**Unit Testing in Java**

**Prework**

* Definir qué es una prueba unitaria.
* Demostrar la importancia de escribir pruebas que garanticen el correcto funcionamiento de un método, por muy simple que sea.
* Analizar técnicas, recomendaciones y buenas prácticas para escribir pruebas unitarias.

Una prueba unitaria comprueba el correcto funcionamiento de las unidades individuales más pequeñas del software para asegurar que cada unidad funcione correcta y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido, entonces el estado final es válido también.

La idea es escribir casos de prueba para cada función no trivial o método en el módulo, de forma que cada caso sea independiente del resto. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

Hay un solo sujeto bajo prueba (Subject Under Test - SUT)

Las pruebas unitarias nos ayudan a:

* Documentar con detalle la funcionalidad de nuestro software
* Verificar el diseño de este
* Proporcionar retroalimentación rápida al desarrollador

Propiedades:

* Aisladas: las pruebas deberían regresar el mismo resultado sin importar el orden en que son ejecutadas.
* Rápidas: las pruebas deben de ejecutarse rápidamente.
* Inspiradoras: el que pasen las pruebas debería inspirar confianza.
* Fáciles de escribir: las pruebas deben ser baratas en relación al código que están probando
* Legibles: las pruebas deben ser comprensibles para el lector, exponiendo de forma clara la razón de ser de una prueba en particular.
* Orientadas al comportamiento: las pruebas deben ser sensibles a cambios en el comportamiento del código probado. Si el comportamiento cambia la prueba también debería hacerlo.
* Insensibles a la estructura: las pruebas no deben cambiar sus resultados si cambia la estructura del código bajo prueba.
* Automatizadas: las pruebas deberían ejecutarse sin intervención humana
* Específicas: si una prueba falla, la razón de la falla debería ser obvia
* Determinísticas: si no ha cambiado nada, el resultado de las pruebas no debería cambiar.
* Predictivas: si todas las pruebas pasan, el código debería estar listo para producción.

**Work**

* Definir que es una prueba unitaria
* Reconocer la importancia de escribir pruebas que garanticen el correcto funcionamiento de un método, por muy simple que sea
* Analizar técnicas, recomendaciones y buenas prácticas para escribir pruebas unitarias.

Reto 1: Bajas de entrevistadores

Se discutió en equipo:

* Mediante terminal permite eliminar entrevistadores existentes.
* El usuario ingresará el correo electrónico del entrevistador a eliminar

Reto 2: Definir modelo de datos para entrevistas

Se discutió en equipo:

* Generar un modelo de datos que permita al sistema tener las siguientes capacidades:
* Una entrevista se considera como el conjunto de los siguientes factores: un candidato, un entrevistador, un tipo de entrevista, una tecnología especifica y una disciplina
* Una entrevista debera contener una descripción, una fecha y tiempo específicos, duración y un estatus.
* De los candidatos se desea almacenar la misma información que de los entrevistadores.

Se creó un diagrama como el siguiente:

Table

Description automatically generated

**Postwork**

* Pasar de modelos de datos abstractos (representaciones gráficas) a implementaciones concretas (clases).
* Aplicar conceptos de reutilización de código.
* Añadir pruebas unitarias adicionales para probar su funcionalidad.
* Aplicar técnicas de diseño de pruebas.
* Explorar la herramienta JUNIT.

Configura tu proyecto e instala las dependencias necesarias

Paralelizar la ejecución de tus pruebas

test {

useJUnitPlatform()

systemProperty 'junit.jupiter.execution.parallel.enabled', 'false'

}