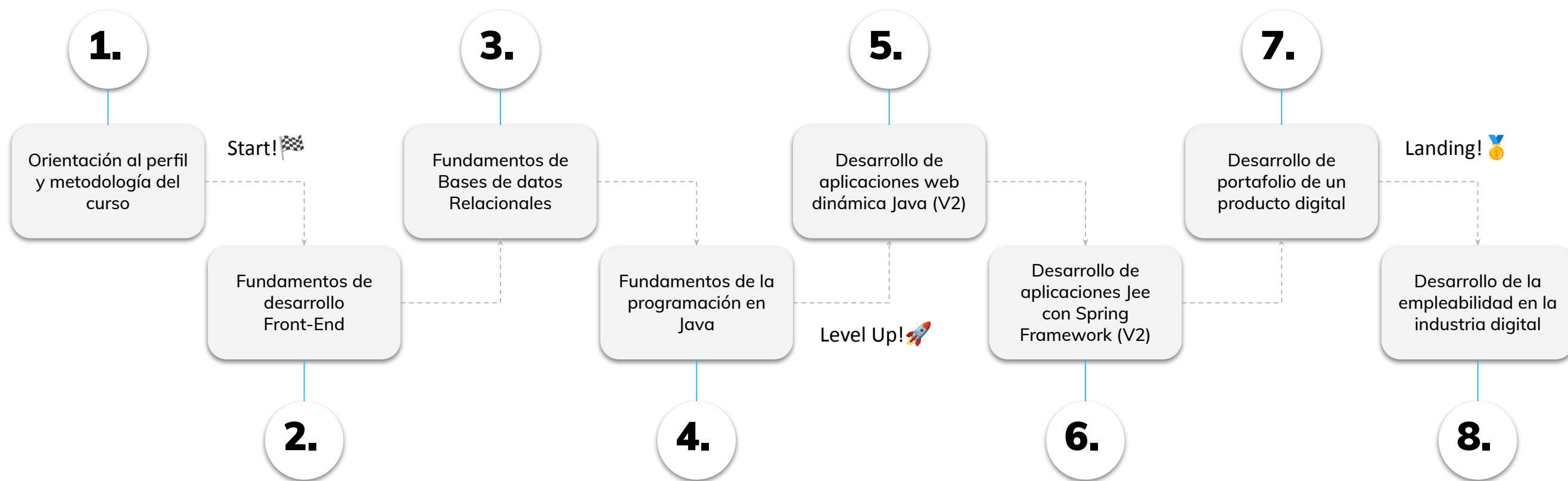


» El gestor de Proyectos

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ *Dependencias Maven*
- ✓ *Repositorio local y remoto*

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.1

Start! 🏁

**El gestor de
Proyectos**

Ciclo de Vida

Compile y Test

El ciclo de vida compilacion:
Compilación
Ejecución de Pruebas

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



Aprender el concepto de ciclo de vida



Conocer las etapas del ciclo de compilación



› Ciclo de vida de compilación



Ciclo de vida de compilación

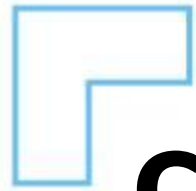


Maven se fundamenta en el concepto central de ciclo de vida de construcción (**build lifecycle**). Esto significa que el proceso de construcción y distribución de un artifact (artefacto = **proyecto**) concreto está definido claramente.



Para la persona que quiera construir un proyecto significa que solamente es necesario aprender un pequeño conjunto de comandos para construir cualquier proyecto Maven, y el POM se asegurará de que obtenga los resultados deseados.





Ciclo de vida de compilación



Existen tres ciclos de vida en el sistema: **default**, **clean** y **site**.

- El ciclo de vida **default** controla el despliegue de tu proyecto.
- El ciclo de vida **clean** controla la limpieza de tu proyecto.
- El ciclo de vida **site** controla la creación del site de documentación de tu proyecto.





Ciclo de vida de compilación



Un ciclo de vida de construcción se compone de fases

Cada uno de estos ciclos de vida de construcción está definido por una lista diferente de fases (phases), donde cada fase representa un estado en el ciclo de vida.

Cada una de estas fases se ejecutan secuencialmente para completar el ciclo de vida.



