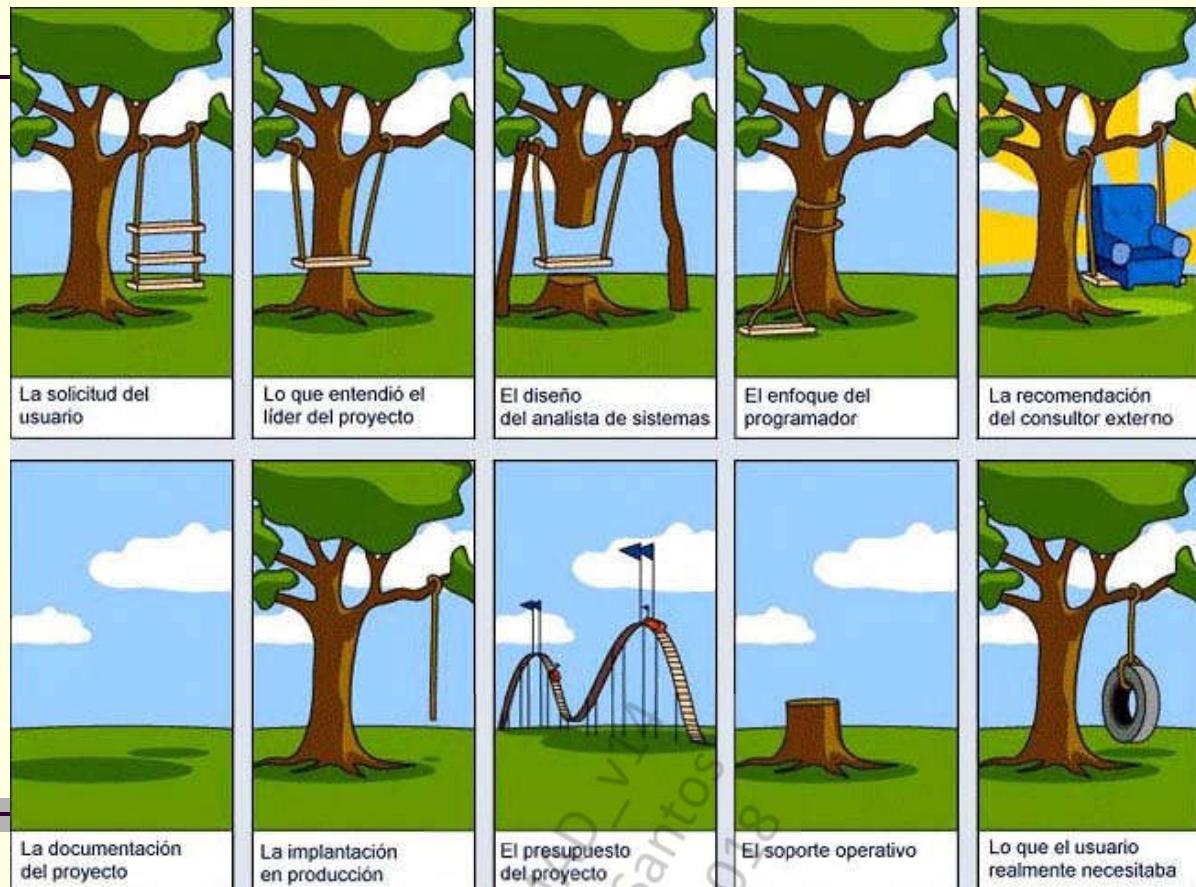
3

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

## T01

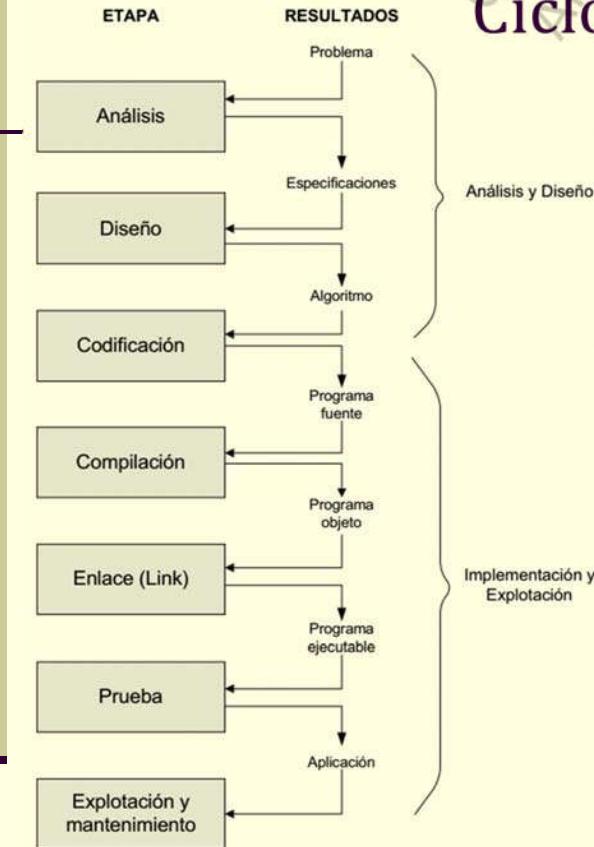
### Software Development Life Cycle (SDLC)

# ¿Así son los proyectos?



5

## Ciclo de Vida



- Análisis del problema
  - Conocer los datos de entrada y salida
  - Entender como se realiza la transformación
- Construcción del algoritmo
  - Se puede expresar mediante gráficos o pseudocódigo
- Codificación (programar)
  - Se codifica el algoritmo con un lenguaje de alto nivel (Java, C, VisualBasic, Objetive C, etc.)
  - De esta fase se obtiene un código denominado Código fuente
- Compilar (traducir)
  - Mediante la ayuda de un programa, se traduce a código máquina para que pueda ser entendido por el ordenador.

# Software. Proceso de construcción

- Conjunto estructurado de actividades requeridas para desarrollar un sistema de software.
- Las fases principales son
  - I. **Especificación:** establecer los requerimientos, las restricciones del sistema y las especificaciones de desarrollo
  - II. **Diseño:** producir un modelo en papel del sistema
  - III. **Desarrollo:** producción e implementación del sistema software
  - IV. **Prueba:** verificar que el sistema cumpla las especificaciones requeridas: desarrollo, aceptación, etc.
  - V. **Validación:** verificar que el software hace lo que el cliente pide.
  - VI. **Instalación:** entregar el sistema al usuario
  - VII. **Mantenimiento:** reparar los fallos descubiertos en el sistema
  - VIII. **Evolución:** cambiar/adaptar el software a las demandas

## Entornos

### Entorno de desarrollo

- Empleado por los programadores web para modificar la aplicación, añadir nuevas características, corregir errores, etc.
- Suele ser local al desarrollador (y necesitar de un repositorio)

### Entorno de pruebas

- Se utiliza para ejecutar las pruebas unitarias, alguna prueba de integración, etc.

## Entornos

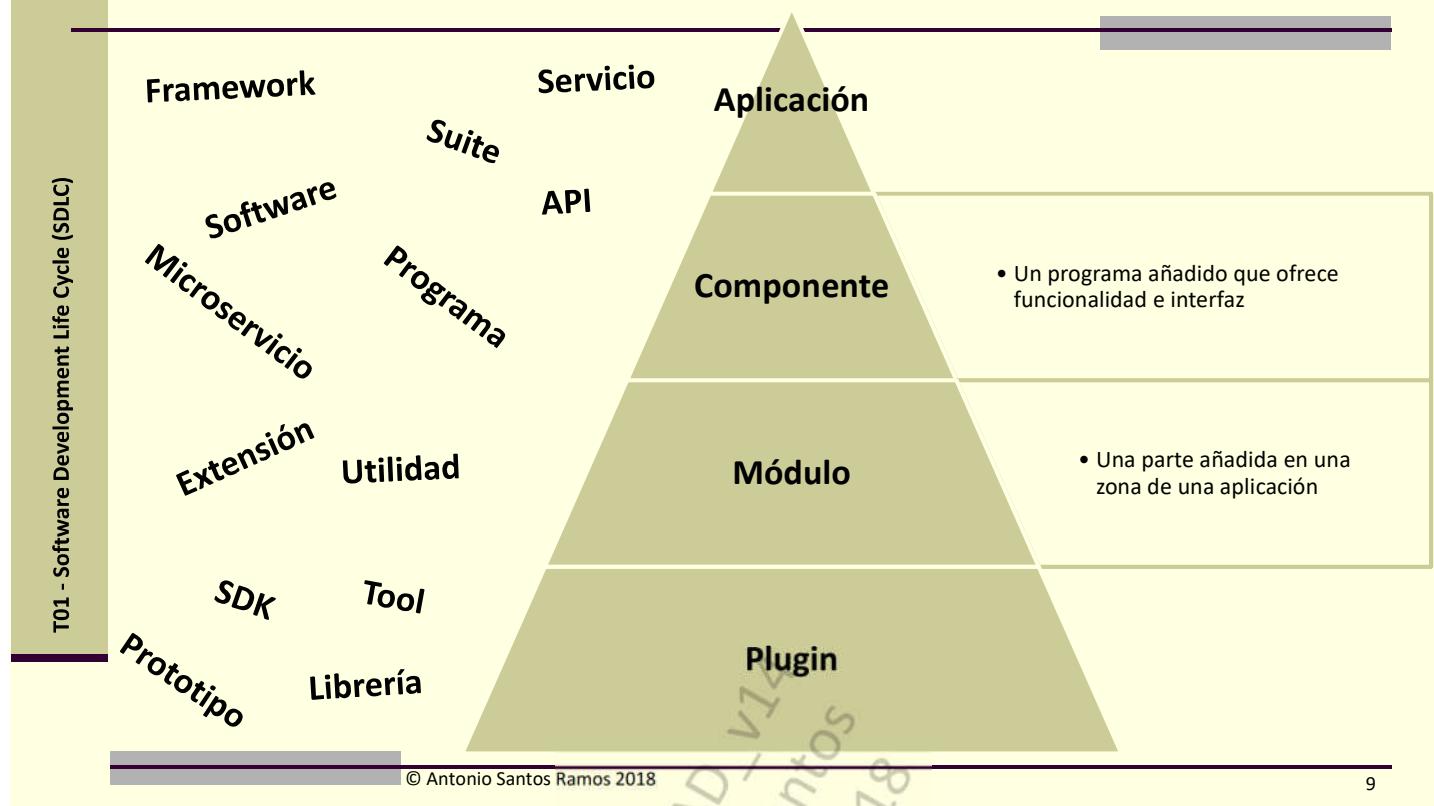
### Entorno de Pre-producción (“staging”)

- Suele ser idéntico al de Producción
- Se emplea para hacer pruebas finales, validar con usuarios, comprobar carencias, etc.

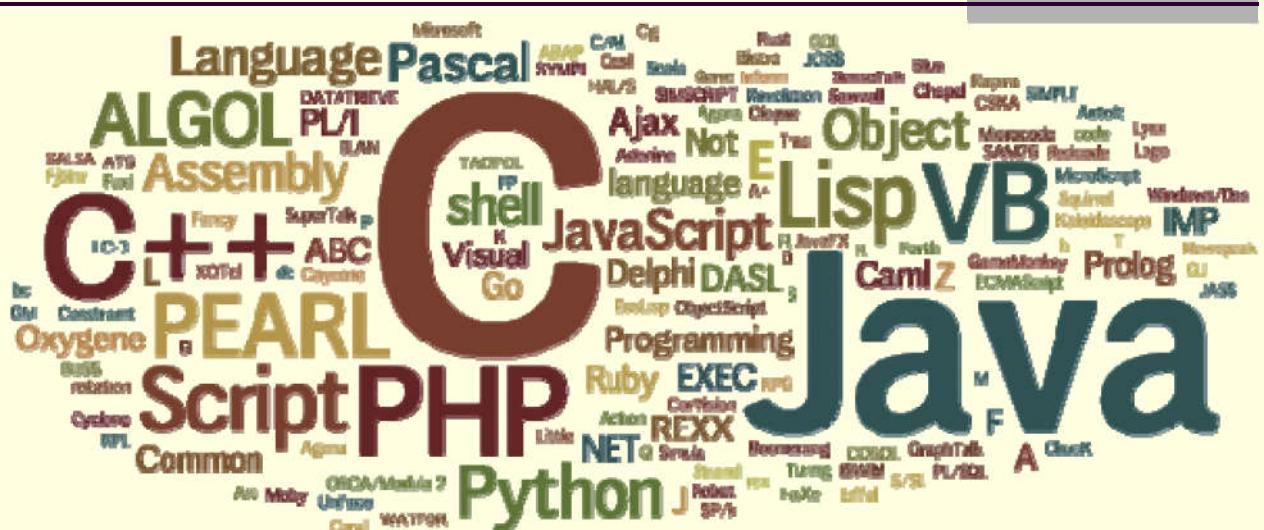
### Entorno de producción

- Entorno real en el que los usuarios finales ejecutan la aplicación.
- Hay que tener cuidado porque suele manejar datos reales

# ¿Qué fabrico? ¿Qué uso? ¿Qué fusilo?



# ¿Qué soy?



# Full Stack Developer

# Back End Developer

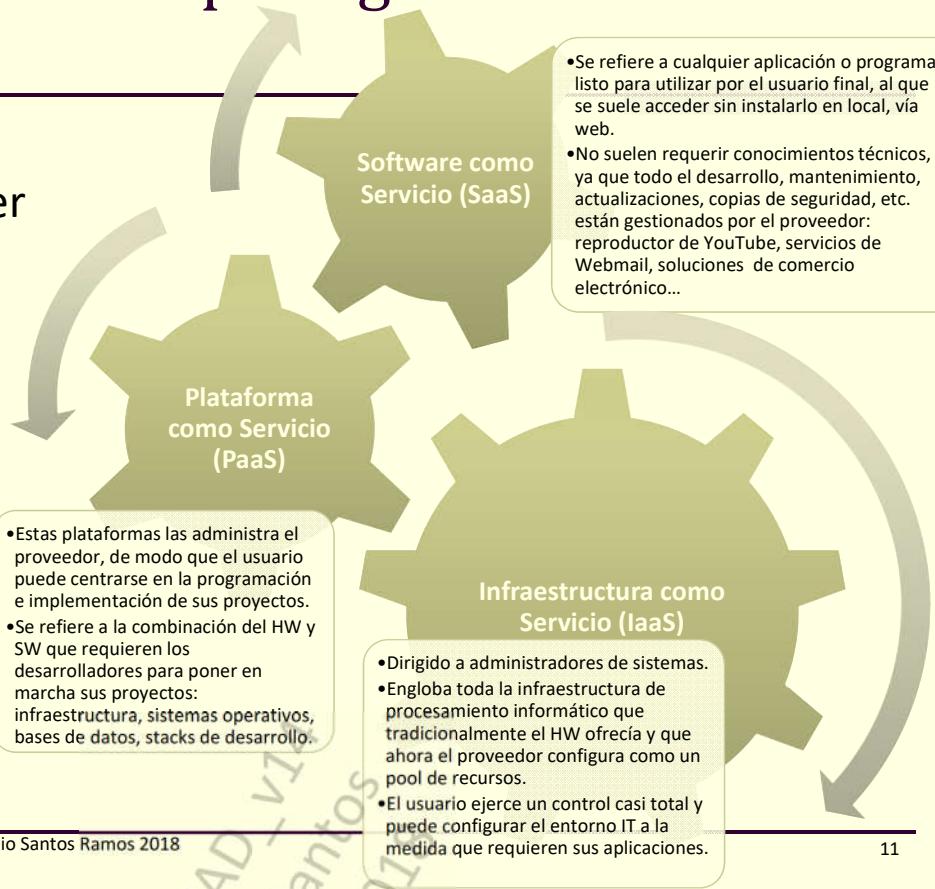
# Front End Developer

# Web Developer

UI  
Developer

# Cloud Computing. Servicios

- Los modelos XaaS (cualquier cosa como un servicio) son flexibles y fáciles de utilizar y se paga por uso de recursos



11

## Leyes y normativas (I/II)

- Propiedad intelectual de los elementos
  - Uso de imágenes
  - Uso de textos apropiados
- Compra de productos royalty-free
- Incluir Política de Privacidad
- Datos personales sensibles
- Licencias
  - Open Source GNU GPL: <http://www.gnu.org/licenses/gpl.html>
  - Open Source MIT: <http://opensource.org/licenses/MIT>
  - CreativeCommons: <http://es.creativecommons.org/blog/licencias>

### Tipos de Licencias de Contenidos Abiertos



creative commons

[http://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n\\_de\\_licencias\\_de\\_software\\_libre](http://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n_de_licencias_de_software_libre)

<http://blog.koalite.com/2013/07/licencias-de-software-cuando-puedo-usar-que/>

<http://pedalogica.blogspot.com.es/2012/06/que-son-los-contenidos-abiertos.html>

# Leyes y normativas (II/II)



**Reconocimiento (by):** Se permite cualquier explotación de la obra, incluyendo una finalidad comercial, así como la creación de obras derivadas, la distribución de las cuales también está permitida sin ninguna restricción.



**Reconocimiento – NoComercial (by-nc):** Se permite la generación de obras derivadas siempre que no se haga un uso comercial. Tampoco se puede utilizar la obra original con finalidades comerciales.



**Reconocimiento – NoComercial – CompartirlGual (by-nc-sa):** No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.



**Reconocimiento – NoComercial – SinObraDerivada (by-nc-nd):** No se permite un uso comercial de la obra original ni la generación de obras derivadas.



**Reconocimiento – CompartirlGual (by-sa):** Se permite el uso comercial de la obra y de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.



**Reconocimiento – SinObraDerivada (by-nd):** Se permite el uso comercial de la obra pero no la generación de obras derivadas.

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

**T02**

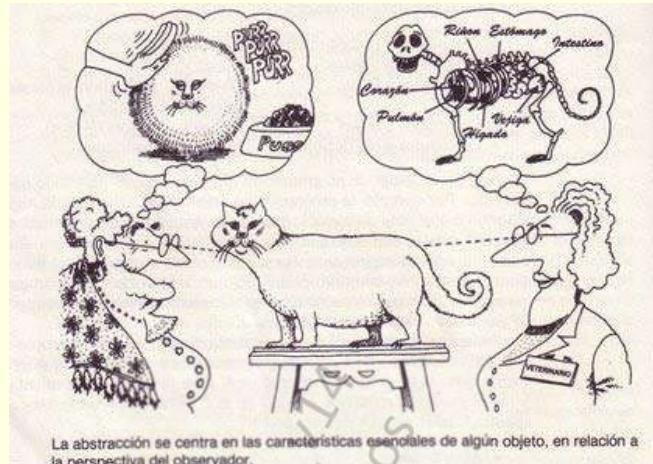
UML Modeling

# POO. Abstracción



## ■ Abstracción

- Capacidad que permite representar las características esenciales (atributos y propiedades) de un objeto sin preocuparse de las restantes características (no esenciales).



La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

© Antonio Santos Ramos 2018

15

# POO. Metodología



Análisis OO	Diseño OO	Programación OO
	<div style="border: 1px solid black; padding: 10px;"><p>Avion</p><hr/><ul style="list-style-type: none"><li>- codigo: String</li><li>- modelo: String</li><li>+ aterrizar() : void</li><li>+ despegar() : void</li></ul></div>	<pre>public class Avion{     private String modelo;     private String codigo;      public void aterrizar(){         ...     }     public void despegar(){         ...     } }</pre>

© Antonio Santos Ramos 2018

16

## ■ UML (Lenguaje Unificado de Modelado)

<http://www.uml.org/>

Lenguaje gráfico para especificar, visualizar y documentar esquemas de sistemas de software orientado a objetos.

- Controlado por el grupo de administración de objetos (OMG).
- Es el estándar de descripción de esquemas de software.
- Incorpora 14 diagramas: de clase, de secuencia, de estado, de colaboración, de actividad...

<http://holub.com/uml/>

<http://www.uml-diagrams.org>

<http://docs.kde.org/stable/es/kdesdk/umbrello/uml-basics.html>

<http://www.tutorialspoint.com/uml/>

## ■ NOTA: Todos los diagramas/gráficos usados en el curso serán UML

© Antonio Santos Ramos 2018

17



# Diagramas UML. Diagramas de estructura

Diagrama de clases	Diagrama de objetos	Diagrama de paquetes	Diagrama de despliegue	Diagrama de estructura compuesta	Diagrama de componentes
<ul style="list-style-type: none"> <li>• Describe los diferentes tipos de objetos en un sistema y las relaciones existentes entre ellos.</li> <li>• Dentro de las clases muestra las propiedades y operaciones, así como las restricciones de las conexiones entre objetos.</li> </ul>	<ul style="list-style-type: none"> <li>• Foto de los objetos de un sistema en un momento del tiempo.</li> <li>• Obsoletos a partir de UML 2.5</li> </ul>	<ul style="list-style-type: none"> <li>• Muestra la estructura y dependencia entre paquetes, los cuales permiten agrupar elementos (no solamente clases) para la descripción de grandes sistemas.</li> </ul>	<ul style="list-style-type: none"> <li>• Muestra la relación entre componentes o subsistemas software y el hardware donde se despliega o instala.</li> </ul>	<ul style="list-style-type: none"> <li>• Descompone jerárquicamente una clase mostrando su estructura interna.</li> </ul>	<ul style="list-style-type: none"> <li>• Muestra la jerarquía y relaciones entre componentes de un sistema software.</li> </ul>

© Antonio Santos Ramos 2018

18

[http://en.wikipedia.org/wiki/List\\_of\\_Unified\\_Modeling\\_Language\\_tools](http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools)

# Diagramas UML.

## Diagramas de Comportamiento

### Diagrama de casos de uso

- Permite capturar los requerimientos funcionales de un sistema.

