Presentación

Java Web Developer | MitoCode

Henry Mendoza Puerta

Profesor en desarrollo de aplicaciones web

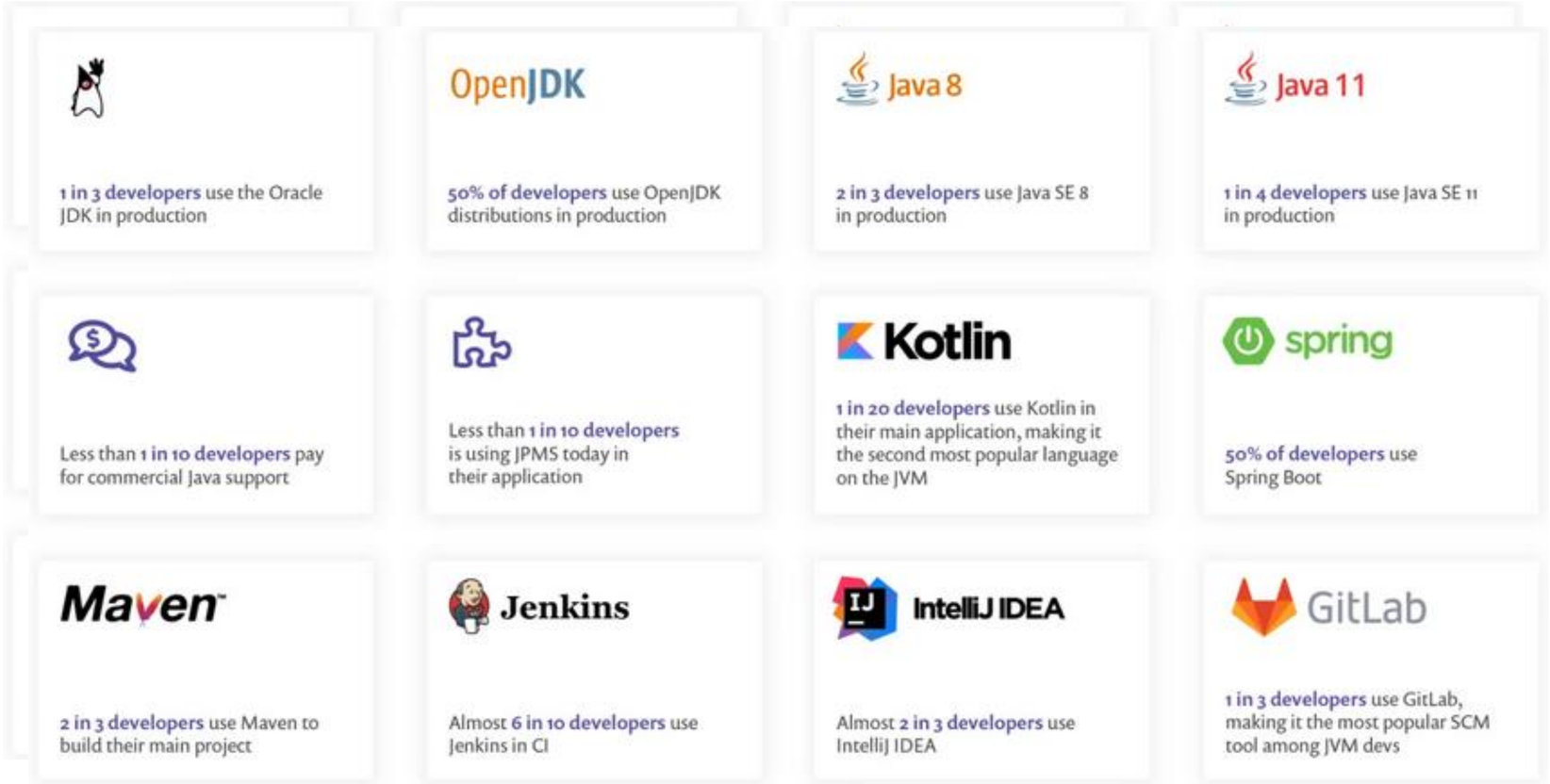
Temario

- Fundamentos de POO y Maven
- Programación Funcional
- Patrones de Diseño, Control de versiones y Gestor de dependencias
- Spring Boot
- Spring Data JPA - Parte 1
- Spring Data JPA - Parte 2
- Thymeleaf + Bootstrap 4 + JasperReports
- Spring Security y Despliegue en Heroku