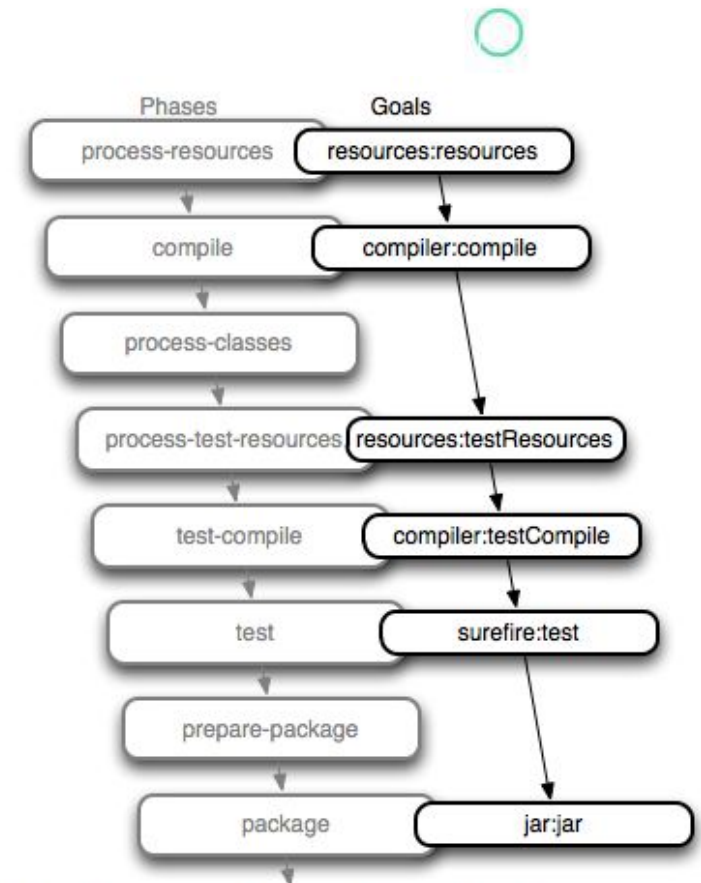
Clean Lifecycle	Default Lifecycle		Site Lifecycle
pre-clean	validate	test-compile	pre-site
clean	initialize	process-test-classes	site
post-clean	generate-sources	test	post-site
	process-sources	prepare-package	site-deploy
	generate-resources	package	
	process-resources	pre-integration-test	
	compile	integration-test	
	process-classes	post-integration-test	
	generate-test-sources	verify	
	process-test-sources	install	
	generate-test-resources	deploy	
	process-test-resources		



Ciclo de vida de compilación

Una fase de construcción se compone de objetivos de plugins

Sin embargo, aunque una fase de construcción es responsable de un paso específico en el ciclo de vida de construcción, el modo en que este se lleva a cabo puede variar. Esto se consigue declarando objetivos (goals) de plugins asociados a fases de construcción.



Note: There are more phases than shown above, this is a partial list



Ciclo de vida de compilación



Un objetivo de un plugin representa una tarea específica que contribuye en la construcción y gestión de un proyecto. Puede estar asociada a cero o más fases de construcción. Un objetivo que no esté asociado a ninguna fase podría ejecutarse fuera del ciclo de vida de construcción mediante la invocación directa.

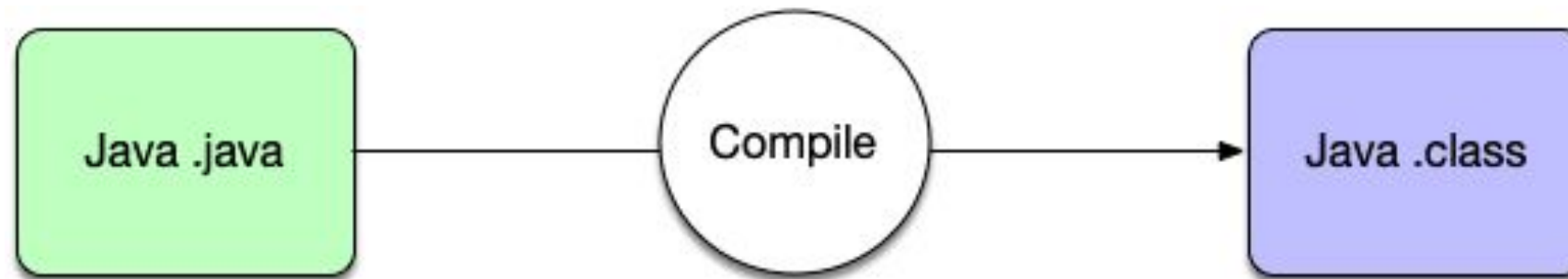


El orden de ejecución depende del orden en el que los objetivos y las fases de construcción sean invocados.