\*

### Diagrama de estado

- Permite mostrar el comportamiento de un objeto a lo largo de su vida.

### Diagrama de actividad

- Describe la lógica de un procedimiento, un proceso de negocio o workflow.

### Diagramas de interacción

- Describen cómo los grupos de objetos colaboran para producir un comportamiento. Son los siguientes:

#### Diagrama de secuencia

- Muestra los mensajes que son pasados entre objetos en un escenario.

\*

#### Diagrama de comunicación

- Muestra las interacciones entre los participantes haciendo énfasis en la secuencia de mensajes.

#### Diagrama de colaboración

- (Sólo en UML 1.x)
- Muestra las interacciones organizadas alrededor de los roles.

#### Diagrama de interacción

- Se trata de mostrar de forma conjunta diagramas de actividad y diagramas de secuencia.

#### Diagrama de tiempo

- Pone el foco en las restricciones temporales de un objeto o un conjunto de objetos.

# POO. Clases



## Clase

- Plantilla o molde para construir objetos.
- Una clase contiene
  - **atributos** (estado): propiedades comunes a los objetos
  - **métodos** (comportamiento): servicios que proporcionan todos los objetos de esa clase para operar con ellos
- En Java se define mediante la palabra **class**.

```
[public] class MyClase {
    // cuerpo de la clase
}
```

```
...
MyClase unObjeto;
```

### Lavadora

- |               |
|---------------|
| - marca       |
| - ...         |
| - ...         |
| + programar() |
| + ...         |
| + ...         |
| + ...         |
| + ...         |

```
public class Lavadora{
    public String marca;
    public int capacidad;
}
```

# POO. Objetos



## ■ Objeto

- Un objeto (o **instancia** de una clase) es una encapsulación abstracta de información, junto con los métodos o procedimientos para manipularla.
  - Informalmente es “una clase con valores concretos”
- En programación un objeto contendrá
    - Operaciones que definen su comportamiento
    - Atributos con valores concretos que definan su estado

lavad1:Lavadora

marca: Balay  
modelo:BL123  
capacidad:5  
estado:centrifugando

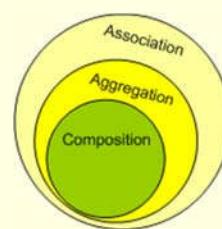
## Diagrama de clases UML (I/II)



## ■ Diagrama de clases

- Diagrama estructural usado en la parte de diseño que muestra las clases, interfaces y sus relaciones
  - Comunica un aspecto de la vista de diseño estática
    - Contiene los elementos esenciales para comprender ese aspecto
    - No debe dejar nociones semánticas “al aire”
- Relaciones UML empleadas entre clases

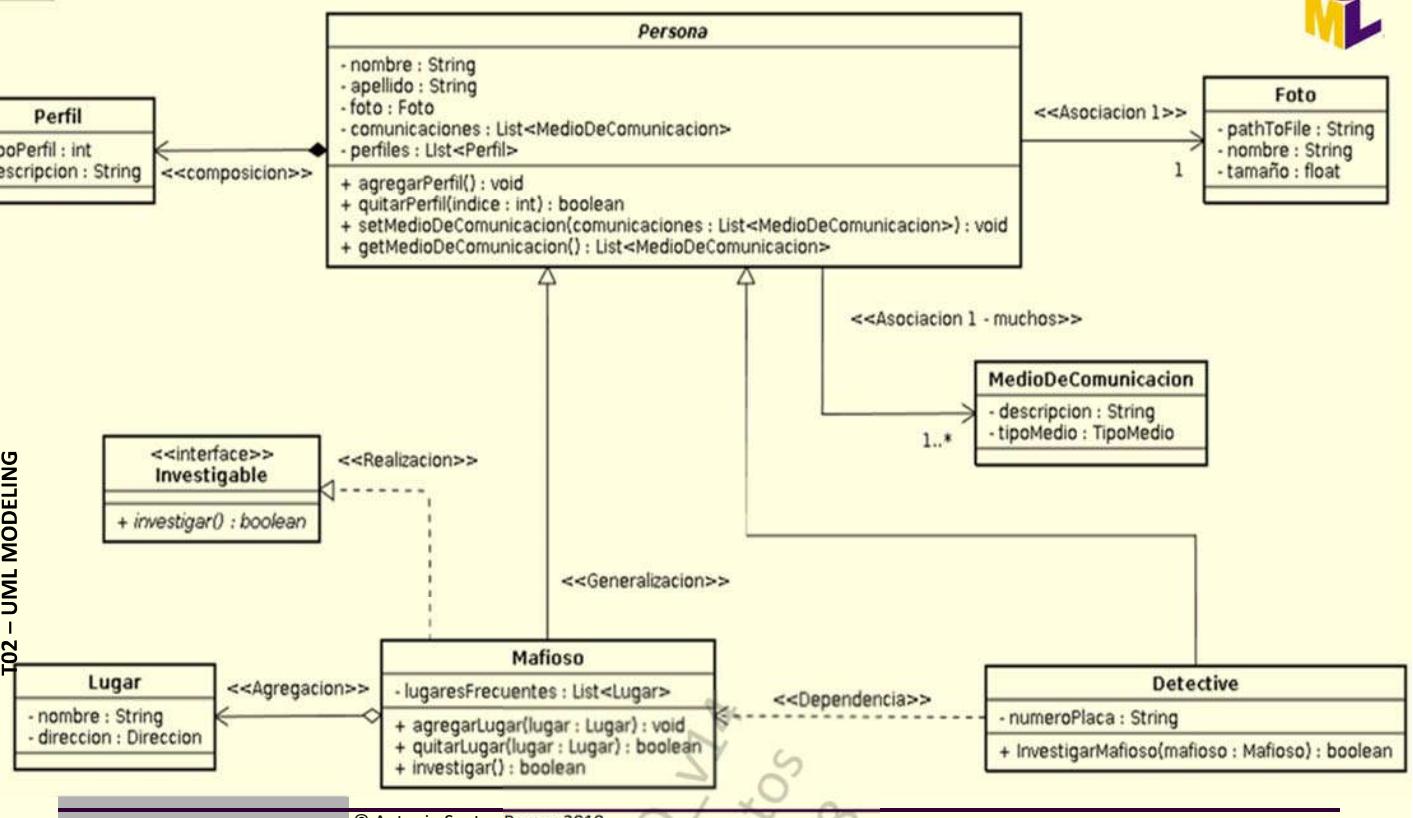
- Dependencia
- Asociación
  - Agregación
  - Composición
- Generalización
- Realización



Además una asociación Implica siempre una dependencia

<http://idiotechie.com/uml2-class-diagram-in-java/>

# Diagrama de clases UML (II/II). Ejemplo

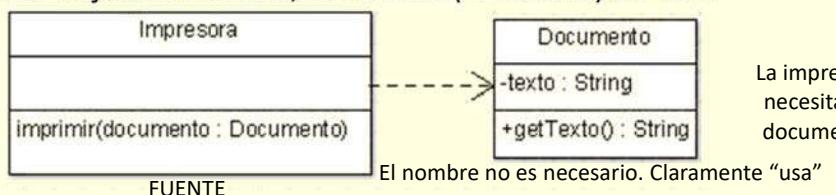


## POO. Relaciones (I/X) Dependencia (I/II)



### ■ Dependencia (...depende de ...) (...restringido a...) (...usa...)

- Relación semántica temporal entre cliente y proveedor (objetivo)
  - La fuente depende, de alguna forma, del objetivo.
  - Si el objetivo cambia, el cliente (la fuente) se ve afectada

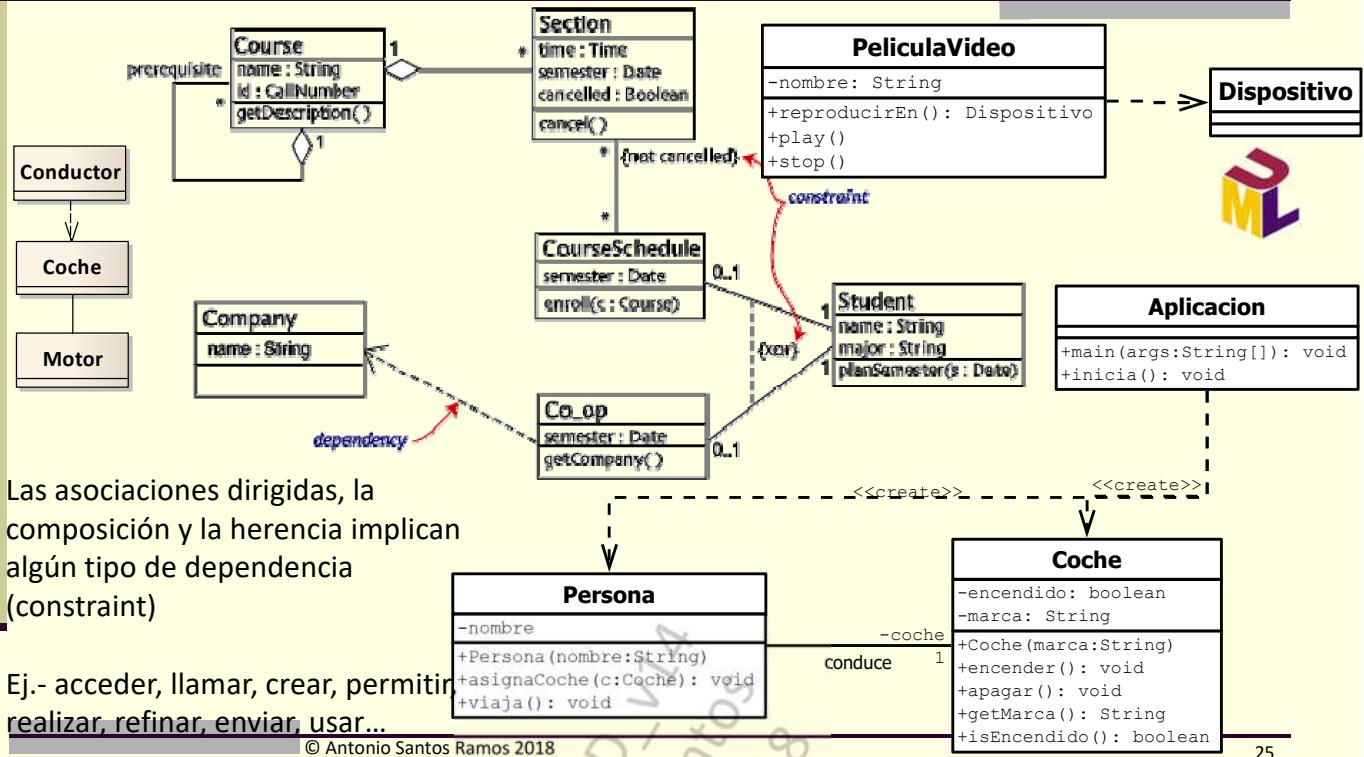


La impresora  
necesita el  
documento

- La dependencia existe si una clase:

- Tiene una variable local basada en otra clase
- Tiene una referencia directa a un objeto
- Tiene una referencia indirecta a un objeto (ej.- a través de los parámetros de operaciones)
- Usa una operación de una clase estática

# POO. Relaciones (II/X) Dependencia (II/II)



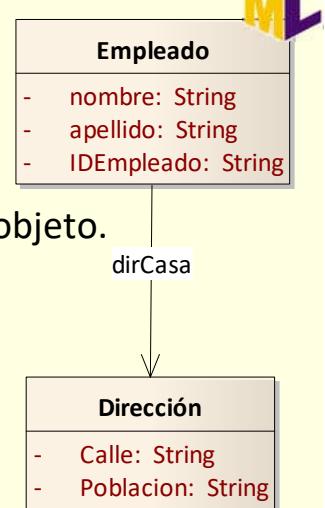
# POO. Relaciones (III/X) Asociación (I/IV) “has...a”

## ■ Asociación (...conoce...)(...trabaja para...)

- Relación estructural permanente (roles) que describe conexiones entre objetos.
  - El estado de la clase A contiene la clase B
  - Un objeto accede a atributos y métodos de otro objeto.

```
public class Empleado {
    private String nombre;
    private String apellido;
    private String IDEmpleado;
    private Direccion dirCasa;
}

public class Direccion {
    private String Calle;
    private String Poblacion;
}
```



Significa que el objeto Empleado contiene en su estado un objeto de tipo Direccion

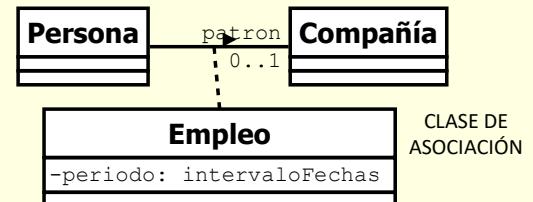
# POO. Relaciones (IV/X) Asociación (II/IV)



- Puede ser “one-way” o bidireccional (no flecha) y tener multiplicidad



- Un atributo también puede ser una clase asociada a otra (en la línea indicas el nombre y un guión)



<http://blog.jotadeveloper.es/tag/multiplicidad/>

# POO. Relaciones (V/X) Asociación (III/IV)

## ■ Agregación (...es accesorio de...) (... usa ...)

- Tipo de asociación que define que un objeto es parte de otro objeto (es un atributo de ese objeto). Comparto info



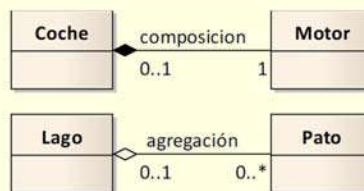
## ■ Composición (...es imprescindible para...) (... pertenece ...)

- Asociación que define que un objeto es parte imprescindible de otro objeto (es decir, es un atributo de ese objeto)
- El Objeto B no tiene sentido sin el objeto A. No lo comarto
- La eliminación de uno supone la eliminación del otro



```

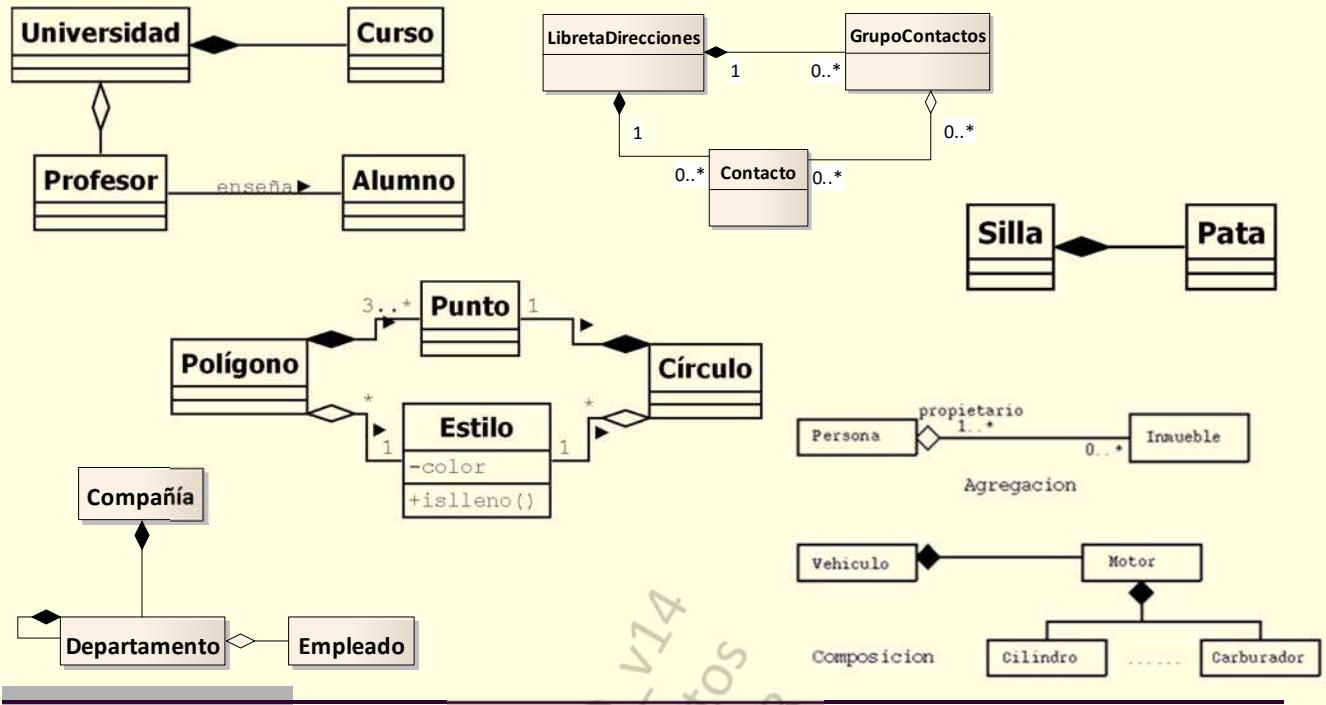
public class Coche {
    private Motor m;
}
  
```



```

public class Lago {
    private Pato[] p;
}
  
```

# POO. Relaciones (VI/X) Asociación (IV/IV)



# POO. Relaciones (VII/X) Generalización(I/II) “is...a”



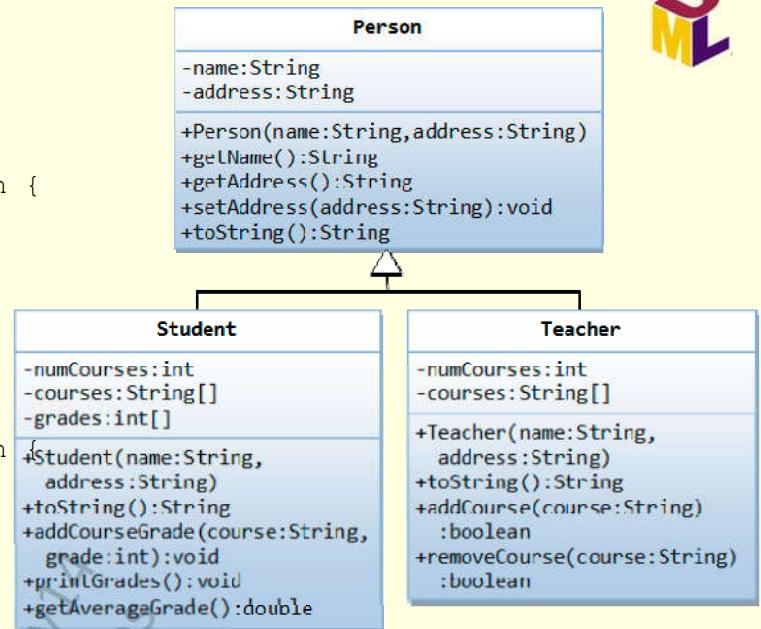
- Definición (alias Herencia) (“es un”)
  - Relación en el cual el hijo (elemento especializado) se basa en la especificación del padre (elemento generalizado)
- ¿Cómo funciona?
  - Las subclases “heredan” atributos y métodos de las superclases (excepto constructores).
    - Se agrupan las partes comunes en una misma clase (clase padre).
    - Se crean otras clases (hijas) con las partes no comunes.
- En Java
  - Heredan de la clase padre mediante la palabra **extends**
  - No existe herencia múltiple (simulada con interfaces)

# POO. Relaciones (VIII/X) Generalización(II/II)

```
public class Person {
    private String name;
    private String address;
    ...
}

public class Student extends Person {
    private int numCourses;
    private String[] courses;
    private int[] grades;
    ...
}

public class Teacher extends Person {
    private int numCourses;
    private String[] courses;
    ...
}
```



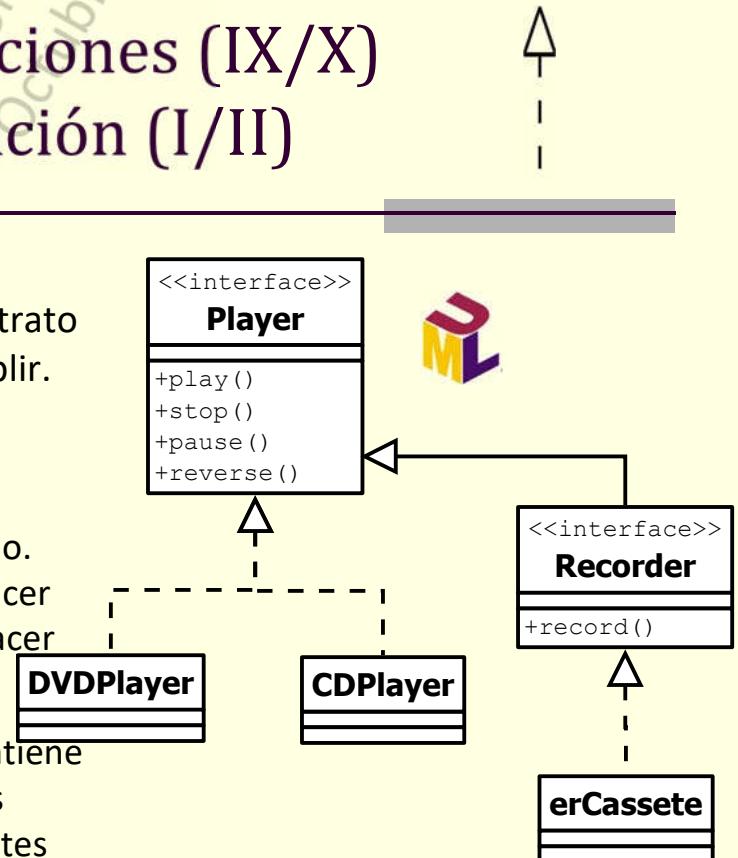
© Antonio Santos Ramos 2018

31



# POO. Relaciones (IX/X) Realización (I/II)

- Realización
  - Una clase especifica un contrato que la otra clase debe cumplir.
- Definiciones
  - Método abstracto
    - Método no implementado. Especifica QUÉ se va a hacer pero no CÓMO se va a hacer
  - Interface
    - Es una clase que sólo contiene métodos abstractos y sus propiedades son constantes
- Semántica: ...se comporta como...



