

JAVA AVANZADO

ÍNDICE

- Generics, Packages, Estructuras de Memoria
- Multithreading en Java
- Introducción al Acceso a Datos con Java
- Creación de Componentes Reutilizables en Java
- Servidores de Aplicaciones JAVA (JBoss, Wildfly)
- Aplicaciones Cliente-Servidor (REST, SOAP)
- Manejo de Dependencias y Empaquetamiento (Maven)
- ORM con JavaEE
- TDD con JUnit 5 y Mockito



Generics, Packages, Estructuras de Memoria



➤ ¿Qué son los genéricos?

- Permiten crear clases, interfaces y métodos donde el tipo de dato se especifica como parámetro.
- Mejoran la reutilización del código y la seguridad de tipos.
- Ejemplo: class Gen<T> {...}

Ventajas de los genéricos

- Reutilización de código
- Seguridad de tipos
- Evitan conversiones explícitas
- Aplicables a múltiples tipos sin reescribir el algoritmo

➤ Antes de los genéricos

- Se usaban referencias de tipo Object.
- Problemas:
 - Conversión explícita requerida
 - Posibles errores de tipo
- Con genéricos: el compilador maneja automáticamente la conversión

➤ Sintaxis básica

```
class Gen<T> {  
    T ob;  
    Gen(T o) { ob = o; }  
    T getOb() { return ob; }  
    void mostrarTipo() {...}  
}  
Gen<Integer> iOb = new Gen<>(28);  
Gen<String> sOb = new Gen<>("Texto");
```

➤ Métodos genéricos

- Se pueden declarar dentro o fuera de clases genéricas.
- Ejemplo:
- `static <T extends Comparable<T>> boolean igualArrays(T[] x, T[] y) {...}`

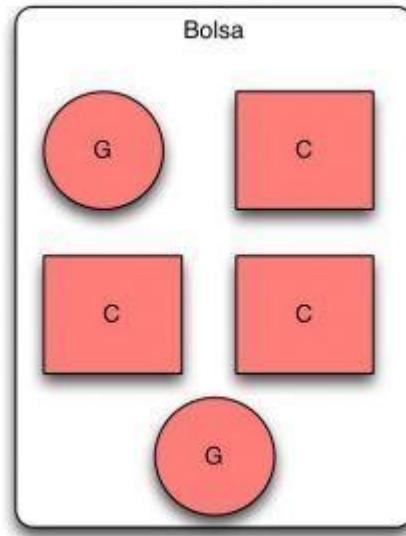
➤ Constructores genéricos

- Una clase no genérica puede tener un constructor genérico.
- Ejemplo:
- class SumaN { <T extends Number> SumaN(T arg) {...} }

➤ Interfaces genéricas

- . interface Contenedor<T> {
 boolean contenido(T o); }
- . class MiClase<T> implements
 Contenedor<T> {...}

Ejemplo



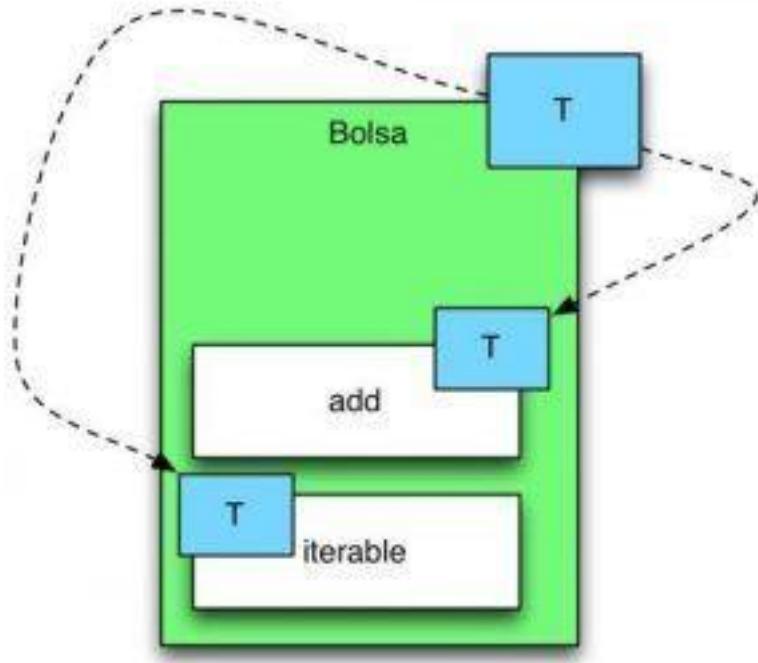


Ejemplo

<Clase>
Golosina

<Clase>
Chocolatina

Ejemplo



Colecciones genéricas - Set

- . - HashSet: alta eficiencia, sin orden, admite null
- . - TreeSet: ordenado, sin duplicados, no admite null
- . - LinkedHashSet: orden de inserción, admite null

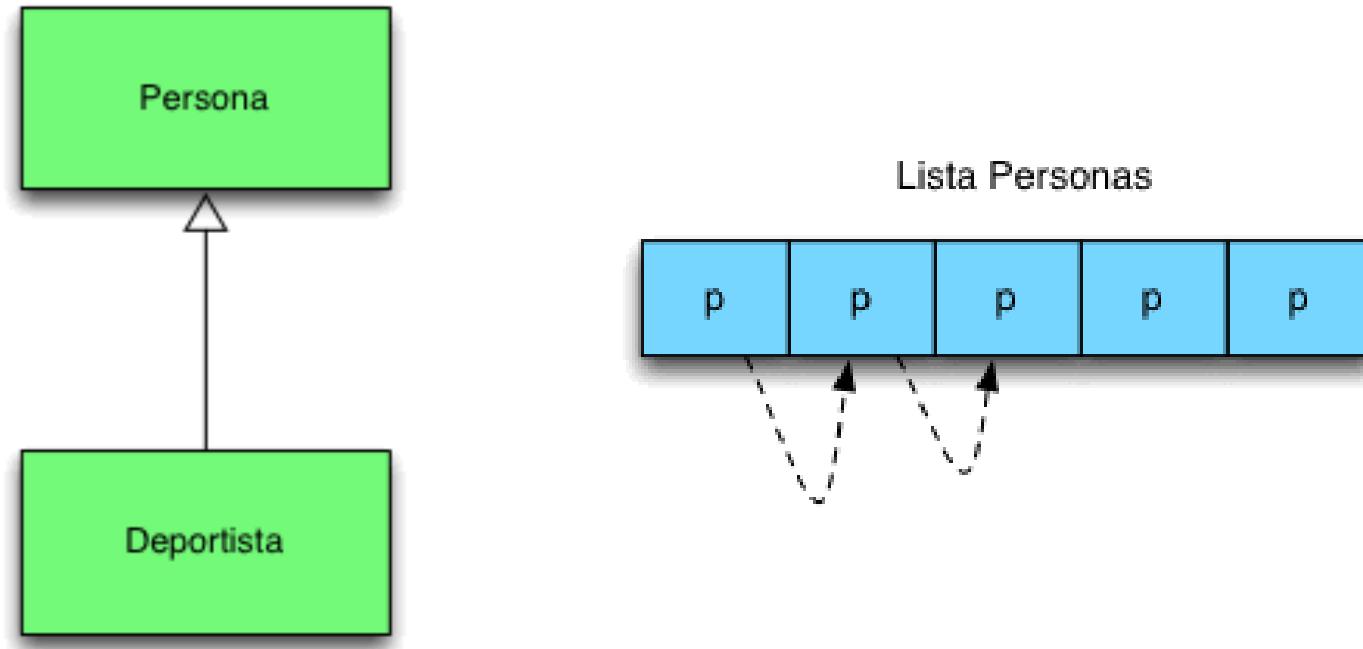
➤ Colecciones genéricas - List

- **ArrayList:** array redimensionable, acceso rápido
- **LinkedList:** lista doblemente enlazada, mejor inserción/borrado

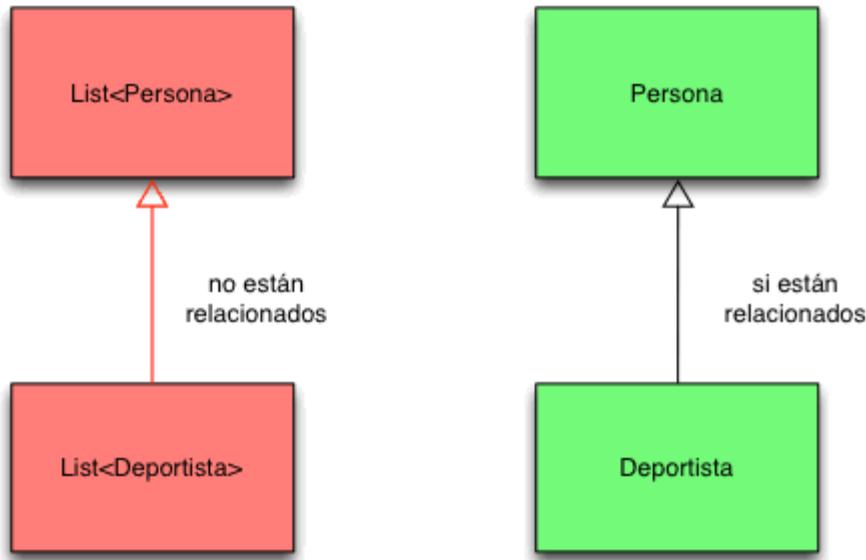
Colecciones genéricas - Map

- . - **HashMap**: sin orden, alto rendimiento
- . - **TreeMap**: ordenado por claves, más lento
- . - **LinkedHashMap**: orden de inserción

Ejemplo



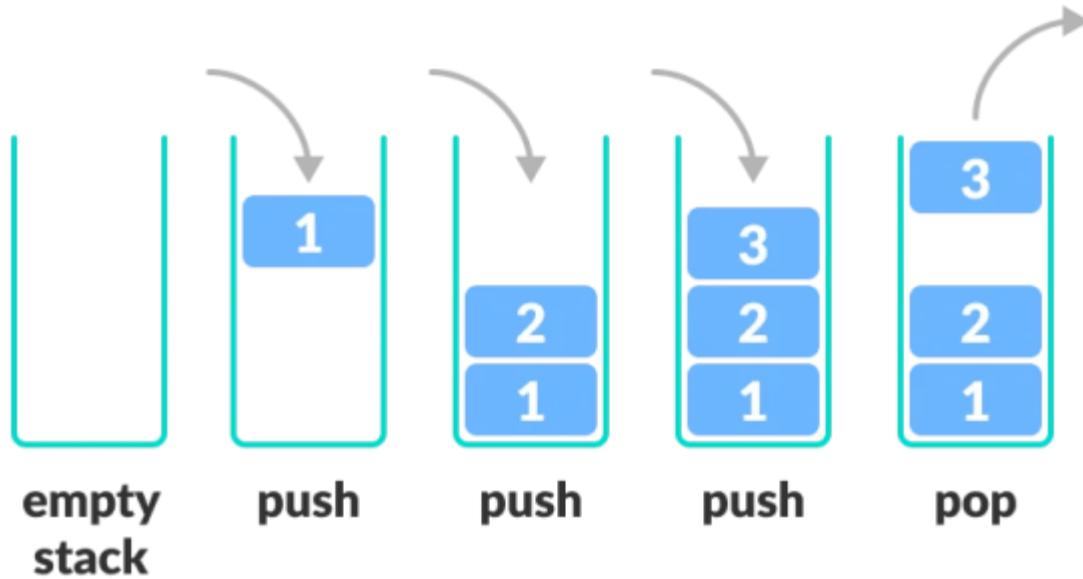
Ejemplo



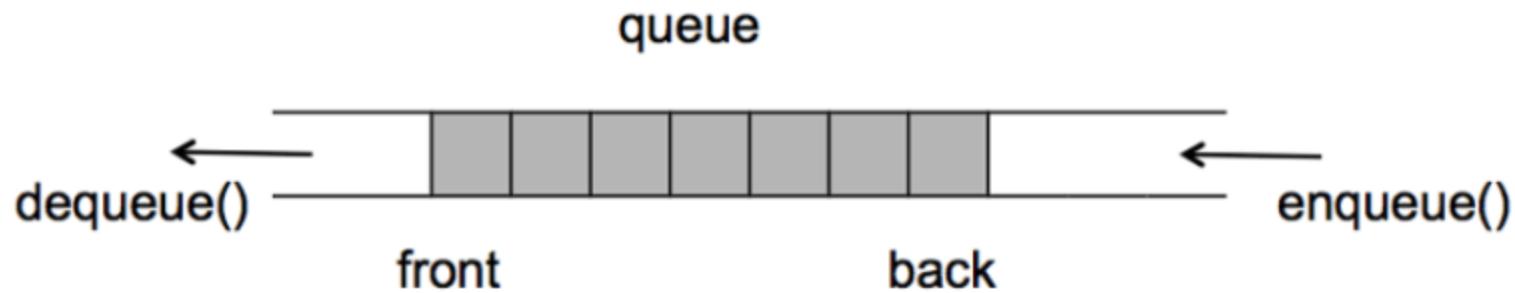
➤ Manejo de memoria en Java

- . - Stack: almacena variables primitivas y llamadas de métodos
- . - Heap: almacena objetos
- . Java usa Garbage Collector para limpiar el heap

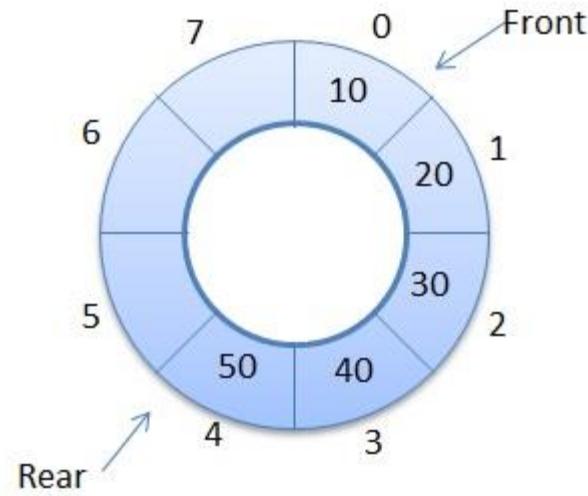
➤ Pila



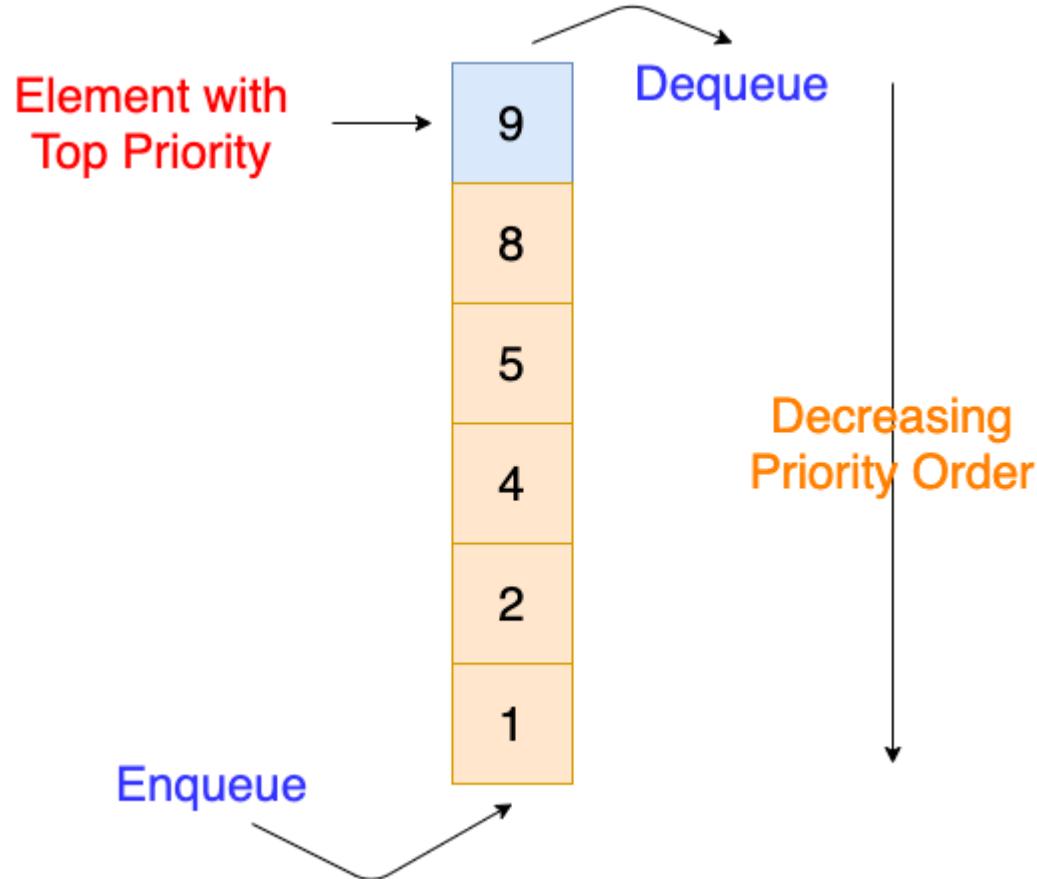
Cola



Cola Circular con Memoria Estática



Cola Dinámica con Prioridad



- Multithreading

en Java





¿Qué es la concurrencia?

- Capacidad para ejecutar múltiples instancias en paralelo.
- Mejora el rendimiento al aprovechar múltiples CPUs.
- Permite mayor eficiencia en sistemas modernos.

➤ Multithreading en Java

- Java soporta múltiples hilos dentro de una aplicación.
- Un hilo es una unidad ligera de ejecución.
- Los subprocessos comparten memoria y pueden cambiar de contexto fácilmente.



Ventajas del Multithreading

- Uso eficiente de la CPU
- Mejora la capacidad de respuesta de la interfaz gráfica
- Reduce tiempos de inactividad
- Mejor aprovechamiento de múltiples CPUs

➤ Hilos en Java

- Tipos:
 - - Hilo de usuario
 - - Hilo daemon
- Se crean mediante:
 - - Extensión de la clase Thread
 - - Implementación de la interfaz Runnable

➤ Ejemplo básico de hilo

```
. Thread hilo1 = new Thread();  
. hilo1.start();  
  
. class MiClase implements  
Runnable {  
    public void run() { ... }  
}  
.
```

➤ Ciclo de vida del hilo

- - New
- - Runnable
- - Running
- - Waiting
- - Dead
- El estado cambia durante la ejecución de acuerdo a la planificación y sincronización.

➤ Ejemplo de ciclo de vida

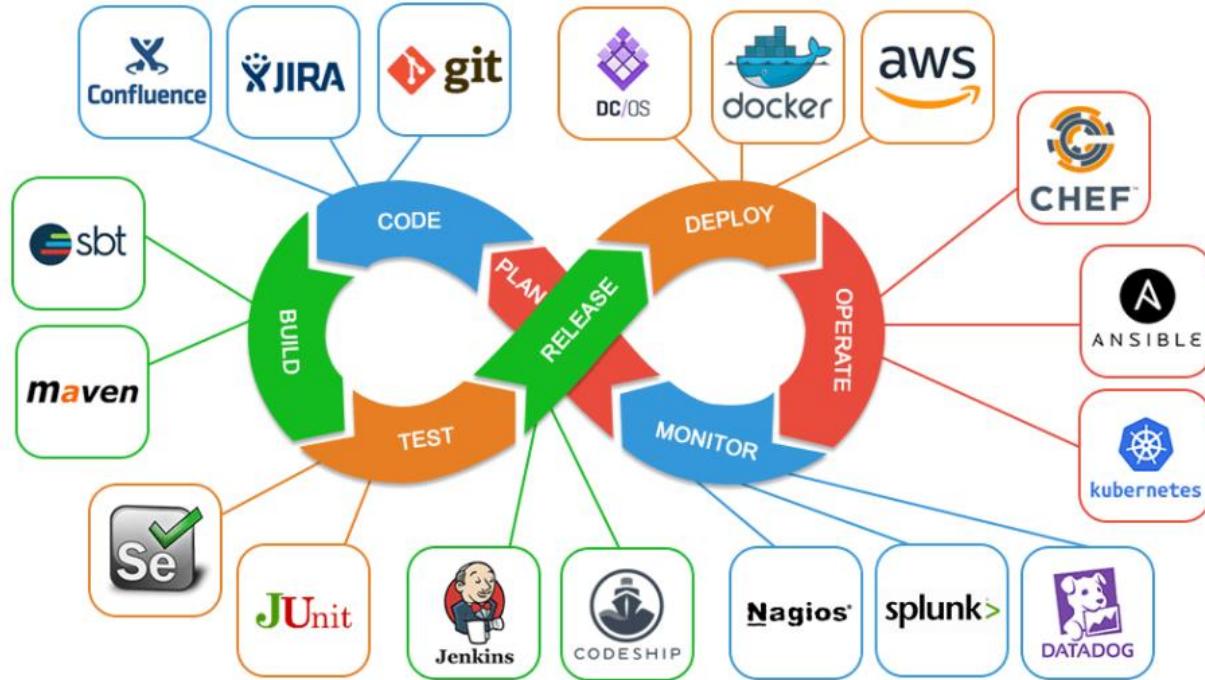
- . Thread t = new Thread();
- . t.start();
- . t.sleep(1000);
- . t.setPriority(1);
- . System.out.println(t.getPriority());

Sincronización de hilos

- Permite evitar condiciones de carrera.
- Solo un hilo accede a un bloque sincronizado a la vez.
- Métodos sincronizados aseguran acceso exclusivo a recursos compartidos.

- Manejo de
**Dependencias y
Empaquetamiento (Maven)**





¿QUE ES DEVOPS?