Ciclo de vida de compilación

El **ciclo de vida de compilación** de Maven es un proceso fundamental para la construcción y gestión de proyectos basados en Maven. Este ciclo de vida consta de tres fases principales: "**compile**", "**test-compile**" y "**process-resources**".





Ciclo de vida de compilación

Pasos y acciones en la fase "compile":

1- Recopilación de código fuente: En esta etapa, Maven busca los archivos fuente en el directorio de código fuente del proyecto, que suele ser "`src/main/java`". Este directorio contiene los archivos Java escritos por los desarrolladores. Maven recopila estos archivos y genera las clases Java compiladas correspondientes.

2- Gestión de dependencias: Antes de la compilación, Maven resuelve las dependencias del proyecto. Esto implica descargar las bibliotecas y dependencias necesarias desde los repositorios configurados. Maven garantiza que las versiones correctas de las dependencias estén disponibles para el proceso de compilación.





Ciclo de vida de compilación



3- Compilación del código fuente: Una vez que las dependencias se han resuelto y los archivos fuente se han recopilado, Maven utiliza el compilador Java para compilar el código fuente en clases Java compiladas. Los archivos `.class` resultantes se almacenan en el directorio de salida, que suele ser "`target/classes`".

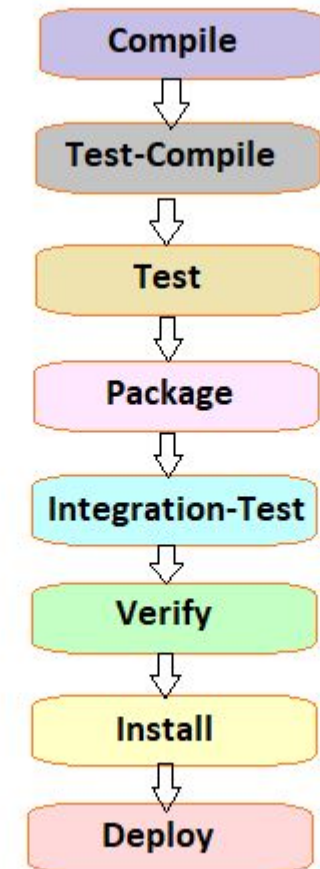
4- Generación de recursos compilados: Además de compilar archivos Java, Maven también puede procesar otros recursos, como archivos de configuración, archivos XML o archivos de propiedades. Los recursos se copian en el directorio de salida para que estén disponibles en la fase de empaquetado.





Ciclo de vida de compilación

5- Generación de archivos JAR/WAR/EAR: Aunque la generación de archivos JAR (Java Archive), WAR (Web Application Archive) o EAR (Enterprise Application Archive) no es parte de la fase "compile" en sí, la fase "compile" prepara el contenido que se incluirá en estos archivos. Las clases compiladas y los recursos procesados se utilizarán en la fase de empaquetado posterior.





Ciclo de vida de compilación



Beneficios y consideraciones de la fase "compile" en Maven:

- **Eficiencia y consistencia:** La fase "*compile*" de Maven automatiza la compilación de código fuente, lo que garantiza un proceso eficiente y consistente. Los desarrolladores no necesitan preocuparse por la configuración del compilador y pueden centrarse en escribir código.
- **Gestión de dependencias:** Maven se encarga de gestionar las dependencias, lo que garantiza que las bibliotecas y componentes necesarios estén disponibles y se utilicen en el proceso de compilación.





Ciclo de vida de compilación



Beneficios y consideraciones de la fase "compile" en Maven:

- **Compatibilidad con múltiples lenguajes y proyectos:** Aunque Maven es ampliamente utilizado en proyectos Java, también se puede configurar para admitir otros lenguajes y proyectos, lo que lo convierte en una herramienta versátil para la compilación.
- **Integración con herramientas de desarrollo:** Maven se integra con varias herramientas de desarrollo y entornos de desarrollo integrados (IDE), lo que facilita la gestión del ciclo de vida del proyecto y la compilación del código fuente.





Ciclo de vida de compilación



En resumen, la fase "compile" en el ciclo de vida de compilación de Maven es un proceso esencial que compila el código fuente de un proyecto, gestiona dependencias, procesa recursos y prepara el contenido que se utilizará en fases posteriores del ciclo de vida. Esta fase automatizada mejora la eficiencia y la consistencia en el desarrollo de proyectos, lo que hace que Maven sea una herramienta valiosa en el mundo del desarrollo de software.