© Antonio Santos Ramos 2018

32

# POO. Relaciones (X/X) Realización (II/II)

## ■ Definición para Java

- Informalmente es una clase con **todos** sus métodos abstractos
- Sólo detalla declaraciones de métodos. No implementa

```
01. import java.awt.Rectangle;  
02.  
03. public interface Drawable {  
04.     int MAX_WIDTH = 1024;  
05.  
06.     public void draw();  
07.     Rectangle getDimensions();  
08.     void resize(int w, int h);  
09. }  
                                public class dibujo implements Drawable
```

- Una interface sería similar a un clase sin atributos ¿¿??
- No puede ser instanciada. Sólo puede ser implementada... es decir la clase que la implemente asume su comportamiento

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)



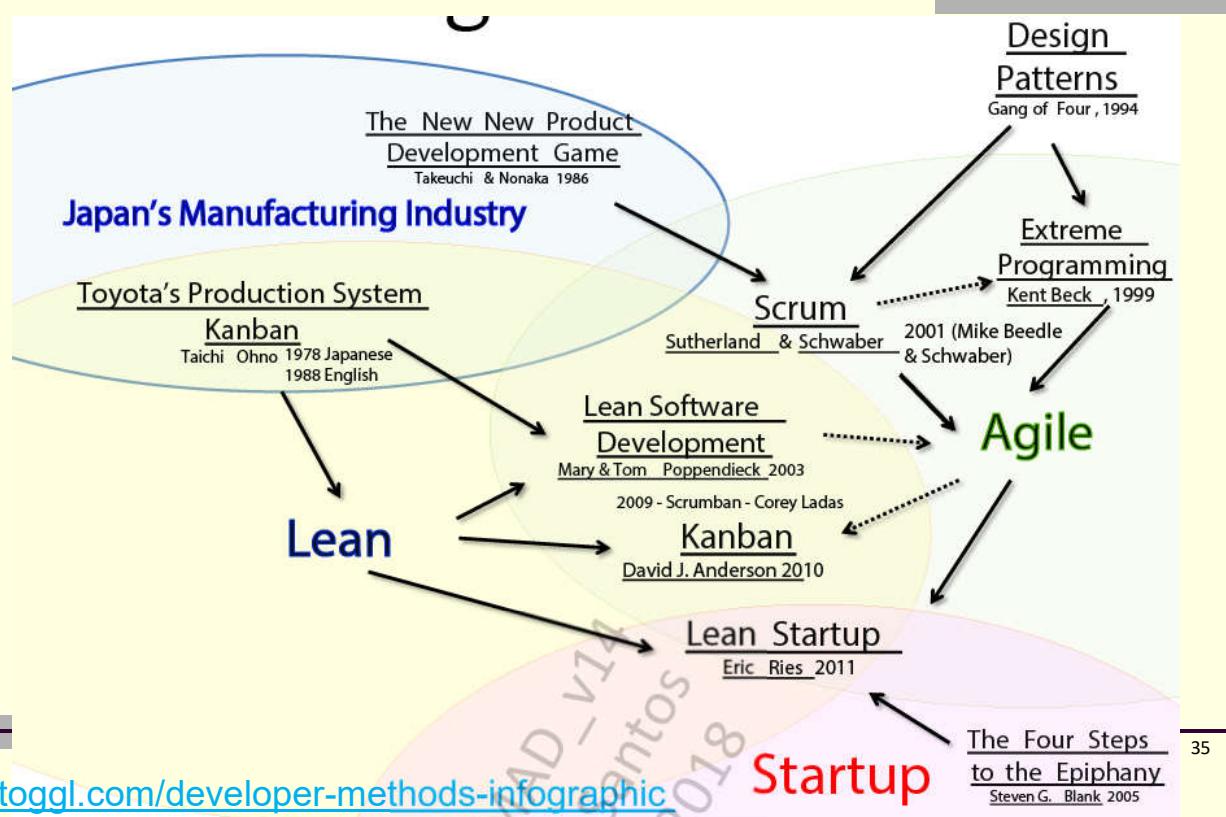
Scrum.org

T03



Metodologías Ágiles: Scrum

# Metodologías...



## Metodologías Ágiles vs. Metodologías tradicionales (I/III)



<https://dzone.com/articles/agile-dictionary-misunderstandings-can-be-fatal>

# Metodologías Ágiles vs. Metodologías tradicionales (II/III)



© Antonio Santos Ramos 2018

37

# Metodologías Ágiles vs. Metodologías tradicionales (III/III)

Metodologías Agiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Mas artefactos
Pocos roles	Mas roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

© Antonio Santos Ramos 2018

38

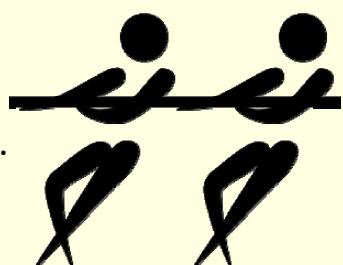
# Metodología Scrum



- **Scrum** es una metodología de desarrollo muy simple, no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto.
- Características
  - Marco de trabajo en procesos agiles.
  - Equipos auto-organizados
  - Producto desarrollado en iteraciones cortas (2–4 semanas)
    - Entregas frecuentes y regulares
  - Valor real del negocio
  - Los requerimientos pueden cambiar de forma rápida
  - Colaboración estrecha con cliente

## Sprint

- Periodo de tiempo durante el que se desarrolla un incremento de funcionalidad.
  - Duración: 2 – 4 semanas. Ni más ni menos.
  - Durante el sprint se diseña, codifica y prueba el incremento.
- Sólo es posible cambiar el curso de un sprint, abortándolo.
  - Sólo lo puede hacer el Scrum Master si decide que no es viable por alguna de las razones siguientes:
    - La tecnología acordada no funciona.
    - Las circunstancias del negocio han cambiado.
    - El equipo ha tenido interferencias.
- Cada proyecto dispondrá de n sprint



# Proceso SCRUM. Básico



- Al iniciar cada iteración, el equipo revisa el trabajo pendiente del proyecto
- Selecciona la parte que terminará como un incremento de funcionalidad incorporado al software al terminar la iteración.
- Al final de la iteración el equipo presenta el incremento de funcionalidad a las partes implicadas en el proyecto.

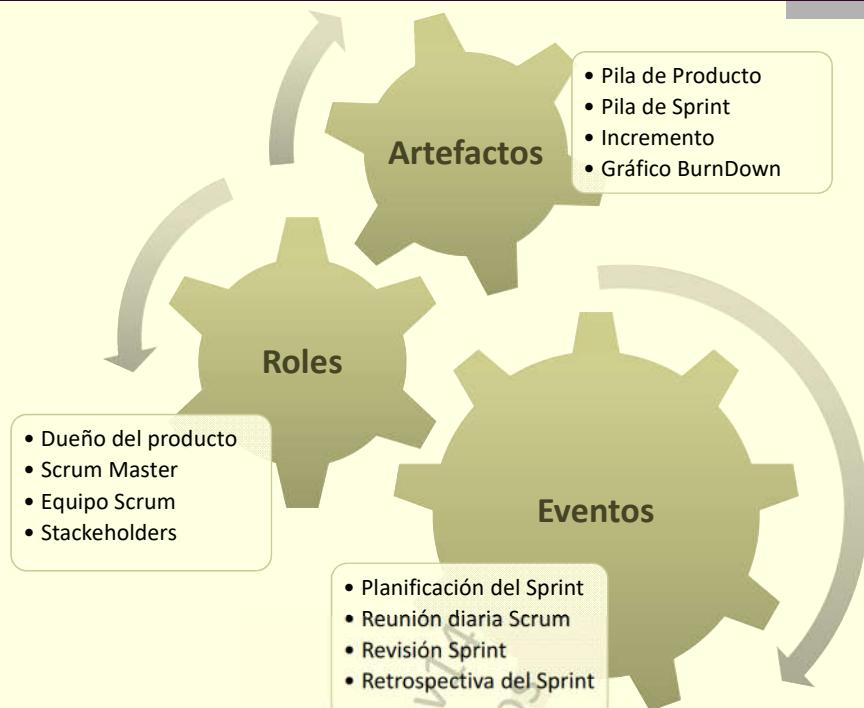
# Proceso SCRUM. Detallado

## The Agile: Scrum Framework at a glance

Inputs from Executives,  
Team, Stakeholders,  
Customers, Users



# Elementos de trabajo de Scrum



© Antonio Santos Ramos 2018

43

## Roles (I/IV)



### Roles



#### DUEÑO DEL PRODUCTO (product owner)

Responsable del alcance y del plan de entregas



#### SCRUM MASTER

Encargado de velar por el óptimo funcionamiento del proceso de desarrollo, facilitando el día a día de cada uno de los miembros del equipo.



#### EQUIPO

Construye el producto



#### INTERESADOS (stakeholders)

Interesados en el éxito del producto



© Antonio Santos Ramos 2018

44

# Roles (II/IV)

## Dueño del producto (Product Owner)

### ■ Realiza las siguientes tareas

- Acepta y/o rechaza los resultados e incrementos
- Decide fechas y contenido de las entregas
- Se responsabiliza de la rentabilidad del producto
- Representa a todos los interesados en el proyecto
- Define las funcionalidades del producto según su visión.
- Prioriza las funcionalidades según el valor del mercado
- Ajusta las funcionalidades y prioriza cada iteración.
- Adapta el producto de cara a sus necesidades de negocio
- Constantemente re-prioriza el Backlog del Producto, ajustando las expectativas a largo plazo



# Roles (III/IV)

## Scrum Master

### ■ Es el responsable de que se cumplan los valores y las prácticas de Scrum.

### ■ Representa “la gestión del proyecto”

### ■ No tiene autoridad en la gestión del equipo

### ■ Realiza las siguientes tareas

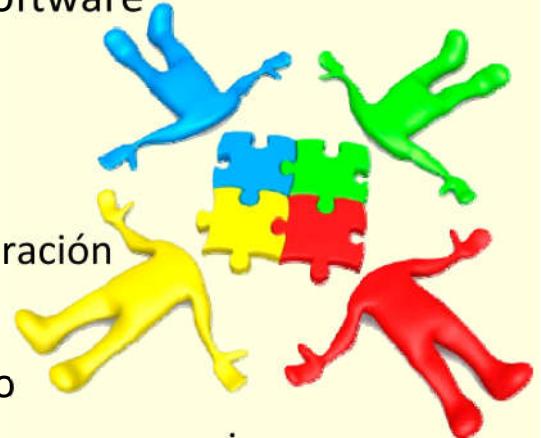
- Elimina obstáculos y resuelve conflictos
- Evita interferencias
- Mantiene enfocada la meta del sprint
- Garantiza que el equipo es funcional y productivo.
- Todas las reuniones son facilitadas por el ScrumMaster



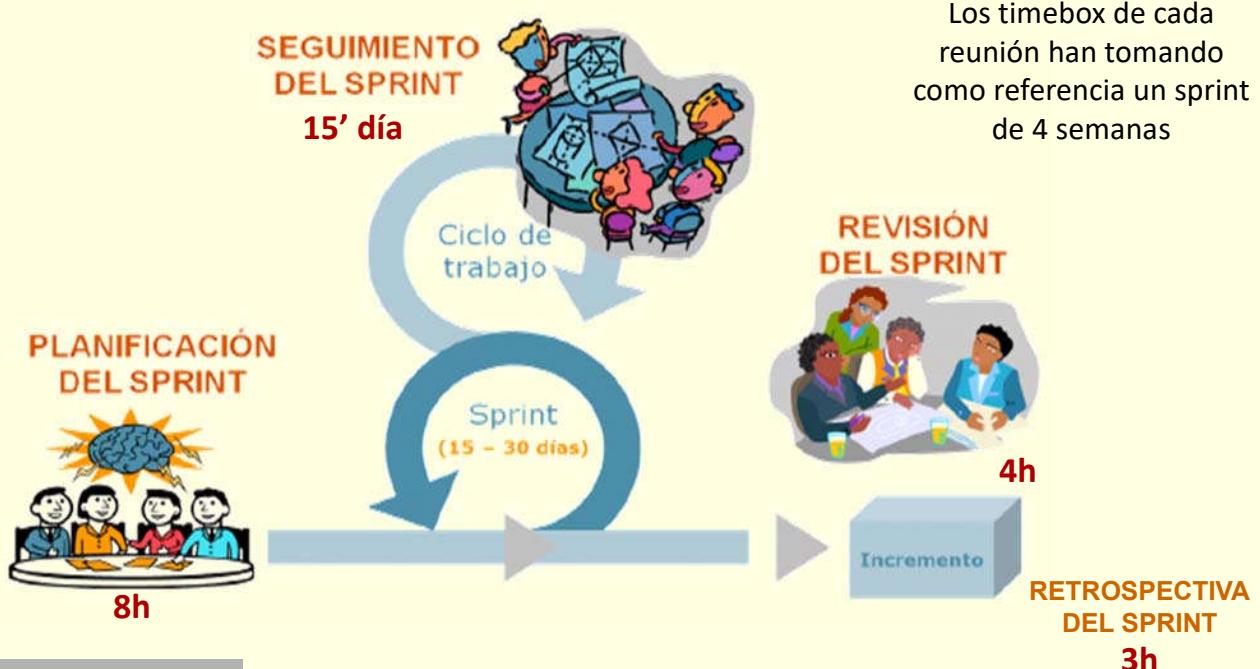
# Roles (IV/IV)

## Equipo de desarrollo (Scrum Team)

- Responsable de transformar la pila del sprint en un incremento de la funcionalidad del software
- Características
  - $7 \pm 2$  personas. Multifuncional
  - Auto-organizado. Alto grado de autonomía, responsabilidad y colaboración
- Realiza las siguientes tareas
  - Define las tareas y estima su esfuerzo
  - Construye el producto que el cliente va a consumir
  - Convierte el Backlog de Producto en software potencialmente entregable.



# Eventos (I/VIII)



# Eventos (II/VIII)

## Tipos de reuniones



© Antonio Santos Ramos 2018

49

<http://www.becominganagilearchitect.com/scrum-eventos-reunion-de-grooming>

# Eventos (III/VIII)

## Planificación del Sprint (I/III) (Max. 8h)

- **Objetivo:** a partir de las prioridades de negocio del cliente, se determinan las funcionalidades que se incorporarán al producto en el siguiente sprint
- **Participantes:** Equipo, ScrumMaster, Product Owner
- **Reglas a seguir:**
  - Reunión dividida en dos partes
    - Parte I: Qué se entregará al terminar el Sprint
      - El Product Owner expone requisitos de mayor prioridad e identifica el objetivo del Sprint.
      - El equipo pregunta y solicita aclaraciones proponiendo sugerencias, modificaciones y alternativas. Se elabora una frase-objetivo.
    - Parte II: Cómo se conseguirá el incremento
      - El equipo desglosa cada funcionalidad en tareas y estima el tiempo para cada una
      - Se crea un sprint backlog con tareas distribuidas por decisión del equipo
      - Se autoasignan las primeras tareas
- **Al finalizar:**
  - El SM asegura que estén establecidos los elementos de la pila a ejecutar, el **objetivo del sprint, la pila del sprint con tareas estimadas y la duración final**

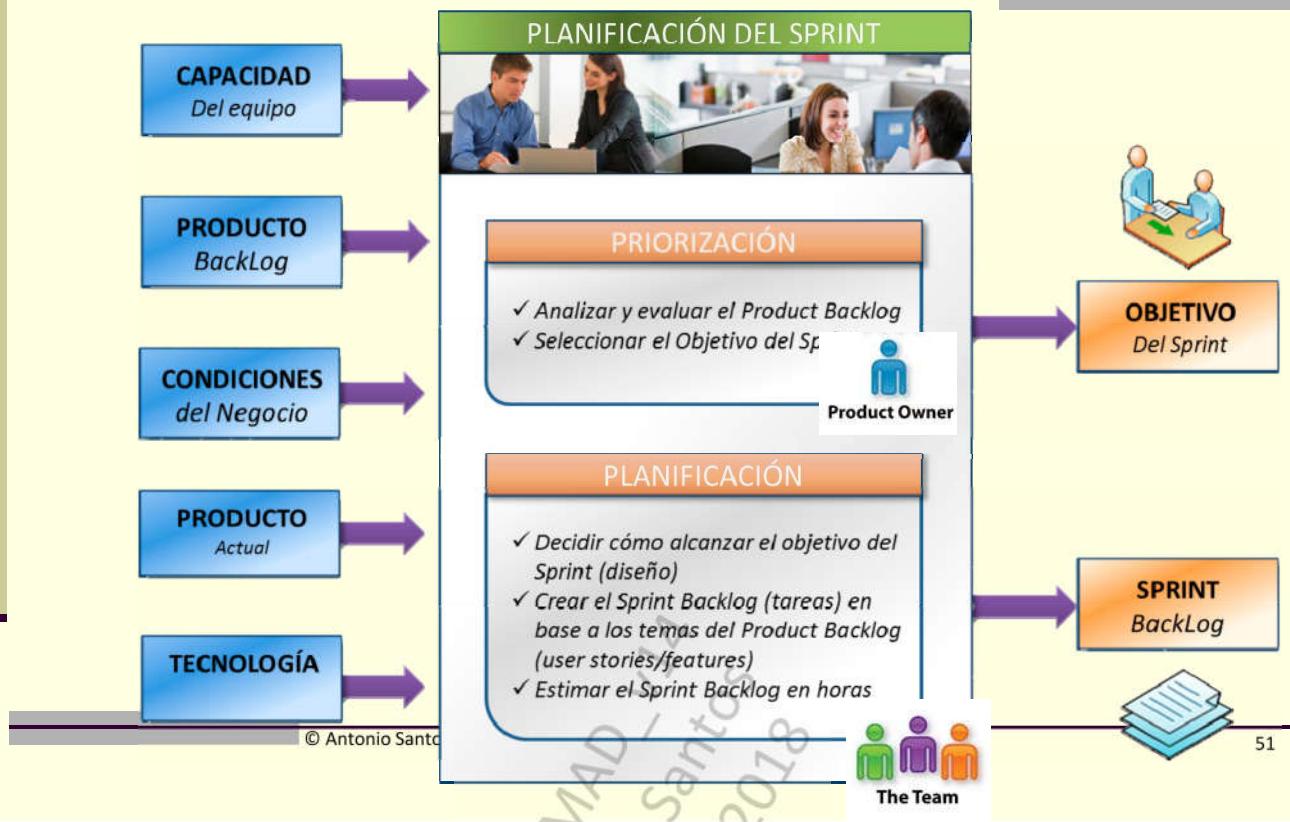
© Antonio Santos Ramos 2018

50

[https://www.scrummanager.net/bok/index.php?title=Planificación\\_del\\_sprint](https://www.scrummanager.net/bok/index.php?title=Planificación_del_sprint)

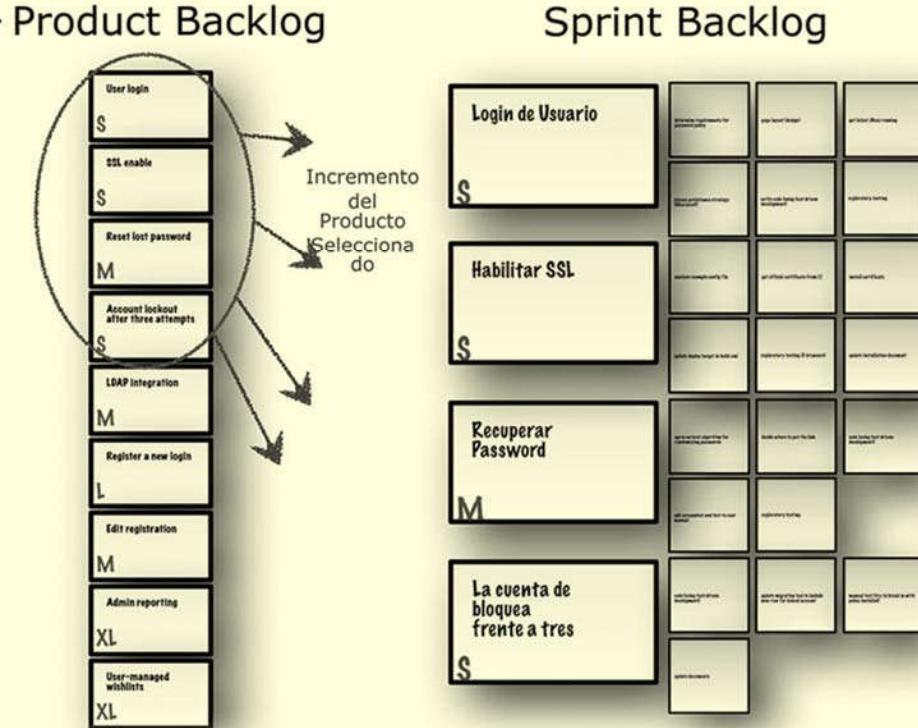
# Eventos (IV/VIII)

## Planificación del Sprint (II/III) (Max. 8h)



# Eventos (V/VIII)

## Planificación del Sprint (III/III)



# Eventos (VI/VIII)

## Reunión de Scrum diaria (Max. 15')

- **Objetivo:** Reunión diaria breve (max. 15'), en la que el equipo sincroniza el trabajo y establece el plan para las 24 horas siguientes.
- **Participantes:** Equipo, ScrumMaster, Product Owner (Opcional)
- **Reglas a seguir:**
  - Moderada por el Equipo. Se pregunta a todos los asistentes
    - ¿Qué trabajo has realizado desde la última reunión?
    - ¿Qué tienes previsto para hoy? ¿Qué necesitas? ¿Tienes algún problema?
  - Sólo habla la persona que informa de su trabajo, el resto escucha y no hay lugar para otras conversaciones.
  - Cuando un miembro informa de algo de interés para otros, o necesita ayuda de otros, estos se reúnen al terminar la revisión diaria.
- **Al finalizar:**
  - Pila del sprint y gráfico de avance (burn-down) actualizados.
  - El SM realiza las gestiones adecuadas para resolver los impedimentos.