Ventajas



Una mejor y más rápida entrega de productos



Resolución de problemas en menos



Mejor escalabilidad y disponibilidad



Entornos de funcionamiento más estables



Mejor utilización de los recursos

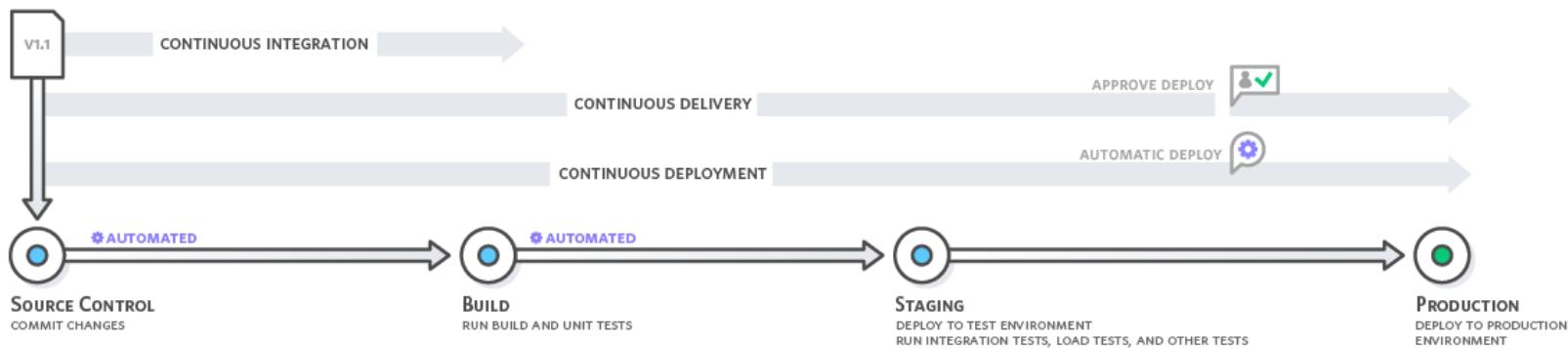
Mayor automatización

Mayor visibilidad de resultados del sistema

Mayor innovación

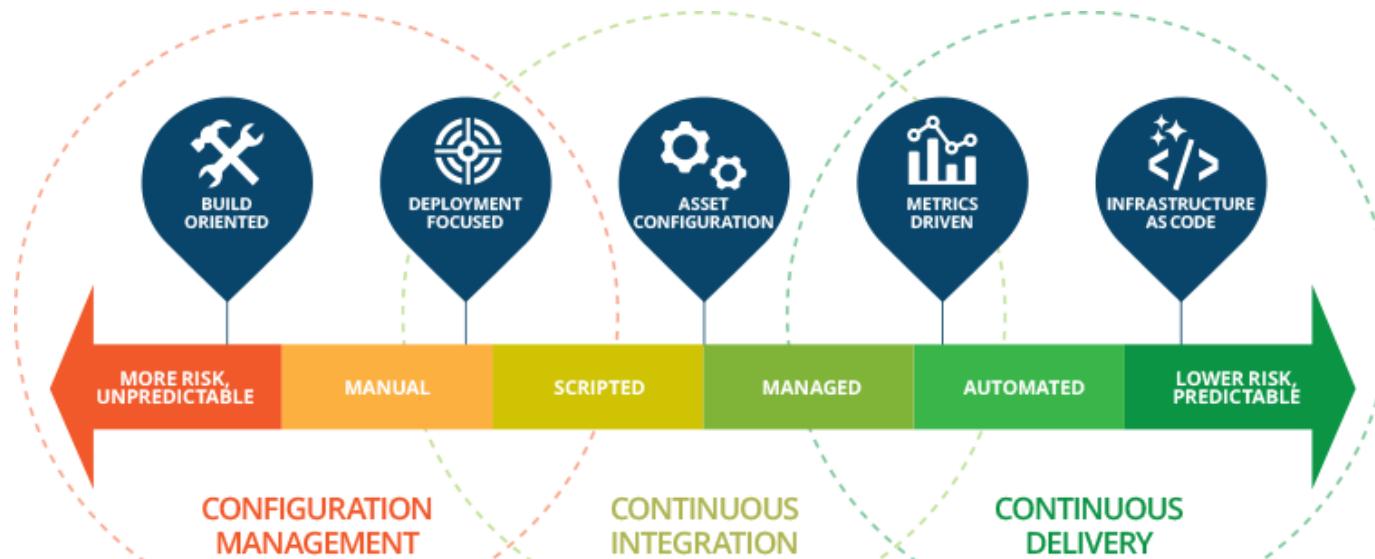


CONCEPTOS: INTEGRACIÓN CONTINUA



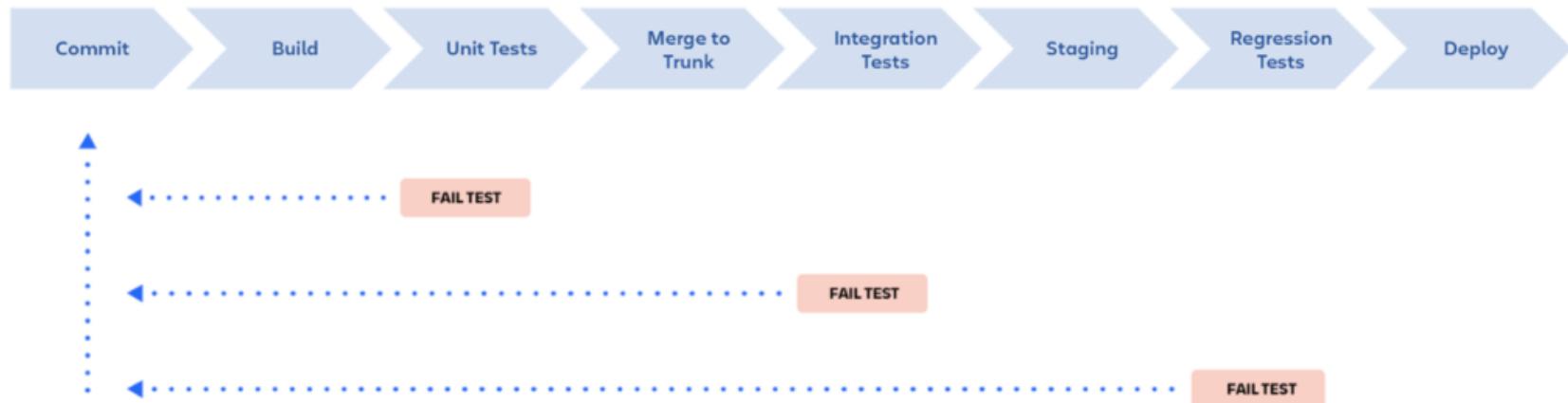


CONCEPTOS: ENTREGA CONTINUA





CONCEPTOS: PIPELINE





CONCEPTOS: REPOSITORIO





Conceptos: gestor de dependencias



Collaborate

Application Lifecycle Mgmt.



Communication & ChatOps



Knowledge Sharing



Build

SCM/VCS



Test

Testing



Deploy

Deployment



Run

Cloud / IaaS / PaaS



CI



Config Mgmt./Provisioning



Orchestration & Scheduling



Build



Artefact Management



BI / Monitoring / Logging



HERRAMIENTAS

0 A● Bugs207 B● Vulnerabilities0 A● New Bugs207 B● New VulnerabilitiesCode Smells P155d B

Deflt

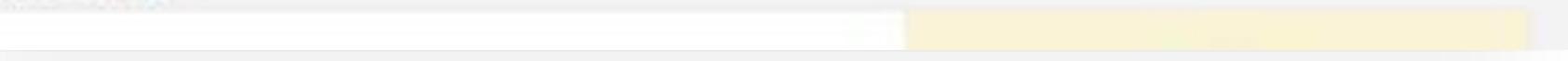
inserted 2 hours ago

4.4k

● Code Smells154d B

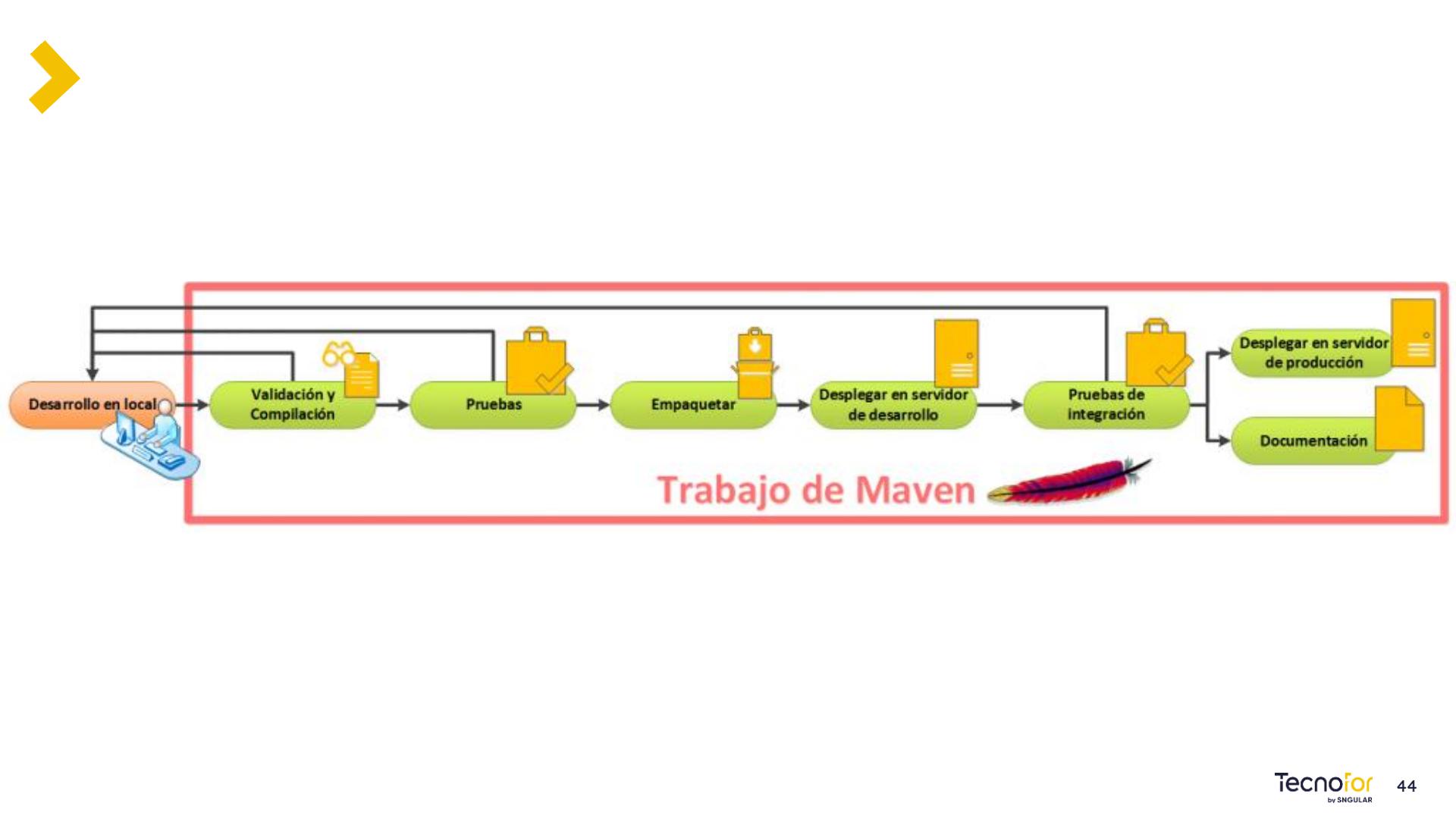
New Deflt

4.3k

● New Code SmellsCoverage P66.7%
Coverage14k
Unit Tests—
Coverage on New CodeDuplications P

Calidad De Software





Artefacto (Artifact)

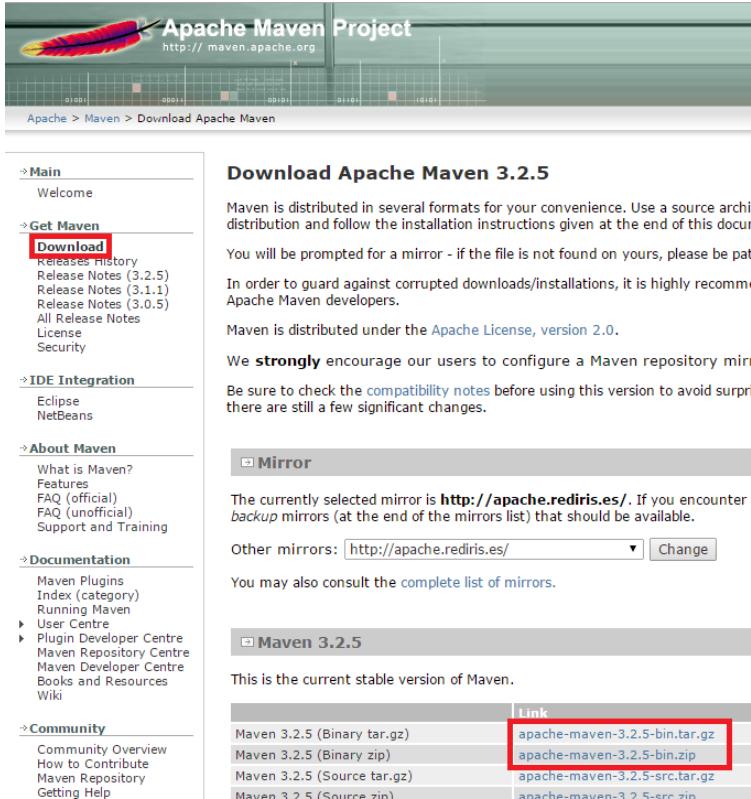
nombreDeMiProyecto

- ▷ src/main/java
- ▷ src/main/resources
- ▷ src/test/java
- ▷ src/test/resources
- ▷ JRE System Library [J2SE-1.5]
- ▷ Maven Dependencies
- ▷ src
- ▷ target

Fichero POM

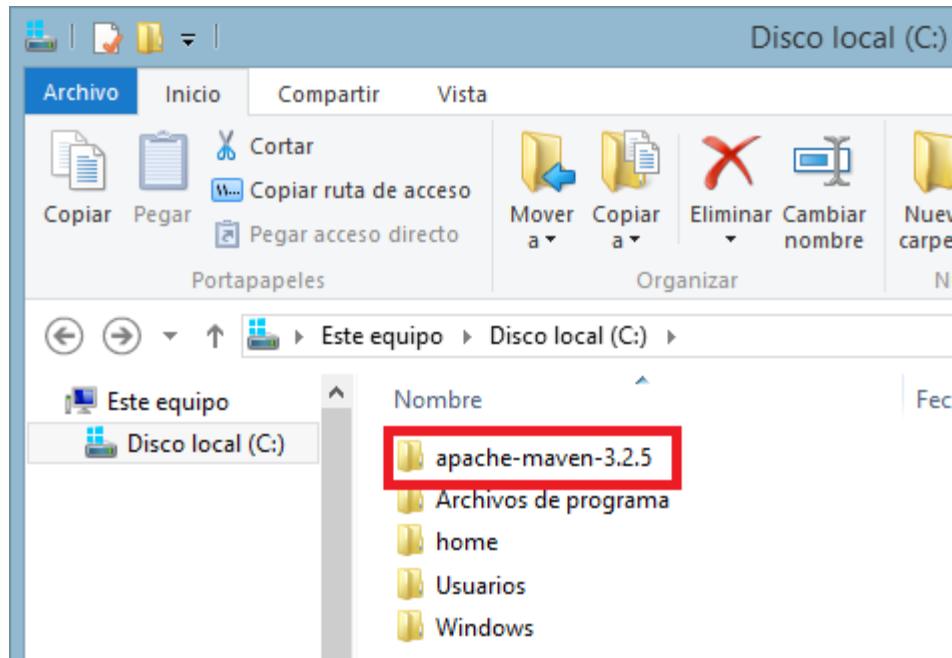
pom.xml

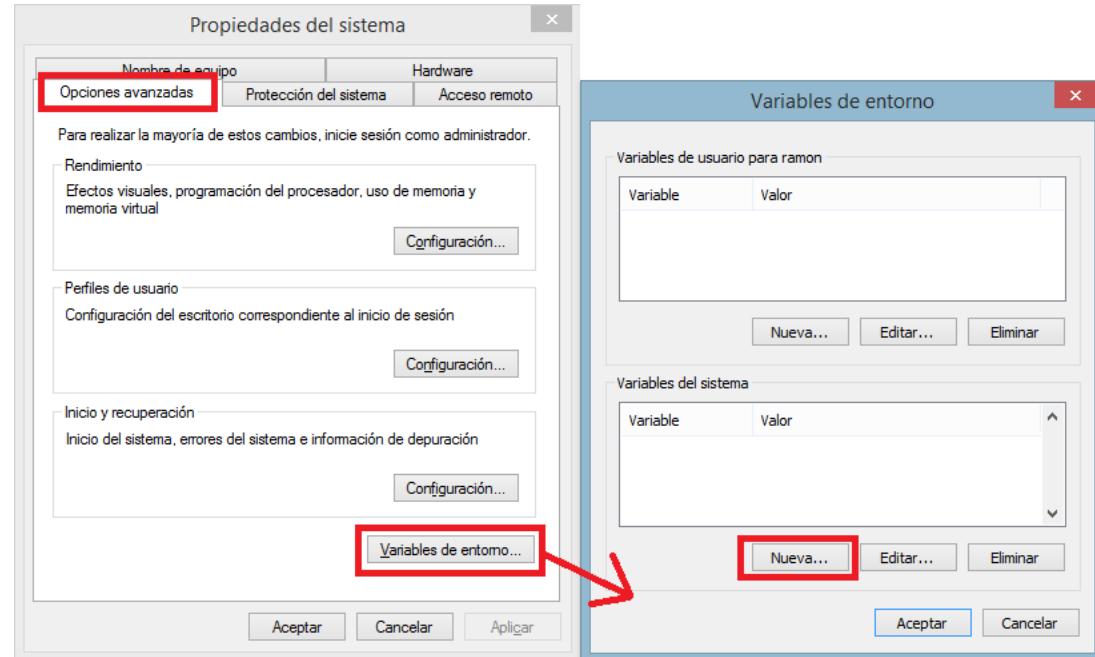
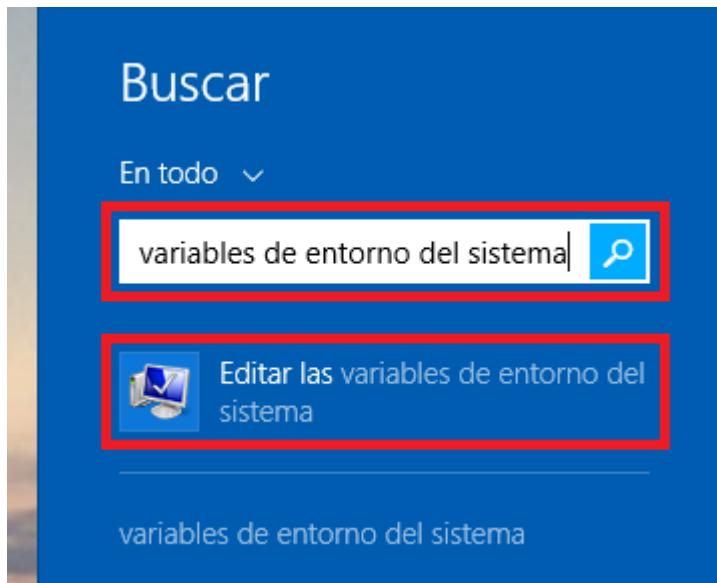
Instalar Maven

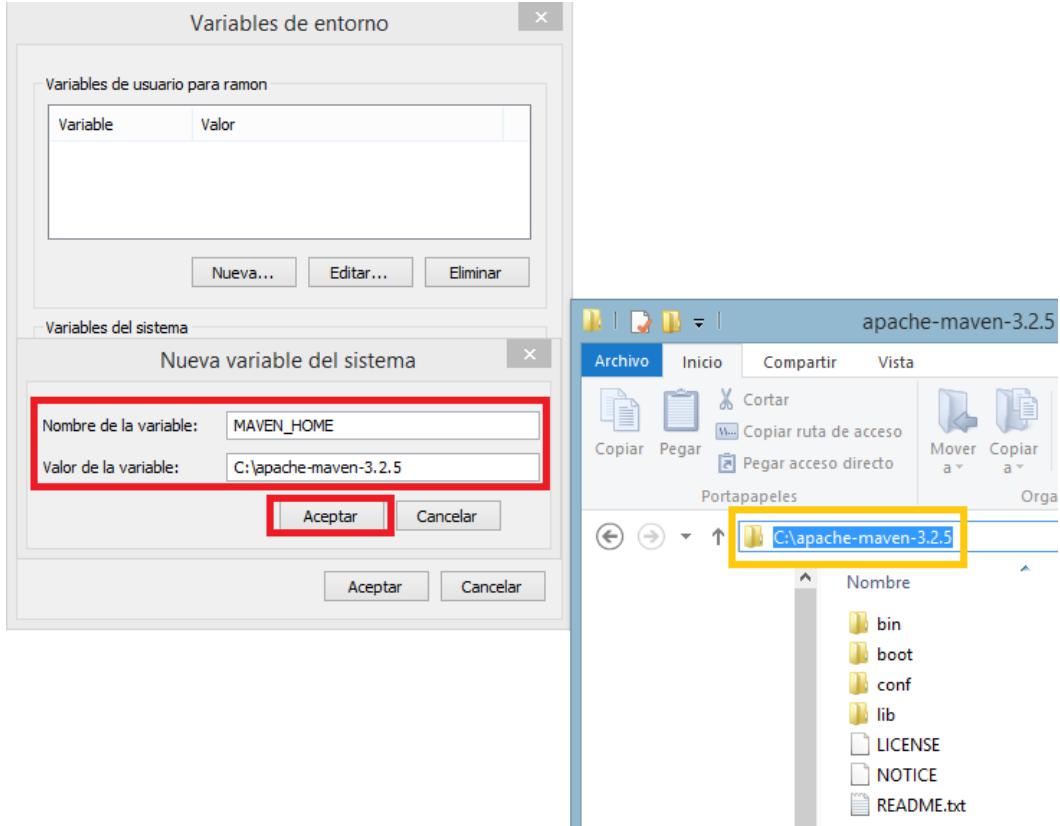


The screenshot shows the Apache Maven Project website. The main navigation menu on the left includes sections like Main, Get Maven (with a red box around the Download link), IDE Integration, About Maven, Documentation, and Community. The 'Get Maven' section is expanded, showing links for Releases History, Release Notes (3.2.5, 3.1.1, 3.0.5), All Release Notes, License, and Security. The 'Download' link is also highlighted with a red box. Below this, there's a 'Mirror' section with a dropdown menu set to 'http://apache.rediris.es/' and a 'Change' button. The 'Maven 3.2.5' section is shown as the current stable version. A table lists download links for Maven 3.2.5 in various formats, with the 'apache-maven-3.2.5-bin.zip' link highlighted with a red box.

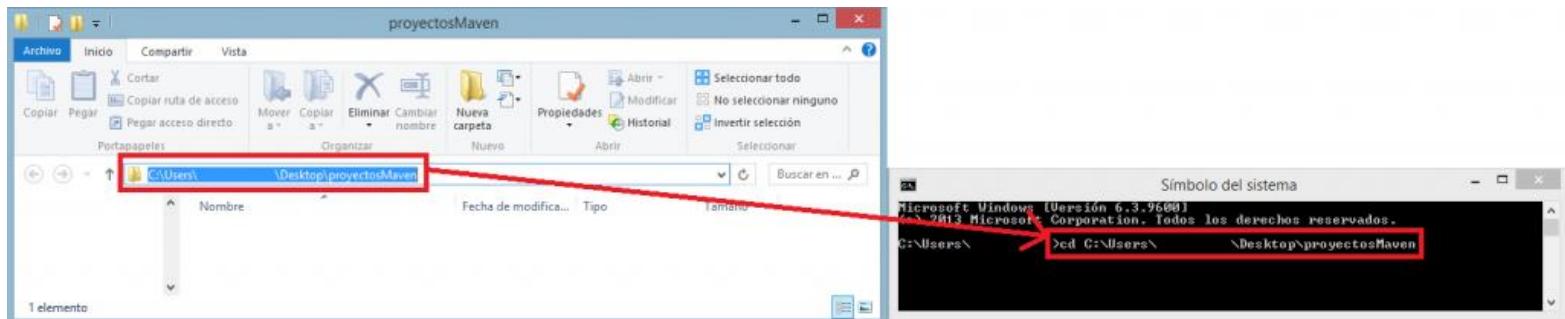
	Link
Maven 3.2.5 (Binary tar.gz)	apache-maven-3.2.5-bin.tar.gz
Maven 3.2.5 (Binary zip)	apache-maven-3.2.5-bin.zip
Maven 3.2.5 (Source tar.gz)	apache-maven-3.2.5-src.tar.gz
Maven 3.2.5 (Source zip)	apache-maven-3.2.5-src.zip



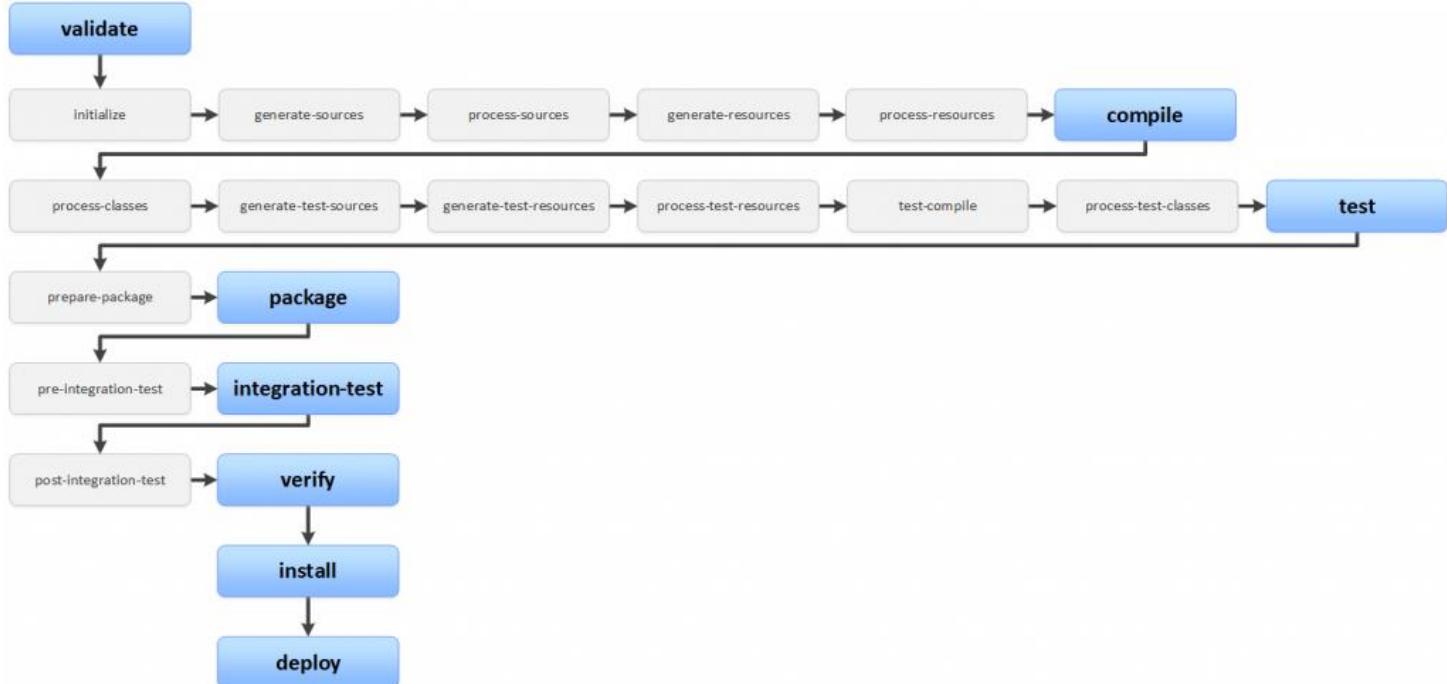




Crear un proyecto con Maven



```
Símbolo del sistema - □ ×
Choose a number or apply filter <format: [groupId:lartifactId, case sensitive contains]: 510:0
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6:0
Define value for property 'groupId': : com.jarroba.ejemplo
Define value for property 'artifactId': : nombreDeMiProyecto
Define value for property 'version': 1.0-SNAPSHOT: :0
Define value for property 'package': com.jarroba.ejemplo: :0
Confirm properties configuration:
groupId: com.jarroba.ejemplo
artifactId: nombreDeMiProyecto
version: 1.0-SNAPSHOT
package: com.jarroba.ejemplo
Y: :0
[INFO] -----
[INFO] Using following parameters for creating project from Old <1.x> Archetype:
maven-archetype-quickstart:1.1
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\     \Desktop\proyectosMaven
[INFO] Parameter: package, Value: com.jarroba.ejemplo
[INFO] Parameter: groupId, Value: com.jarroba.ejemplo
[INFO] Parameter: artifactId, Value: nombreDeMiProyecto
[INFO] Parameter: packageName, Value: com.jarroba.ejemplo
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old <1.x> Archetype in dir: C:\Users\     \Desktop\proyectosMaven\nombreDeMiProyecto
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:49 min
[INFO] Finished at: 2015-01-02T15:11:52+01:00
[INFO] Final Memory: 13M/155M
[INFO] -----
C:\Users\     \Desktop\proyectosMaven>
```



- *El fichero “pom.xml” es el núcleo una configuración de un proyecto en Maven. Simplemente es un fichero de configuración, que contiene la mayoría de la información necesaria para construir (build) un proyecto de la manera en la que tú quieras. El POM es enorme e intimida su complejidad, no siendo necesario comprender todos sus entresijos para utilizarlo de manera eficaz.*

MVN Repository

The screenshot shows the MVNRepository search interface. A red box highlights the search bar containing the query "junit". The search results page displays a chart titled "Artifacts/Year" showing a sharp increase in artifact count from 2004 to 2015, reaching approximately 828k. The main search results area shows "Found 472 results" and lists the top result, "1. JUnit", which has 26,341 usages. The result card for JUnit includes categories "Testing Frameworks" and a brief description: "JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck." Below the description is a link "junit » junit".

