Introducción

¡Hola! 👋 Te damos la bienvenida a la sección de Mockito.

Como desarrollador Java, es esencial escribir pruebas unitarias efectivas para garantizar la calidad y el correcto funcionamiento de nuestro código. En ocasiones, nuestras pruebas pueden depender de objetos externos o de comportamientos complejos, lo que dificulta la escritura de pruebas aisladas y confiables. Aquí es donde entra en juego Mockito.

¡Que comience el viaje! 🚀

Mockito

Mockito es un framework de simulación (mocking) en Java ampliamente utilizado en el desarrollo de pruebas unitarias. Proporciona una forma sencilla y flexible de crear objetos simulados, especificar su comportamiento y verificar las interacciones con ellos.

Instalar mockito

Ahora que usamos Maven solo necesitamos agregar la dependencia de mockito al pom.xml y él se encargará de descargar la librería para usarlo en nuestro proyecto.

```
...
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>5.4.0</version>
    <scope>test</scope>
</dependency>
```

```
...
```

Cómo se usa Mockito

- **Creación de objetos simulados (mocks):** En Mockito, puedes crear objetos simulados utilizando el método mock(). Estos objetos simulados actúan como sustitutos de los objetos reales y pueden ser interfaces, clases concretas o abstractas (las interfaces y clases abstractas las veremos en las clases siguientes).
- **Configuración de comportamientos:** Una vez que tienes un objeto simulado, puedes especificar su comportamiento utilizando métodos como when().thenReturn() o doAnswer(). Esto te permite definir qué debe hacer el objeto simulado cuando se llame a un método específico.
- **Verificación de interacciones:** Con Mockito, puedes verificar si se han realizado ciertas interacciones con los objetos simulados. Puedes comprobar si un método se ha llamado, cuántas veces se ha llamado y con qué argumentos se ha llamado utilizando el método verify().
- **Argumentos de captura:** Mockito también permite capturar los argumentos pasados a los métodos llamados en los objetos simulados. Esto es útil cuando necesitas verificar el valor de los argumentos pasados o hacer aserciones más complejas.
- **Especificación de comportamientos condicionales:** Puedes configurar diferentes comportamientos en función de ciertas condiciones usando métodos como when().thenReturn() o doAnswer(). Esto es útil para simular escenarios complejos y condicionales en las pruebas.
- **Simulación de dependencias:** Uno de los beneficios clave de Mockito es la simulación de dependencias externas. Puedes simular objetos y servicios externos para aislar la unidad que estás probando y centrarte en su comportamiento específico. Esto se logra reemplazando las dependencias reales con objetos simulados en las pruebas.
- **Creación de objetos espías:** En Mockito, un espía (spy) es una característica que permite rastrear y, opcionalmente, modificar el comportamiento de un objeto real durante las pruebas. A diferencia de los objetos simulados (mocks), los espías conservan la implementación

original del objeto y solo reemplazan o registran ciertos métodos según sea necesario. En los mocks si no reemplazamos un método y lo invocamos, devolverá el valor por defecto del tipo del dato que devuelve el método, 0 si devuelve un int por ejemplo.