© Antonio Santos Ramos 2018

53

# Eventos (VII/VIII)

## Revisión del Sprint (Max 4h)

- **Objetivo:** Presentar al Product Owner y demás involucrados el incremento realizado durante el Sprint.
- **Participantes:** Equipo, ScrumMaster, Product Owner, Stakeholder
- **Reglas a seguir:**
  - Preparación previa: max. 1 hora
  - Sólo se presentan funcionalidades acabadas
  - Las demos son en vivo. No son un informe
  - Al finalizar la reunión se pide opiniones a participantes. Estos pueden aportar cambios y mejoras.
  - El Product Owner decide elementos acabados. Los no acabados vuelven a la Pila del Producto
- **Al finalizar:**
  - Se actualiza y vuelve a priorizar el Product Backlog
  - El ScrumMaster anuncia el lugar y fecha de la próxima



© Antonio Santos Ramos 2018

54

# Eventos (VIII/VIII)

## Retrospectiva del Sprint (Max 3h)

- **Objetivo:** Identificar aspectos a cambiar para hacer el trabajo más productivo
- **Participantes:** Equipo, ScrumMaster, Product Owner (Opcional)
- **Reglas a seguir:**
  - El Scrum Manager anota pero no proporciona respuestas
  - Propone dos preguntas que todos responden
    - ¿Qué cosas hicimos bien?
    - ¿Qué cosas podemos mejorar?
  - Se debate todo aquello que afecta a como el equipo construye software
- **Al finalizar:**
  - Tras la exposición y comentario se deciden las acciones que se van a tomar, como grupo, en el siguiente Sprint.



© Antonio Santos Ramos 2018

55

<https://dosideas.com/noticias/retrospectivas/876-las-5-etapas-de-una-retrospectiva-efectiva>

<https://gssolutionsgroup.wordpress.com/2013/05/01/seven-steps-to-remarkable-retrospectives/>

# Artefactos (I/IX)

## Pila del producto (product backlog)

- Lista de requisitos de usuario, que a partir de la visión inicial del producto crece y evoluciona durante el desarrollo.
- Es responsabilidad del PO.

## Pila del sprint (sprint backlog)

- Lista de los trabajos que debe realizar el equipo durante el sprint para generar el incremento previsto.

## Gráficos BurnDown

- Gráfico que se realiza a lo largo de un sprint y que muestra la desviación actual con respecto a la planificación

## Incremento

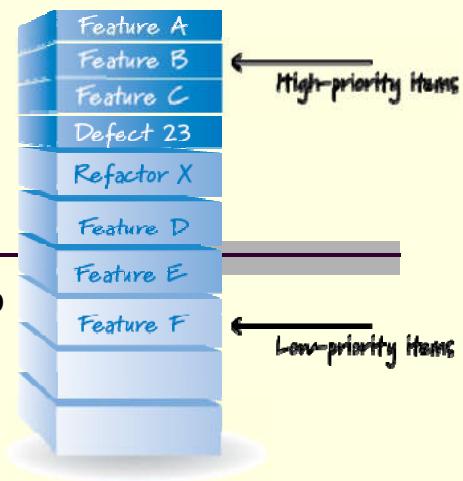
- Parte del producto desarrollada en un sprint, listas para salir a producción (pruebas, codificación limpia y documentada).

# Artefactos (II/IX)

## Pila de producto (Backlog del producto)

- Requisitos funcionales de alto nivel del producto o “Historias de usuario” que desea el cliente.
- Similar a la Especificación de Requisitos clásica
  - La Pila empleada para planificar el proyecto es sólo una estimación inicial de requisitos
- Es una lista de lo qué se va a realizar
  - Priorizada por el dueño del producto.
  - No menciona cómo se va a realizar
  - Expresado de la forma que cada requisito proporcione valor al cliente.
- Características
  - Al principio no es una lista completa y definitiva
  - Visible a todos los miembros del equipo
  - Incorpora de forma constante las necesidades actuales del sistema
  - **Se mantiene durante el ciclo de vida del producto. Cambia y evoluciona**

© Antonio Santos Ramos 2018



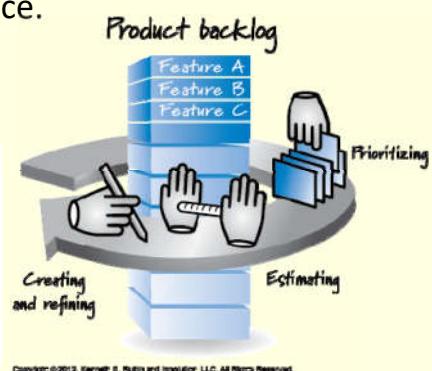
57

# Artefactos (III/IX)

## Pila de producto. Preparación

- Elementos a incorporar
  - Identificador único de la funcionalidad o trabajo.
  - Descripción de la funcionalidad/requisito, denominado “historia de usuario”.
  - Campo o sistema de priorización.
  - Estimación del esfuerzo necesario.
- Possible campos a incorporar
  - Observaciones.
  - Criterio de validación.
  - Persona asignada.
  - Nº de Sprint en el que se realiza.
  - Módulo del sistema al que pertenece.

Id	Prioridad	Módulo	Descripción	Est.	Por
1	Muy alta		Plataforma tecnológica	30	JM
2	Muy alta		Prototipos interfaz usuario	40	LR
3	Muy alta		Diseño de datos	40	LR
4	Alta	Trastienda	El operador define el flujo y textos de un expediente	60	JM
5	Alta	Trastienda	Etc...	999.	XX



Copyright ©2012, Kenner S. Rubin and Involution LLC. All Rights Reserved.

# Artefactos (IV/IX)

## Pila de Sprint (Sprint backlog)

- Es el resultado de dividir las funcionalidades de la pila de producto (“Historias de Usuario”) en tareas asignables.
  - Refleja los requisitos vistos desde el punto de vista del equipo
- Negociados entre el Dueño del producto y el Scrum Team durante la planificación del sprint
- Características
  - Se genera al principio de cada sprint
  - Incluye las tareas de la iteración actual, a quien se asigna y el esfuerzo previsto
  - Lo actualiza diariamente el equipo. El SM rehace el Sprint Burndown
  - Una tarea debería durar una jornada como máximo
- Se visualiza mediante un tablero, hoja de cálculo, etc.

# Artefactos (V/IX)

## Pila de Sprint (Sprint backlog)

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

# Artefactos (VI/IX)

## Pila de Sprint. Transformación

Product Backlog			Complejidad	Estimación inicial	Trabajo pendiente			
ID	Elemento	Sprint			1	2	3	4
1	Nuevo formulario para peticiones de clientes		2	0.2	2,4	2,4	0	0
2	Configuración de respuestas automáticas		3	0.2	3,6	3,6	0	0
3	Envío automático de respuestas		1	0.2	1,2	1,2	0	0
4	Consulta para los clientes de peticiones enviadas		1	0.2	1,2	1,2	0	0
5	Modificación del cliente de sus peticiones enviadas		2	0.2	2,4	2,4	0	0
6	Acceso a peticiones sólo para clientes del portal jurídico		5	0.2	6	6	0	6
7	Consulta de peticiones por parte del staff		1	0.2	1,2	1,2	0	0
<b>SPRINT 1</b>			15		18	18	0	0
8	Inserción de comentarios y reasignación a peticiones (staff)		2	0.2	1,2	1,2	1,2	0
9	Consultas por clientes, fechas y temas		3	0,2	3,6	3,6	3,6	0
10	[Continúa]....							

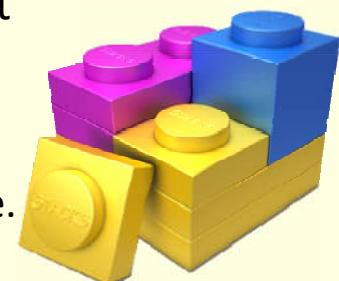
© Antonio Santos Ramos 2018

61

# Artefactos (VII/IX)

## Incremento

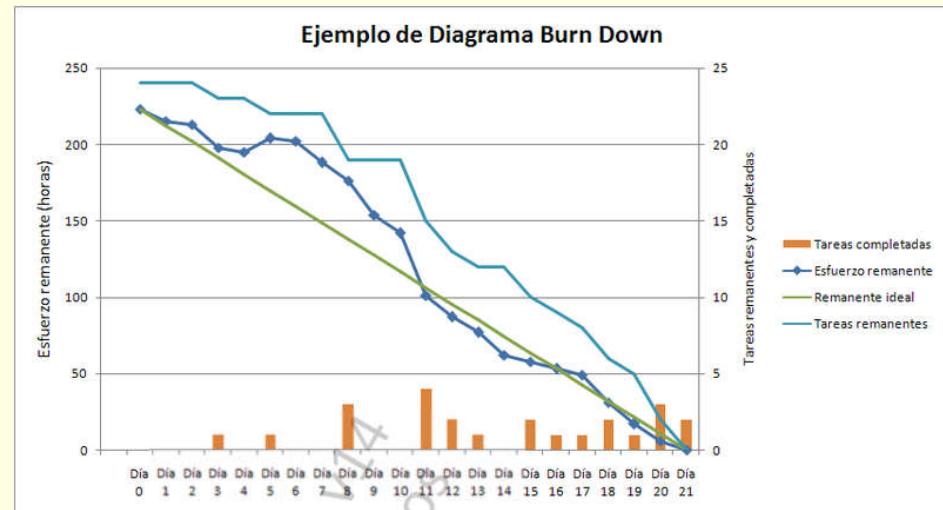
- Parte del producto producida en un sprint
- Características
  - Completamente terminada y operativa
  - En condiciones de ser entregada al cliente.
  - Cada Sprint produce un incremento
- ¿Qué no es un incremento?
  - Prototipos, módulos o sub-módulos, partes pendientes de pruebas o integración, etc.
  - El primer sprint es una excepción, ya que puede implicar especificación de la plataforma, trabajos de diseño o desarrollo de prototipos...



# Artefactos (VIII/IX)

## Gráficos burndown

- Permite saber el tiempo que falta para completar el trabajo
- Muestra la velocidad a la que se está completando los objetivos/requisitos dentro de un sprint



© Antonio Santos Ramos 2018

[https://www.scrummanager.net/bok/index.php?title=Gr%C3%A1fico\\_de\\_avance](https://www.scrummanager.net/bok/index.php?title=Gr%C3%A1fico_de_avance)<https://luis-goncalves.com/burndown-chart-ultimate-guide/>

63

# Artefactos (IX/IX)

## Gráficos burndown (Pila de Sprint)

SPRINT	INICIO	DURACIÓN	J	V	L	M	X	J	V	L	M	J	V	L
1	1-mar-07	12	1-mar	2-mar	5-mar	6-mar	7-mar	8-mar	9-mar	12-mar	13-mar	15-mar	16-mar	19-mar
			23	23	19	16	16	13	9	9	9	9	9	9
			276	246	216	190	178	158	110	110	110	110	110	110
Tareas pendientes			23	23	19	16	16	13	9	9	9	9	9	9
Horas de trabajo pendientes			276	246	216	190	178	158	110	110	110	110	110	110
SPRINT BACKLOG														
Tarea	Estado	Responsal												
Descripción de la tarea 1	Terminada	Luis	16	16	16	16	16	16	12	12	12	12	12	12
Descripción de la tarea 2	Terminada	Luis	12	8										
Descripción de la tarea 3	Terminada	Luis	4	4	4	4	4	4						
Descripción de la tarea 4	Terminada	Elena	8	4										
Descripción de la tarea 5	Terminada	Elena	16	16	4									
Descripción de la tarea 6	Terminada	Elena	6	6	2									
Descripción de la tarea 7	Terminada	Antonio	16	4										
Descripción de la tarea 8	Terminada	Antonio	16	16	20	12	4							
Descripción de la tarea 9	Terminada	Antonio	12	2										
Descripción de la tarea 10	En curso	Luis	12	12	12	12	12	12	12	12	12	12	12	12
Descripción de la tarea 11	Pendiente	Luis	8	8	8	8	8	8	8	8	8	8	8	8
Descripción de la tarea 12	Terminada	Luis	14	14	14	14	14	14	14	14	14	14	14	14
Descripción de la tarea 13	En curso	Antonio	8	8	8	8	8	8	6					

También puede indicarse el Tipo

© Antonio Santos Ramos 2018

64

# Estimaciones Historias de Usuario

- Características de las HU (o escenarios)
  - Expresan una necesidad muy específica que tiene un Usuario
  - Deben escribirse en el lenguaje del usuario para que pueda entenderlas. NO son técnicas.
  - Deben aportar valor de negocio, ser pequeñas, estimables, “testables”, ...
  - La precisión en la estimación de una historia decrece a medida que incrementa el tamaño de la misma.
- Es mejor redactar las historias haciendo referencia a roles específicos. Utiliza esta plantilla

As a <who>, I want <what> so that <why>

## Estimaciones Historia de Usuario. ¿Qué necesitan?



# Estimaciones

## Historia de Usuario. Puntos

- Las historias se estimarán preferentemente por puntos.
  - Los Puntos de Historia son el tamaño relativo de una HU comparada con otras historias estimadas por el mismo equipo.
  - No importa quién implementa la historia.
    - Una historia estimada en 5 puntos de historia, sigue siendo 5 puntos sea desarrollada por un senior o un junior.
    - Si se estima en horas o días, puede ser una tarea de 2 días para un Senior y de 1 semana para un Junior

<https://gssolutionsgroup.wordpress.com/2013/01/20/your-car-can-show-you-how-agile-estimation-works/>

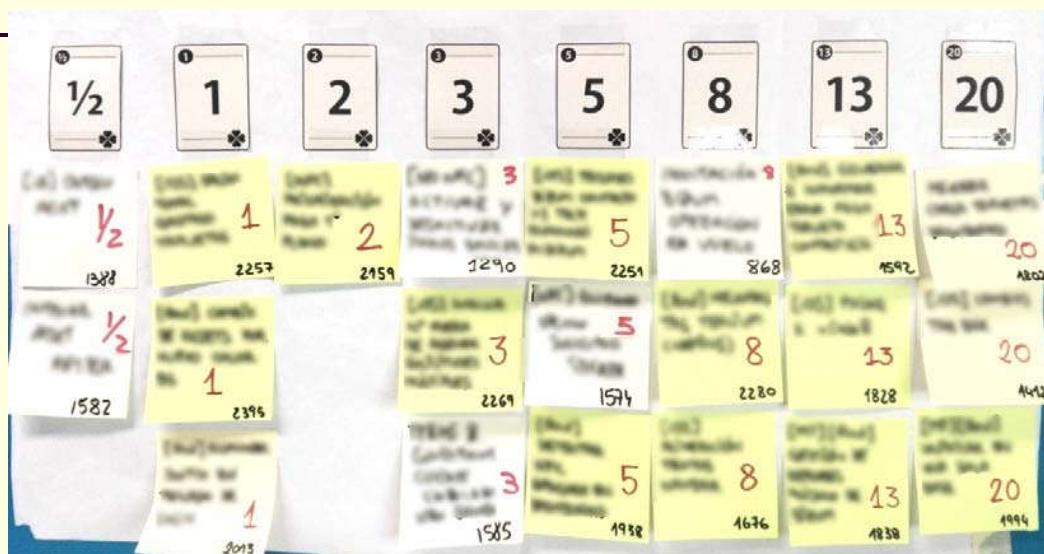
© Antonio Santos Ramos 2018

67

<https://www.safaribooksonline.com/library/view/head-first-software/9780596527358/ch04.html>

# Estimaciones

## Historia de Usuario. Puntos



<http://samuelcasanova.com/2017/04/puntos-historia-en-agile/>

<http://www.javiergarzas.com/2015/06/puntos-historia-para-estimar-y-no-horas.html>

<https://geeks.ms/devnettips/2011/02/02/estimando-en-puntos-de-historia/>

© Antonio Santos Ramos 2018

68

<https://www.visualstudio.com/en-us/docs/work/tfs-ps-sync/make-agile-team-progress-visible-to-the-pmo>

# Kanban (I/II)

- Metodología de trabajo similar a Scrum.
- Acepta la posibilidad de replantear las prioridades del trabajo a medida que las necesidades cambian.
- Kanban tiene tres reglas
  - Trabajo continuo priorizando (en Scrum hablamos de Sprint)
  - Tareas limitadas (Work In Progress) por fase (desarrollo, pruebas, etc.)
  - Incluye dos métricas: Lead (tiempo desde que algo está pendiente hasta que se termina) y Cycle (tiempo desde que se empieza a trabajar en algo hasta que se termina)

## Scrumban = Scrum + Kanban

<https://jeronimopalacios.com/2017/01/scrum-vs-kanban-combate-definitivo/>

© Antonio Santos Ramos 2018

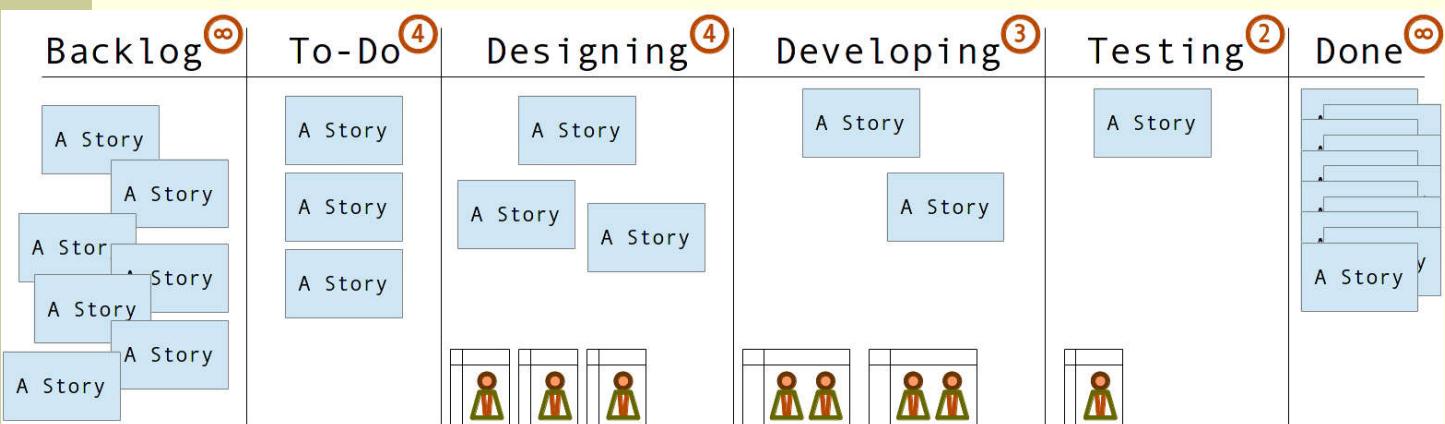
69

<http://comunidad.iebschool.com/universoagile/2015/01/05/quiero-ser-agil-mejor-scrum-o-kanban/>