➤ Ejecución de pruebas

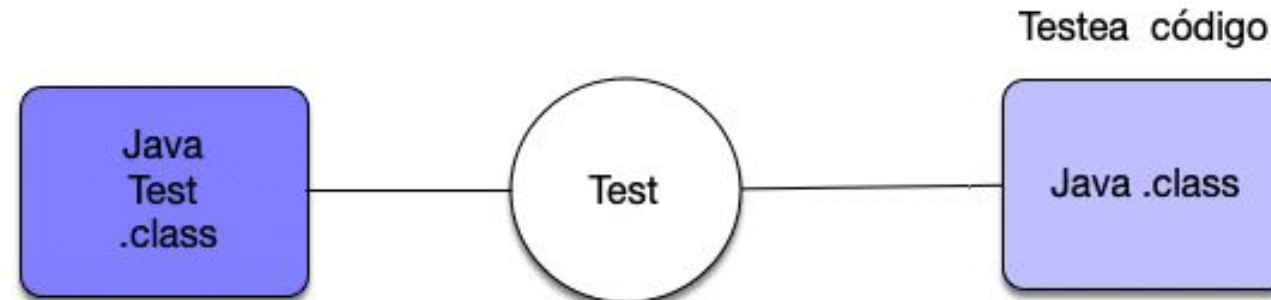


Ejecución de Pruebas



Maven Test:

Esta es otra de las fases principales en las cuales, **una vez se ha compilado el código, se ejecutan las pruebas unitarias** que se han construido para él . De esta forma nos aseguramos que nuestro código es correcto y no nos llevaremos ninguna sorpresa en producción.

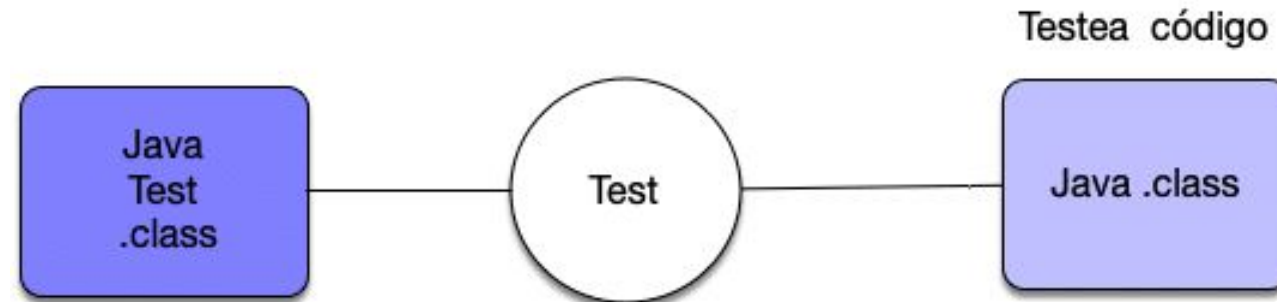




Ejecución de Pruebas



El ciclo de vida de ejecución de pruebas en Maven es una parte fundamental del proceso de desarrollo de software. Esta fase, conocida como "**test**", se centra en la ejecución de pruebas unitarias y de integración en una aplicación.





Ejecución de Pruebas



Pasos y acciones en la fase "test":

Su objetivo principal es llevar a cabo pruebas automatizadas en una aplicación para garantizar su calidad y funcionamiento correcto. Estas pruebas **pueden incluir pruebas unitarias, pruebas de integración y pruebas funcionales.**