MVNREPOSITORY

junit

Search

Artifacts/Year

Found 472 results

1. JUnit

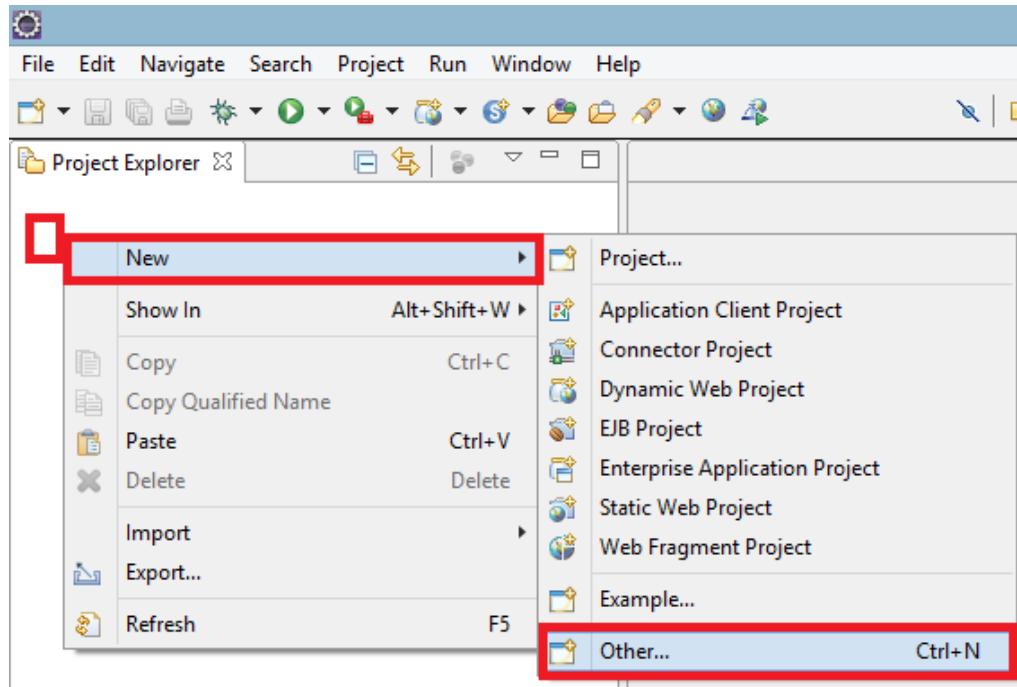
26,341 usages

Categories: Testing Frameworks

JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck.

junit » junit

> Eclipse





Estructura

- src/main/java. Contiene el código fuente. Aquí escribiremos (o se incluirán si se carga el proyecto) las clases con extensión .java. El contenido de este directorio se conoce como módulo.
- src/main/resources. Contiene los recursos estáticos (XML, propiedades, imágenes...) que necesita nuestro módulo para funcionar correctamente.
- src/test/java. Contiene los ficheros de prueba (testing) para verificar el correcto funcionamiento del módulo.
- src/test/resources. Almacena los archivos que genera Maven cuando usamos los comandos (compile, package...).

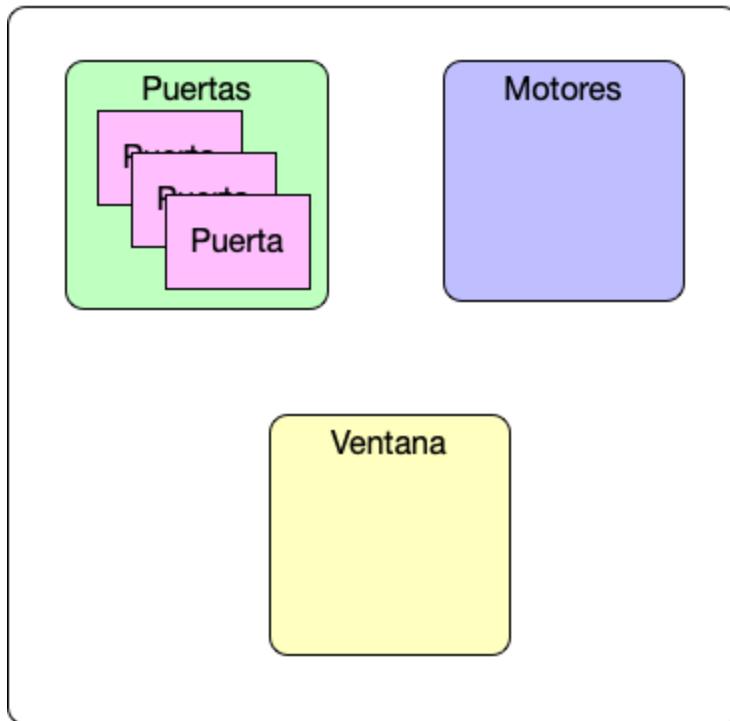


Dependencias

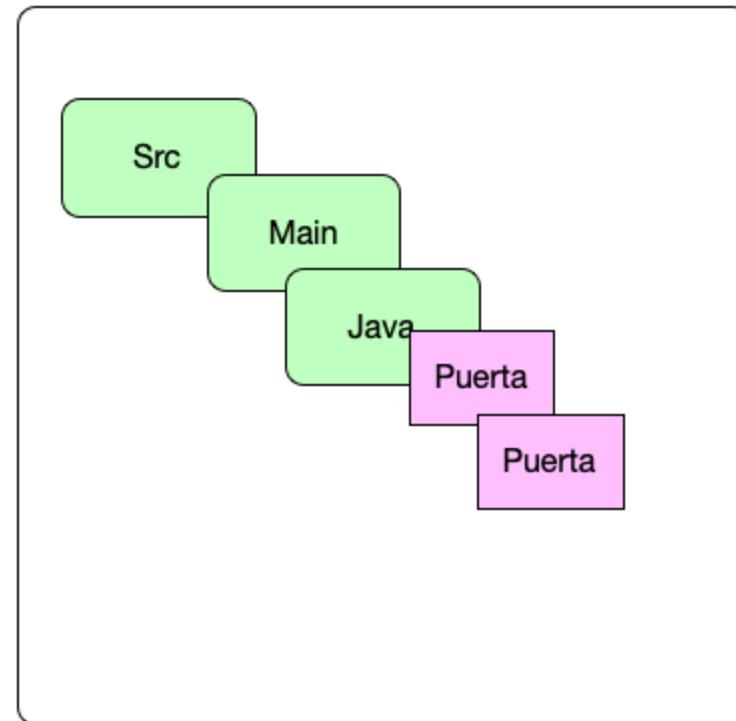
- Dependency: Elemento contenedor padre mediante el que se especifica que se trata de una dependencia.
- GroudId: Dependiendo del caso, puede no resultar obligatorio, aunque si se añade no supondrá un problema.
- ArtifactId: Su definición resulta obligatoria.
- Version: Si no la especificamos, Maven seleccionará la última versión o la que especifiquemos en el pom padre (superpom.xml). Aunque no está de más especificarla, si no lo hacemos, tampoco supondrá un problema, siempre y cuando esta dependencia no entre en conflicto con otras.



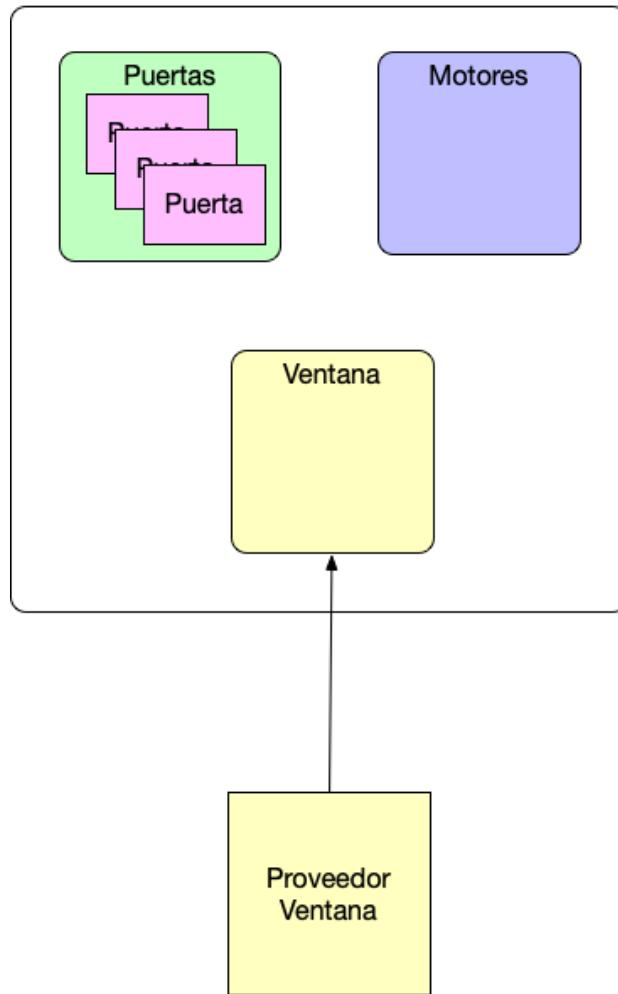
Almacen



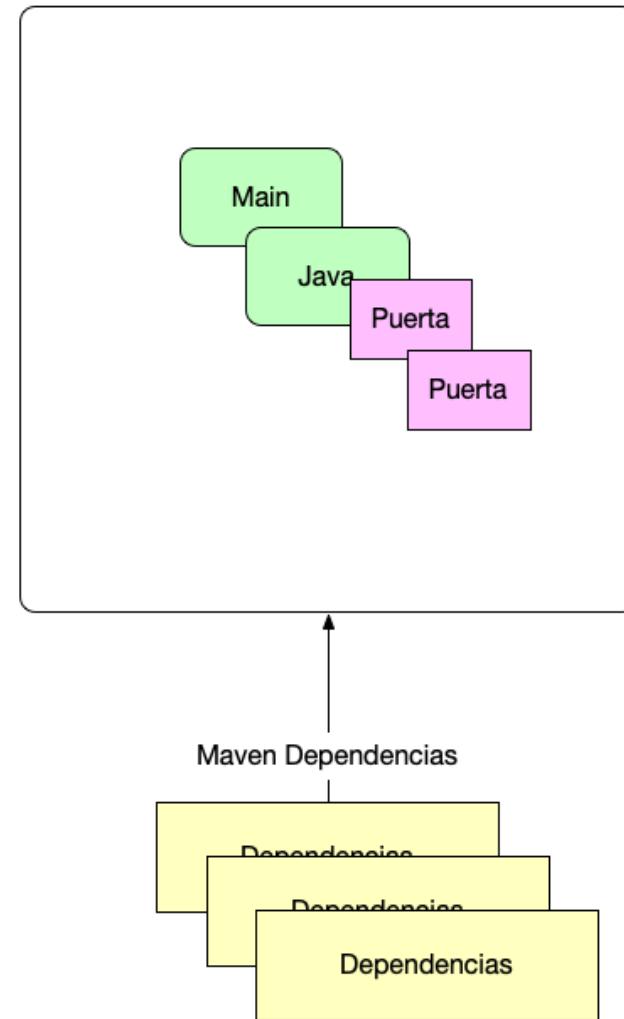
Carpetas

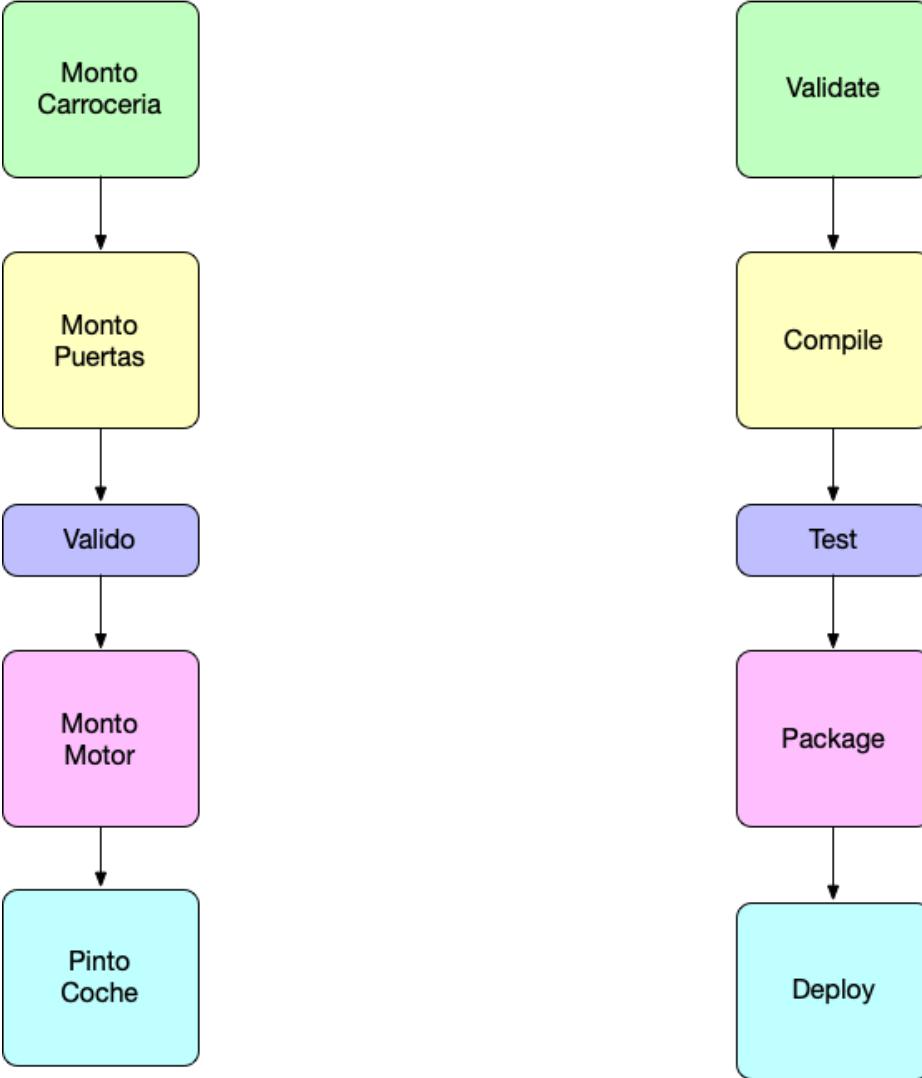


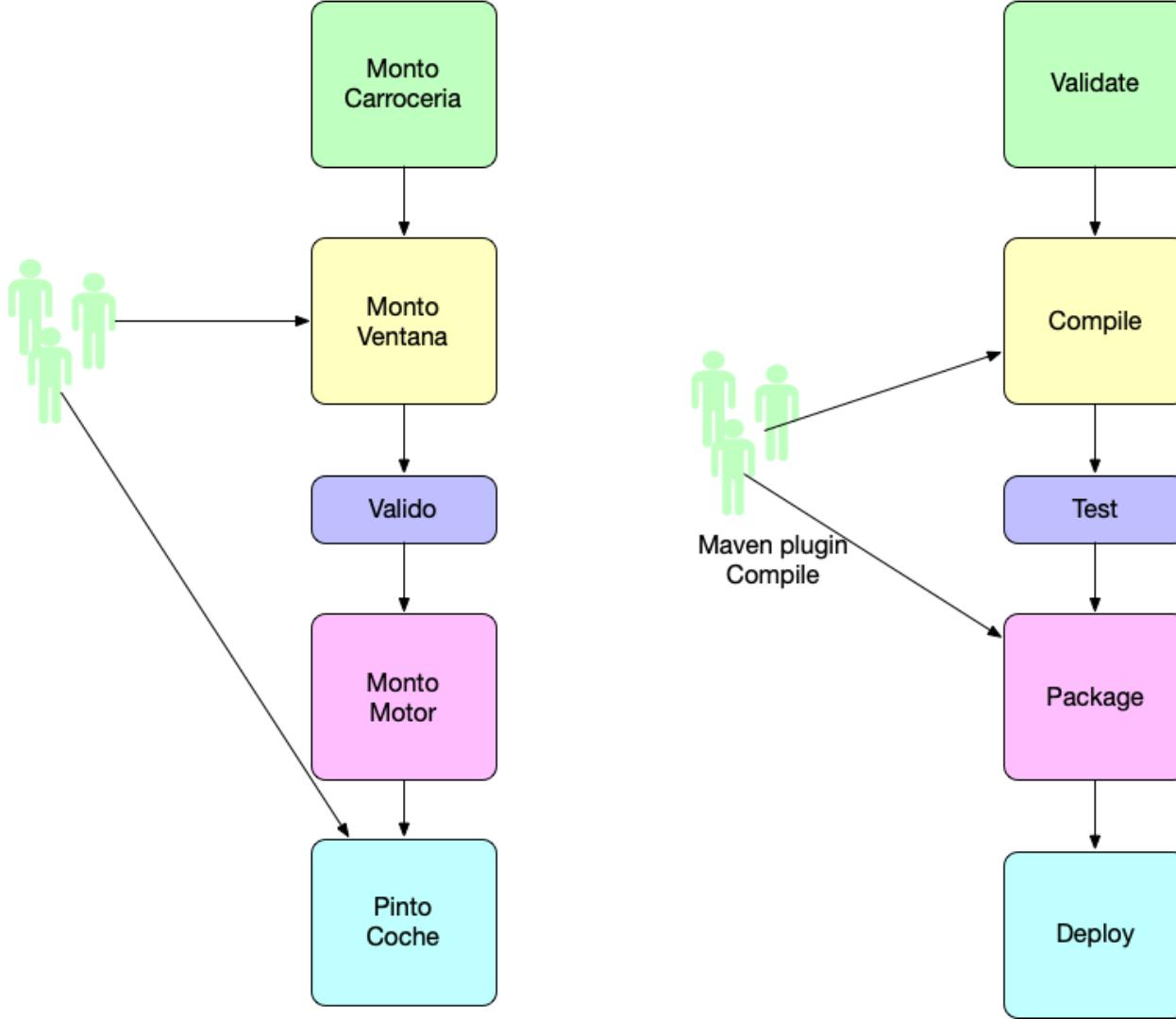
Almacen

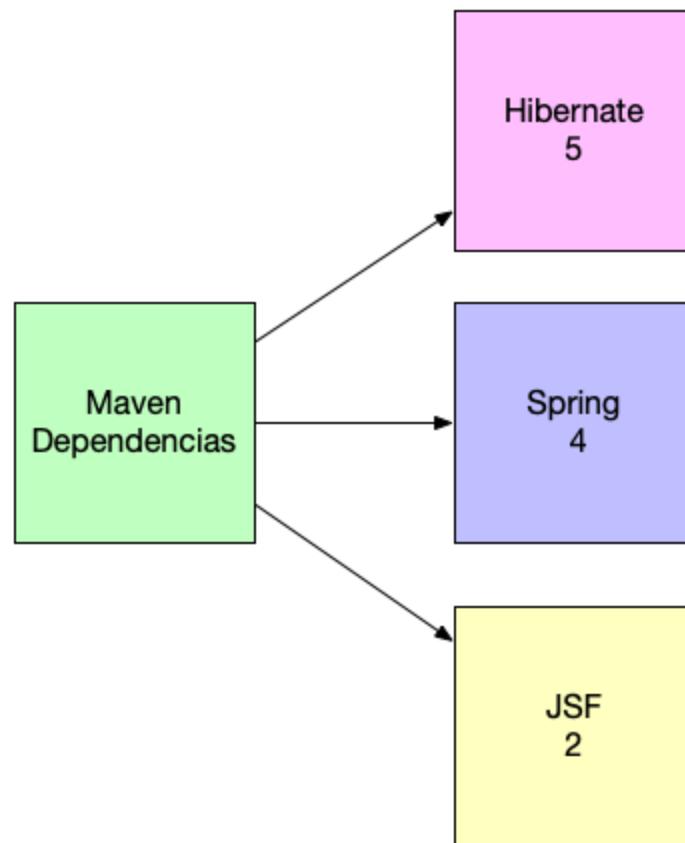
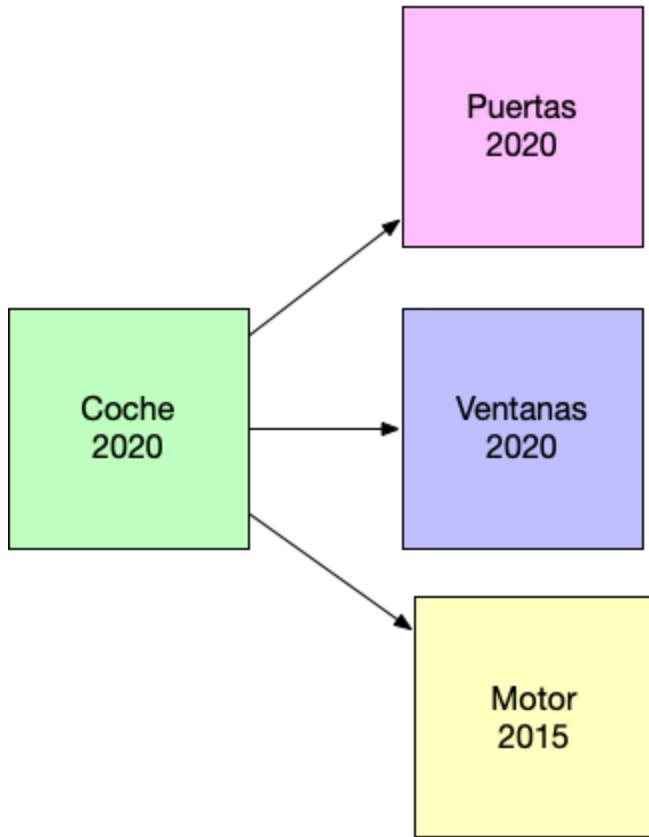


Carpetas









• **Introducción
al Acceso a
Datos con
Java**



➤ BBDD





Características

- Independencia lógica y física de los datos.
- Redundancia mínima.
- Acceso concurrente por parte de múltiples usuarios/as.
- Integridad de los datos.
- Consultas complejas optimizadas.
- Seguridad de acceso y auditoría.
- Respaldo y recuperación.
- Acceso a través de lenguajes de programación estándar.