<http://www.javiergarzas.com/2016/04/scrum-o-kanban.html>

# Kanban (II/II)

- Su tablero es similar al de Scrum pero incluye todos los elementos (en Scrum dividen por Sprint)



<https://es.slideshare.net/SimoneBrighina/simone-brighina-implantacion-metodologia-kanban-para-ioplanto>

© Antonio Santos Ramos 2018

<https://jeronimopalacios.com/kanban/>

70

<https://toggl.com/developer-methods-infographic>

<http://leankanban.com/guide/>

# Dudas SCRUM-eras

## ■ ¿BBDD aparte o una historia de usuario?

- <https://softwareengineering.stackexchange.com/questions/105242/agile-methods-and-databases-at-the-start-of-the-project>
- <http://www.vertabelo.com/blog/notes-from-the-lab/database-modeling-in-scrum-teams>

## ■ WebSite

- <https://www.osudio.com/en/blog/postings/how-to-the-3-day-design-sprint>

## ■ Tools

- <http://technologyadvice.com/blog/information-technology/kanban-tools-project-managers-developers/>
- <https://unitid.nl/english/7-ux-design-tools-for-an-effective-scrum-workflow/>
- <https://www.daxx.com/article/free-agile-project-management-tools-for-your-scrum>
- <https://www.justinmind.com/blog/7-awesome-tools-for-agile-design-and-development-collaboration/>

© Antonio Santos Ramos 2018

71

# Ejemplo Tablero Kanban



© Antonio Santos Ramos 2018

72

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

T04

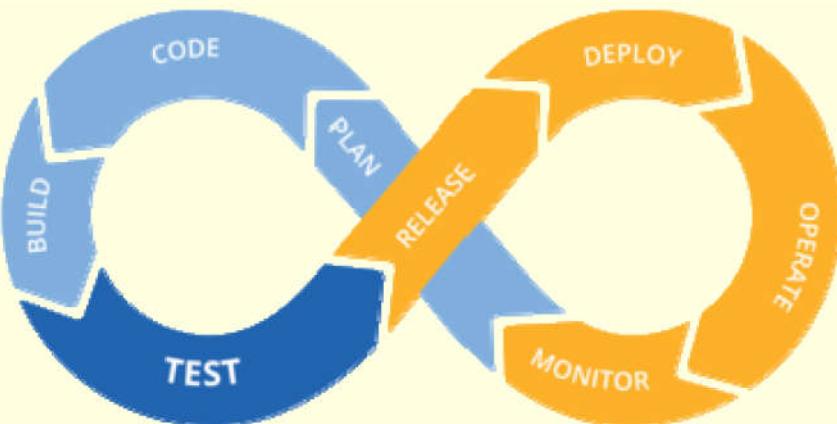
Introducción a DevOps

© Antonio Santos Ramos 2018

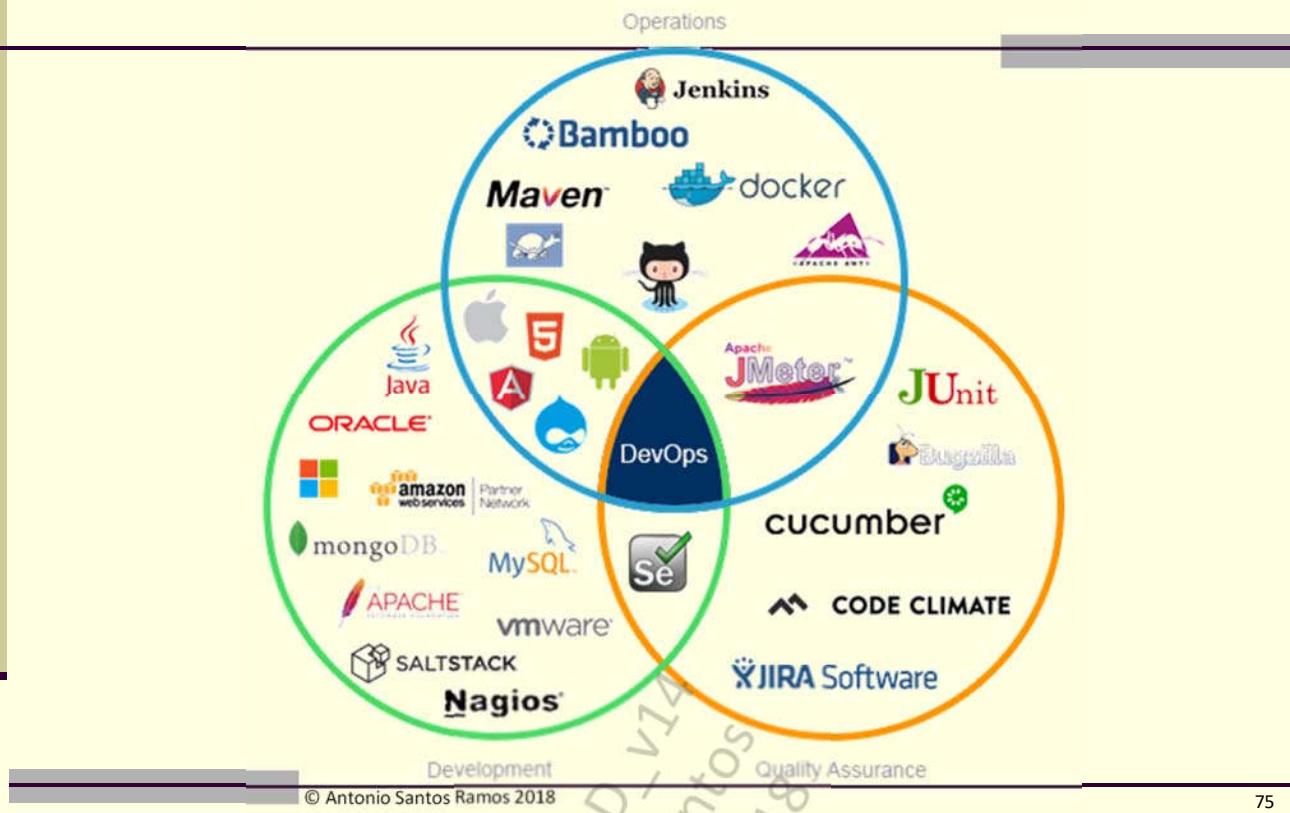
73

## DevOps

- Acrónimo de development y operations
- Se centra en la comunicación, colaboración e integración entre desarrolladores de software y los profesionales de operaciones en las tecnologías de la información (IT)



# DevOps en Java



Todo un ecosistema...

<https://stackshare.io/stackups>



# ... y más Open Source



© Antonio Santos Ramos 2018

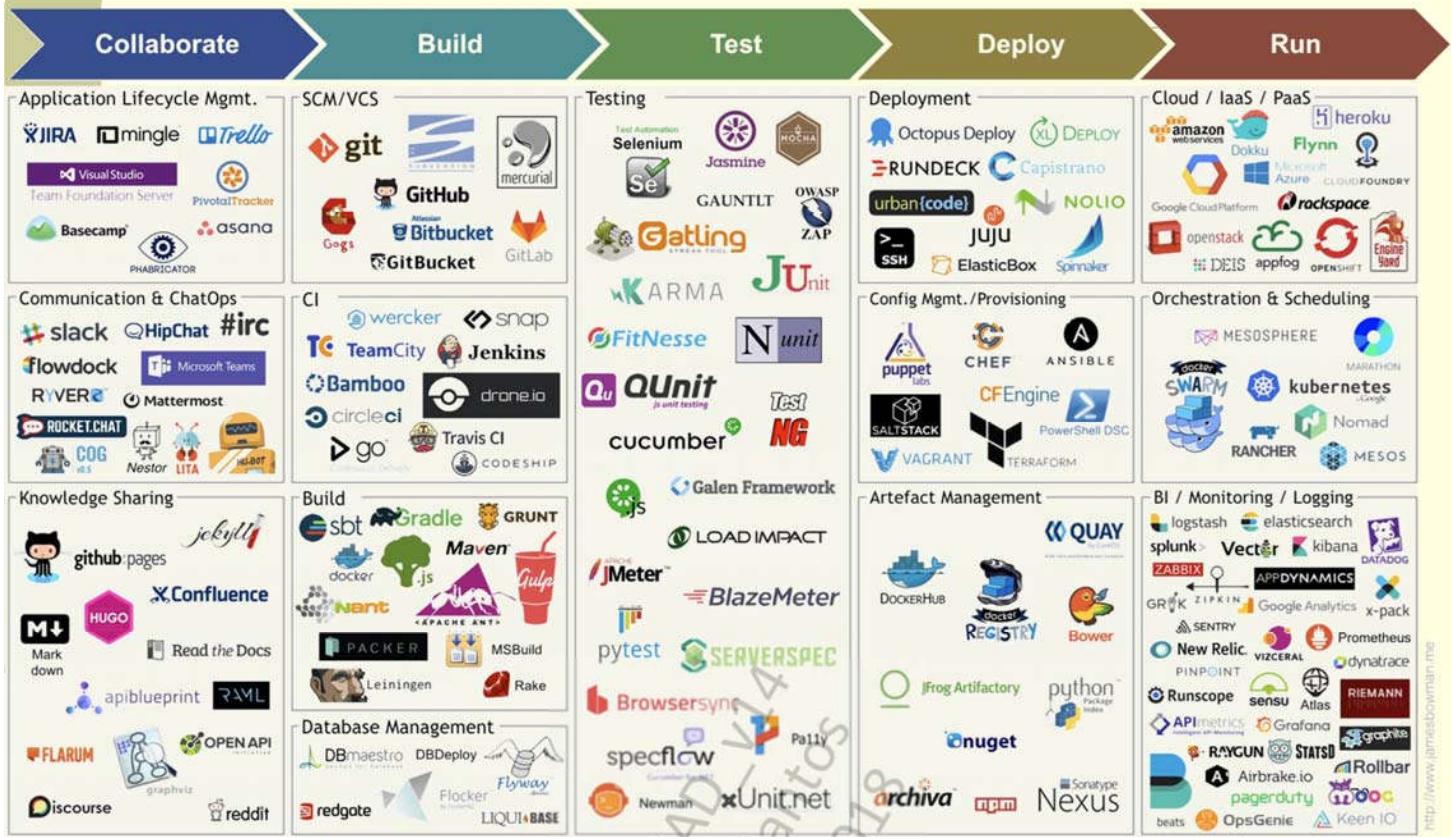
77

<https://dreamix.eu/blog/java/elasticsearch-spring-boot-and-angular-js-into-action-for-building-b2c-web-applications>

## ... y más???



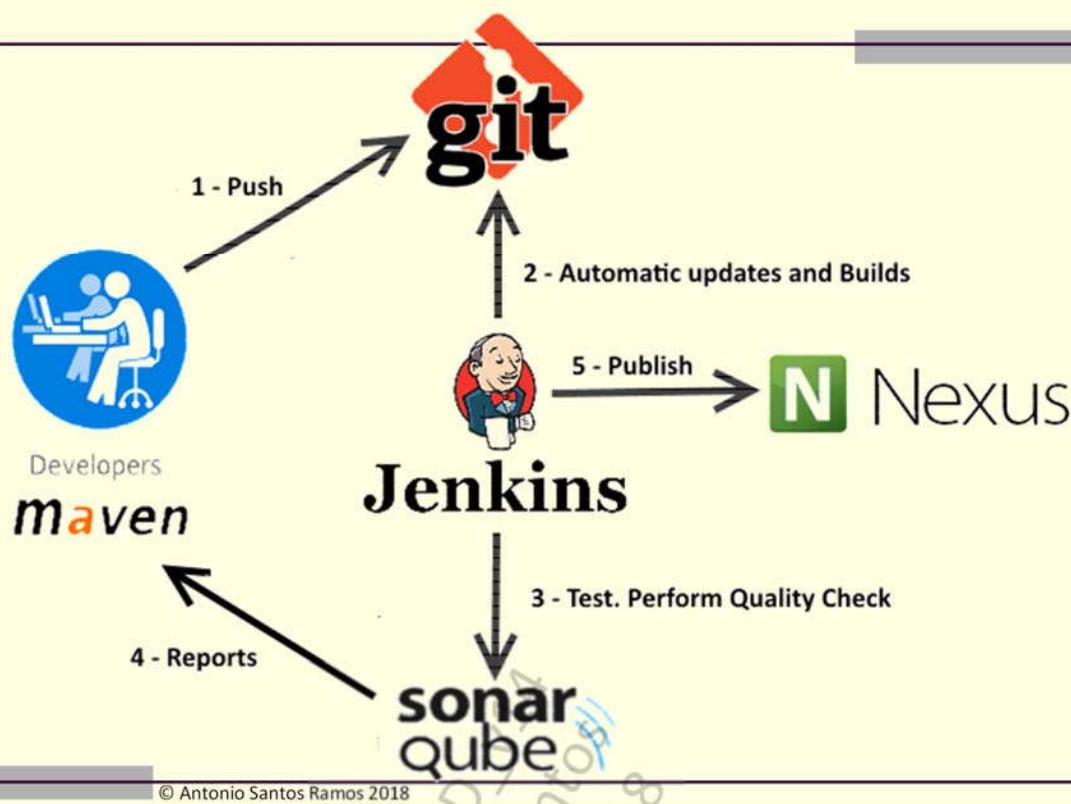
# Siempre hay más...



## Integración Continua

- Integración Continua (Martin Fowler)(2006)
  - “La integración continua es una práctica de desarrollo de software en la cual los miembros de un equipo integran su trabajo frecuentemente, como mínimo de forma diaria.
  - Cada integración se verifica mediante una herramienta de construcción automática para detectar los errores de integración tan pronto como sea posible.”
- Por lo tanto, permite configurar y lanzar una serie de procesos incrementales diseñados para automatizar las tareas más importantes dentro del desarrollo software; sencillas (construcción periódica y programada del software) o complejas.

# Flujo de Integración Continua



## Herramientas empleadas

- **Control de Versiones:** Git, Subversion (SVN), SourceSafe, ...
  - Permite a un equipo trabajar de forma sincronizada
    - Commits diarios
      - El almacenamiento de los cambios debería ser diario y continuo.
    - Build diario
      - Disponer de una versión compilada y probada del sistema al final del día.
- **Pruebas unitarias**
  - Comprobar que el proyecto funciona correctamente y automatizar el proceso con pruebas unitarias.
- **Herramientas de construcción:** Ant, Maven
- **Herramientas de integración:** Jenkins, CruiseControl, ...
  - Permiten realizar la compilación automática y ejecución de pruebas integradas al control de versiones

# Entornos

## Entorno de desarrollo

- Empleado por los programadores web para modificar la aplicación, añadir nuevas características, corregir errores, etc.
- Suele ser local al desarrollador (y necesitar de un repositorio)

## Entorno de pruebas

- Se utiliza para ejecutar las pruebas unitarias, alguna prueba de integración, etc.

## Entornos

## Entorno de Pre-producción (“staging”)

- Suele ser idéntico al de Producción
- Se emplea para hacer pruebas finales, validar con usuarios, comprobar carencias, etc.

## Entorno de producción

- Entorno real en el que los usuarios finales ejecutan la aplicación.
- Hay que tener cuidado porque suele manejar datos reales

© Antonio Santos Ramos 2018

83

# Repositorios

- Nexus es un gestor de repositorio de Maven y permite guardar librerías propias, de la empresa, de terceros....



- Al usar las librerías, Maven primero busca en el repositorio local, si no están buscará en el Nexus y si no irá a Maven Central y las instalará en el Nexus.

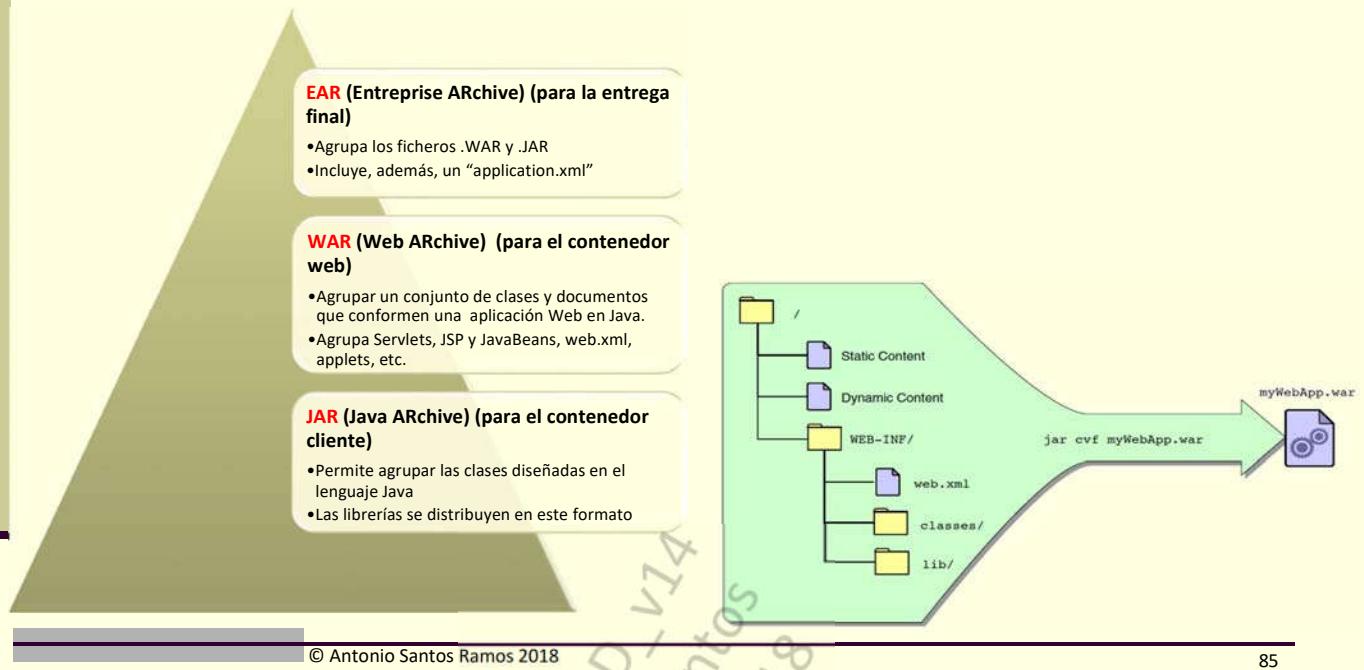
<http://www.javiergarzas.com/2014/08/nexus-artifactory-10-min.html>

© Antonio Santos Ramos 2018

84

# Empaquetado de archivos Java EE

## ■ ¿Cómo distribuir aplicaciones Java EE?



© Antonio Santos Ramos 2018

85

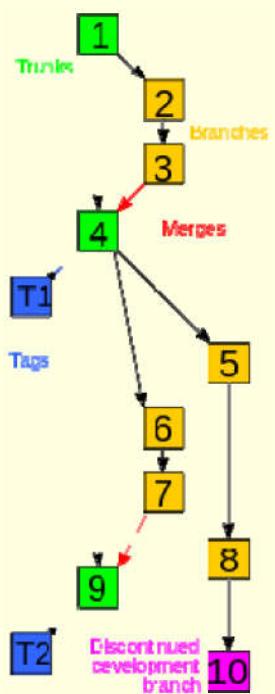
# Control de versiones (I/II)



- Versión, revisión o edición de un producto
  - Estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación.
- Control de versiones
  - Gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo.
- Sistemas de control de versiones o SVC (System Version Control).
  - Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

<http://nvie.com/posts/a-successful-git-branching-model/>

© Antonio Santos Ramos 2018

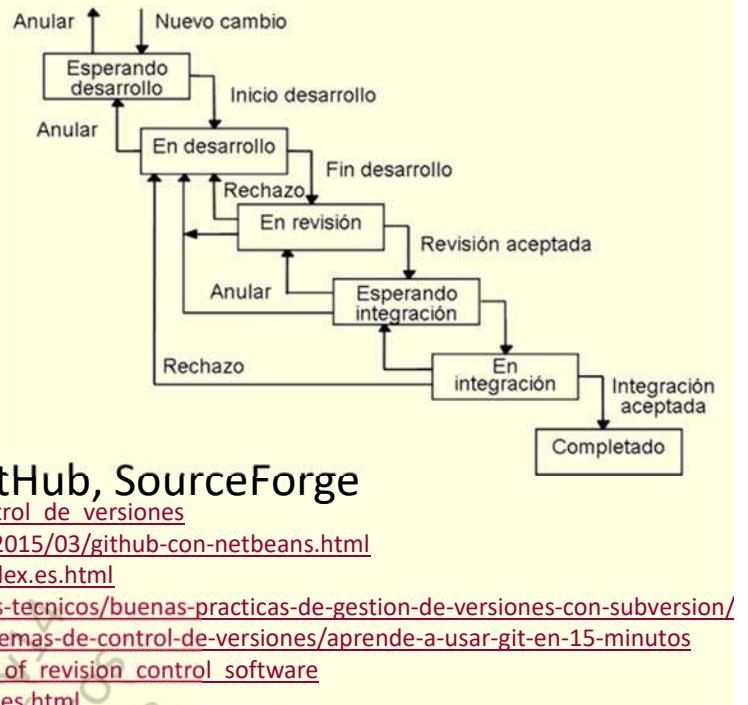


86

# Control de versiones (II/II)

## ■ Principales Sistemas de Control de Versiones

- SVN, Subversion, Git, SourceSafe, ClearCase, Darcs, Bazaar, Plastic SCM, CVS, Mercurial, Perforce



## ■ Principales repositorios: GitHub, SourceForge

[http://es.wikipedia.org/wiki/Control\\_de\\_versiones](http://es.wikipedia.org/wiki/Control_de_versiones)

<http://avbravo.blogspot.com.es/2015/03/github-con-netbeans.html>

<http://svnbook.red-bean.com/index.es.html>

<http://blogs.tecsisa.com/articulos-tecnicos/buenas-practicas-de-gestion-de-versiones-con-subversion/>

<http://www.genbeta.com/sistemas-de-control-de-versiones/aprende-a-usar-git-en-15-minutos>

[http://en.wikipedia.org/wiki/List\\_of\\_revision\\_control\\_software](http://en.wikipedia.org/wiki/List_of_revision_control_software)

<http://lml.ls.fi.upm.es/en/versiones.html>

© Antonio Santos Ramos 2018

87

# Herramientas de construcción (Build)

- Estas herramientas permiten automatizar el proceso de generación y/o actualización de un conjunto de ficheros.
- Emplean un lenguaje XML y cada lenguaje tiene los suyos
- Permiten gestionar las dependencias con otros ficheros, empaquetar el proyecto, construir tareas, etc.