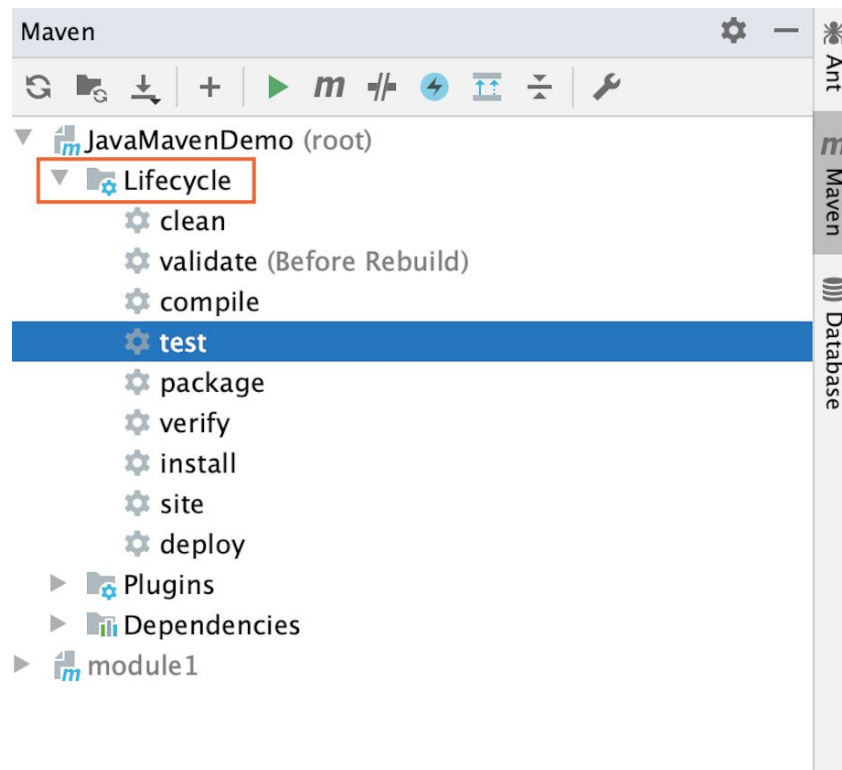


1- Configuración de pruebas: Antes de ejecutar las pruebas, es necesario configurar el entorno de pruebas. Maven utiliza el directorio "**src/test**" para almacenar los archivos de código fuente de pruebas y los recursos relacionados con las pruebas. Las pruebas se organizan generalmente en paquetes y se crean archivos de prueba (por ejemplo, clases JUnit o TestNG) que contienen casos de prueba.





Ejecución de Pruebas



2- Compilación de pruebas: En esta etapa, Maven compila el código fuente de las pruebas. Los archivos de prueba, así como las clases y recursos de prueba, se compilan en clases Java compiladas (**.class**). Estas clases se almacenan en el directorio de salida de pruebas, que suele ser "**target/test-classes**".





Ejecución de Pruebas



3- Ejecución de pruebas: Una vez que las pruebas están compiladas, Maven inicia la ejecución de pruebas automatizadas. Esto implica la ejecución de casos de prueba individuales en busca de errores o problemas de lógica. Las pruebas pueden incluir pruebas unitarias, que se centran en componentes aislados, así como pruebas de integración que evalúan la interacción entre varios componentes.



4- Generación de informes de pruebas: Durante la ejecución de pruebas, Maven recopila información sobre las pruebas realizadas, los resultados obtenidos y cualquier error o fallo encontrado. Esta información se utiliza para generar informes detallados de pruebas. Los informes pueden estar en formatos como XML, HTML o texto y proporcionan una visión general de la calidad de la aplicación.





Ejecución de Pruebas



5- Cobertura de código: La fase "**test**" también puede incluir la medición de la cobertura de código. Esto implica determinar qué partes del código fuente se ejecutaron durante las pruebas y cuáles no. La cobertura de código es una métrica importante para evaluar cuán exhaustivas son las pruebas.



6- Detención de construcción en caso de fallos: Maven puede configurarse para detener la construcción en caso de que las pruebas fallen. Esto garantiza que no se produzca la generación de artefactos si la aplicación no pasa las pruebas.





Ejecución de Pruebas



Beneficios y consideraciones de la fase "test" en Maven:

- **Garantía de calidad:** La fase "test" de Maven asegura que las pruebas automatizadas se realicen de manera sistemática, lo que mejora la calidad del software y reduce la probabilidad de errores en la producción.
- **Retroalimentación rápida:** La ejecución de pruebas proporciona una retroalimentación rápida a los desarrolladores, lo que les permite corregir errores y problemas de manera oportuna.
- **Automatización y repetibilidad:** Las pruebas automatizadas son repetibles y se pueden ejecutar en múltiples entornos, lo que garantiza que el software se comporte de manera consistente.





Ejecución de Pruebas



Beneficios y consideraciones de la fase "test" en Maven:

- **Cobertura de código:** La medición de la cobertura de código ayuda a identificar áreas del código que no se prueban adecuadamente, lo que puede guiar la creación de pruebas adicionales.
- **Integración continua:** La fase "test" es esencial en un entorno de integración continua (CI) donde las pruebas se ejecutan automáticamente después de cada confirmación de código.
- **Mejora la colaboración:** Al garantizar que las pruebas sean parte del proceso de desarrollo, la fase "test" fomenta una mayor colaboración entre los miembros del equipo de desarrollo y mejora la confianza en la calidad del software.