DDL

- DDL comprende sentencias para la creación (CREATE), modificación (ALTER) y borrado (DROP) de los componentes principales de una base de datos:
- Base de datos (DATABASE).
- Tablas (TABLE).
- Vistas (VIEW).
- Índices (INDEX).
- Procedimientos almacenados (PROCEDURE).
- Disparadores (TRIGGER)

- DML comprende sentencias destinadas:
- A consultar (SELECT).
- A insertar (INSERT).
- A modificar (UPDATE).
- A borrar (DELETE).

- DCL está integrado por sentencias SQL destinadas a controlar las funciones de administración:
- Confirmar la operación (COMMIT).
- Retroceder en la operación (ROLLBACK).
- Proporcionar permisos (GRANT).
- Retirar permisos (REVOKE).

CREATE

- . CREATE DATABASE mibasededatos;
- . USE mibasededatos;
- . CREATE TABLE mitabla1
(Columna1 (TipoDeDato), Columna2
(TipoDeDato)...);

➤ SELECT

- `SELECT * FROM nombretabla;`
- `SELECT columna1, columna2 FROM nombretabla;`

WHERE

- La cláusula WHERE se utiliza para aplicar filtros a las consultas, es decir, para seleccionar únicamente aquellas filas de la tabla que cumplan una determinada condición.
- El valor de la condición debe situarse entre comillas simples (").
- Así, por ejemplo, aplicaremos un filtro para seleccionar aquellas personas cuyo nombre sea ANTONIO:
- `SELECT * FROM personas WHERE nombre = 'ANTONIO';`



INSERT

- . **INSERT INTO** nombretabla
- . **VALUES** (valor1, valor2, valor3...);
- . **INSERT INTO** nombretabla
(columna1, columna2, columna3...)
VALUES (valor1, valor2, valor3...);

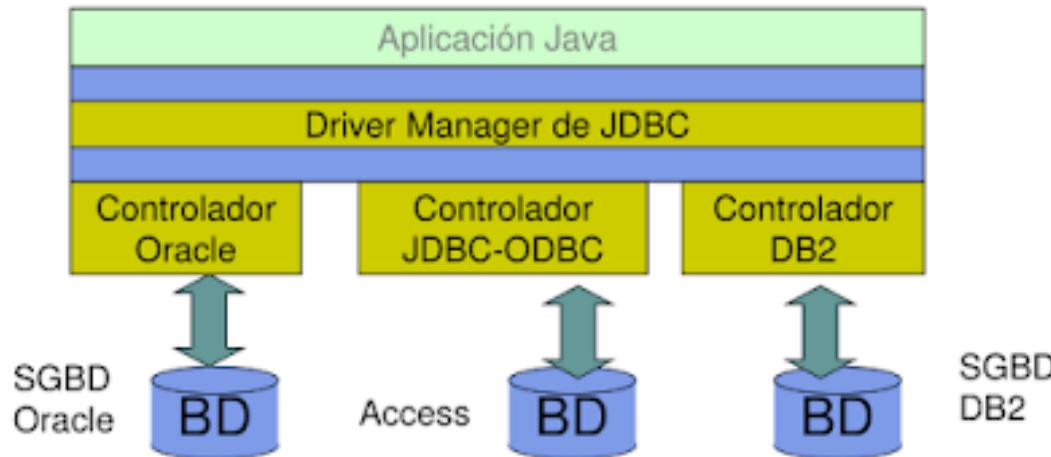
➤ UPDATE

- La sentencia UPDATE se utiliza para modificar valores en una tabla. Su sintaxis es la siguiente:
- UPDATE nombretabla
- SET columna1 = valor1, columna 2 = valor2
- WHERE condición;
- La cláusula SET establece los nuevos valores para las columnas indicadas, mientras que la cláusula WHERE selecciona las filas que queremos modificar.
- Atención: si omitimos la cláusula WHERE, se modificarán los valores en todas las filas de la tabla por defecto.

➤ DELETE

- La sentencia DELETE está destinada a borrar filas de una tabla. Su sintaxis es la siguiente:
- `DELETE FROM nombretabla`
- `WHERE nombrecolumna = valor;`
- Si queremos borrar todos los registros o filas de una tabla, utilizaremos la sentencia:
- `DELETE * FROM nombre tabla;`

➤ Arquitectura JDBC



➤ JDBC en MySQL

- Hay que añadir la librería mysql-connector-java-x.x.x-bin.jar a nuestro proyecto.
- Por ejemplo, si compilamos desde línea de comandos, añadimos el fichero a la variable del sistema CLASSPATH, y si usamos Eclipse, vamos a Project → Properties → Java Build Path → Libraries → Add External JARs...
- El driver es com.mysql.jdbc.Driver y la URL para conectar a la BD es
`jdbc:mysql://localhost:3306/simple`



Introducción a JDBC

- JDBC (Java DataBase Connectivity) es la API estándar de acceso a base de datos desde Java.
- Está incluida en Java SE (Standard Edition). En Java SE 6 se incluye JDBC 4.0, pero actualmente la mayoría de bases de datos soportan JDBC 3.0.
- Si se desea más información se puede visitar los siguientes enlaces
 - <http://java.sun.com/javase/technologies/database>
 - <http://java.sun.com/docs/books/tutorial/jdbc>

➤ ODBC (Open DataBase Connectivity)

- ODBC (Open DataBase Connectivity) es un estándar de acceso a base de datos desarrollado por Microsoft. Sun ha desarrollado un driver que hace de puente entre JDBC y ODBC aunque no suele usarse.
- Los pasos para que una aplicación se comunique con una base de datos son:
 - Cargar el driver necesario para comprender el protocolo que usa la base de datos concreta.
 - Establecer una conexión con la base de datos, normalmente a través de red.
 - Enviar consultas SQL y procesar el resultado
 - Liberar los recursos al terminar
 - Manejar los errores que se puedan producir

➤ Carga del driver

- Class.forName("com.mysql.jdbc.Driver");
- Antes de poder conectarse a la base de datos es necesario cargar el driver JDBC, y solo hay que hacerlo una única vez al comienzo de la aplicación.
- El nombre del driver debe venir especificado en la documentación de la base de datos.

➤ Establecer una conexión

- Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/sample", "root", "password");
- Las bases de datos actúan como servidores y las aplicaciones como clientes que se comunican a través de la red.
- Un objeto Connection representa una conexión física entre el cliente y el servidor. Para crear una conexión se usa la clase DriverManager, especificando la URL, el nombre y la contraseña de nuestra base de datos.

Ejecutar una sentencia SQL

- Statement stmt = conn.createStatement();
- ResultSet rs = stmt.executeQuery("SELECT
 titulo, precio FROM Libros WHERE precio > 2");
- Una vez que tienes una conexión puedes ejecutar sentencias SQL.
- Primero se crea el objeto Statement desde la conexión, y posteriormente se ejecuta la consulta y su resultado se devuelve como un ResultSet.

Acceso al conjunto de resultados

- ```
while(rs.next()){
 String name = rs.getString("titulo");
 float price = rs.getFloat("precio");
 System.out.println(name + "\t" + price);
}
```
- El ResultSet es el objeto que representa el resultado.
- No carga toda la información en memoria, internamente tiene un cursor en memoria que apunta a una fila concreta del resultado en la base de datos, por lo que hay que posicionar el cursor en cada fila y obtener toda la información de la misma.

## Result Set

| StudentId        | First_Name | Last_Name | GPA  |
|------------------|------------|-----------|------|
| BEFORE FIRST ROW |            |           |      |
| 1                | Jim        | Tackett   | 2.3  |
| 2                | J.D.       | Poe       | 2.29 |
| 3                | Angela     | Kincaid   | 2.5  |
| 4                | Aaron      | Shoopman  | 3.4  |
| 5                | Donna      | Brown     | 3.53 |
| 6                | Michael    | Hamby     | 3.22 |
| 7                | Chris      | Roden     | 3.01 |
| AFTER LAST ROW   |            |           |      |

cursor

ResultSet.next()

The diagram illustrates the state of a Result Set after executing `ResultSet.next()`. The cursor is positioned at the second row, which contains the data for J.D. The previous row (Jim) is shaded gray, and the current row (J.D.) is also shaded gray, indicating it has been fetched. The rows before the first row and after the last row are labeled "BEFORE FIRST ROW" and "AFTER LAST ROW" respectively. Arrows show the movement of the cursor from the first row to the second row.

# Liberar recursos

- rs.close();
- stmt.close();
- conn.close();
- Cuando se termina de usar una Connection, un Statement o un ResultSet es necesario liberar los recursos que necesitan.
- Puesto que la información de un ResultSet no se carga en memoria, existen conexiones de red abiertas.

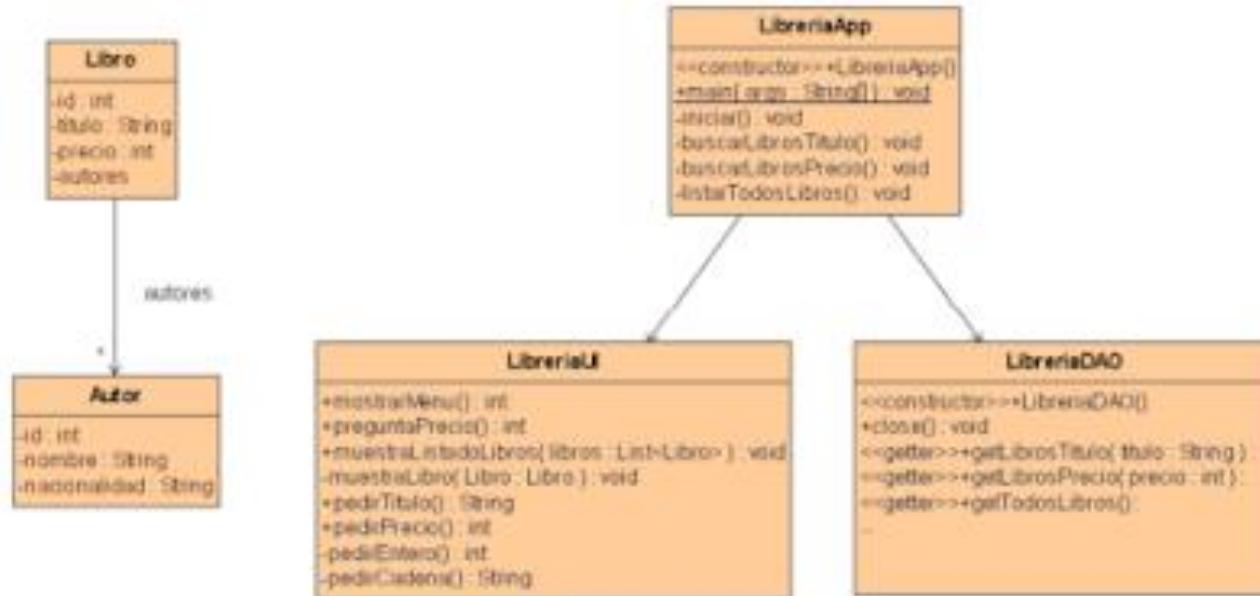
# ➤ Métodos close():

- `ResultSet.close()` – Libera los recursos del `ResultSet`. Se cierran automáticamente al cerrar el `Statement` que lo creó o al reejecutar el `Statement`.
- `Statement.close()` – Libera los recursos del `Statement`.
- `Connection.close()` – Finaliza la conexión con la base de datos.

# ➤ Manejar los errores

- throws ClassNotFoundException, SQLException
- Hay que gestionar los errores apropiadamente, ya que se pueden producir excepciones ClassNotFoundException si no se encuentra el driver, o excepciones SQLException al interactuar con la base de datos. Por ejemplo:
  - SQL mal formado
  - Conexión de red rota
  - Problemas de integridad al insertar datos (claves duplicadas)

# Diseño de una aplicación con BD



# Conexiones a una base de datos

- Cada objeto Connection representa una conexión física con la base de datos.
- Se pueden especificar más propiedades además del usuario y la contraseña al crear una conexión.
- Estas propiedades se pueden especificar:
- Codificadas en la URL (ver detalles de la base de datos)
- Usando métodos getConnection(...) sobrecargados de la clase DriverManager.

# Ejemplo 1

```
String url =
"jdbc:mysql://localhost:3306/sample";
String name = "root";
String password = "pass";
Connection c =
DriverManager.getConnection(url,
user, password);
```

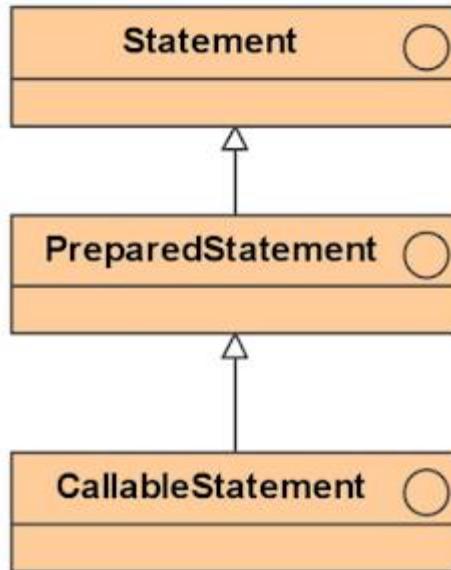
## Ejemplo 2

```
. String url =
"jdbc:mysql://localhost:3306/sam
ple?user=root&password=pass";
. Connection c =
DriverManager.getConnection(url
);
```

# Ejemplo 3

```
. String url =
"jdbc:mysql://localhost:3306/sample";
. Properties prop = new Properties();
. prop.setProperty("user", "root");
. prop.setProperty("password", "pass");
. Connection c =
DriverManager.getConnection(url, prop);
```

# ➤ Sentencias SQL



# Uso de Statement

- Tiene diferentes métodos para ejecutar una sentencia.
- `executeQuery(...)`. Se usa para sentencias SELECT. Devuelve un ResultSet
- `executeUpdate(...)`. Se usa para sentencias INSERT, UPDATE, DELETE o sentencias DDL. Devuelve el número de filas afectadas por la sentencia.
- `execute(...)`. Método genérico de ejecución de consultas. Puede devolver uno o más ResultSet y uno o más contadores de filas afectadas.

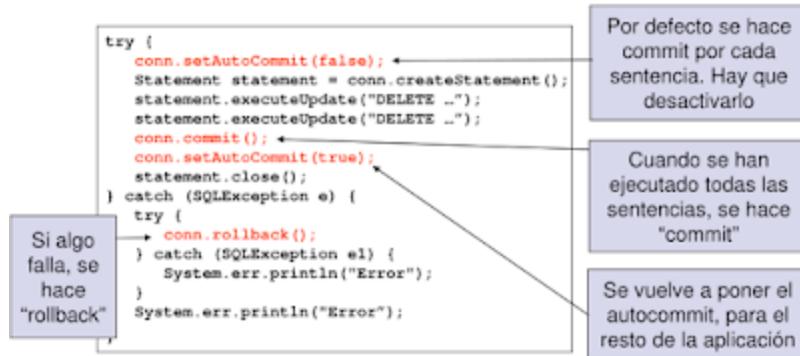
# Uso de PreparedStatement

```
PreparedStatement ps =
conn.prepareStatement("INSERT
INTO Libros VALUES (?, ?, ?)");
ps.setInt(1, 23);
ps.setString(2, "Bambi");
ps.setInt(3, 45);
ps.executeUpdate();
```

# Uso de CallableStatement

- . CallableStatement cstmt = conn.prepareCall("{call getEmpName (?, ?)}");
- . cstmt.setInt(1, 111111111);
- . cstmt.registerOutParameter(2, java.sql.Types.VARCHAR);
- . cstmt.execute();
- . String empName = cstmt.getString(2);

# Transacciones



# ➤ **resultSetType**

- `createStatement(int resultSetType, int resultSetConcurrency);`
- `prepareStatement(String SQL, int resultSetType, int resultSetConcurrency);`
- `prepareCall(String sql, int resultSetType, int resultSetConcurrency);`

# Actualización de datos

- rs.updateString("campo", "valor");
- rs.updateInt(1, 3);
- rs.updateRow();

# Inserción de datos

- rs.moveToInsertRow();
- rs.updateString(1, "AINSWORTH");
- rs.updateInt(2, 35);
- rs.updateBoolean(3, true);
- rs.insertRow();
- rs.moveToCurrentRow();

# ➤ Posicionamiento del cursor

Result Set

| Result Set       |            |           |      |
|------------------|------------|-----------|------|
| StudentId        |            |           |      |
|                  | First_Name | Last_Name | GPA  |
| BEFORE FIRST ROW |            |           |      |
| 1                | Jim        | Tackett   | 2.3  |
| 2                | J.D.       | Poe       | 2.29 |
| 3                | Angela     | Kincaid   | 2.5  |
| 4                | Aaron      | Shoopman  | 3.4  |
| 5                | Donna      | Brown     | 3.53 |
| 6                | Michael    | Hamby     | 3.22 |
| 7                | Chris      | Roden     | 3.01 |
| AFTER LAST ROW   |            |           |      |

cursor

ResultSet.next()

The diagram illustrates the state of a cursor on a result set. The cursor is positioned at row 2 (J.D.). A double-headed arrow labeled "ResultSet.next()" indicates the current position of the cursor. The result set contains 7 rows of student information: StudentId, First\_Name, Last\_Name, and GPA. The rows are numbered 1 through 7. The first row is "BEFORE FIRST ROW" and the last row is "AFTER LAST ROW". The rows alternate in shading between white and light gray.

# Métodos

- Los métodos que permiten un movimiento por el ResultSet son:
  - next(). Siguiente fila
  - previous(). Fila anterior
  - beforeFirst(). Antes de la primera
  - afterLast(). Después de la última.
  - first(). Primera fila
  - last(). Última fila
  - absolute(). Movimiento a una fila concreta
  - relative(). Saltar ciertas filas hacia delante

# EJERCICIO



## STRUCTURED DATA



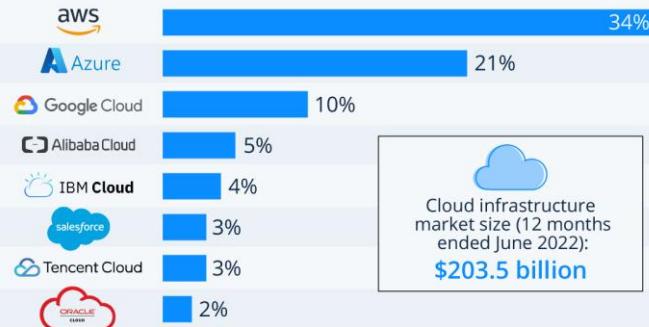
## UNSTRUCTURED DATA





## Amazon Leads \$200-Billion Cloud Market

Worldwide market share of leading cloud infrastructure service providers in Q2 2022\*



Cloud infrastructure  
market size (12 months  
ended June 2022):  
**\$203.5 billion**

\* includes platform as a service (PaaS) and infrastructure as a service (IaaS)  
as well as hosted private cloud services

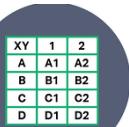
Source: Synergy Research Group



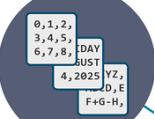
statista



Can be displayed  
in rows, columns and  
relational databases



Numbers, dates  
and strings



Estimated 20% of  
enterprise data (Gartner)



Requires less storage

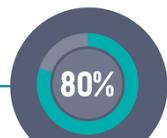


Cannot be displayed  
in rows, columns and  
relational databases

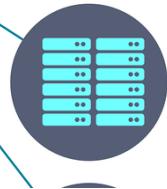


Images, audio, video,  
word processing files,  
e-mails, spreadsheets

Estimated 80% of  
enterprise data (Gartner)