Fundamentos de POO y Maven

Sesión 1

Java


1 in 3 developers use the Oracle JDK in production
OpenJDK
50% of developers use OpenJDK distributions in production
Java 8
2 in 3 developers use Java SE 8 in production
Java 11
1 in 4 developers use Java SE 11 in production
Less than 1 in 10 developers pay for commercial Java support
Less than 1 in 10 developers is using JPMS today in their application
Kotlin
1 in 20 developers use Kotlin in their main application, making it the second most popular language on the JVM
spring
50% of developers use Spring Boot
Maven
2 in 3 developers use Maven to build their main project
Jenkins
Almost 6 in 10 developers use Jenkins in CI
IntelliJ IDEA
Almost 2 in 3 developers use IntelliJ IDEA
GitLab
1 in 3 developers use GitLab, making it the most popular SCM tool among JVM devs

Alternativas a JDK

OpenJDK

Amazon Corretto

AdoptOpenJDK

Dependecias

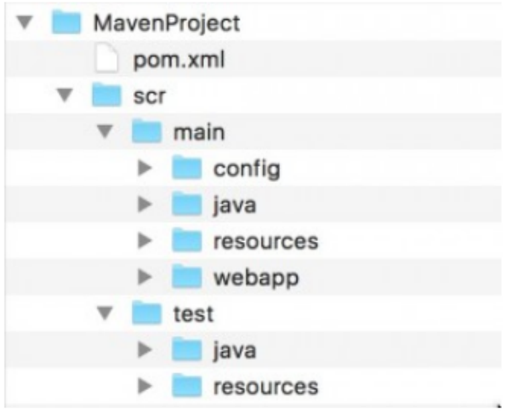
Maven

Clean Lifecyle
pre-clean
clean
post-clean

Default Lifecyle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
processs-test-resources	

Site Lifecyle
pre-site
site
post-site
site-deploy

Maven



Cada una de las carpetas de la jerarquía contiene lo siguiente:

- **src**: código fuente, ficheros de configuración, recursos y demás. Es decir, todo salvo el directorio de salida (target) y el pom.xml
- **main**: desarrollo de la aplicación, independientemente de los test
- **test**: desarrollo de los test de la aplicación
- **config**: ficheros de configuración en el proyecto
- **java**: clases java que contienen el código fuente de la aplicación
- **resources**: recursos que se incluirán en el empaquetado y serán usados por la aplicación, como pueden ser ficheros de texto o scripts
- **webapp**: contiene todos los ficheros correspondientes a la propia aplicación web que no sean código java

Maven

- **project**: Es la etiqueta que recoge todas las demás.
- **modelVersion**: Indica qué versión de la especificación POM se está utilizando.
- **groupId**: Identificador único del grupo de proyectos dentro de la organización en la que nos encontremos. Suele coincidir con el nombre base del paquete java, añadiendo el nombre del proyecto.
- **artifactId**: Identificador único del artefacto correspondiente al proyecto. Por defecto será el nombre de paquetería.
- **version**: Versión del proyecto en la cual estamos trabajando. Se le puede añadir el sufijo SNAPSHOT para indicar que es una versión en desarrollo.
- **packaging**: Tipo de empaquetado del proyecto (jar, war, esb...)
- **name**: Nombre del proyecto.
- **description**: Descripción del proyecto.
- **dependencies**: Dentro de esta etiqueta incluiremos las dependencias necesarias, cada una de ellas precedida de la etiqueta **dependency**. En la web http://mvnrepository.com/ podremos encontrar cualquiera de ellas, facilitándonos la plataforma el texto que debemos añadir en el pom.xml para incluirlas en el proyecto.

A continuación tenemos un ejemplo muy sencillito de pom.xml, en el cual se pueden ver todas las etiquetas enumeradas.

Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="<a href="http://www.w3.org/20(

	<modelVersion>4.0.0</modelVersion>
	<groupId>com.autentia</groupId>
	<artifactId>autentiaNegocio</artifactId>
	<packaging>jar</packaging>
	<version>1.0-SNAPSHOT</version>
	<name>Maven Quick Start Archetype</name>
	<dependencies>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>4.12</version>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>log4j</groupId>
			<artifactId>log4j</artifactId>
			<version>1.2.17</version>
		</dependency>
	</dependencies>
</project>
```

Maven

Guide to naming conventions on groupId, artifactId, and version

- **groupId** uniquely identifies your project across all projects. A group ID should follow Java's package name rules. This means it starts with a reversed domain name you control. For example,
  `org.apache.maven`, `org.apache.commons`

  Maven does not enforce this rule. There are many legacy projects that do not follow this convention and instead use single word group IDs. However, it will be difficult to get a new single word group ID approved for inclusion in the Maven Central repository.

  You can create as many subgroups as you want. A good way to determine the granularity of the `groupId` is to use the project structure. That is, if the current project is a multiple module project, it should append a new identifier to the parent's `groupId`. For example,

  `org.apache.maven`, `org.apache.maven.plugins`, `org.apache.maven.reporting`

- **artifactId** is the name of the jar without version. If you created it, then you can choose whatever name you want with lowercase letters and no strange symbols. If it's a third party jar, you have to take the name of the jar as it's distributed.
  eg. `maven`, `commons-math`