Ejecución de Pruebas



En resumen, la fase "test" en el ciclo de vida de ejecución de pruebas de Maven es una parte crucial del proceso de desarrollo de software. Proporciona un enfoque sistemático y automatizado para garantizar la calidad y la confiabilidad del software, al tiempo que permite a los desarrolladores recibir retroalimentación temprana y continua sobre el estado de sus aplicaciones. La ejecución de pruebas automatizadas es una práctica esencial en el desarrollo de software moderno y contribuye significativamente a la entrega de software de alta calidad a los usuarios finales.




LIVE CODING

Ejemplo en vivo

Test:

En este ejemplo en vivo, veremos la carpeta Test de un proyecto Maven para comprender de manera gráfica cómo se crean las carpetas y las pruebas de ejecución.

  **Tiempo: 15 minutos**



Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro...

Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase.





Ejercicio N° 1

Compile y Test



Compile y Test



Manos a la obra: 🙌

En este ejercicio, vamos a crear un proyecto Java simple y escribir pruebas unitarias para él. Además, ejecutaremos los comandos Maven necesarios para realizar la compilación y la ejecución de pruebas.



Paso a paso: ✍️

1- **Configuración del entorno:** Asegúrate de tener instalado Java en tu sistema y Maven correctamente configurado.

Tiempo 🕒: 30 minutos



Compile y Test



Paso a paso: 🛠️

2- **Creación de un proyecto Maven:** Abre una terminal y crea un directorio para tu proyecto. Luego, ejecuta el siguiente comando para crear un proyecto Maven:



```
mvn archetype:generate -DgroupId=com.ejercicio  
-DartifactId=maven-compile-test  
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Esto creará una estructura de proyecto Maven básica con un directorio de código fuente ("src/main/java") y un directorio de pruebas ("src/test/java").



Compile y Test



Paso a paso:

3- **Escribir código fuente:** Dentro del directorio "src/main/java/com/ejercicio", crea una clase Java simple llamada "Calculadora" con un método para sumar dos números. Por ejemplo:



```
package com.ejercicio;

public class Calculadora {
    public int sumar(int a, int b) {
        return a + b;
    }
}
```





Compile y Test

Paso a paso: 

4- Escribir pruebas unitarias: Dentro del directorio "src/test/java/com/ejercicio", crea una clase llamada "CalculadoraTest" para escribir pruebas unitarias. Utilizaremos el framework de pruebas JUnit para esto. Por ejemplo:

```
package com.ejercicio;

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

public class CalculadoraTest {

    @Test
    public void testSumar() {
        Calculadora calculadora = new Calculadora();
        int resultado = calculadora.sumar(2, 3);
        assertEquals(5, resultado);
    }
}
```





Compile y Test

Paso a paso: 🛠️

5- Compilación y ejecución de pruebas: Abre una terminal en el directorio raíz de tu proyecto y ejecuta los siguientes comandos:

Para compilar el código fuente, utiliza el siguiente comando:

`mvn compile`

Maven compilará el código fuente y colocará las clases compiladas en "`target/classes`".

Para ejecutar las pruebas unitarias, utiliza el siguiente comando:

`mvn test`

Maven ejecutará las pruebas JUnit y generará un informe con los resultados.



Compile y Test

Paso a paso: 

6- Verificar los resultados

Maven generará un informe en el directorio "**target/surefire-reports**" con los resultados de las pruebas. Asegúrate de que las pruebas se ejecuten con éxito y que los resultados sean como se esperaba.

Este ejercicio te permite practicar la fase "compile" de Maven al compilar el código fuente de tu proyecto y la fase "test" al ejecutar pruebas unitarias. Puedes expandir el ejercicio agregando más pruebas o funcionalidad a tu código y asegurándose de que las pruebas continúen funcionando correctamente.

○

¿Alguna consulta?

+



RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender el concepto de ciclo de vida de compilación y pruebas**
- ✓ **Aprender a utilizar los comandos “compile” y “test”**



#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 🙌 🙌 🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
 - a. *Lectura Modulo 6, Lección 1: páginas 6 - 7*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

¡Muchas Gracias!

Nos vemos en la próxima clase 🙌



Momento: ✚

Time-out!

🕒 5 min.