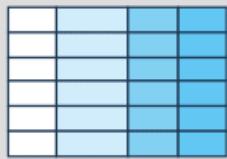
Requires more storage



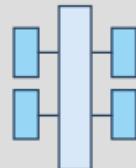


## SQL

### Relational

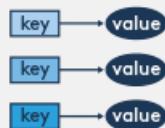


### Analytical (OLAP)

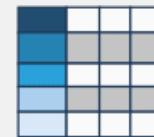


## NoSQL

### Key-Value



### Column-Family

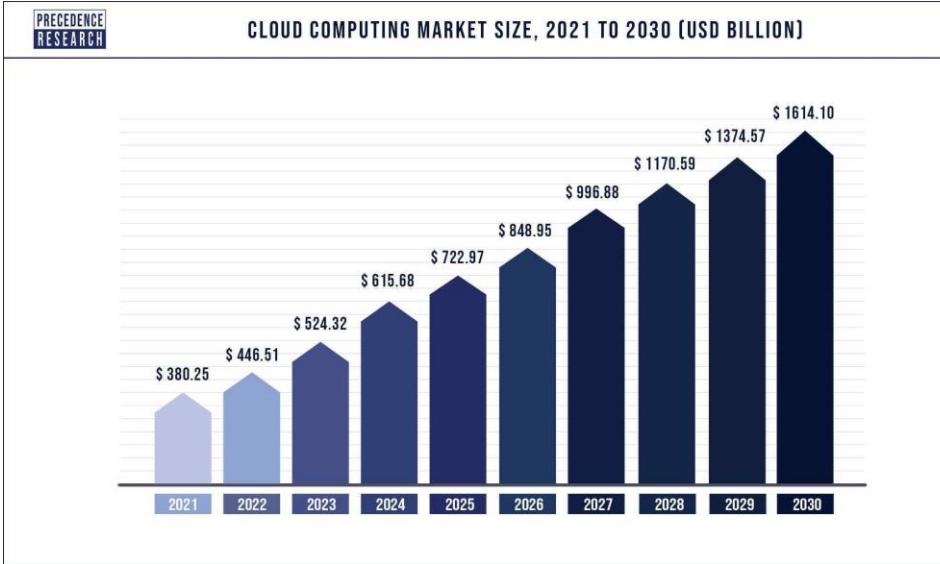


### Graph



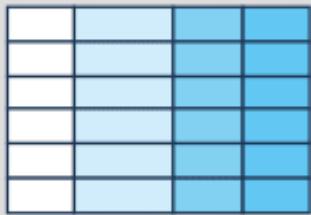
### Document



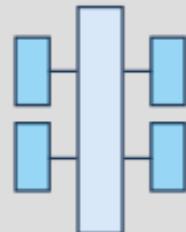


# SQL

## Relational

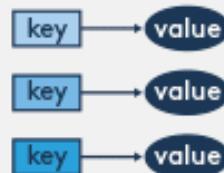


## Analytical (OLAP)

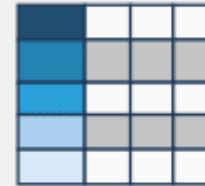


# NoSQL

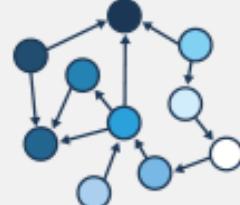
## Key-Value



## Column-Family



## Graph

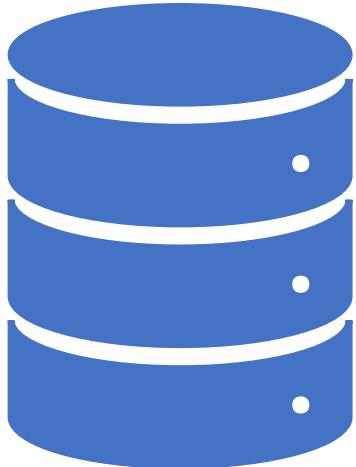


## Document



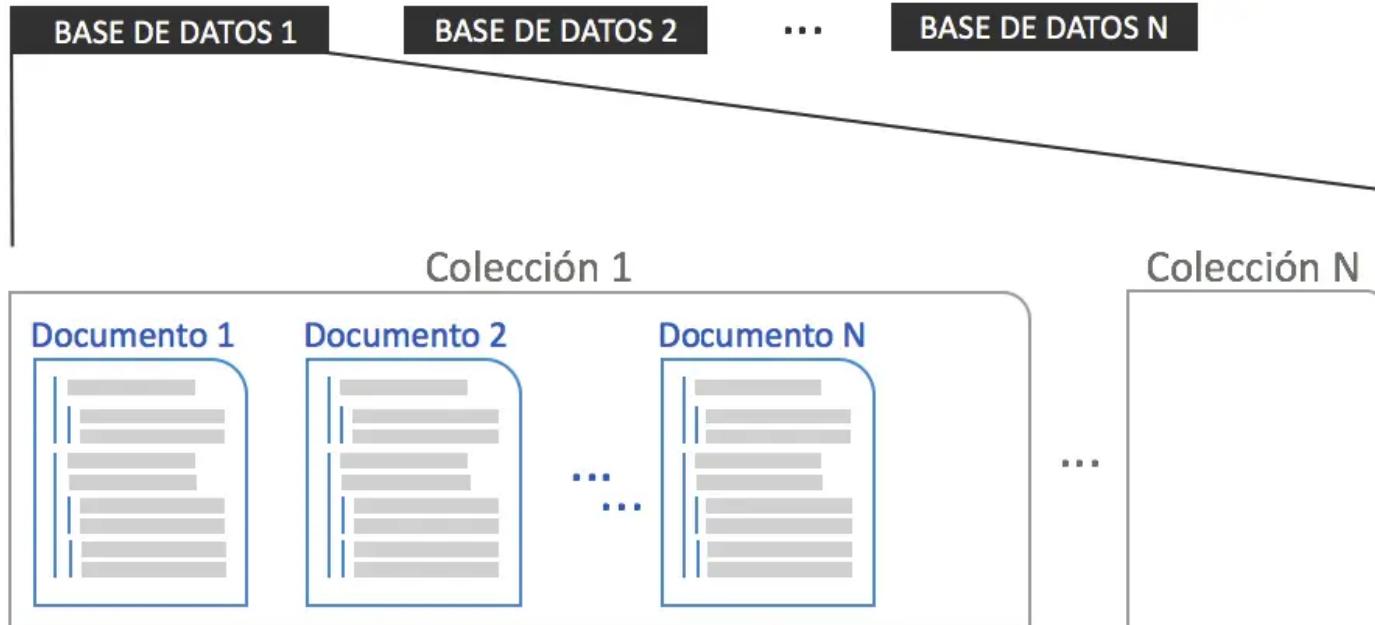


# MONGODB



- El gestor de base de datos MongoDB se lo puede asociar a un conjunto de gestores de bases de datos que no tienen como lenguaje principal el SQL para su manipulación.
- Los gestores de bases de datos NoSQL no requieren estructuras fijas como tablas, normalmente no soportan operaciones join y presentan como gran ventaja que pueden escalar en forma sencilla.

# BASE DE DATOS DOCUMENTAL





## MOTORES NOSQL



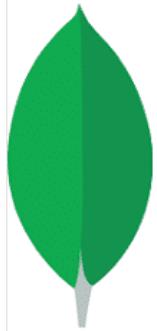
# CARACTERÍSTICAS MONGODB

- Indexación
- Replicación
- Balanceo de carga
- Almacenamiento de archivos
- Agregación

A photograph of a stack of books on a light-colored wooden surface. The top book has a yellow cover and a white spine. The background is blurred.

# JSON

```
{
 codigo: 1,
 nombre: 'El aleph',
 autor: 'Borges',
 editoriales: ['Planeta','Siglo
XXI']
}
```



**mongoDB**<sup>®</sup> Atlas



# **CREAR CUENTA DE MONGODB ATLAS Y LANZAR PRIMER CLUSTER**

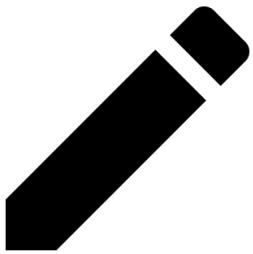
# MONGODB COMPASS

The screenshot shows the MongoDB Compass interface for the `flightStats-cut` database. The left sidebar lists 8 databases and 15 collections. The main panel displays the following information:

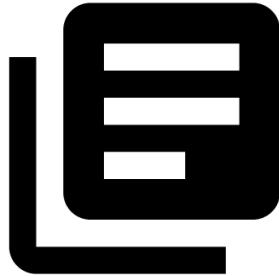
- Flight Statistics Cut**: A report for 9,993 documents based on a sample of 100 documents (1.00%).
- Documents**: 10.0k total size 6.5 MB avg. size 684 B.
- Indexes**: 1 total size 566.9 KB avg. size 566.9 KB.
- Sampled Documents**: A list of 10 flight identifiers: EWR-EV-4467-542273341, EWR-EV-4382-544626514, LGA-9E-3457-542758157, JFK-9E-4093-544640472, LGA-ZW-3910-545111616, JFK-SE-41-544640167, EWR-YX-4904-544626872, LGA-ZW-3815-544184788.
- arrivalAirportF**: A histogram showing the distribution of arrival airports. The x-axis is labeled "String (100%)" and the y-axis ranges from 0% to 7%. The distribution is highly skewed, with the top few categories accounting for most of the data.
- carrierFsCode**: A histogram showing the distribution of carrier codes. The x-axis is labeled "string" and the y-axis ranges from 0% to 19%. The distribution is highly skewed, with the top few categories accounting for most of the data.
- codeshares**: A section showing an array of documents with 3 nested fields. It includes a histogram of array lengths (min: 1, average: 2.73, max: 11) and a table of sample documents.



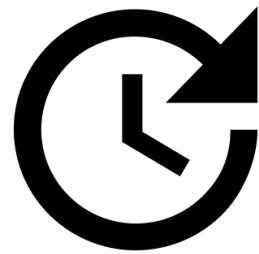
# **RECORRIDO POR MONGODB COMPASS Y CONECTARNOS A NUESTRO CLUSTER**



C  
reate



R  
ead



U  
pdate



D  
elete

A photograph showing a stack of several books. On top of the stack, a pair of round-rimmed glasses lies horizontally. The background is dark and out of focus.

# INSERTAR DOCUMENTOS

Para insertar un documento o un conjunto de documentos disponemos de los métodos:

- `insertOne`: Inserta un documento en una colección.
- `insertMany`: Inserta múltiples documentos en una colección.



## CAMPO OBLIGATORIO \_ID

- En MongoDB, cada documento almacenado en una colección requiere un único \_id que actúa como clave principal. Si se inserta documento omite el \_id, el controlador MongoDB automáticamente genera un ObjectId para el \_id.

# Dependencias

- Comenzaremos creando un proyecto Maven incluyendo la depedencia siguiente en el pom.xml. En el momento de crear este documento (Mayo 2015) la última versión del driver es la 3.0.1.
- <dependencies>
- <dependency>
- <groupId>org.mongodb</groupId>
- <artifactId>mongo-java-driver</artifactId>
- <version>3.0.1</version>
- </dependency>
- </dependencies>

# Conexión a MongoDB

- La forma básica de obtener una conexión a MongoDB es instanciando directamente la clase MongoClient. Si no indicamos nada, nos conectaremos al host y puerto por defecto.
- ```
MongoClient mongoClient = new  
MongoClient();
```

Conexión a MongoDB

- El host y el puerto se pueden pasar como parámetro al constructor de MongoClient.
 - La base de datos será seleccionada a través del método getDatabase() de la clase MongoClient.
-
- ```
public MongoDatabase connect(String host, int port, String database) {
 // Getting a connection
 mongoClient = new MongoClient(host, port);
 // Select the database
 db = mongoClient.getDatabase(database);
 return db;
}
```

# Conexión a MongoDB

- También podemos crear la conexión usando un objeto ServerAddress al que pasaremos el host y el puerto.
- ```
public MongoDBDatabase connect(String host, int port, String database) {  
    // Create serverAddress, the location a MongoDB server  
    ServerAddress serverAddress = new ServerAddress(host, port);  
    // Connect to the MongoDB server with internal connection pooling  
    mongoClient = new MongoClient(serverAddress);  
    // Getting a connection easily avoiding ServerAddress  
    // mongoClient = new MongoClient(host, port);  
    // Select the database  
    db = mongoClient.getDatabase(database);  
    return db;  
}
```

Desconexión a MongoDB

- La desconexión de MongoDB se hace mediante el método `close()` de la clase `MongoClient`.
- ```
public void disconnect() {
 mongoClient.close();
}
```

# ➤ Selección de la colección

- La clase MongoDatabase nos ofrece el método getDatabase() que nos permite seleccionar la base de datos con la que queremos trabajar. La clase MongoDatabase nos ofrece otros métodos interesantes aplicables a la base de datos (p.e. para cuestiones relacionadas con write concern)
- A continuación vemos cómo crear un objeto para una base de datos de ejemplo denominada ggvdTest.
- `MongoCollection<Document> collection = db.getCollection("ggvdTest");`

# Creación de documentos

- En la interacción con MongoDB usando el driver de Java se suele utilizar para casi todo la clase Document. Utilizaremos esta clase no sólo para los documentos que tengamos que introducir, sino para definir los criterios de eliminación, actualización, ordenación, y demás.
- El constructor de la clase Document admite dos parámetros: una clave y un valor. Para crear documentos con más de pareja clave-valor aplicaremos el método append() al constructor tantas veces como sea necesario encadenando las llamadas con un punto (.). Esta será la técnica básica para construir las listas JSON.



# Inserción de documentos

- Los documentos se insertarán con los métodos `insertOne()` o `insertMany()` dependiendo de si queremos insertar uno o varios documentos simultáneamente. Ambos métodos se aplican sobre la colección correspondiente.
- A continuación se muestran un par de métodos para insertar documentos con `insertOne()`. El primer método inserta un documento sencillo. El segundo método inserta un documento complejo con un array en una clave y un documento anidado en otra.

# Inserción de documentos

- El código siguiente ilustra un ejemplo para insertar una lista de documentos con `insertMany()`.



# Busqueda de documentos

- La búsqueda de documentos se realiza mediante el método `find()` aplicado a una colección. El método `find()` devuelve un objeto de la clase `FindIterable`. Sobre un objeto `FindIterable` podemos usar el método `first()`, que como su nombre indica nos devuelve el primer elemento de un objeto `FindIterable`.
- A continuación se muestra un ejemplo combinado con el método `toJson()` que muestra el resultado de una consulta en forma de documento en JSON.
-

- Si en lugar de recuperar el primer documento queremos recorrer el conjunto de resultados (p.e. para mostrarlos), tenemos que convertir el objeto FindIterable que devuelve el método `find()` en un cursor. Para ello usaremos el método `iterator()`. El método `iterator()` devuelve un objeto `MongoCursor` que podremos recorrer con el método `next()`.
- Una vez sobre el documento, podremos aplicar métodos `getString()`, `getInteger()` y demás, similares a los usados con JDBC.

# Ordenación, Proyección, Skip y Limit

- Además de first(), la clase FindIterable nos proporciona métodos interesantes, como son los métodos para la ordenación (sort()), proyección (projection()), salto (skip()) y limitación (limit()).
- sort() toma como un documento como argumento indicando los campos por los que se quiere ordenar y el criterio de ordenación: ascendente (1) o descendente (-1), de la misma forma que en la shell de MongoDB.
- projection() toma como un documento como argumento indicando los campos que se quieren mostrar (1) u ocultar (0), de la misma forma que en la shell de MongoDB.
- skip() y limit() toman un entero como argumento que representa el número de documentos a saltar y limitar, respectivamente.
- ATENCION: sort(), projection(), skip() y limit() devuelven un objeto FindIterable. Para poder recorrerlo hay que convertirlo en un cursor aplicando el método iterator().

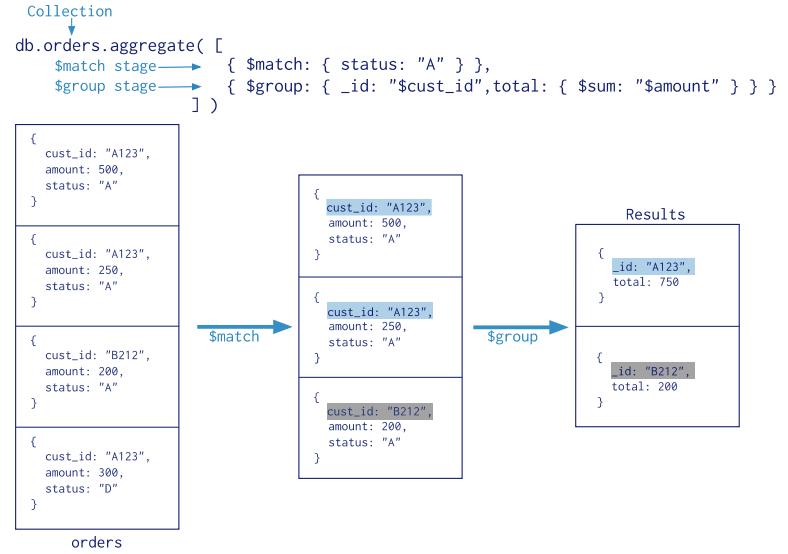
# ➤ Eliminación de documentos

- Para eliminar documentos con el driver de MongoDB podemos usar varios métodos (`deleteOne()`, `deleteMany()` y `findOneAndDelete()`). `findAndDelete()` permite realizar de forma atómica la búsqueda de los documentos a eliminar y su posterior eliminación. A este método le pasaremos el documento con los criterios, y de forma optativa las opciones de eliminación.
-

# Actualización de documentos

- Para actualizar documentos con el driver de MongoDB podemos usar varios métodos (`updateOne()`, `updateMany()` y `findOneAndUpdate()`), que como se puede comprobar guardan un gran parecido con sus homólogos para eliminar documentos. `findAndUpdate()` permite realizar de forma atómica la búsqueda de los documentos a eliminar y su posterior eliminación. A este método le pasaremos el documento con los criterios, la actualización y de forma optativa las opciones de actualización.

# AGGREGATION PIPELINE



# Framework de Agregación

- Las operaciones relacionadas con el framework de agregación se realizan con el método `aggregate()`. A este método le proporcionaremos el pipeline de operaciones (`$match`, `$project`, `$group`, `$sort`, ...) en forma de lista. El método devolverá un objeto `AggregationIterable` que se podrá recorrer como un cursor.

- TDD con  
**JUnit 5 y  
Mockito**





# INTRODUCCIÓN A PRUEBAS



vs.



**Pruebas  
dinámicas**

**Pruebas  
estáticas**

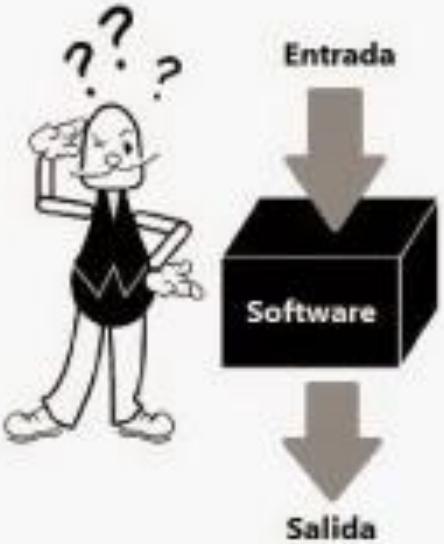
# TIPOS DE SOFTWARE TESTING

## Pruebas funcionales

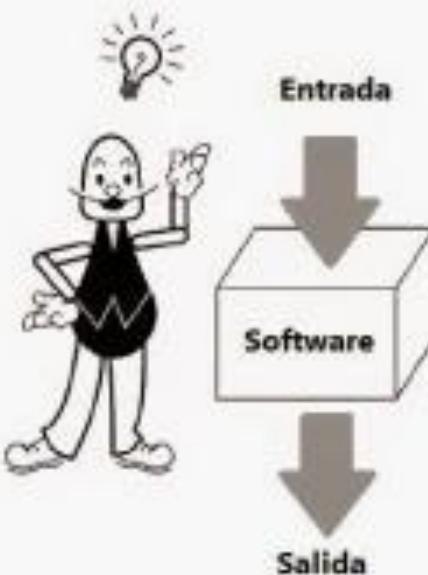
- Pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de sanidad
- Pruebas de humo
- Pruebas de interfaz
- Pruebas de regresión
- Pruebas de aceptación

## Pruebas no funcionales

- Pruebas de rendimiento
- Prueba de carga
- Pruebas de estrés
- Pruebas de volumen
- Pruebas de seguridad
- Pruebas de compatibilidad
- Pruebas de instalación
- Pruebas de recuperación
- Pruebas de confiabilidad
- Pruebas de usabilidad
- Pruebas de conformidad
- Pruebas de localización



Test de Caja Negra



Test de Caja Blanca

# Estrategia de Pruebas

- Planificación: Definición de estrategias, alcance, recursos y calendario de pruebas.
- Diseño de Casos de Prueba: Creación de escenarios y casos de prueba basados en requisitos.
- Ejecución de Pruebas: Ejecución de casos de prueba y registro de resultados.
- Análisis de Resultados: Evaluación de los resultados de las pruebas y seguimiento de los defectos encontrados.
- Informe y Retest: Creación de informes de pruebas y reevaluación tras la corrección de defectos.



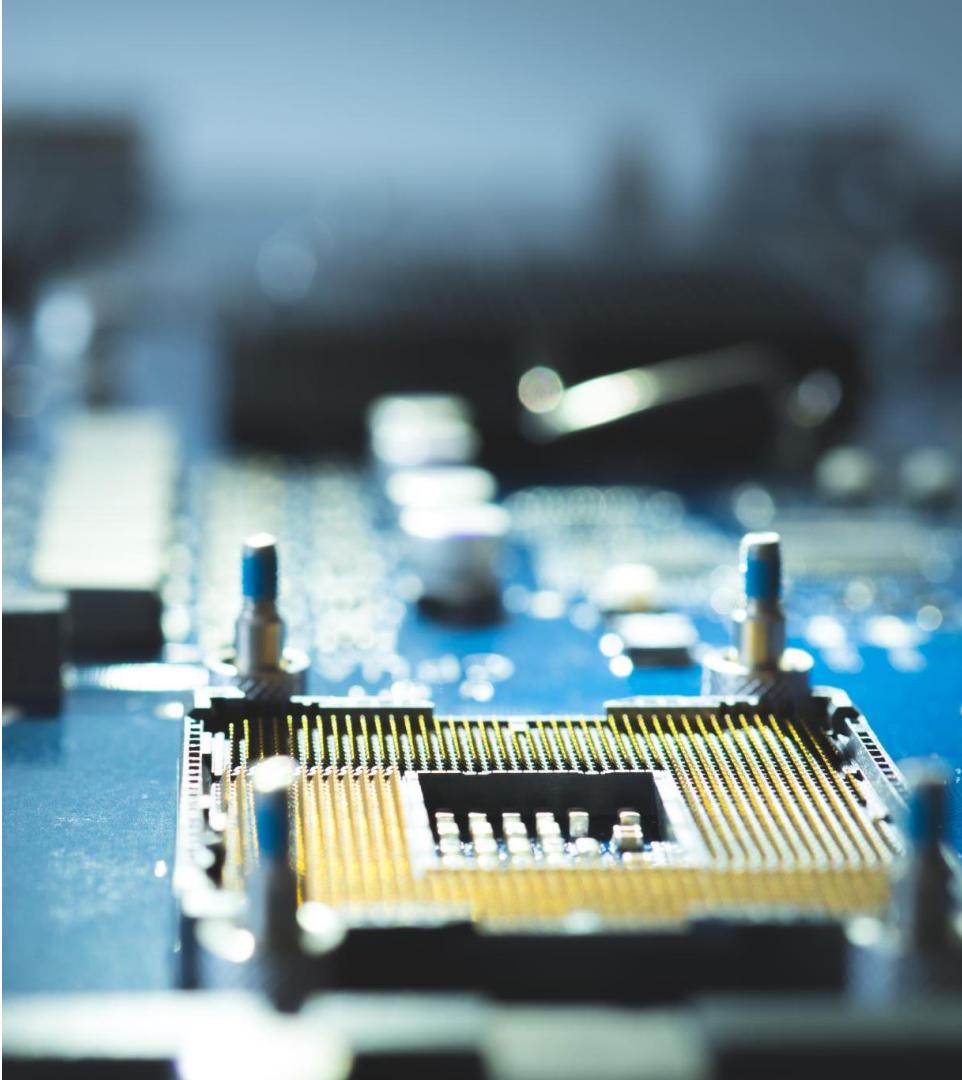
# Diseño de Pruebas

- Conocimiento de los Requisitos
- Reducción de la funcionalidad a probar
- Definición de Casos de Prueba
- Relacionar con los Casos de Usuario (historias)
- Pensar bien los costes de las pruebas
- Enfocarse en las funcionalidades más críticas
- Cada prueba requiere su herramienta



# HERRAMIENTAS

- Pruebas Unitarias: Junit
- Pruebas de Integración: Mockito
- Pruebas de Aceptación/Funcionalidad Web: Selenium
- Bases de Datos/Web Rendimiento: Jmeter/Blazemeter/Gatling
- Servidor de Integración Contínua: Jenkins/TravisCI/CircleCI
- Herramienta de Seguimiento de Proyectos e Incidencias: Jira/Mantis
- Automatizador de la Compilación, Despliegue y Ejecución de proyectos: Maven/Gradle/NPM
- Despliegue de aplicaciones: Docker/Kubernetes



# AUTOMATIZACIÓN DE LAS PRUEBAS





# EJEMPLO

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class MiClaseTest {

 @Test
 public void pruebaSuma() {
 int resultado = MiClase.suma(3, 5);
 assertEquals(8, resultado);
 }
}
```



# MENSAJES PERSONALIZADOS

```
assertEquals(2, calculadora.suma(1, 1), "La suma debería ser 2");
```

```
assertEquals(2, calculadora.suma(1, 1), () -> "La suma debería ser 2")
```

# LA ANATOMIA DE JUNIT

```
package examples.nbank;

public class Conversion {

 public double tempConversion (double temperature, String unit) {
 if (unit.equals("F"))
 return (temperature - 32) * (5.0/9.0);
 else
 return (temperature * (9.0/5.0)) + 32;
 }
}
```

```
package examples.nbank;