- **version** if you distribute it, then you can choose any typical version with numbers and dots (1.0, 1.1, 1.0.1, ...). Don't use dates as they are usually associated with SNAPSHOT (nightly) builds. If it's a third party artifact, you have to use their version number whatever it is, and as strange as it can look. For example,
  `2.0`, `2.0.1`, `1.3.1`

Maven

Versionado Semántico 2.0.0

Resumen

Dado un número de versión MAYOR.MENOR.PARCHE, se incrementa:

1. la versión MAYOR cuando realizas un cambio incompatible en el API,
2. la versión MENOR cuando añades funcionalidad que compatible con versiones anteriores, y
3. la versión PARCHE cuando reparas errores compatibles con versiones anteriores.

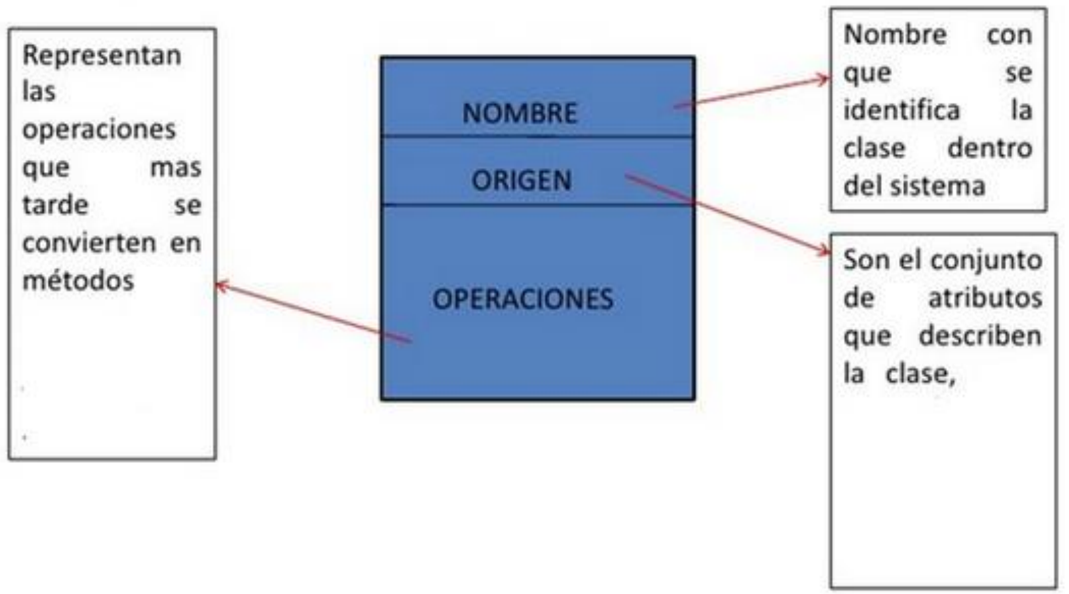
Hay disponibles etiquetas para prelanzamiento y metadata de compilación como extensiones al formato MAYOR.MENOR.PARCHE.

POO

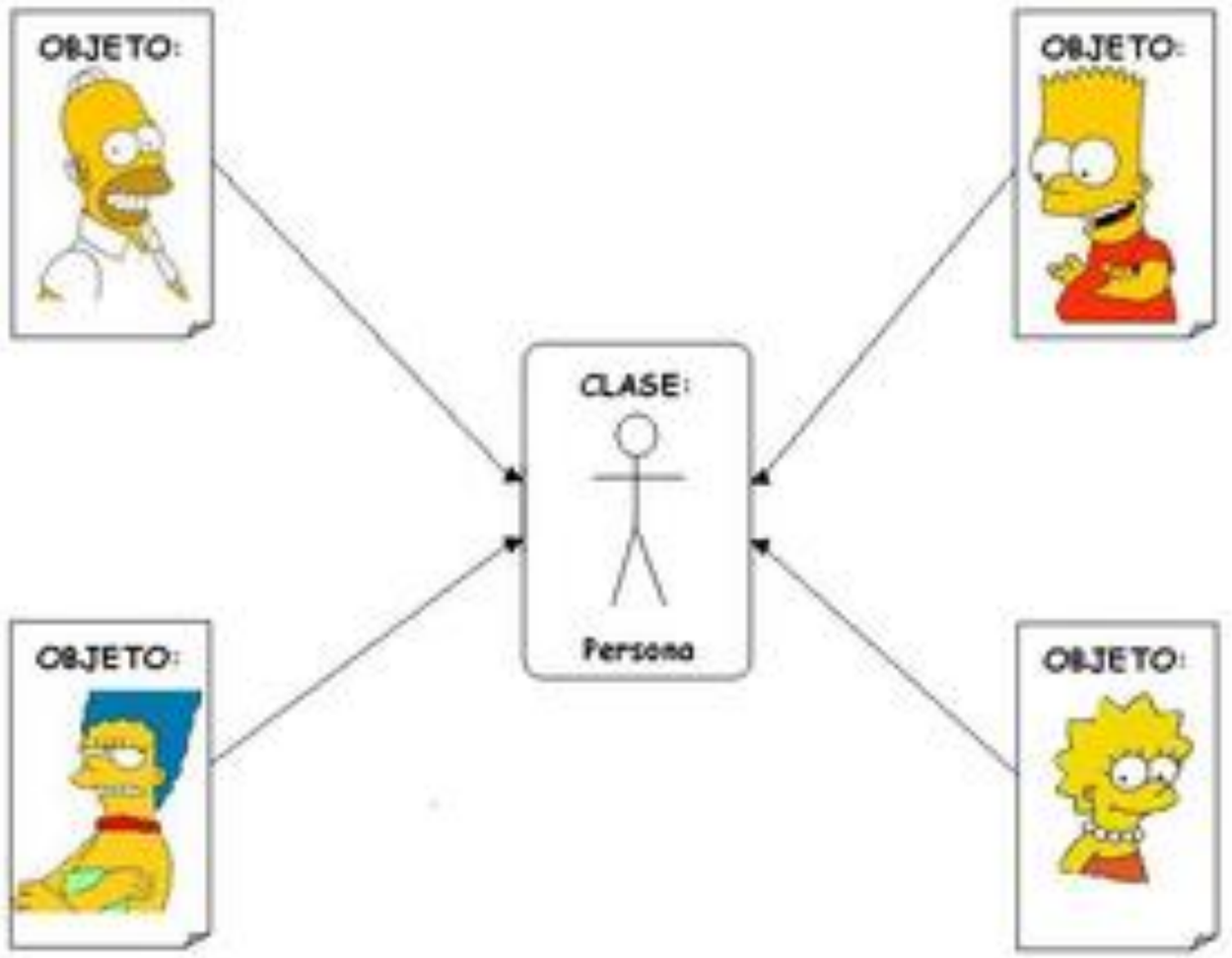

Abstracción
Expresa las características esenciales de un objeto.
Encapsulamiento
ocultamiento del estado, es decir, de los datos de un objeto de manera que sólo se pueda cambiar mediante las operaciones definidas para ese objeto.
Herencia
posibilidad de crear nuevas clases basadas en una clase existente.
Polimorfismo
palabra griega que significa "con muchas formas". Tareas similares son realizadas por métodos con mismo nombre.

POO
Clase

- Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase).
- A través de ella podemos modelar el entorno que se desea estudiar (una Casa, un Auto, una Cuenta Corriente, etc.).
- Generalmente representa un sustantivo (persona, lugar o cosa)



POO
Objeto



POO Diagrama de Clases

Los diagramas de clases son uno de los tipos de diagramas más útiles en UML, ya que trazan claramente la estructura de un sistema concreto al modelar sus clases, atributos, operaciones y relaciones entre objetos. Con nuestro
software de generación de diagramas UML, la creación de estos diagramas no es tan abrumadora como podría parecer. Esta guía te ayudará a entender, planificar y crear tu propio diagrama de clases.

Demo