**Maven™**

 gradle

<https://technologyconversations.com/2014/06/18/build-tools/>

<https://zereturnaround.com/rebellabs/java-build-tools-part-2-a-decision-makers-comparison-of-maven-gradle-and-ant-ivy/8/>

© Antonio Santos Ramos 2018

88

<http://prietopa.wordpress.com/2010/01/29/herramientas-de-construccion-de-java-ant-vs-maven/>

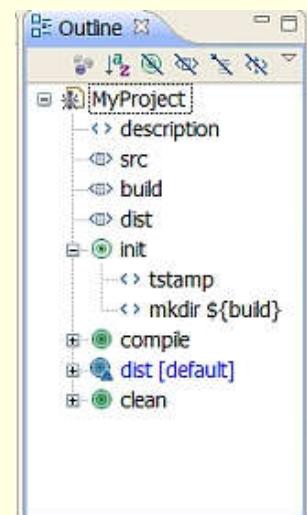
# Ant

<http://ant.apache.org/>



## ■ Apache Ant (Another Neat Tool)

- Herramienta para la realización de tareas mecánicas y repetitivas (normalmente en la fase de compilación y construcción)
- Desarrollado en Java. Basado en XML
- Nacida en Tomcat (independiente desde 2000)
- Debe crearse un fichero build.xml y un build.properties
- Desarrollos flexibles y rápidos
- Para gestionar dependencias usa Ivy <http://ant.apache.org/ivy/>



<http://www.java-forums.org/java-tutorial/8513-ant-another-neat-tool.html>

<http://es.slideshare.net/OliverCenteno/herramientas-java-23875690>

© Antonio Santos Ramos 2018

89

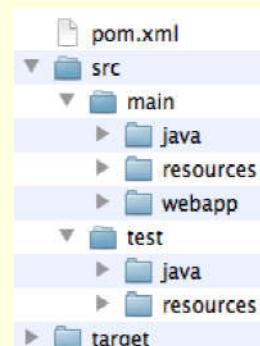
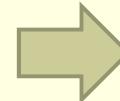
# Maven (I/X)



## ■ Apache Maven (2002)

- Sw para gestionar los elementos del proyecto y controlar el ciclo de vida de la aplicación
- Incluye una serie de carpetas

Item	Default
source code	<code> \${basedir}/src/main/java</code>
Resources	<code> \${basedir}/src/main/resources</code>
Tests	<code> \${basedir}/src/test</code>
Complied byte code	<code> \${basedir}/target</code>
distributable JAR	<code> \${basedir}/target/classes</code>



- Utiliza un fichero denominado pom.xml (Project Object Model)
  - Maneja la configuración y permite controlar las dependencias

© Antonio Santos Ramos 2018

<http://www.javiergarzas.com/2014/06/maven-en-10-min.html>

<http://www.developer.com/java/data/what-is-maven.html>

90

# Maven (II/X)

## Conceptos

### ■ Repositorio

- Es la carpeta dónde Maven va a ir a localizar los distintos artefactos a utilizar.
- Básicamente, hay dos tipos de repositorios
  - Repositorio Local: %userprofile%\m2\repository
  - Repositorio central: <http://mvnrepository.com/>

### ■ Artefacto <artifactid>

- Componente de software que se puede incluir en un proyecto como dependencia: un jar, war, etc.

Algunos repositorios

<http://mvnrepository.com/>  
<http://maven-repository.com/>  
<http://search.maven.org/>

### ■ Grupo <groupid>

#### ■ Conjunto de artefactos

© Antonio Santos Ramos 2018

91

# Maven (III/X)

## pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ejemplos.spring</groupId>
  <artifactId>SpringMvcHibernate_Anot</artifactId>
  <name>SpringMvcHibernate Anot</name>
  <packaging>war</packaging>
  <version>1.0.0-BUILD-SNAPSHOT</version>

  <properties>
    <org.springframework-version>4.0.3.RELEASE</org.springframework-version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${org.springframework-version}</version>
      <scope>runtime</scope>
    </dependency>
  </dependencies>
```

#### Elementos del pom.xml

- Espacio de nombres
- Propiedades del artefacto
- Sección de properties
- Dependencias
- Repositorios
- Build

<<< Object model versión  
 <<< Group /Organization Id  
 <<< ID of the Project itself  
 <<< Display name of the Project  
 <<< Packaging Type  
 <<< Version of the Project

# Maven (IV/X)

## pom.xml

### Elementos del pom.xml

- Espacio de nombres
- Propiedades del artefacto
- Sección de properties
- Dependencias
- Repositorios
- Build

### ■ Elementos del POM

- Propiedades del artefacto
  - Las importantes son groupId\*, artifactId\*, version\* y packaging.
- Sección <properties>
  - Esta sección es opcional
  - Permite parametrizar las versiones de las dependencias que pertenecen al mismo grupo
- Dependencias del proyecto
  - Su función es gestionar las dependencias.
  - Campos obligatorios: <groupId>, <artifactId>.
  - Campos optativos: <version>, <type>, <scope> (por defecto la versión última, jar y compile –scope- respectivamente)

# Maven (V/X)

## pom.xml

### Elementos del pom.xml

- Espacio de nombres
- Propiedades del artefacto
- Sección de properties
- Dependencias
- Repositorios
- Build

### ■ Scope <scope>

- Alcance de la dependencia. Por defecto es compile

SCOPE	DESCRIPCIÓN
<b>compile</b>	La dependencia es necesaria para compilar y está disponible en el classpath
<b>provided</b>	Indica que la dependencia la proporcionará el JDK o el servidor donde se ejecute la aplicación (esperas que el contenedor ya tenga esa librería)
<b>runtime</b>	La dependencia es necesaria en tiempo de ejecución, no en compilación.
<b>test</b>	La dependencia es para testing; una de las fases de compilación de Maven
<b>system</b>	Se indica la ruta donde se buscará la dependencia (mediante la etiqueta <systemPath>). Maven no busca el artefacto en el repositorio local.
<b>import</b>	Para dependencias que están en otro POM

# Maven (VI/X)

## pom.xml

### Elementos del pom.xml

- Espacio de nombres
- Propiedades del artefacto
- Sección de properties
- Dependencias
- Repositorios
- Build

### ■ Elementos del POM

#### ■ Repositorios

- Permite definir repositorios (en vez los del setting.xml de Maven)
- Ej.- repositorio de snapshots, milestones, integration, maintenance

#### ■ Build

- Permite configurar el build del proyecto mediante plugins (de hecho, Maven está construido así)

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>${java-version}</source>
                <target>${java-version}</target>
            </configuration>
        </plugin>
        ...
    </plugins>
</build>
```

# Maven (VII/X)

### ■ Ciclos de Vida en maven

#### Default Lifecycle (to build a project)

- validate: Validates that all project information is available and is correct. There are 21 phases...
- compile: Compiles the project's source code
- test: Runs unit tests inside the proper framework
- package: Packages the compiled code in its distribution format (for example, JAR, WAR, and the like)
- integration-test: Takes the package in the integration-test environment
- verify: Checks the package validity
- install: To re-use the package locally in other projects, Maven installs the package in the local repository
- deploy: The final package is installed in the remote repository and is ready to be shared by other projects/developers.



#### Clean Lifecycle (cleaning the project for files generated at compilation)

- pre-clean: Executes before project cleaning
- clean: Deletes all files from previous builds
- post-clean: Finalizes project cleaning

#### Site Lifecycle (to generate and deploy the project's site documentation)

- pre-site: Executes before generation of the site
- site: Generates the project's site documentation
- post-site: Finalizes the site generation and gets ready for site deployment
- site-deploy: Deploys the site documentation to the specified server

# Maven (VIII/X)

## ■ Los “Goal” de Maven

- Es un comando que recibe maven como parámetro para hacer algo

mvn plugin:comando

- Incluye unos cuantos por defecto aunque se pueden incorporar otros. Permite lanzar las fases de los ciclo de vida

mvn clean	Limpia todas las clases compiladas del proyecto.
mvn compile	Compila el proyecto
mvn package	Empaquea el proyecto (si es un proyecto java simple genera un jar, si es un proyecto web genera un war, etc...)
mvn install	Instala el artefacto en el repositorio local

# Maven (IX/X)

## ■ Arquetipos de Maven

- Definen lo que debe contener un proyecto en su creación
  - Crea la estructura del proyecto, el contenido del pom, la estructura de carpetas y los ficheros que incluye por defecto
- Están disponibles a través de los catálogos como...
  - <http://repo.maven.apache.org/maven2/archetype-catalog.xml>
  - <http://repo1.maven.org/maven2/archetype-catalog.xml>
  - <http://servicemix.apache.org/tooling/<version>/archetype-catalog.xml>
- Se puede usar desde comando con archetype:generate, (antes se usaba archetype:create)
 

```
mvn archetype:generate
-DgroupId=com.ejemplos -DartifactId=maven-jar-sample
```

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-j2ee-simple	RELEASE
org.apache.maven.archetypes	maven-archetype-marmalade-mojo	RELEASE
org.apache.maven.archetypes	maven-archetype-mojo	RELEASE
org.apache.maven.archetypes	maven-archetype-portlet	RELEASE
org.apache.maven.archetypes	maven-archetype-profiles	RELEASE
org.apache.maven.archetypes	maven-archetype-quickstart	RELEASE
org.apache.maven.archetypes	maven-archetype-site	RELEASE
org.apache.maven.archetypes	maven-archetype-site-simple	RELEASE
org.apache.maven.archetypes	maven-archetype-webapp	RELEASE

# Maven (X/X)

- <http://www.genbeta.com/java-j2ee/introduccion-a-maven>
- <https://www.slideshare.net/mulderbaba/intelligent-projects-with-maven-devfest-istanbul>
- <http://www.genbeta.com/java-j2ee/introduccion-a-maven-ii-project-object-model>
- <http://jdiezfoto.es/informatica/crear-un-jar-de-una-aplicacion-java-y-sus-dependencias-usando-maven/>
- <http://www.tutorialspoint.com/maven/>
- <http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>
- <http://fruzenshtein.com/setup-of-dynamic-web-project-using-maven>
- [http://www.javahispano.org/storage/contenidos/Tutorial\\_de\\_Maven\\_3\\_Erick\\_Camacho.pdf](http://www.javahispano.org/storage/contenidos/Tutorial_de_Maven_3_Erick_Camacho.pdf)
- <http://stackoverflow.com/questions/15683299/how-do-i-configure-a-java-ee-maven-project-in-eclipse>
- <https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>
- <http://myjeeva.com/exclusive-maven-archetype-list.html>
- <http://zeroturnaround.com/rebellabs/java-build-tools-part-2-a-decision-makers-comparison-of-maven-gradle-and-ant-ivy/>
- <http://technologyconversations.com/2014/06/18/build-tools/>

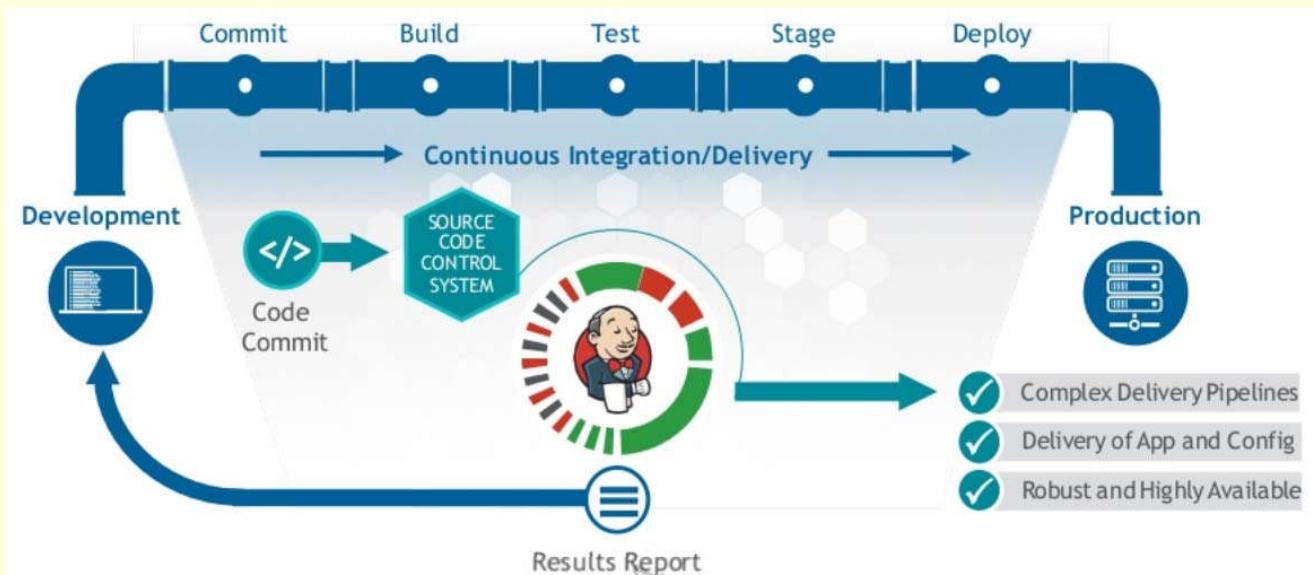
# Jenkins (I/IV)



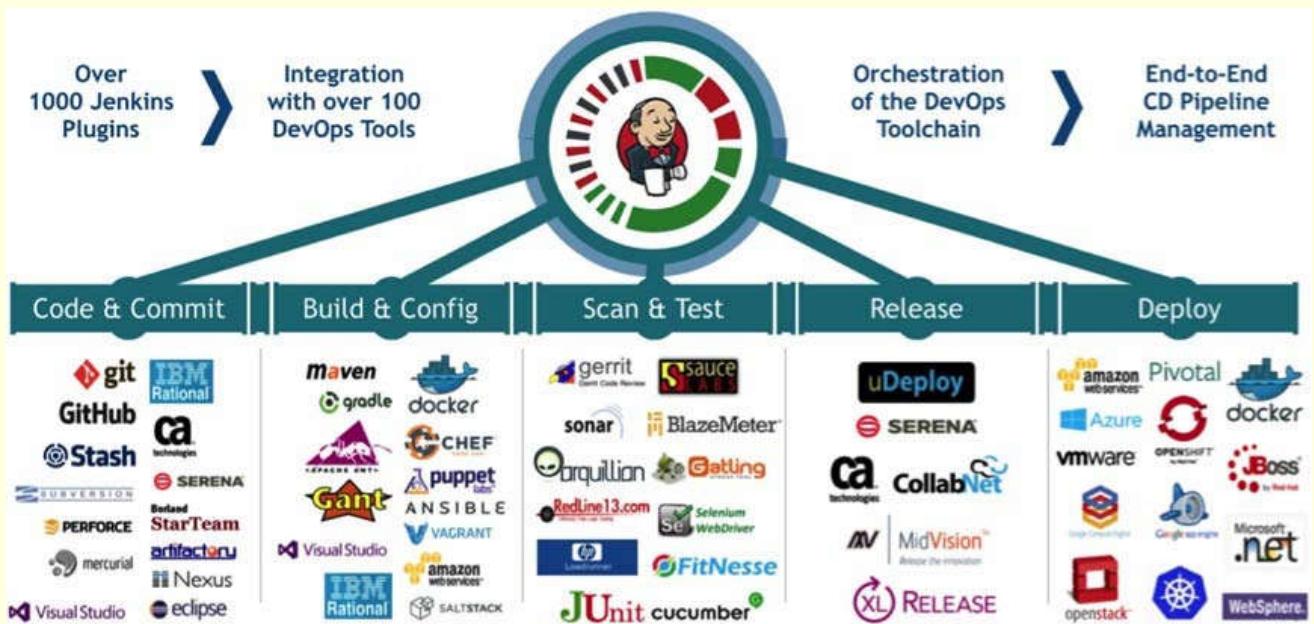
- Jenkins es un servidor de integración continua, licencia MIT
- Es un fork (en Feb 11) de Hudson (Feb 05) que emplea plugins para añadir funcionalidad
- Funciona dentro de un contenedor (ej Tomcat) soportando un control de versiones y ejecutando proyectos Ant y Maven.
- Permite definir tareas automatizadas
  - Ej.- Monitorizar un SVC y si hay cambios, compilar, ejecutar pruebas, pasar métricas de calidad e informar de los resultados

<https://jenkins.io/>

# Jenkins (II/IV)



# Jenkins (III/IV)



# Jenkins (IV/IV)

Active Projects | All | Bee | Eden | Mondidinamici | ProjectAutomation | **Prova** | Releases | Statistics | Training | Vendor | z\_Maintenance\_z | z\_Old\_z | +

S	W	Job ↓	Last Success	Last Failure	Last Duration
●	●	<a href="#">Eden-ActiveWorlds</a>	2 hr 43 min (#82)	1 mo 7 days (#54)	3 min 54 sec
●	●	<a href="#">Eden-Eden</a>	2 hr 24 min (#631)	2 days 20 hr (#617)	18 min
●	●	<a href="#">Eden-MondiDinamiciWebservice</a>	5 hr 53 min (#48)	7 days 2 hr (#20)	49 sec
●	●	<a href="#">Eden-Palm</a>	1 hr 53 min (#31)	3 days 4 hr (#18)	20 sec

Icon: S M L      Legend: for all for failures for just latest builds

<b>Tests</b>		<b>Success</b>		<b>Failed</b>		<b>Skipped</b>		<b>Total</b>
Job		#	%	#	%	#	%	#
<a href="#">Eden-ActiveWorlds</a>		127	100%	0	0%	0	0%	127
<a href="#">Eden-Eden</a>		266	100%	0	0%	0	0%	266
<a href="#">Eden-MondiDinamiciWebservice</a>		155	100%	0	0%	0	0%	155
<a href="#">Eden-Palm</a>		20	100%	0	0%	0	0%	20
<b>Total</b>		<b>568</b>	<b>100%</b>	<b>0</b>	<b>0%</b>	<b>0</b>	<b>0%</b>	<b>568</b>

**Test statistics**

skipped = 0 (0%)
failed = 0 (0%)
success = 568 (100%)

**Test trend**

The chart shows a significant increase in the number of tests run over time, starting around 100 in September and reaching nearly 500 by October.