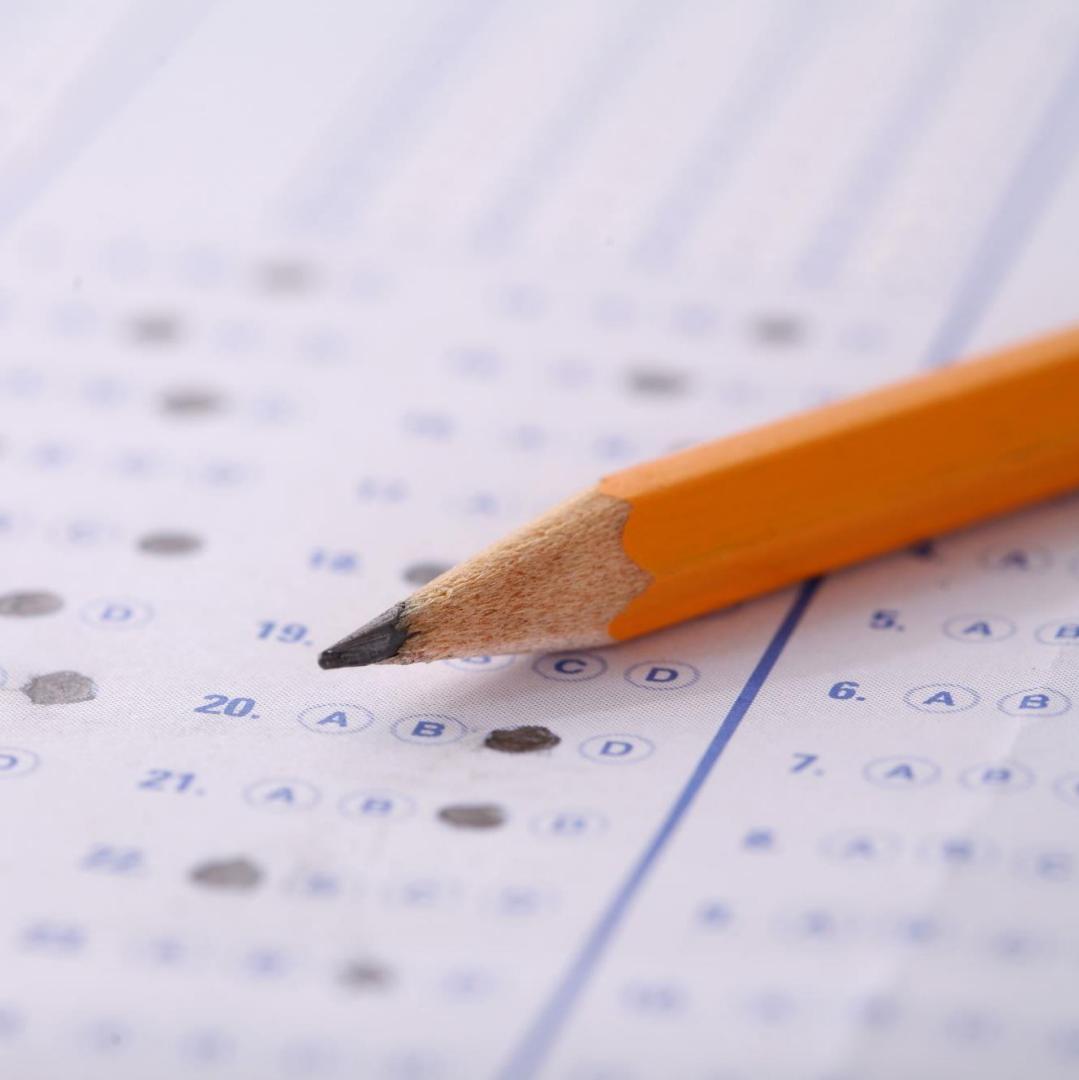
import static org.junit.Assert.assertEquals;
import org.junit.*;

public class ConversionTest {

 @Test
 public void testTempConversion() throws Throwable {
 // Given
 Conversion underTest = new Conversion();

 // When
 double temperature = 80.0d;
 String unit = "";
 double result = underTest.tempConversion(temperature, unit);

 // Then - assertions for result of method tempConversion(double, String)
 assertEquals(176.0d, result, 0.0);
 }
}
```



**TRABAJAN  
DO CON  
TEST**



# EXCEPCIONES CONTROLADAS

- JUnit nos permite comprobar que un método lanza una excepción controlada.
  - Deben extender de Throwable. Por ejemplo RuntimeException.
  - `assertThrows(Exception.class, () -> {});`
  - `assertThrows(Exception.class, () -> {}, message);`

## Assert all

- Difícil seguimiento de asserts cuando hay muchos en un test.
- Si un assert falla, no se ejecutan los siguientes y no sabemos su evaluación.
- assertAll ejecuta todos los assert independientemente del posible fallo de uno de ellos.
- Reporta todos los fallos. Dónde se han producido y por qué.



# Ejemplo de uso

- `assertAll(String message, () -> {});`
- Pueden incluir varias expresiones lambda.
- `assertAll("probando compras", () ->  
assertNotNull(store.getProducts()),  
() -> assertEquals(2,  
store.getProducts().size()), () ->  
assertTrue(new  
BigDecimal("300.00").compareTo(ac  
tualAmount) == 0));`





## Dar nombre a los tests

- JUnit 5 nombra a los tests con el nombre del método.
- Permite especificar un nombre personalizado anotando el método con `@DisplayName("Nombre del test")`;
  - `@Test`
  - `@DisplayName("Comprobación suma calculadora")`
  - `void sumaCalculadora() { ... }`



## CICLO DE VIDA

- Proceso por el cual se crea, se ejecuta y se destruye una instancia encargada de la realización de las pruebas
- Se encarga el motor de JUnit 5.
- Se crea una nueva instancia con cada test que se ejecuta.
- JUnit 5 permite ejecutar hooks en diferentes momentos del ciclo de vida.
- Hooks de JUnit 5: @BeforeAll / @AfterAll  
@BeforeEach/ @AfterEach

## @BeforeEach / @AfterEach

- Se ejecuta una vez que se crea una nueva instancia, es decir, cada vez que se ejecuta un test.
- **@BeforeEach** - Se ejecuta antes de la ejecución del test.
- **@AfterEach** - Se ejecuta después de la ejecución del test.

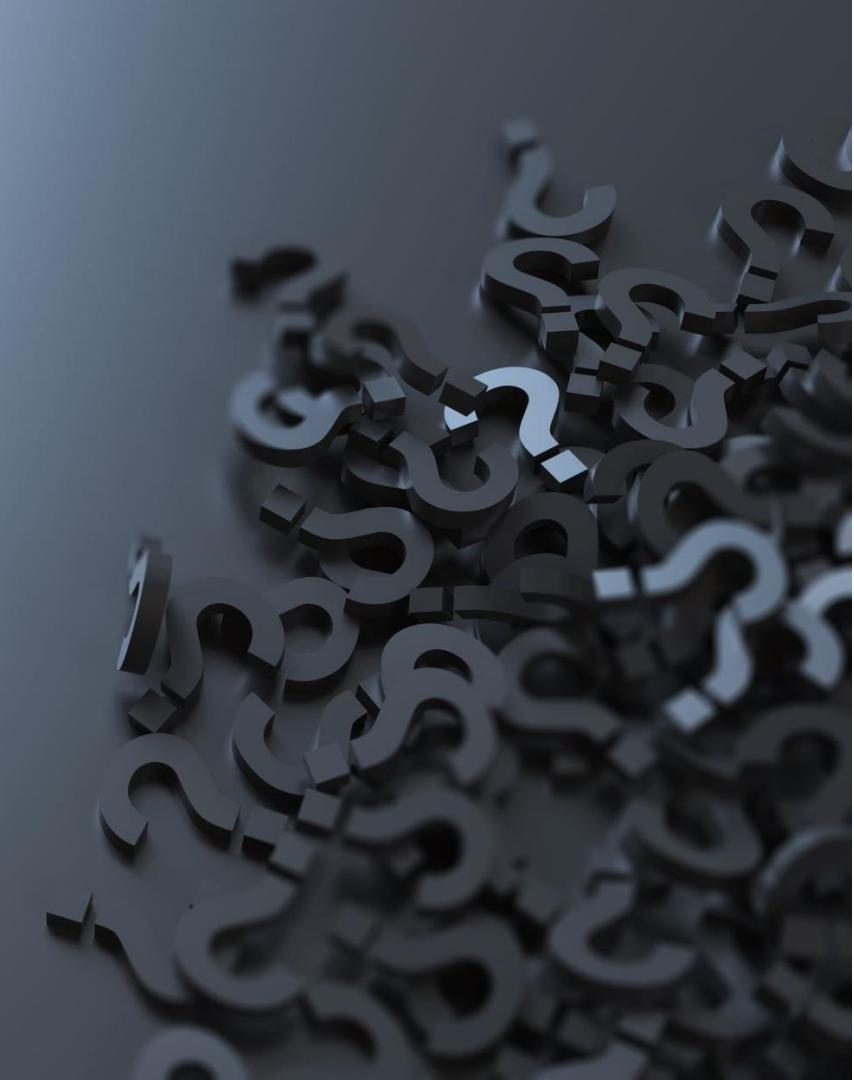
```
public class TiendaTest {
 private List<String> products;
 @BeforeEach
 void setup() {
 products = Arrays.asList("product1",
"product2");
 }

 @AfterEach
 void teardown() {
 products.clear();
 }
}
```



# @BeforeAll/@AfterAll

- Se ejecuta antes de crear/después de destruir la instancia, por lo que se implementa en un método estático.
- Si se anota en un método no estático, éste fallará, ya que la instancia no existe.
- Se puede forzar que la instancia sólo se cree una vez, aunque es mala práctica, pues compartes el estado de la clase entre tests.
- @Creando una instancia por clase, nos permite quitar el static a @BeforeAll/@AfterAll



# DESHABILITANDO TESTS UNITARIOS

- Los tests pueden deshabilitarse para evitar su ejecución.
- Anotamos el test con `@Disabled` ○ JUnit 5 recomienda especificar un motivo:
  - `@Disabled("Se deshabilita este test hasta que el bug @B54 se resuelva")`

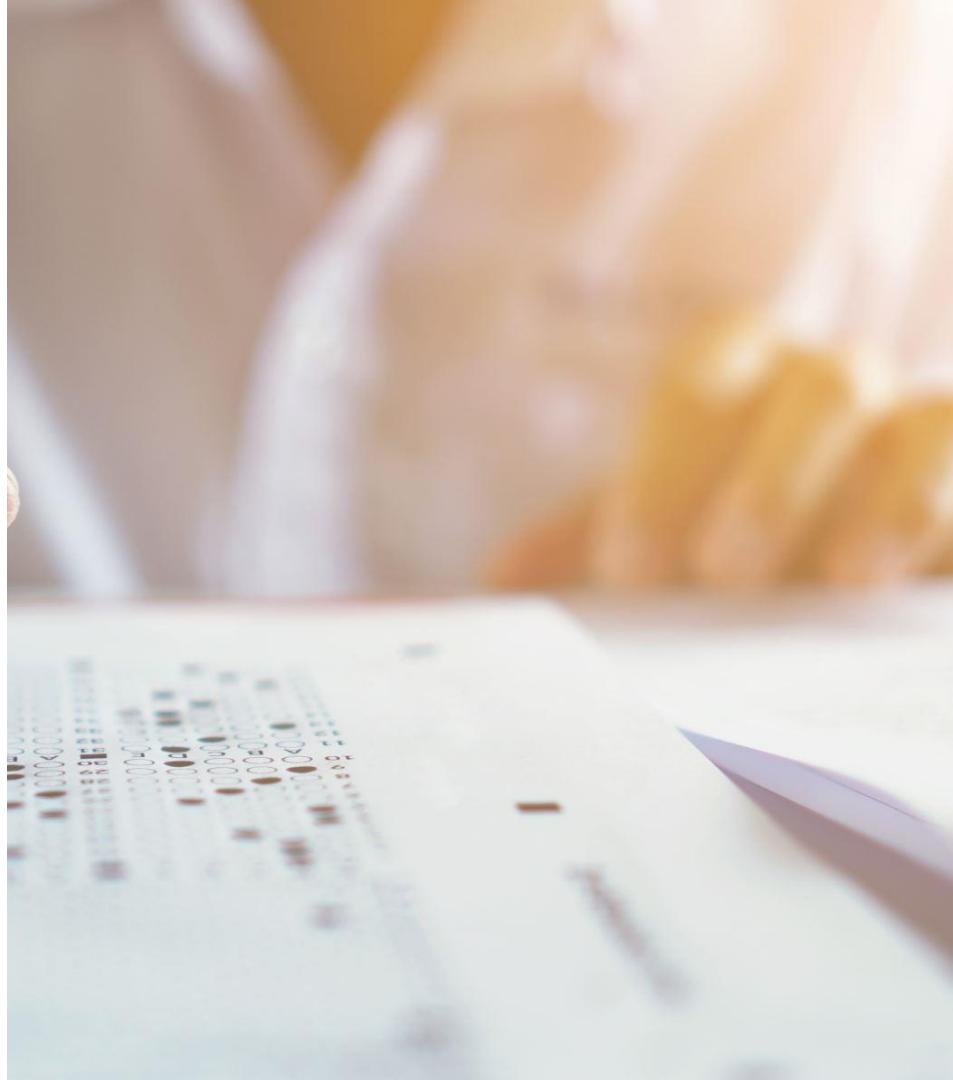
# ➤ TESTS CONDICIONALES - ANOTACIONES

- Las pruebas unitarias se pueden ejecutar conforme a diferentes condiciones.
- JUnit ofrece anotaciones para habilitar o no dichos tests:  
  @EnabledOnOS  
  @EnabledOnJre  
  @EnabledIfSystemProperty  
  @EnabledIfEnvironmentVariable

```
@EnabledIf("hasStock")

@Test
void it_should_decrease_stock() {
}

boolean hasStock(){
}
```



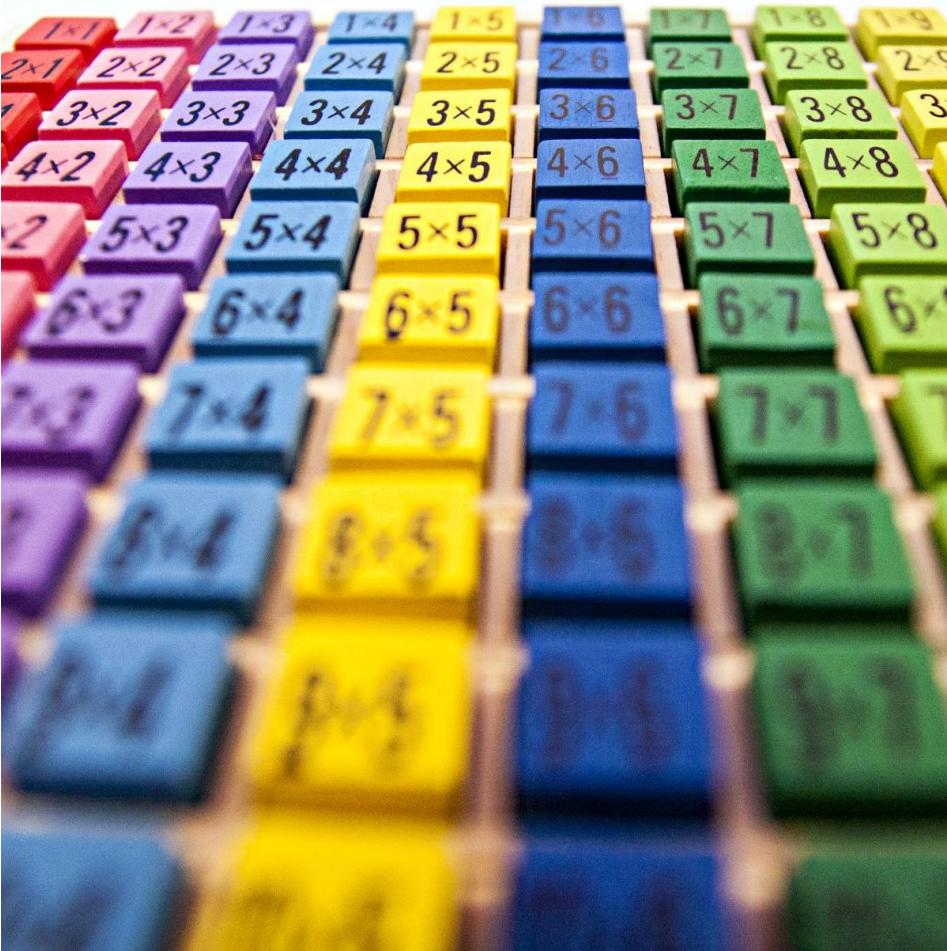
# TESTS CONDICIONALES

- Permiten habilitar parte de un test en función de si se cumple una condición. `assumeTrue(boolean condition); assumeFalse(boolean condition);`
- Si la assumption no se cumple, el código a partir de ahí se deshabilita, evitando el fallo
- JUnit 5 permite ejecutar o no parte del método encapsulándolo en una expresión lambda.
  - `assumingThat(boolean, () -> {});`
- La expresión lambda sólo se va a ejecutar si la expresión evalúa a true.
- El código fuera de la expresión lambda, sí se ejecutará.



# Clases anidadas

- Las clases anidadas permiten organizar los tests por diferentes criterios: funcionalidad, condicionalidad... Se anotan las clases con @Nested.
- Se puede incluir una descripción en dichas clases y los métodos que contiene con @DisplayName. Los tests aparecerán en el reporting agrupados por clases. Si falla un test de una clase @Nested, aparecerá como fallo el test y las clases contenedoras del mismo (hasta la clase raíz).



# Repetir tests

- JUnit 5 permite ejecutar varias veces el test. Útil en métodos que presentan cierta aleatoriedad.
  - Por ejemplo, crean valores random. Se anotan con `@RepeatedTest(int repetitions)`
- En el reporting, aparece la ejecución de las repeticiones. Es personalizable el nombre en cada ejecución del test. `@RepeatedTest(int, message)`
- Podemos usar variables para la creación de ese mensaje: `{currentRepetition}{totalRepetitions}`
- Podemos combinar el nombre con `@DisplayName`.  
`@DisplayName: Será el título principal.`  
`@RepeatedTest: Nombre en cada repetición.`  
`@DisplayName puede ser inyectado en el mensaje de @RepeatedTest:`
  - `{displayName}`



# TESTS PARAMETRIZADOS

- Otra forma de repetir tests en JUnit 5.
- Permite que en cada repetición se ejecute con datos diferentes. Se inyectan mediante variables en los métodos.
- JUnit 5 permite proporcionar dichos datos mediante:  
`@ValueSource(strings={})` Otros tipos:  
ints, doubles, booleans.  
`@CsvSource({índice, valor})`  
`@CsvFileSource(resources, delimiter,  
numLinesToSkip)`  
`@MethodSource(static methodName)`



# Filtrar tests

- Permite ejecutar los tests de forma selectiva.
- Útil para identificar un test con el id de una tarea. Posibilidad de ejecutar grupos de tests.
- Se anota el test o la clase con @Tag. Se puede anotar con varios @Tag.
- Se especifican en el RunConfiguration / Maven.

- 
2. A B C D E
  3. A B C D E
  4. A B C D E
  5. A B C D E
  6. A B C D E
  7. A B C D E
  8. A B C D E
  9. A B C D E
  10. A B C D E
  11. A B C D E
  12. A B C D E
  13. A B C D E
  14. A B C D E
  15. A B C D E
  16. A B C D E
  17. A B C D E
  18. A B C D E
  19. A B C D E
  20. A B C D E
  21. A B C D E
  22. A B C D E
  23. A B C D E
  24. A B C D E
  27. A B C D E
  28. A B C D E
  29. A B C D E
  30. A B C D E
  31. A B C D E
  32. A B C D E
  33. A B C D E
  34. A B C D E
  35. A B C D E
  36. A B C D E
  37. A B C D E
  38. A B C D E
  39. A B C D E
  40. A B C D E
  41. A B C D E
  42. A B C D E
  43. A B C D E
  44. A B C D E
  45. A B C D E
  46. A B C D E
  47. A B C D E
  48. A B C D E
  49. A B C D E

# Ejercicio

- ▶ examples
- ▶ now
- ▶ packages
- ▶ scripts
- ▶ src
- ▶ test
- ▶ types

.babelrc.js

.editorconfig

.eslintignore

.eslintrc.js

.flowconfig

.gitignore

BACKERS.md

LICENSE

v2.6.0-beta.2 build: release 2.6.0-beta.2

build: build 2.6.0-beta.2

feat: dynamic directive arguments for v-on, v-bind and custom directives (#9371)

origin/dynamic-directive-arguments feat: dynamic args for custom directives

perf: improve scoped slots change detection accuracy (#9371)

test: test cases for v-on/v-bind dynamic arguments

refactor: v-bind dynamic arguments use bind helper

test: fix tests, resolve helper conflict

fix: fix middle modifier

feat: handle dynamic argument for v-bind.sync

origin/slot-optimization perf: improve scoped slots change detection

feat: dynamic directive arguments for v-bind and v-on

refactor: extend dom-props update skip to more strict key equality when

fix: fix checkbox event edge case in Firefox

test: fix tests in IE/Edge

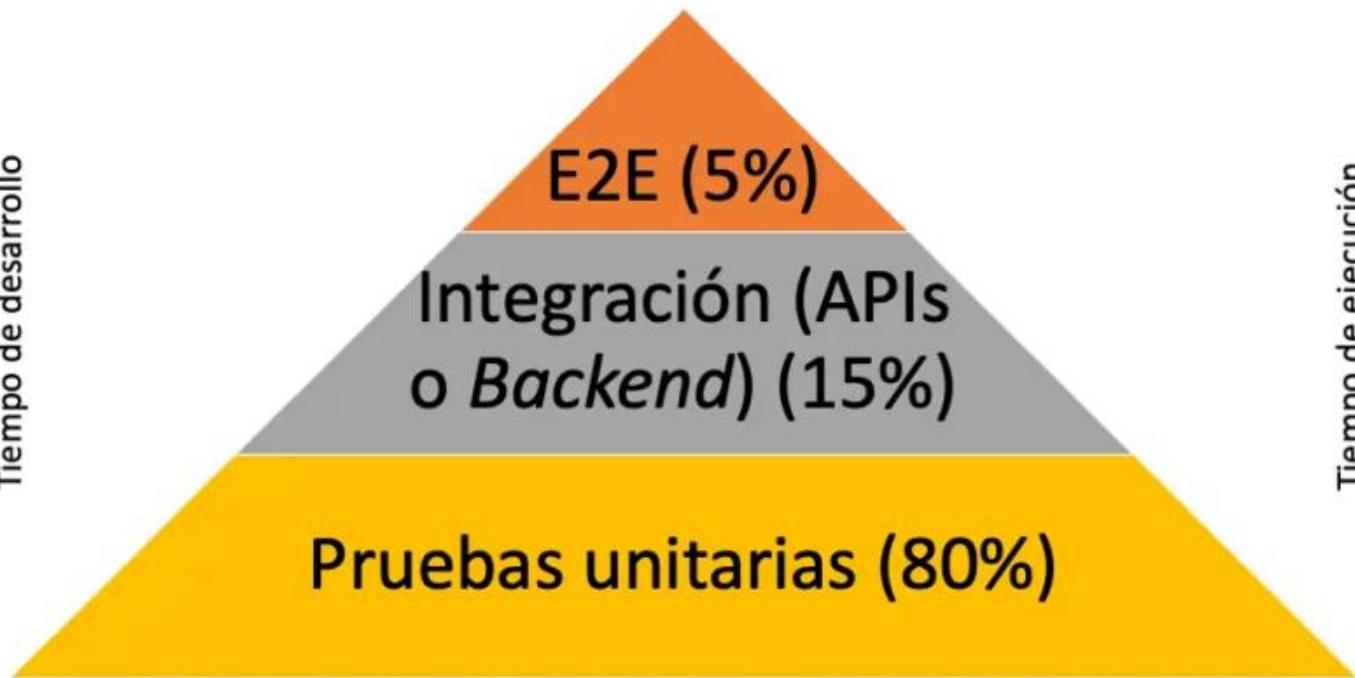
refactor: simplify timestamping check



\$\$\$

Tiempo de desarrollo

₡

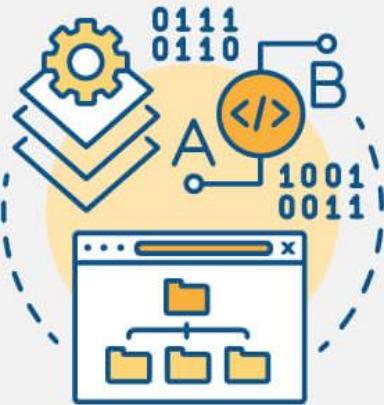


Tiempo de ejecución





Unit testing



Integration testing



Functionality testing

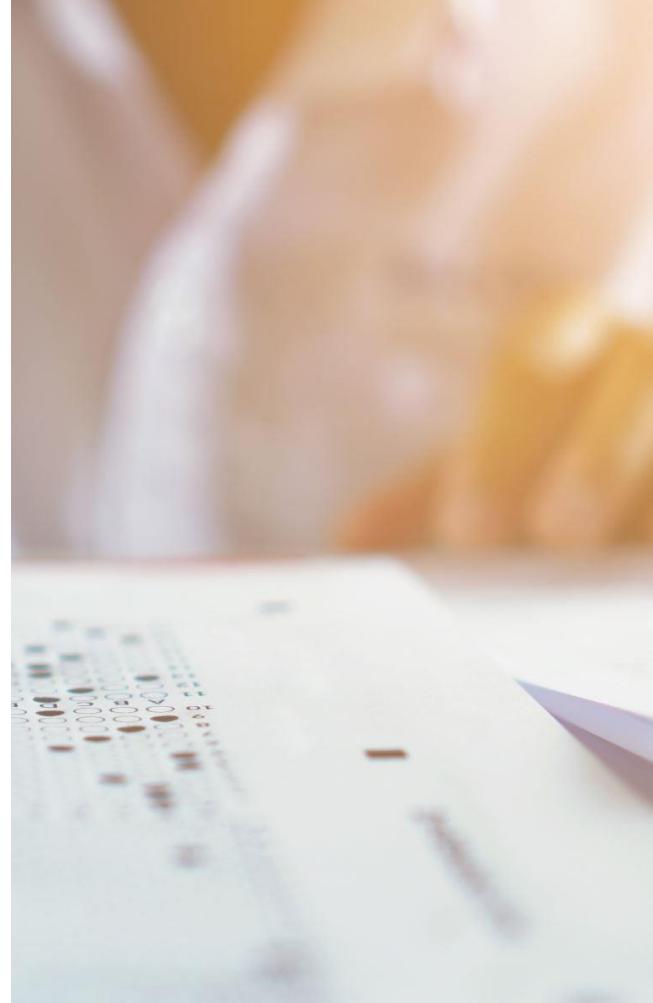


# mockito



# ¿Por qué utilizarlo?

- Permite escribir y ejecutar tests unitarios de código integrado por varios componentes.
- Simula el comportamiento de componentes (mock, spy).
- Proporciona rapidez en las pruebas.
- Útil para seguir el paradigma TDD/BDD.
- Ejemplos de uso: Conexiones con BD. Servicios web. Clases de lenta ejecución. Clases con side-effects. Clases con un comportamiento indefinido.





Mockito permite basarnos en el comportamiento de otros componentes:



Dado que (Given): Preparación estado inicial.



Cuando (When): Se invoca al método.



Entonces (Then): Validamos el comportamiento esperado