# SonarQube (I/V)

- SonarQube es una plataforma de código abierto para el análisis de la calidad de código.
- <https://www.sonarqube.org/>
- Esta plataforma recibe el código fuente como entrada de datos.
  - En base a la entrada de datos, comienza a aplicar reglas predefinidas y a controlar si se cumplen.
  - Para cada lenguaje se puede descargar un conjunto de normas a aplicar

<https://www.sonarsource.com/products/codeanalyzers/sonarjava.html>

## SonarQube (II/V)

Dashboards Projects Measures Issues Settings Log in Search

**Helicopter View**

Activity Java Projects Javascript Projects Languages Panel

**TOOLS**

Dependencies Compare

**sonarqube**  
Sonar as a Service for your project with

**CloudBees**

**All Projects**

SOALE Rating B Remediation Cost 69,102.9 days Lines of Code 10,637K 9  
5,280.4 days to 9 9

**All Projects**

Issues 524,089 ▶ Blocker 889 Critical 3,019 ▶ Major 441,509 ▶ Minor 20,256 ▶ Info 58,416 ▶  
Rules compliance 87.1%

**Forges**

Name	LOCs	SOALE Rating
Forges	8,114,396	B
Apache	4,149,049	B
Others	1,984,743	B
JBoss	560,876	B
OW2	535,057	B
Sourceforge	367,201	B
Codehaus	257,051	B
GoogleCode	137,790	B
OPS4J	71,501	B
SpringSource	51,128	B

9 results

**All Projects**

Size: Lines of code Color: Rules compliance 0.0% 100.0%

This dashboard provides a comprehensive overview of project health and activity. It includes sections for Helicopter View, Tools (Dependencies, Compare), SonarQube integration, CloudBees integration, and detailed project status. The main area features three cards: 'All Projects' showing SOALE Rating, Remediation Cost, and Lines of Code; 'All Projects' showing Issues by severity; and 'Forges' listing various open source projects with their LOCs and SOALE Ratings. A large central area displays a heatmap of various Apache projects and their dependencies, with color coding for rules compliance. Below this is a chart showing trends in Lines of code, Duplicated lines, and Unit tests from 2010 to 2013.

**Quality Gate** Failed

since 1.4 since 1.4

1 New Bugs 1 Open Issues

Fix this to be able to release to production

Changes in the leak period

Bugs & Vulnerabilities

Leak Period: since 1.4 started 3 months ago

1 C Bugs 0 A Vulnerabilities 1 New Bugs 0 New Vulnerabilities

These have been around a while, they can wait a little longer

Fix these first

Code Smells

3h A Debt 10 Code Smells 10min New Debt 1 New Code Smells

started 2 years ago

Coverage

92.6% Coverage 64 Unit Tests — Coverage on New Code

Duplications

5.1% Duplications 5 Duplicated Blocks — Duplications on New Code

# SonarQube (IV/V)

The screenshot shows the SonarQube dashboard for a Java project. Key metrics include:

- Lines Of Code:** 9,458 (Java)
- Files:** 156
- Functions:** 772
- Duplications:** 8.0% (8,040 lines, 47 blocks, 27 files)
- Complexity:** 3.1 /function, 15.0 /class, 15.3 /file (Total: 2,383)
- SOALE Rating:** A
- Technical Debt Ratio:** 1.5%
- Technical Debt:** 8d 5h
- Issues:** 166 (0 Blocker, 0 Critical, 81 Major, 78 Minor, 7 Info)
- Directory Tangle Index:** 0.0%
- Dependencies To Cut:** Between Directories: 0, Between Files: 0
- Unit Tests Coverage:** 64.7% (Line Coverage: 71.8%, Condition Coverage: 51.4%)
- Unit Test Success:** 100.0% (0 Failures, 0 Errors, 300 Tests, 14 sec Execution Time)
- Overall Coverage:** 64.7% (Line Coverage: 71.8%, Condition Coverage: 51.4%)

Events history from Jan 22 2016 to Dec 15 2015 is also displayed.

© Antonio Santos Ramos 2018

107

# SonarQube (V/V)

The screenshot shows the rule configuration interface for SonarQube. It lists rules categorized by language (Language, Tag) and provides a search bar and navigation buttons.

**Reglas**

**Buscar:** "important" annotation should be placed at the end of the declaration

**Language:**

- Java: 1484
- JavaScript: 116
- TS: 64
- SCSS: 63
- CSS: 53
- Web: 53
- bw: 18
- XML: 6

**Tag:**

**CONDICIONES:** Solamente las medidas a nivel de proyecto se comprueban contra los umbrales. Se ignoran los sub-proyectos, directorios y ficheros. [Más](#)

Añadir Condición:

Blocker issues	Valor	es mayor que	0	0	Actualizar	Borrar
Comments (%)	Valor	es menor que	0	20	Actualizar	Borrar
Critical issues	Valor	es mayor que	0	5	Actualizar	Borrar
Major Issues OBSOLETO	Valor	es mayor que	5	7	Actualizar	Borrar
Complexity /function	Valor	es mayor que	2.5	3	Actualizar	Borrar
Directory tangle index	Valor	es mayor que	3	5	Actualizar	Borrar
Public documented API (%)	Valor	es menor que	100	100	Actualizar	Borrar

108

# Prácticas recomendadas (I/II)

- Mantener el código en un repositorio:
  - Todos los artefactos requeridos para la compilación deben encontrarse en el repositorio.
- Automatizar las compilaciones:
  - Un simple comando permite construir o compilar el sistema.
- Automatizar las pruebas:
  - Una vez compilado se deben ejecutar todos los test para validar que el sistema se comporta como esperaban
- Hacer commits diarios:
- Cada commit debe compilar:
  - El sistema debe compilar el cambio hecho para la versión actual

# Prácticas recomendadas (II/II)

- Mantener la compilación rápida:
- Realizar pruebas en un entorno parecido al de producción:
  - Permite detectar fallas antes que se despliegue en producción.
- Facilita la entrega de los últimos entregables:
  - Tener las compilaciones listas para ejecutar reduce el tiempo para generar el entregable de una característica revisada
- Todos pueden ver el resultado de la compilación:
  - Ver cuando falla, detalles del cambio que origina el error, etc.
- Automatizar despliegues:
  - Una vez compilado se despliega a un entorno de pruebas o producción

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

T05

Pruebas (Testing)

© Antonio Santos Ramos 2018

111

QA (I/II)

- Testing (o ‘Control de la calidad’):
  - Realiza actividades para comprobar la calidad al final del proceso. Interesa el desarrollo del producto en si mismo.
- QA (quality Assurance)
  - QA se realiza durante todas las etapas del proyecto.
  - Se centra más en la prevención que en la detección.
  - Las tareas de QA están interesadas en el proceso de desarrollo del producto.
  - Hay dos tipos
    - QA Funcional (Diseño y Ejecución de Pruebas) y
    - QA Técnico (Rendimiento y Optimización, o Calidad de Software).

## QA (II/II)

### ■ Tipos de QA

#### ■ QA Funcional:

- Orientado a asegurar la corrección funcional de las soluciones TI

#### ■ QA Técnico:

- Basado en el rendimiento, optimización y la calidad del software



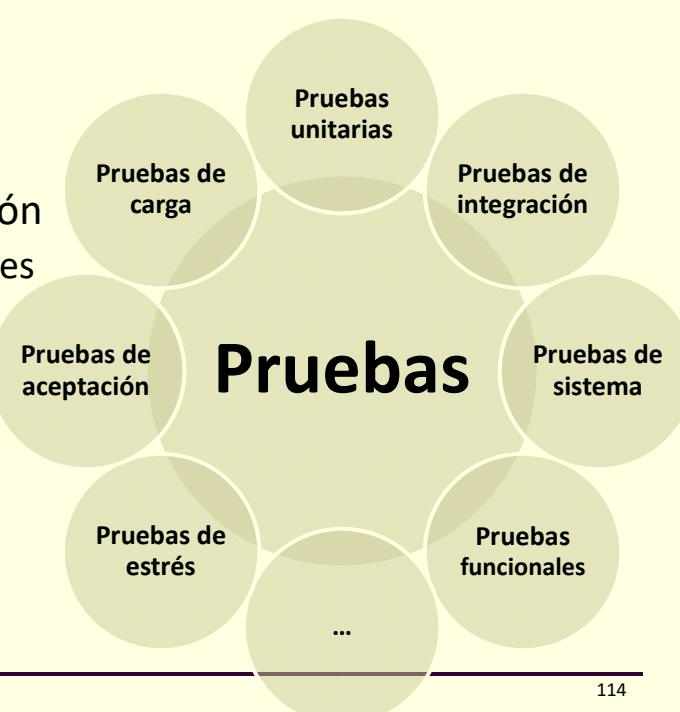
© Antonio Santos Ramos 2018

113

## Tipos de pruebas (I/III)

### ■ Se pueden realizar numerosos tipos de pruebas:

- Pruebas Unitarias
- Pruebas de Usabilidad
- Pruebas previas a Producción
  - Pruebas de Funcionalidades y Operación
  - Pruebas de Carga
  - Pruebas de Seguridad
  - Pruebas de Respaldo y Recuperación
  - Pruebas de Interfaces y Contenidos



© Antonio Santos Ramos 2018

114

# Tipos de pruebas (II/III)

## Pruebas de integración

- Se prueba cómo es la interacción entre dos o mas unidades del software.
- Se verifica que los componentes de la aplicación funcionan de forma correcta actuando en conjunto.
- Son pruebas dependientes del entorno en el que se ejecutan.

## Pruebas de aceptación

- Se comprueba si el software cumple con las expectativas del cliente

## Pruebas de sistema

- Suelen realizarse después de las pruebas de integración.
- Su objetivo es probar todo el sistema software completo e integrado desde el punto de vista de requisitos de la aplicación.
- Engloban **pruebas funcionales, pruebas de carga, de estrés etc.**

# Tipos de pruebas (III/III)

## Pruebas funcionales (o de caja negra)

- Se comprueba que el software creado cumple con la función para la que se había pensado.
- Importan las entradas y salidas al software, es decir, si ante una serie de entradas el software devuelve los resultados esperados.
- No se comprueba que el software esté bien hecho, bien diseñado, etc. Se estudia el software desde la perspectiva del cliente, no del desarrollador.
- Pueden ser manuales o automatizarse (Selenium)
- Ej. llenar el campo de un teléfono con un texto y no deje registrar.

## Pruebas de carga

- Comprueban el rendimiento del sistema.
- Se observa la respuesta de la aplicación ante un determinado número de peticiones.
- Ej.- comportamiento del sistema ante el acceso concurrente de 1000 usuarios.

## Pruebas de estrés

- Comprueban el rendimiento del sistema.
- Se somete el software a situaciones extremas (intentando "tirarlo") para ver cómo se comporta, si es capaz de recuperarse, puede tratar el error, etc.

# Testing

## Test-Driven Development (TDD)

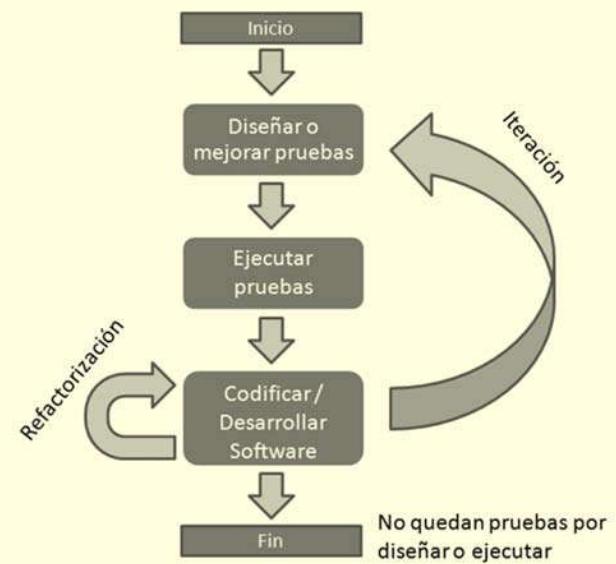
- TDD: Desarrollo Guiado por Pruebas
- Es una metodología consistente en escribir las pruebas antes que el código.
- El diseño viene gobernado por las pruebas del programador.
- TDD es muy efectiva ya que automatiza las pruebas de programador (programmer unit test).

<http://librosweb.es/libro/tdd/>

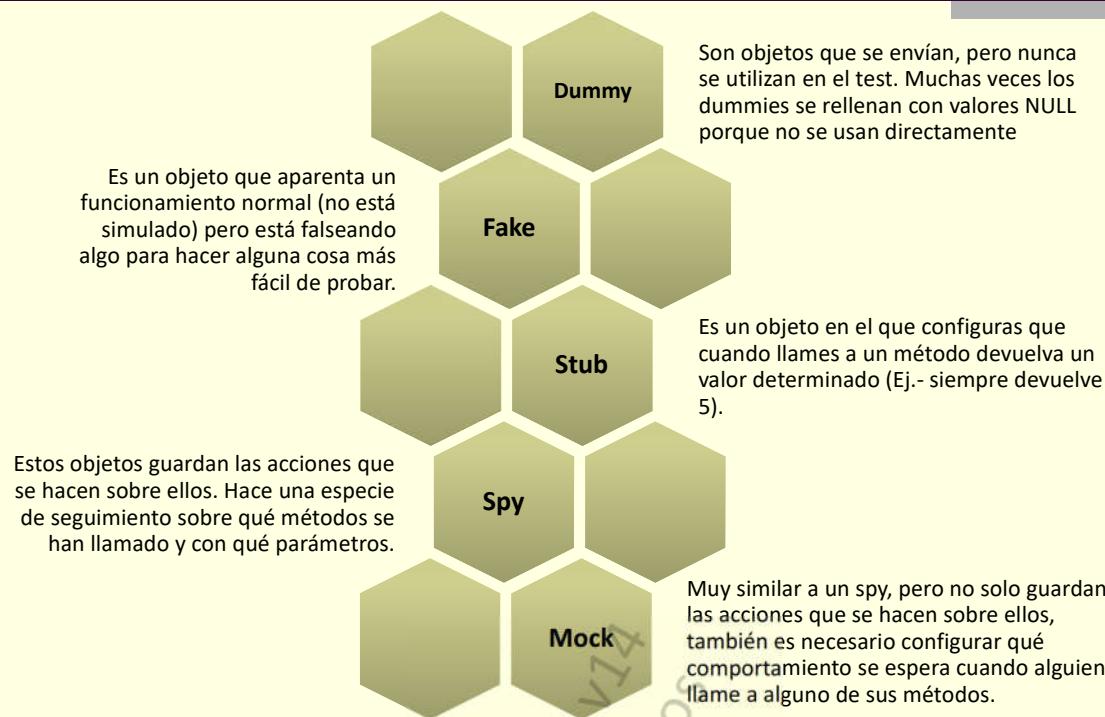


## Test-Driven Development (TDD) Fases

- Elegir un requisito
- Escribir una prueba para ese requisito
- Hacer una prueba para un caso de un método de una clase  
Comprobar si falla la prueba
- Escribir el código que pase dicha prueba.
- Realizar la automatización de la prueba
- Hacer refactoring sobre el código



# Tipos de test doubles

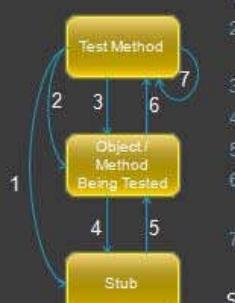


© Antonio Santos Ramos 2018

119

[http://librosweb.es/libro/tdd/capitulo\\_6/cuando\\_usar\\_un\\_objeto\\_real\\_un\\_stub\\_o\\_un\\_mock.html](http://librosweb.es/libro/tdd/capitulo_6/cuando_usar_un_objeto_real_un_stub_o_un_mock.html)

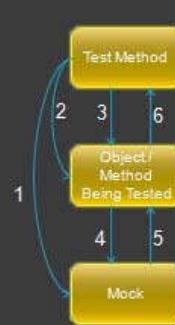
## Stub



Success is determined based on how the method being tested responds back to the test – “State”

## Stub vs. Mock

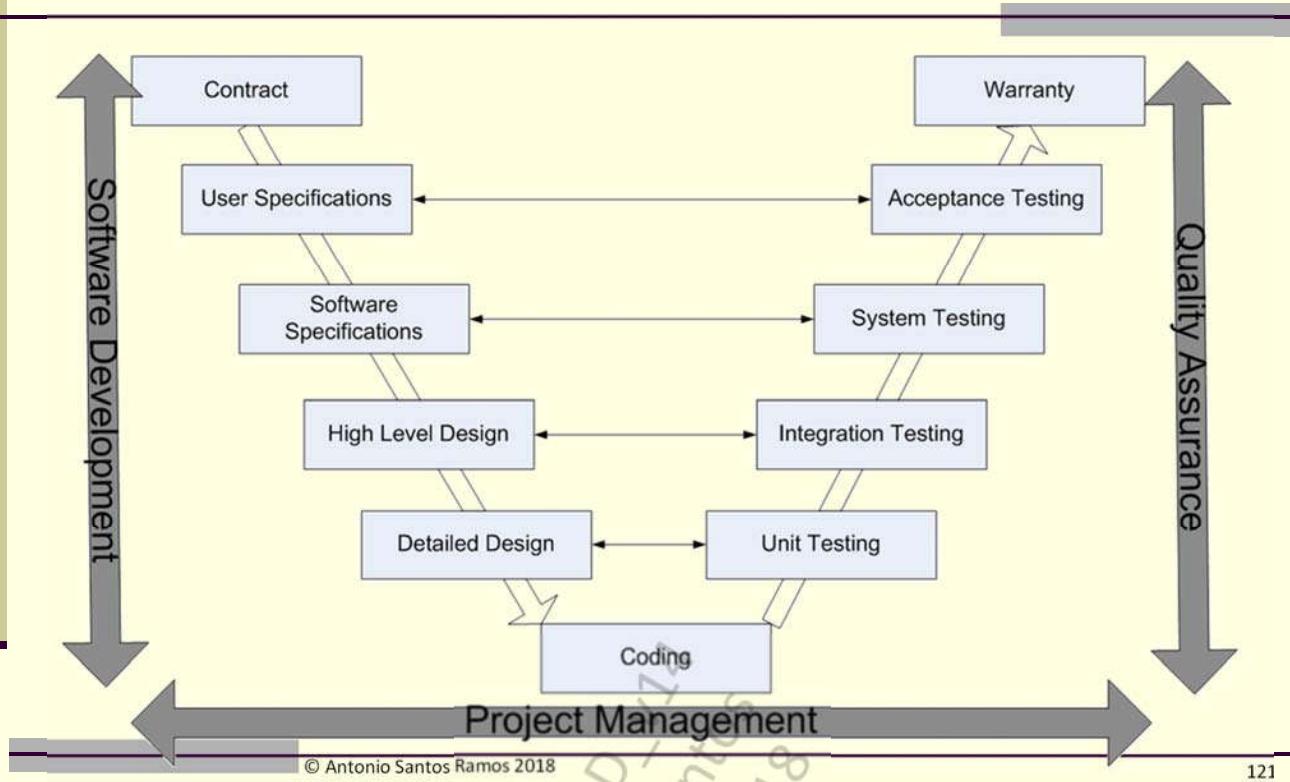
## Mock



Mock communicates back to the test method and success is determined based on the calls that were made – “Behavior”

[http://librosweb.es/libro/tdd/capitulo\\_6.html](http://librosweb.es/libro/tdd/capitulo_6.html)

# V-Model



<https://stackoverflow.com/questions/3370334/difference-between-acceptance-test-and-functional-test>

## Pruebas unitarias

- También conocidas como pruebas de unidad
  - Permiten probar las unidades del software (normalmente métodos).
  - Se escriben desde el punto de vista del desarrollador.
  - Interesa como funciona la unidad, no la interacción entre componentes (eso es una prueba de integración).
- Consisten en comprobaciones (manuales o automatizadas)
  - Las comprobaciones verificarán que el código correspondiente a un módulo concreto de un sistema software funciona de acuerdo con los requisitos del sistema.

[www.cc.uah.es/drg/docencia/Pruebas/Pruebas4x1.pdf](http://www.cc.uah.es/drg/docencia/Pruebas/Pruebas4x1.pdf)

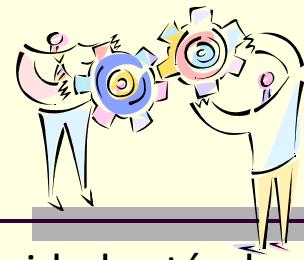
# Pruebas unitarias... correctas

- Una prueba unitaria es correcta si es:
  - Automatizable: sin intervención manual.
  - Completa: cubre todo el código.
  - Reutilizable: las pruebas se pueden ejecutar más de una vez.
  - Independiente: la ejecución de una prueba no afecta la ejecución de otra.
  - Valorada: es tan importante o más que el código generado.
- Simplifica el proceso de integración.
  - Aseguramos que el código funciona.
- Acotación de errores.
  - Se pueden localizar de forma más sencilla.