```
@Test
Run Test | Debug Test| ✓
void our_first_test_with_mockito() {
 // Creamos el objeto ficticio.
 List<String> first_mock = mock(ArrayList.class);

 // Simulamos el comportamiento
 when(first_mock.get(0)).thenReturn("first_element");

 String first_element = first_mock.get(0);

 // Verificamos
 assertEquals("first_element", first_element);
 verify(first_mock).get(0);
}
```

# ARGUMENT MATCHERS

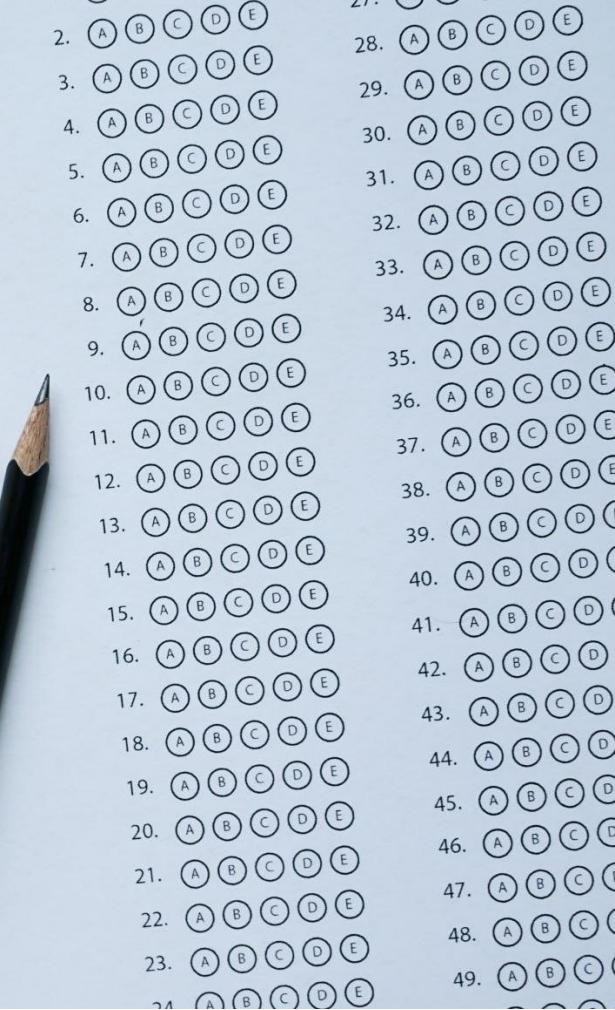
- Los ArgumentMatchers de Mockito permiten:
- Verificar argumentos con los que son llamados métodos de los mocks.
- Simular el comportamiento de un componente.
- eq(), any(), anyInt(), anyString()... isNull(), isNotNull(), isA(Class c)

A blackboard filled with mathematical calculations and diagrams. At the top, there's a diagram of a rectangle divided into two triangles by a diagonal line. Below it, a circle is shown with its radius labeled 'c'. To the right, there's a right-angled triangle with legs labeled 'x' and 'y'. Several equations are written around these diagrams, including:  
1.  $x^2 + y^2 = c^2$   
2.  $c(x, y) \left\{ \begin{array}{l} xy = c \\ cx - cy = 0 \\ 2\pi = c \end{array} \right.$   
3.  $\frac{2x}{y} + \frac{x^2 + y^2}{c^2} + \frac{x^2}{y^2} = 1$   
4.  $= 584. + n^{av} (x^2 + 3c)$   
5.  $\sum N_{30} - x - \frac{1}{2} [364 + x]$   
6.  $\beta = 9 + x^2$



# VERIFY

- Método estático.
- Permite testear la ejecución de un mock.
- `verify(mock, [times(int number)]).método()`
- `verify(mock, never()).método()`
- `verifyNoInteractions()`
- `verify(mock, {atLeast(int), atMost(int), atLeastOnce(), atMostOnce()})`









Apache JMeter (5.0 r1840935)

The screenshot shows the Apache JMeter 5.0 interface. The left sidebar contains a tree view of the 'Test Plan' structure, with 'HTTP(S) Test Script Recorder' currently selected. The main panel displays the configuration for this recorder. At the top, there are fields for 'Name' (set to 'HTTP(S) Test Script Recorder') and 'Comments'. Below these are three buttons: 'Start' (green triangle), 'Stop' (grey circle), and 'Restart' (blue square). Under 'Global Settings', the 'Port' is set to 8888 and the 'HTTPS Domains' field is empty. The 'Test Plan Creation' tab is active, showing 'Test plan content' with 'Target Controller' set to 'Use Recording Controller'. The 'Grouping' option is set to 'Put each group in a new transaction controller'. There are several checkboxes at the bottom of this section: 'Capture HTTP Headers' (checked), 'Add Assertions' (unchecked), and 'Regex matching' (checked). The 'HTTP Sampler settings' section includes a dropdown for 'Transaction name' (set to a dropdown arrow), a field for 'Create new transaction after request (ms)', and three checkboxes: 'Retrieve All Embedded Resources' (unchecked), 'Redirect Automatically' (unchecked), and 'Use KeepAlive' (checked). To the right of these settings are two checkboxes: 'Follow Redirects' (checked) and a large empty text area labeled 'Type:'.

. ORM con  
**JavaEE**



# Introducción a los ORM

- Las siglas ORM significan “Object-Relational mapping” 1) y en castellano es “Mapeo Objeto-Relacional”. El ORM es simplemente el código que escribimos para guardar el valor de nuestras clases en una base de datos relacional. Simplemente éso.

# Ejemplo

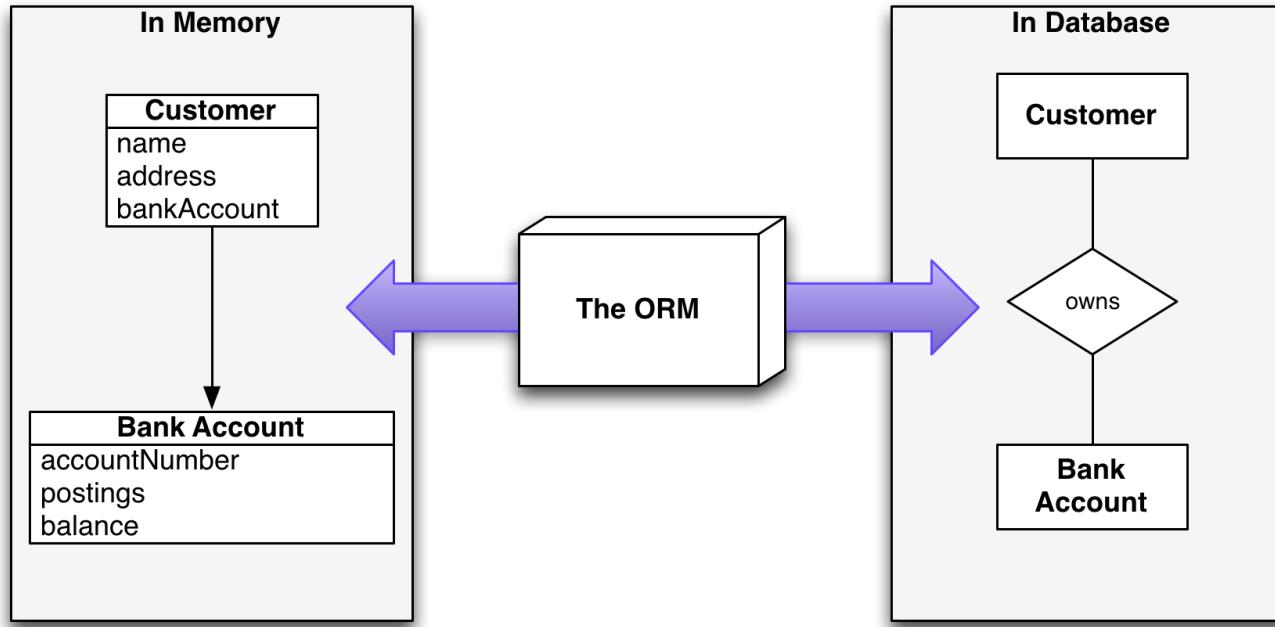
| C      | Profesor |
|--------|----------|
| int    | id       |
| String | nombre   |
| String | ape1     |
| String | ape2     |

| C       | «Table» Profesor |
|---------|------------------|
| INTEGER | id               |
| VARCHAR | nombre           |
| VARCHAR | ape1             |
| VARCHAR | ape2             |

# ➤ Desfase objeto-relacional

- El desfase objeto-relacional surge cuando en el desarrollo de una aplicación con un lenguaje orientado a objetos se hace uso de una base de datos relacional. Hay que tener en cuenta que esta situación se da porque tanto los lenguajes orientados a objetos como las bases de datos relacionales están ampliamente extendidas.

# ➤ ¿Qué es el mapeo objeto-relacional?



```
 . @Entity
 . @Table(name="personajes")
 public class Personaje {
 . @Id // Marca el campo como la clave de la tabla
 . @GeneratedValue(strategy = IDENTITY)
 . @Column(name="id")
 private int id;
 . @Column(name="nombre")
 private String nombre;
 . @Column(name="descripcion")
 private String descripcion;
 . @Column(name="vida")
 private int vida;
 . @Column(name="ataque")
 private int ataque;
 .
 public Personaje(. . .) {
 .
 }
 .
 // getters y setters
 }
```

# Hibernate

- En nuestro caso usaremos Hibernate como librería ORM, concretamente Hibernate 5.2. Habrá que tenerlo en cuenta puesto que algunas clases/métodos pueden variar entre diferentes versiones de este framework, especialmente a la hora de configurar (`hibernate.cfg.xml`) e implementar el gestor de sesiones (`HibernateUtil.java`)

- Las anotaciones que nos podemos encontrar para anotar una clase Java que debe ser mapeada con una tabla son:
- `@Entity` Indica que la clase es una tabla en la base de datos
- `@Table(name = “nombre_tabla”, catalog = “nombre_base_datos”)` Indica el nombre de la tabla y la base de datos a la que pertenece (este último parámetro no es necesario ya que esa información viene en el fichero de configuración)

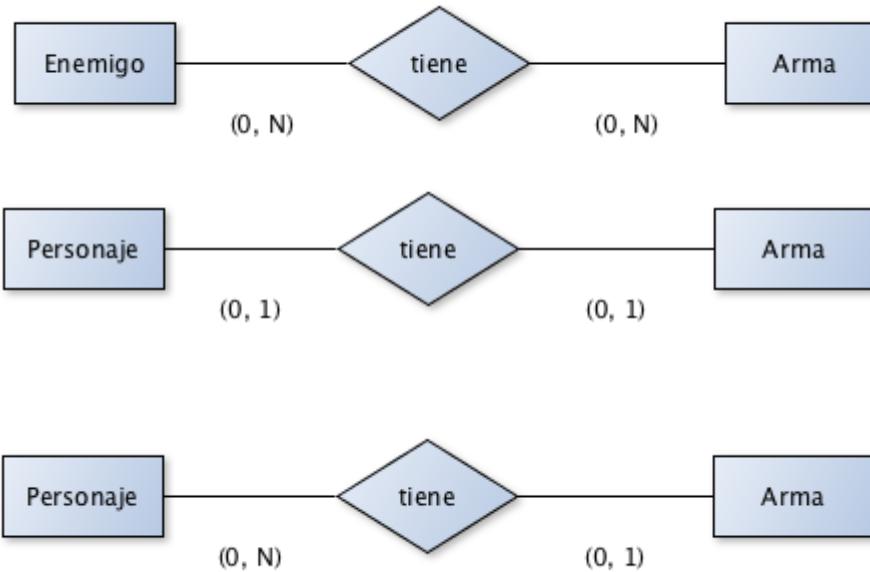
- 
- En el caso de los atributos simples que deben ser mapeados con los campos de la tabla correspondiente:
  - `@Id` Indica que un atributo es la clave
  - `@GeneratedValue(strategy = GenerationType.IDENTITY)` Indica que es un valor autonumérico (PRIMARY KEY en MySQL, por ejemplo)
  - `@Column(name = “nombre_columna”)` Se utiliza para indicar el nombre de la columna en la tabla donde debe ser mapeado el atributo



# Relaciones

- **@OneToOne** Indica que el objeto es parte de una relación 1-1
- **@ManyToOne** Indica que el objeto es parte de una relación N-1. En este caso el atributo sería el lado 1
- **@OneToMany** Indica que el objeto es parte de una relación 1-N. En este caso el atributo sería el lado N
- **@ManyToMany** Indica que el objeto es parte de una relación N-M. En este caso se indica la tabla que mantiene la referencia entre las tablas y los campos que hacen el papel de claves ajenas en la base de datos

# Tipos



# ➤ Registrar un objeto

- UnaClase unObjeto = new UnaClase();
- ...
- Session sesion =  
HibernateUtil.getCurrentSession();
- sesion.beginTransaction();
- sesion.save(unObjeto);
- sesion.getTransaction().commit();
- sesion.close();
- ...

# Modificar un objeto

- · ·
- Session sesion =  
HibernateUtil.getCurrentSession();
- sesion.beginTransaction();
- sesion.save(unObjeto);
- sesion.getTransaction().commit();
- sesion.close();
- · ·

# ➤ Eliminar un objeto

- · ·
- Session sesion =  
HibernateUtil.getCurrentSession();
- sesion.beginTransaction();
- sesion.delete(unObjeto);
- sesion.getTransaction().commit();
- sesion.close();
- · ·

# ➤ Obtener un objeto identificado por el id

- · ·
- int id = ...;
- Cliente cliente =  
HibernateUtil.getCurrentSession().get(Clien  
te.class, id);
- · ·

# ➤ Obtener todos los objetos de una clase

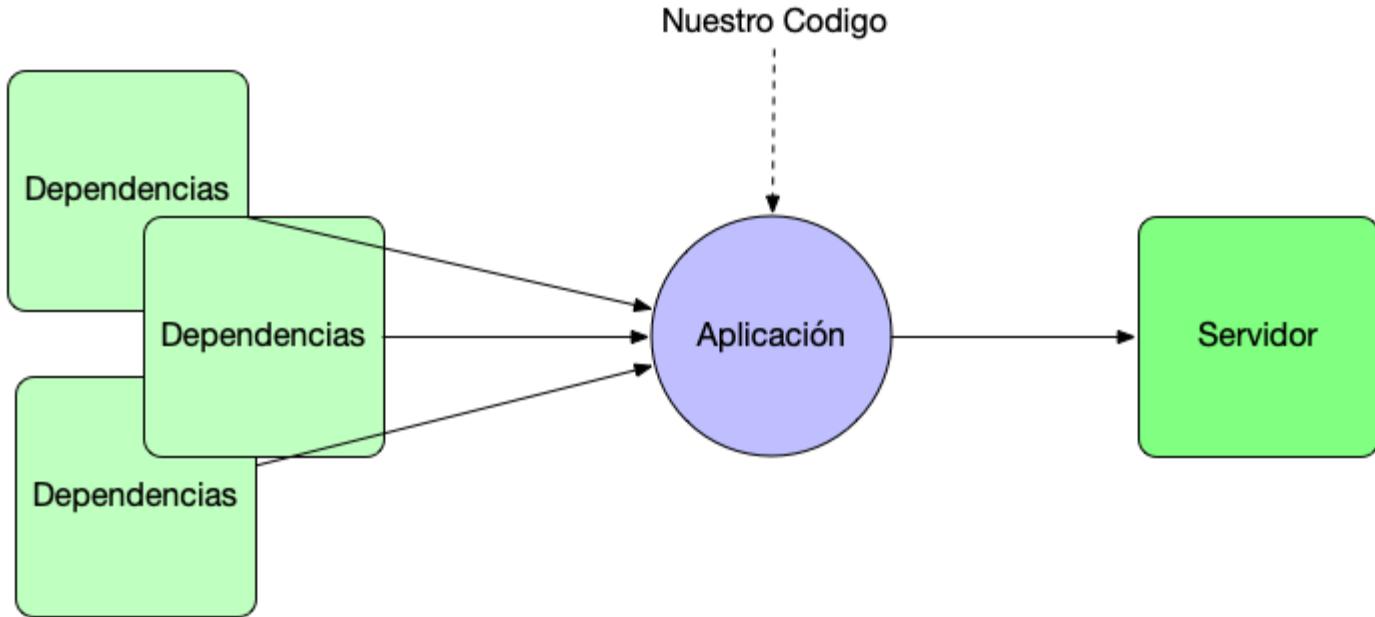
- ...
  - Query query = HibernateUtil.getCurrentSession().createQuery("FROM Cliente");
  - ArrayList<Cliente> clientes = (ArrayList<Cliente>) query.list();
  - ...

- **Servidores de  
Aplicaciones  
JAVA**

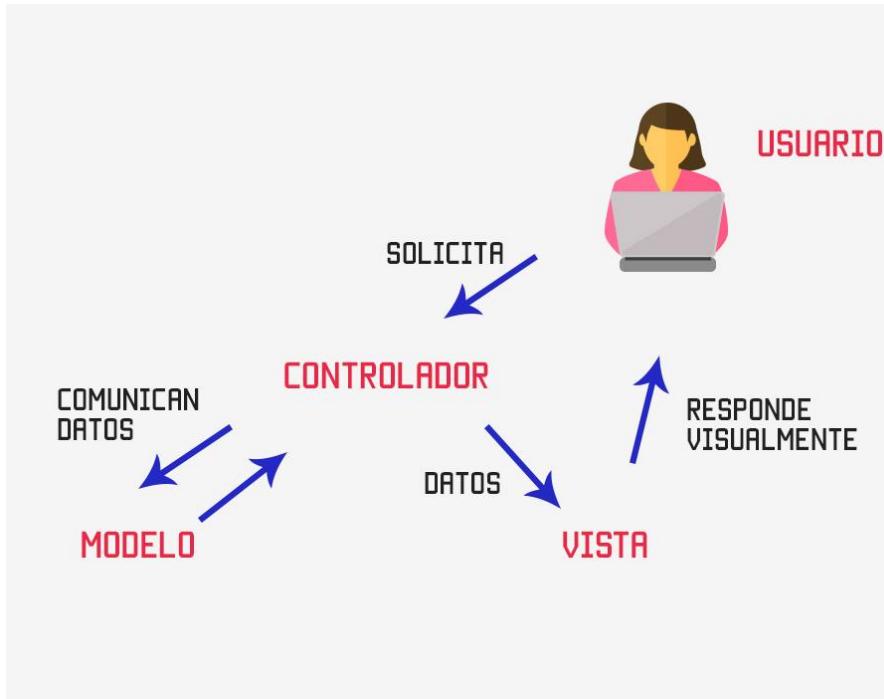


# ➤ Creación de aplicaciones web. Spring Boot





# > MVC





# spring initializr

Project      Language

Gradle - Groovy     Gradle - Kotlin     Java     Kotlin     Groovy  
 Maven

Spring Boot

3.2.0 (SNAPSHOT)     3.2.0 (M3)     3.1.5 (SNAPSHOT)     3.1.4  
 3.0.12 (SNAPSHOT)     3.0.11     2.7.17 (SNAPSHOT)     2.7.16

Project Metadata

Group: com.arquitecturajava

Artifact: HolaMundo

Name: HolaMundo

Description: HolaMundo

Package name: com.arquitecturajava.HolaMundo

Packaging:  Jar     War

Java:  21     17     11     8

Dependencies

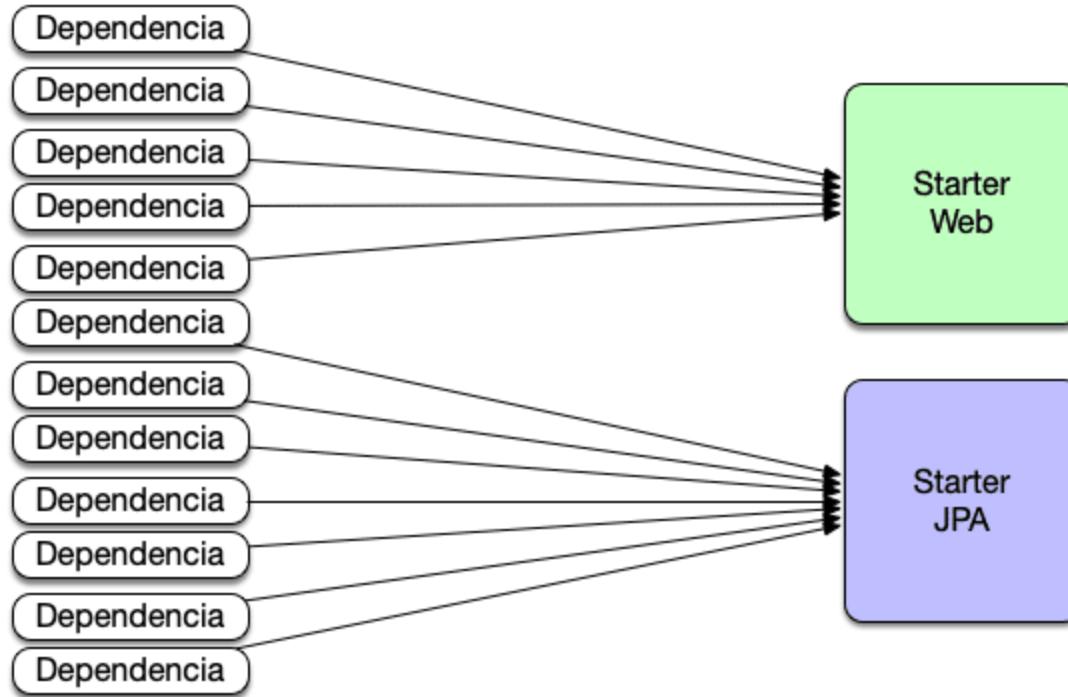
**Spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Thymeleaf** TEMPLATE ENGINES

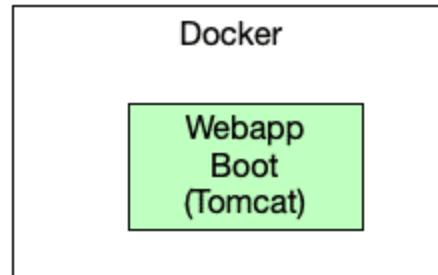
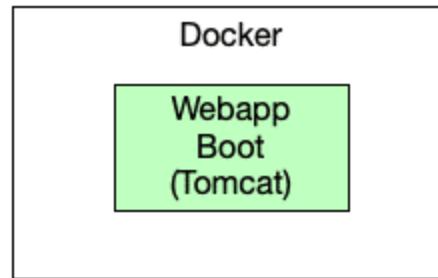
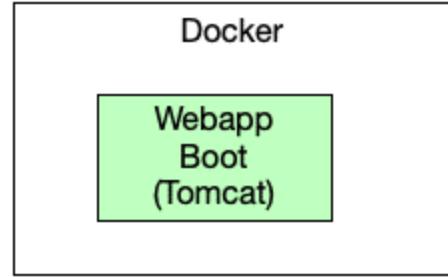
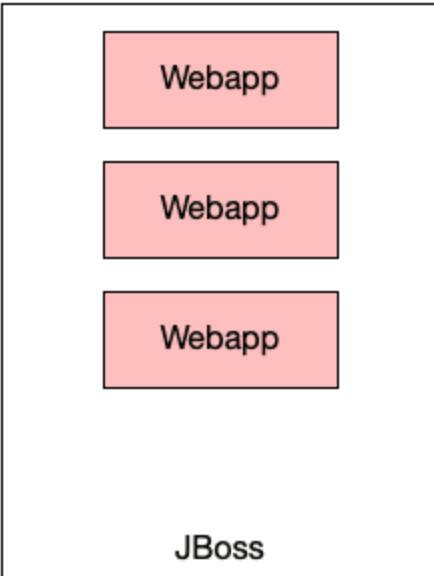
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

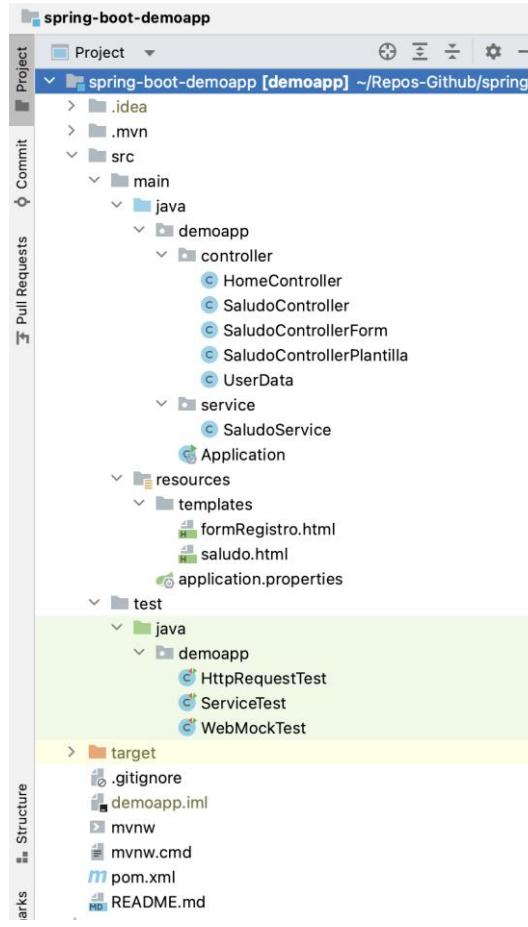
**ADD DEPENDENCIES...** + 8





|                                |            |
|--------------------------------|------------|
| mvnw                           | ✓ hoy 7:29 |
| mvnw.cmd                       | ✓ hoy 7:29 |
| pom.xml                        | ✓ hoy 7:29 |
| src                            | ✓ hoy 8:31 |
| main                           | ✓ hoy 8:31 |
| java                           | ✓ hoy 8:31 |
| com                            | ✓ hoy 8:31 |
| arquitecturajava               | ✓ hoy 8:31 |
| HolaSpringBootApplication.java | ✓ hoy 7:29 |
| resources                      | ✓ hoy 8:31 |
| application.properties         | ✓ hoy 7:29 |
| static                         | ✓ hoy 7:29 |
| templates                      | ✓ hoy 7:29 |
| test                           | ✓ hoy 8:31 |
| java                           | ✓ hoy 7:29 |
| com                            | ✓ hoy 7:29 |





# Controladores

- Los controladores definen el código a ejecutar como respuesta a una petición HTTP. Son clases que se suelen colocar en el paquete controller y están anotadas con @Controller.
- Vemos un ejemplo en la clase SaludoController.



# Clases de Servicio

- Es recomendable definir clases de servicio en las que se implementa la lógica de negocio de la aplicación. Las clases controller llaman a las clases servicio, que son las que realmente realizan todo el procesamiento.
- De esta forma se separan las responsabilidades. Las clases controller se encargan de procesar las peticiones y las respuestas HTTP y las clases de servicio son las que realmente realizan la lógica de negocio y devuelven el contenido de las respuestas. Si en algún momento hay que añadir una nueva capa de presentación en la que, por ejemplo, se trabaje con objetos JSON, no será necesario cambiar la capa de servicios, sólo añadir nuevas clases controller.
- La separación de la lógica de negocio en las clases de servicio permite también realizar tests que trabajan sobre objetos Java, independientes de los formatos de entrada/salida manejados por los controladores.

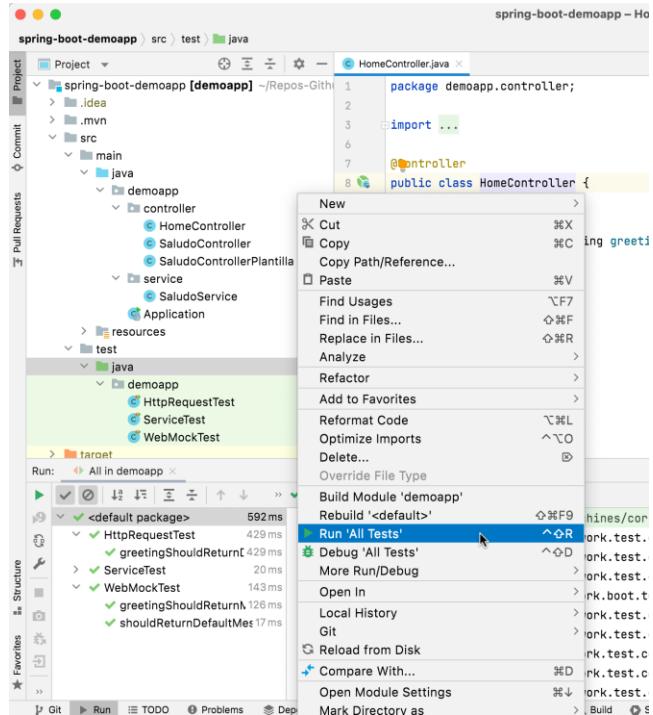
# Plantillas Thymeleaf

- Las páginas HTML devueltas se pueden construir utilizando un lenguaje de plantillas con el framework Thymeleaf. Las plantillas thymeleaf son páginas HTML en las que se introducen objetos Java pasados por los controllers.
- Las plantillas se incluyen en el directorio resources/templates.

# Formularios y validación

```
• package demoapp.controller;
•
• import javax.validation.constraints.Size;
•
• public class UserData {
• @Size(min=3, max=30)
• String nombre;
•
• public void setNombre(String nombre) {
• this.nombre = nombre;
• }
•
• public String getNombre() {
• return nombre;
• }
• }
```

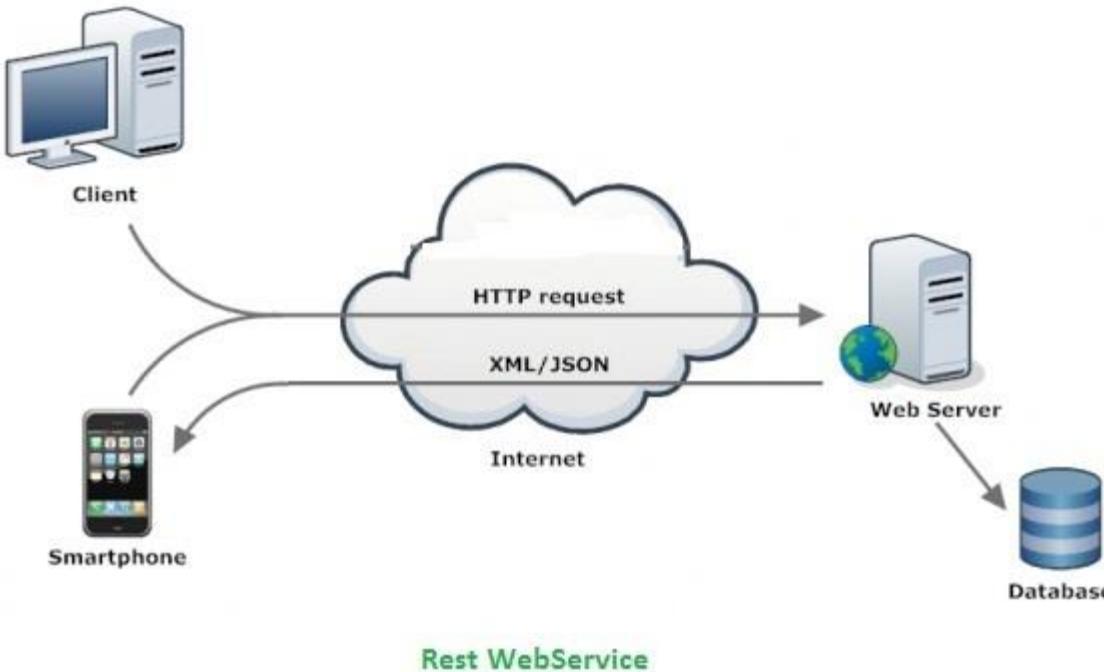
# Tests



- **Aplicaciones  
Cliente-  
Servidor  
(REST, SOAP)**



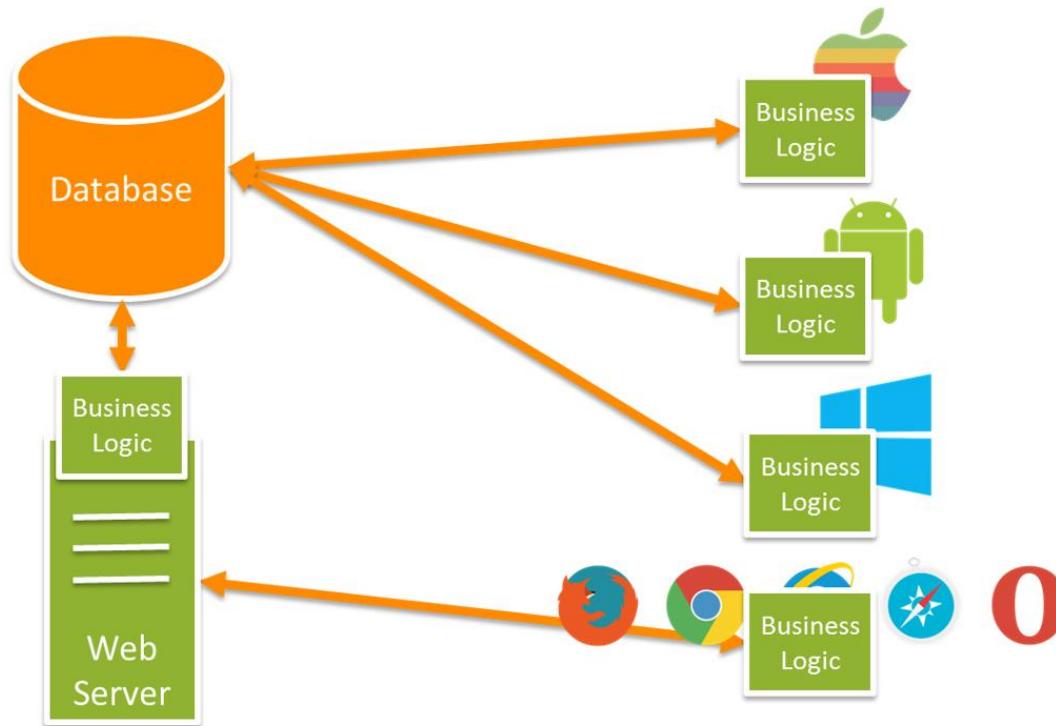
# ¿Qué son los servicios web REST?

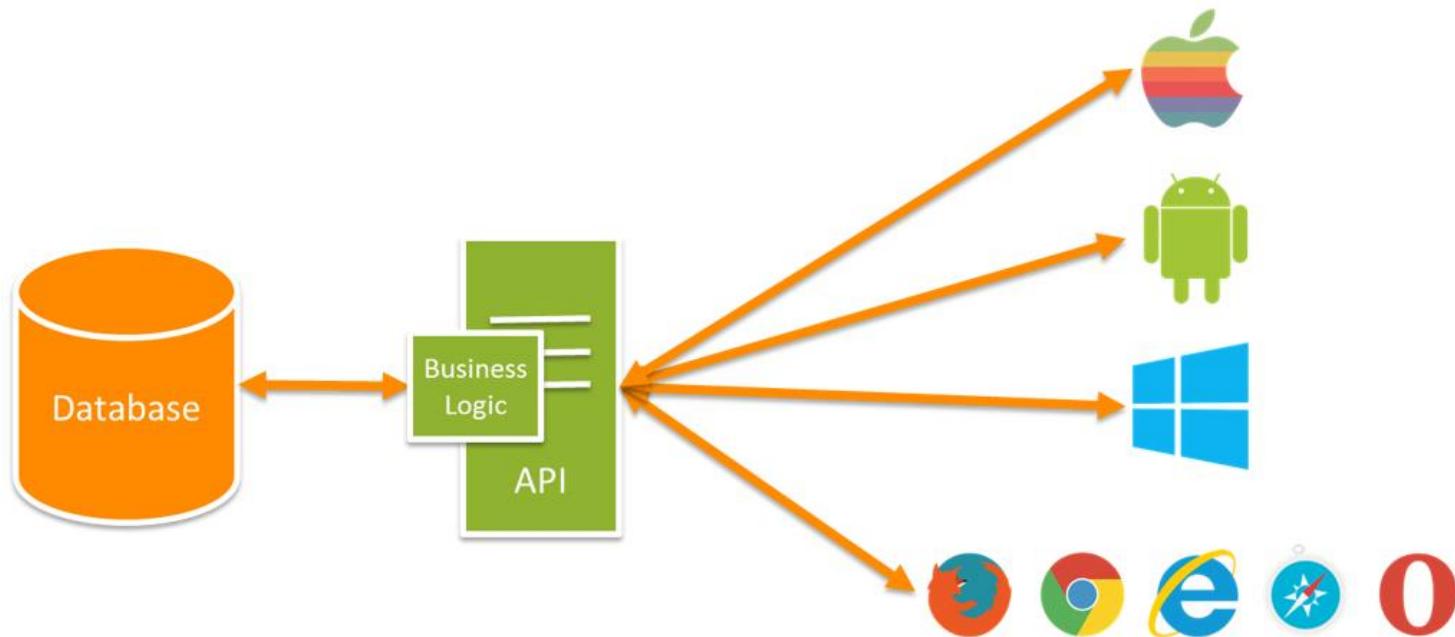


# REST Web Service

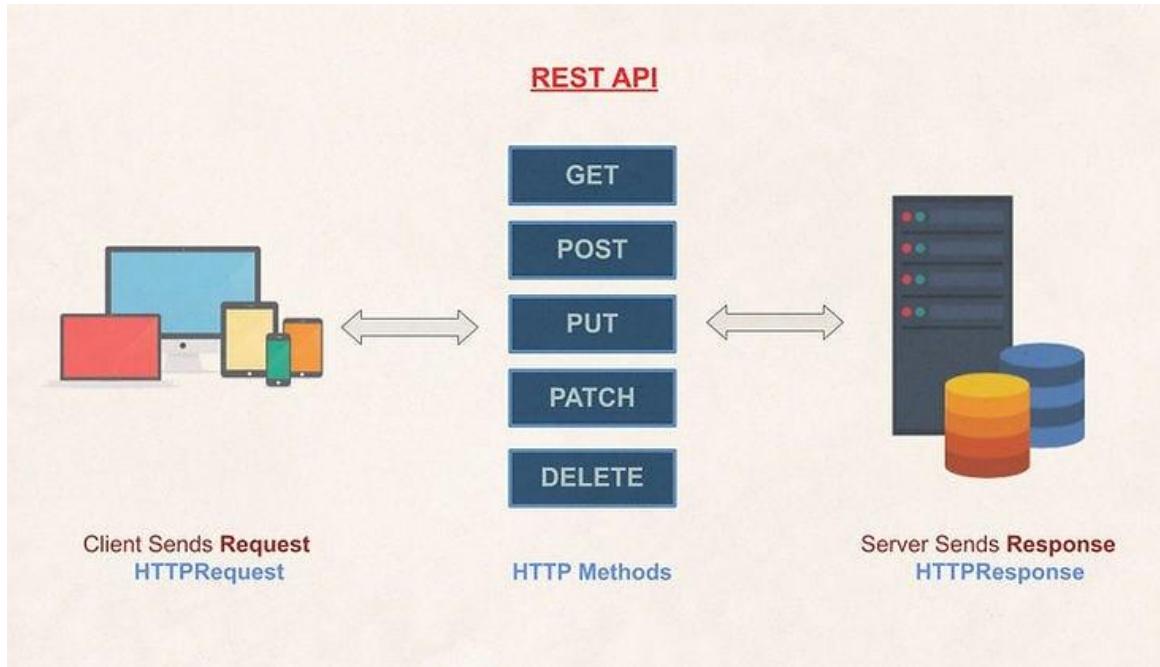
| Task                    | Method | Path        |
|-------------------------|--------|-------------|
| Create a new task       | POST   | /tasks      |
| Delete an existing task | DELETE | /tasks/{id} |
| Get a specific task     | GET    | /tasks/{id} |
| Search for tasks        | GET    | /tasks      |
| Update an existing task | PUT    | /tasks/{id} |

# Web API

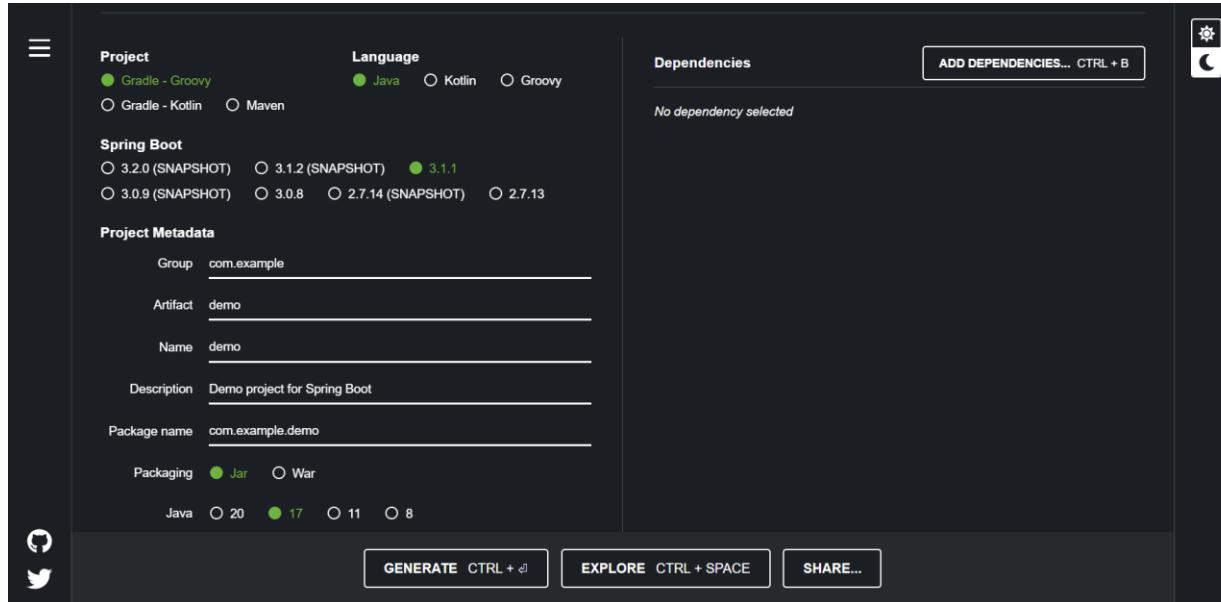




# ➤ Implementación del controller



# Paso 1: Configuración inicial con Spring Initializr





## Paso 2: Configuración de la base de datos

- . CREATE DATABASE empleados\_db;

## Paso 3: Configuración de la conexión a la base de datos

- # Configuración de la conexión a la base de dato
  - spring.datasource.url=jdbc:mysql://localhost:3306/empleados\_db
  - spring.datasource.username=root
  - spring.datasource.password=pass
- 
- # Actualizar o crear la base de datos automáticamente
  - spring.jpa.hibernate.ddl-auto=update

# Paso 4: Creación de la entidad "Empleados"

```
• package com.empleados.entity;

• import javax.persistence.*;
• import lombok.Data;

• @Entity
• @Data
• @Table(name = "empleados")
• public class Empleados {
• @Id
• @GeneratedValue(strategy = GenerationType.IDENTITY)
• @Column(name = "id_Empleado")
• private Integer idEmpleados;

• @Column(name = "nombre")
• private String nombre;

• @Column(name = "apellido")
• private String apellido;

• @Column(name = "email")
• private String email;
• }
```

## Paso 5: Creación del repositorio "EmpleadoRepository"

- package com.empleados.repository
- 
- import com.empleados.entity.Empleados;
- import org.springframework.data.jpa.repository.JpaRepository;
- 
- public interface EmpleadoRepository extends JpaRepository<Empleados, Integer> {
- }

# Paso 6: Creación del controlador "EmpleadoController"

- Hemos creado un controlador llamado "EmpleadoController" y lo hemos marcado con la anotación @RestController. Esta anotación le dice a Spring que este controlador gestionará peticiones HTTP y devolverá datos en formato JSON.
- @Autowired se utiliza para inyectar automáticamente una instancia de EmpleadoRepository en el controlador, lo que nos permite acceder a las operaciones del repositorio.
- Hemos definido diferentes métodos para las operaciones CRUD:  
obtenerTodosLosEmpleados(): Devuelve una lista de todos los empleados en la base de datos.
- agregarEmpleado(empleado): Agrega un nuevo empleado a la base de datos.
- editarEmpleado(id, empleado): Actualiza la información de un empleado existente en la base de datos.
- obtenerEmpleadoPorId(id): Obtiene un empleado específico por su ID.
- eliminarEmpleado(id): Elimina un empleado específico por su ID.

# Postman

The screenshot shows the Postman application interface. At the top, there is a header with a yellow arrow icon, the word "Postman", and a search bar. Below the header, the main interface has a dark theme.

The top navigation bar includes:

- Method: **POST**
- URL: `http://localhost:8080/api/v1/empleados/nuevo`
- Buttons: **Send** (blue) and a dropdown menu.

The main workspace is divided into sections:

- Params**, **Authorization**, **Headers (8)** (highlighted in green), **Body** (highlighted in orange), **Pre-request Script**, **Tests**, **Settings**.
- Body** tab options: none, form-data, x-www-form-urlencoded, raw (selected), binary, GraphQL, JSON (highlighted in blue).
- Body** content area (JSON format):

```
1 {
2 "nombre": "borrar",
3 "apellido": "borrador",
4 "email": "borrador@mail.com"
5 }
```
- Cookies** and **Beautify** buttons.

At the bottom, the results section shows:

- Body** tab (selected).
- Cookies**, **Headers (5)**, **Test Results**.
- Status: **200 OK**, Time: **2.59 s**, Size: **249 B**.
- Buttons: **Save as Example**, **...**, **Copy**, **Find**.
- Pretty**, **Raw**, **Preview**, **Visualize**, **JSON** (selected).
- Body** content area (JSON format):

```
1 {
2 "idEmpleados": 4,
3 "nombre": "borrar",
4 "apellido": "borrador",
5 "email": "borrador@mail.com"
6 }
```

# Jorge López Blasco

*Formato IT*

+34 693303309

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

C/ Manuel Tovar, 31 28034-  
Madrid