# Pruebas unitarias en Java

- **JUnit** (<http://www.junit.org/>) es el indiscutible líder. Simple y sencillo permite realizar de forma muy sencilla tests unitarios sobre nuestras clases
- **jMock** (<http://www.jmock.org/>) permite generar mock objects al vuelo para usarlos en los tests unitarios.
- **EasyMock** (<http://www.easymock.org/>) permite generar mock objects al vuelo para usarlos en los tests unitarios.
- **DbUnit** (<http://dbunit.sourceforge.net/>) para hacer pruebas unitarias de la base de datos.
- **HttpUnit** (<http://httpunit.sourceforge.net/>) para hacer pruebas de integración, aceptación o funcionales de aplicaciones Web.
- **JWebUnit** (<http://jwebunit.sourceforge.net/>) es un wrapper sobre HttpUnit y otros motores de pruebas para Web.
- **JMeter** (<http://jmeter.apache.org/>) para realizar pruebas funcionales, de rendimiento y de estrés de aplicaciones Web
- **Mockito** (<http://site.mockito.org/>) framework para unit test en Java

# JUnit



- JUnit (2002) es la biblioteca de pruebas de unidad estándar de facto para Java.
- Sus dos propósitos básicos son:
  - Ofrecer una estructura que permita la especificación de casos de prueba
  - Proporcionar un controlador capaz de llevar a cabo las pruebas de forma automática.
- Fue desarrollado por Kent Beck y Erich Gamma
- Se ejecuta con unos datos de entrada para controlar que la salida se ajusta a un valor esperado.
- Se deben crear casos de prueba.

## Junit. Funcionamiento

`extends junit.framework.TestCase o TestCase`

- Métodos de prueba
  - `public void testXXX() [throws ...]`
  - 1 o más aserciones por método
  - se ejecutarán en algún orden, no obligatorio
- Aserciones:
  - `assertEquals(T esperado, T resultado)`
  - `assertNull(Object resultado)`
  - `assertNotNull(Object resultado)`
  - `assertTrue(Object resultado)`
  - `assertFalse(Object resultado)`
- Si una prueba falla, no se ejecutará la siguiente.

No es necesario desde Junit 4.x

En Junit 4.x se coloca @ a cada uno

# Junit. Anotaciones

- Se pueden realizar anotaciones en los métodos

Junit 4.x	Junit 5.x	¿Qué hace?
@Test	@Test	El método es un test
@Before	@BeforeEach	El método se ejecuta antes de cada test
@BeforeClass	@BeforeAll	El método se ejecuta antes de empezar a ejecutar los test. Solo se ejecuta 1 vez
@After	@AfterEach	El método se ejecuta después de cada test
@AfterClass	@AfterAll	El método se ejecuta después de terminar todos los test
@Test @Ignore	@Test @Disabled	El test al que se le añade se ignora y no se ejecuta

<https://softwarebysergio.wordpress.com/2018/09/24/junit-5-introduction/>

## Junit. miContador.java

```
public class Contador {  
    private int cuenta= 0;  
    public int incrementa() { return ++cuenta; }  
    public int decrementa() { return --cuenta; }  
    public int dameCuenta() { return cuenta; }  
}
```

# Junit. TestContador.java

```
public class TestContador {
    private Contador contador;
    @Before
    public void executedBeforeEach() {
        contador = new Contador();
    }
    @Test
    public void testIncrementa() {
        assertEquals(1, contador.incrementa());
        assertEquals(2, contador.incrementa());
    }
    @Test
    public void testDecrementa() {
        assertEquals(-1, contador.decrementa());
        assertEquals(-2, contador.decrementa());
    }
    @After
    public void executedAfterEach() {
    }
}
```

© Antonio Santos Ramos 2018

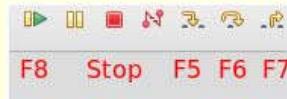
129

<http://javarevisited.blogspot.com.es/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html>

## “Debugging”



- Proceso para corregir los errores de código existentes
- Cada IDE tiene su sistema pero son, en general parecidos
- Se trabaja mediante breakpoint. El código se ejecuta hasta ese punto y a partir de ahí se emplea



Shortcut	Toolbar	Description
F5 (Step Into)		Steps into the call
F6 (Step Over)		Steps over the call
F7 (Step Return)		Steps out to the caller
F8 (Resume)		Resumes the execution
Ctrl + R (Run to Line)		Run to the line number of the current caret position
Drop to Frame		Rerun a part of your program
Shift + F5 ( Use Step Filters)		Skipping the packages for Step into
Ctr + F5 / Ctrl + Alt + Click		Step Into Selection

© Antonio Santos Ramos 2018

130

<http://www.vogella.com/tutorials/EclipseDebugging/article.html>

[http://chuwiki.chuidiang.org/index.php?title=Empezando con el debugger de eclipse](http://chuwiki.chuidiang.org/index.php?title=Empezando_con_el_debugger_de_eclipse)

# Curso de Programación para Internet con Java EE (Parte V - Proyectos)

T06

Best Practices

© Antonio Santos Ramos 2018

131

## Principios básicos de Diseño

T06 – BEST PRACTICES

- **Tight Encapsulation** (Encapsulación rígida)
  - Sólo se accede a los atributos de la clase mediante métodos
  - Atributos privados y acceso mediante métodos setter/getter
- **Loose Coupling** (Pérdida de acople)
  - Poca dependencia entre objetos
  - Cambiar el código de una clase no debería afectar a otra
- **High Cohesion** (alta cohesión)
  - Como de relacionadas se encuentran las tareas de una clase
- **DRY** (“Don’t Repeat Yourself”)
- **KISS** (“Keep It Simple, Stupid”)
- **YAGNI** (“You Aren’t Gonna Need It”)

<http://es.slideshare.net/ikercanarias/buenas-prcticas-para-la-construccin-de-software>

© Antonio Santos Ramos 2018

Ej.- Libro Raposa (Pag 382-389)

132

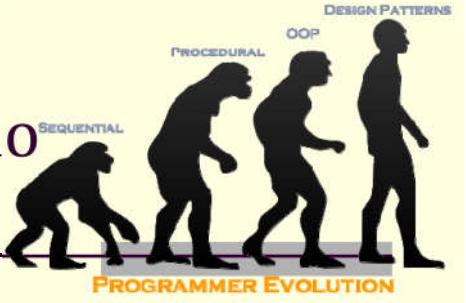
# Principios SOLID

- Desarrollados por Robert C. Martin. Hay 5 principios
  - (**SRP**) Single Response Principle
    - Una clase solo debería tener una única responsabilidad.
  - (**OCP**) Open-Close Principle
    - Las entidades software deben estar abiertas a su extensión, pero cerradas para su modificación.
  - (**LSP**) Liskov Substitution Principle
    - Los “objetos de un programa deberían ser reemplazables por las instancias de sus subtipos sin alterar el correcto funcionamiento del programa”.
  - (**ISP**) Interface Segregation Principle
    - Es mejor disponer de muchos interfaces cliente específicos en vez de una única interfaz de propósito general
  - (**DIP**) Dependency Inversion Principle
    - “Depender de Abstracciones, No Depender de Concreciones”.

# GRASP

- **GRASP**
  - General Responsibility Assignment Software Patterns
  - GRASP son una serie de buenas prácticas enfocadas a la calidad del software y reúne estos consejos
    - 1. Alta cohesión y bajo acoplamiento
    - 2. Controlador
    - 3. Creador
    - 4. Experto en información
    - 5. Fabricación pura
    - 6. Indirección
    - 7. Polimorfismo
    - 8. Variaciones protegidas

# Patrones de diseño



## Definiciones

- “Una solución (probada) a un problema en un determinado contexto” (Erich Gamma)
- “A Design Pattern names, abstracts and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design.” (Erich Gamma)

## Tipos

- **Creacionales:** abstraen el proceso de instanciación
- **Estructurales:** se ocupan de generar estructuras entre clases y objetos, se estudian con los diagramas de clases/objetos
- **De Comportamiento:** se encargan de la asignación de responsabilidades entre objetos y cómo se comunican entre sí.

© Antonio Santos Ramos 2018

135

## Patrones Básicos (I/II)

### Patrones creacionales

Para abstraer el proceso de instanciación y ocultar detalles de la creación

Abstract Factory (Fábrica abstracta)	Builder (Constructor virtual)	Factory Method (Método de fabricación)	Prototype (Prototipo)	Singleton (Instancia única)
<ul style="list-style-type: none"> <li>• Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se está usando.</li> </ul>	<ul style="list-style-type: none"> <li>• Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.</li> </ul>	<ul style="list-style-type: none"> <li>• Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.</li> </ul>	<ul style="list-style-type: none"> <li>• Crea nuevos objetos clonándolos de una instancia ya existente.</li> </ul>	<ul style="list-style-type: none"> <li>• Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.</li> </ul>

### Patrones estructurales

Cómo combinar clases y objetos para formar estructuras más grandes y nuevas funcionalidades

Adapter (Adaptador)	Bridge (Puente)	Composite (Objeto compuesto)	Decorator (Envoltorio)	Facade (Fachada)	Flyweight (Peso ligero)	Proxy
<ul style="list-style-type: none"> <li>• Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.</li> </ul>	<ul style="list-style-type: none"> <li>• Desacopla una abstracción de su implementación.</li> </ul>	<ul style="list-style-type: none"> <li>• Permite tratar objetos compuestos como si de uno simple se tratase.</li> </ul>	<ul style="list-style-type: none"> <li>• Añade funcionalidad a una clase dinámicamente.</li> </ul>	<ul style="list-style-type: none"> <li>• Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.</li> </ul>	<ul style="list-style-type: none"> <li>• Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.</li> </ul>	<ul style="list-style-type: none"> <li>• Mantiene un representante de un objeto.</li> </ul>

# Patrones Básicos (II/II)

## Patrones de comportamiento

Definen la interacción y comunicación entre objetos

### Chain of Responsibility (Cadena de responsabilidad)

- Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.

### Command (Orden)

- Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.

### Interpreter (Intérprete)

- Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.

### Iterator (Iterador)

- Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.

### Mediator (Mediador)

- Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.

### Memento (Recuerdo)

- Permite volver a estados anteriores del sistema.

### Observer (Observador)

- Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

### State (Estado)

- Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

### Strategy (Estrategia)

- Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.

### Template Method (Método plantilla)

- Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

### Visitor (Visitante)

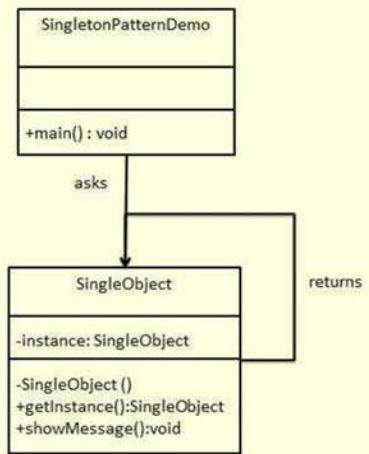
- Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

# Patrón Singleton

- Patrón que permite que una clase sea instanciada una única vez

```
public class SingletonC {  
    // 1-Es necesario una referencia static al objeto  
    private static final SingletonC instance =  
        new SingletonC();  
  
    //2-El constructor es privado  
    private SingletonC() {}  
  
    //3-Un método público devuelve un acceso al objeto  
    public static SingletonC getInstance() {  
        return instance;  
    }  
}
```

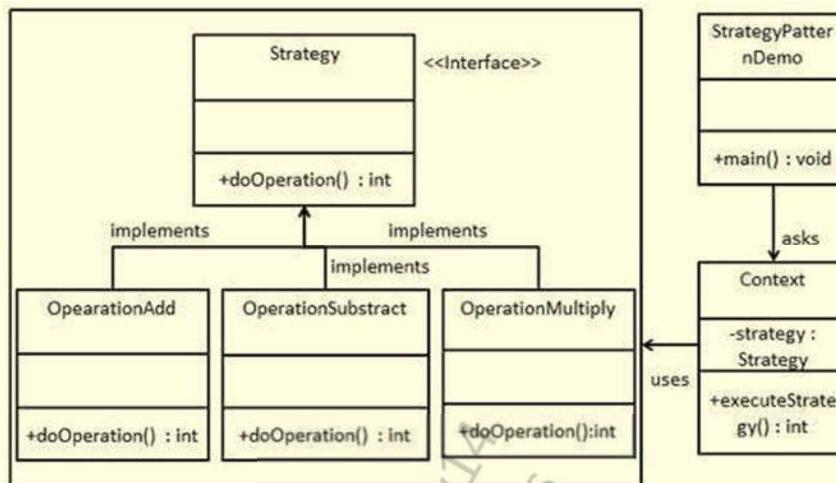
```
SingletonC ref = SingletonC.getInstance();
```



[http://www.tutorialspoint.com/design\\_pattern/singleton\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/singleton_pattern.htm)

# Patrón “Estrategia”

- Patrón de comportamiento que permite crear objetos que representan diferentes estrategias y un objeto de contexto cuyo comportamiento varíe dependiendo de la estrategia



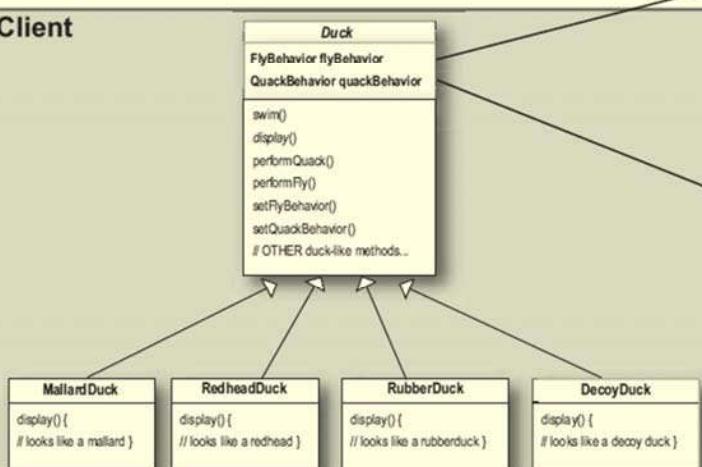
© Antonio Santos Ramos 2018

139

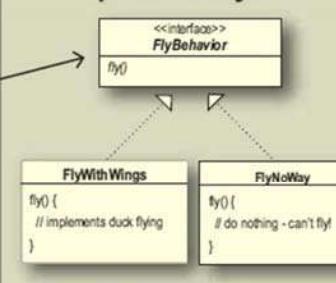
# Patrón “Estrategia”

Client makes use of an encapsulated family of algorithms for both flying and quacking.

## Client

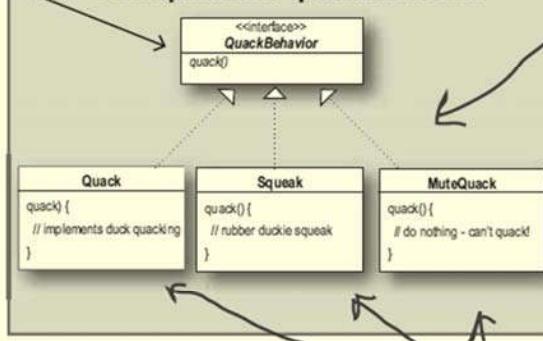


## Encapsulated fly behavior



Think of each set of behaviors as a family of algorithms.

## Encapsulated quack behavior



The **Strategy Pattern** defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

© Antonio Santos Ramos 2018

140

# Patrón “Observer”

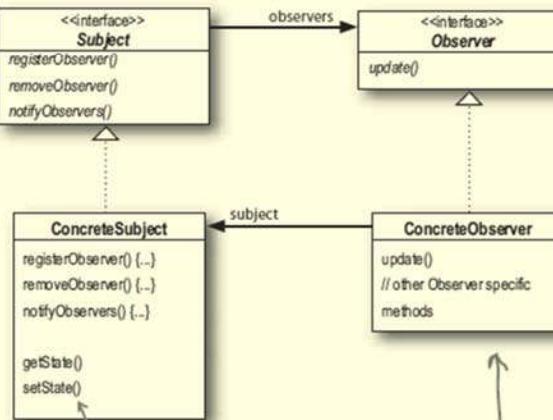
## the class diagram

Here's the Subject interface. Objects use this interface to register as observers and also to remove themselves from being observers.

Each subject can have many observers.

All potential observers need to implement the Observer interface. This interface just has one method, update(), that gets called when the Subject's state changes.

A concrete subject always implements the Subject interface. In addition to the register and remove methods, the concrete subject implements a notifyObservers() method that is used to update all the current observers whenever state changes.



The concrete subject may also have methods for setting and getting its state (more about this later).

Concrete observers can be any class that implements the Observer interface. Each observer registers with a concrete subject to receive updates.

© Antonio Santos Ramos 2018

141

# Patrón “Observer”

The Observable class keeps track of all your observers and notifies them for you.

Observable is a CLASS not an interface, so WeatherData extends Observable.

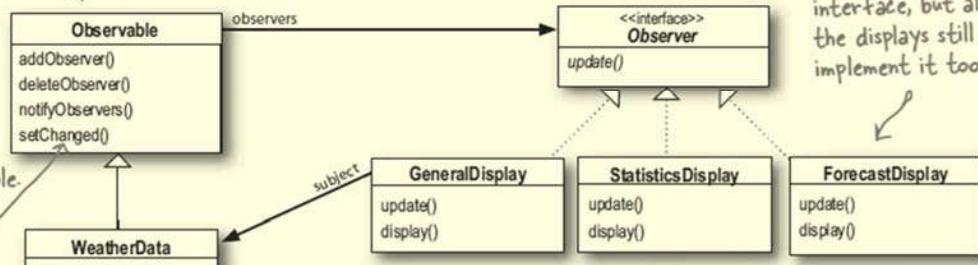
This doesn't look familiar! Hold tight, we'll get to this in a sec...

Here's our Subject, which we can now also call the Observable. We don't need the `register()`, `remove()` and `notifyObservers()` methods anymore; we inherit that behavior from the superclass.

This should look familiar. In fact, it's exactly the same as our previous class diagram!

We left out the **DisplayElement** interface, but all the displays still implement it too.

There will be a few changes to make to the `update()` method in the concrete Observers, but basically it's the same idea... we have a common **Observer** interface, with an `update()` method that's called by the Subject.

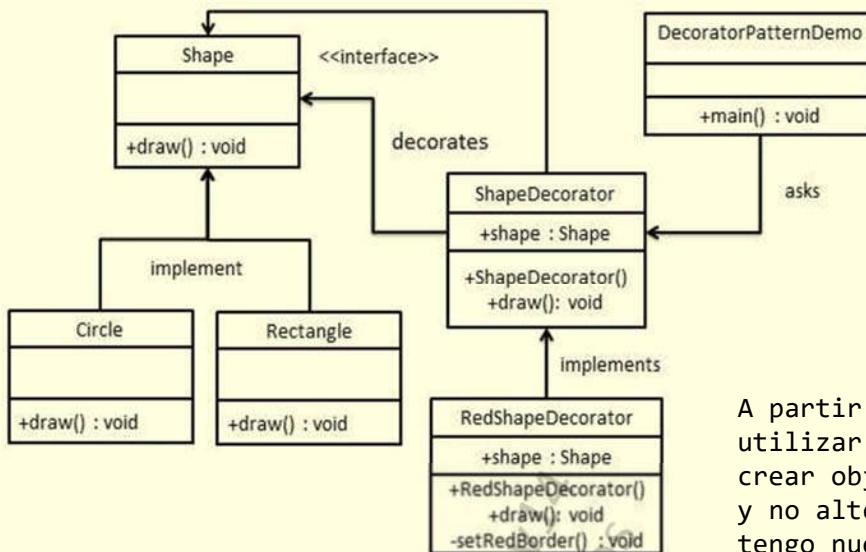


© Antonio Santos Ramos 2018

142

# Patrón “Decorator”

- Permite añadir nuevas funcionalidades a un objeto existente sin alterar su estructura



A partir de ahora puedo utilizar esta clase para crear objetos redefinidos y no altero el resto y tengo nuevos métodos

© Antonio Santos Ramos 2018

143

## “Estándares” de programación

```

package package_name
import class_names
import static static_member_name

public class class_name
    constants (using static final)
    static variables (using static)
    private instance variables
    private constructors
    public instance methods
        getters / setters
        instance methods
    protected instance methods
        getters / setters
        instance methods
    private instance methods
        getters / setters
        instance methods
    static methods
  
```

- Secure Coding Guidelines for Java SE
  - <https://www.oracle.com/technetwork/java/seccodeguide-139067.html>
- Upper Case, Camel Case, Snake Case, Kebab Case,...
- ¿Se te ocurren más normas?

# Bibliografía (Diseño avanzado de aplicaciones)

- Patrones de diseño (Erich Gamma)
- Design Patterns Java Woorkbook (Steven John)
- Head First design Patterns (Eric Freeman & Elisabeth Freeman)
- <http://sourcecodemania.com/grasp-design-patterns/>
- <http://migranitodejava.blogspot.com.es/search/label/Patrones> (Jun11)
- [http://raibledesigns.com/rd/entry/the\\_modern\\_java\\_web\\_developer](http://raibledesigns.com/rd/entry/the_modern_java_web_developer)
- [http://www.tutorialspoint.com/design\\_pattern/](http://www.tutorialspoint.com/design_pattern/)
- [http://sourcemaking.com/design\\_patterns](http://sourcemaking.com/design_patterns)
- <http://hillside.net/patterns/onlinepatterncatalog.htm>
- [http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=SOLID\\_5](http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=SOLID_5)
- <http://sourcecodemania.com/learn-to-use-design-patterns/>