Pruebas unitarias y TDD

Sesión Conceptual 01

{desafío} latam_

- Objetivo de la sesión
- Activación de conceptos clave

- Conceptualización
- Ejercicios
- Quiz

- Cierre

Inicio

{desafío}
latam_

Objetivo

- Comprender la importancia de las pruebas unitarias para su uso en diversos códigos de Java.
- Comprender cómo cambia el flujo de desarrollo de un código al implementar pruebas unitarias.

Desarrollo

{desafío}
latam_

Pruebas Unitarias

{desafío}
latam_

Pruebas Unitarias

Son una pieza de código que verifica el correcto funcionamiento, del código productivo ya escrito.

Ventajas

- Se escriben pequeñas pruebas a la vez, obliga a que el código sea más modular.
- Las pruebas sirven de documentación.
- El código es más fácil de mantener y refactorizar.

Desventajas

- Las pruebas deben mantenerse.
- Inicialmente, ralentiza el desarrollo.

Herramientas de automatización de builds

Maven

Características

- Núcleo de la aplicación pom.xml
- Describe pom en XML
- Permite gestionar informes y documentación

Maven es una herramienta de gestión y comprensión de proyectos de software.

Añadir dependencia en a Maven

Se agregan las librerías adicionales en el archivo pom.xml en la raíz del proyecto.

Dentro de las etiquetas **dependencies**

```
<dependencies>

  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.4.2</version>
    <scope>test</scope>
  </dependency>

</dependencies>
```

{desafío} latam_

JUnit

JUnit

Librería de pruebas mayormente utilizada en el mundo Java

Características

- Posee una gran comunidad, adoptar la herramienta es sencillo por la buena documentación.
- Permite agregar anotaciones para hacer la prueba más fácil de leer.
- Fácil de usar y agregar al proyecto.

@Test

Se usa la anotación @Test para definir que el método será un método de prueba.

```
@Test
@DisplayName("Test testCrearPersona")
void testCrearPersona() {
    logger.info("info test crear persona");
    Persona juanito = new Persona("1234-1", "Juanito");
  String respuestaServicio =
servicioPersona.crearPersona(juanito);
    assertEquals("Creada", respuestaServicio);
}
```

Asserts

Las aserciones son métodos de utilidad para respaldar las condiciones en las pruebas; estos métodos son accesibles a través de la clase Assertions.

- **assertEquals** recibe dos parámetros, lo esperado y actual para afirmar si son iguales.
- **assertNotNull** recibe un parámetro y se encarga de validar que este no sea nulo.

```
import static org.junit.jupiter.api.Assertions.assertEquals;

assertEquals("OK", respuestaEsperadaQueDebeSerOK);
```

TestFixtures

Es una pieza de código donde escribes las tareas en la misma clase con el objetivo de reutilizar código y eliminar código duplicado. Se identifican con estas anotaciones.

@BeforeAll

@BeforeEach

@AfterAll

@AfterEach

```
package cl.desafiolatam.servicios;

import org.junit.jupiter.api.*;

@DisplayName("Tests Clase ServicioPersona")
public class ServicioPersonaTest {

    @BeforeAll
    static void setup() {
        System.out.println(“Inicializa algun objeto...”);
    }

   @AfterAll
    void tearDown() {
        System.out.println(“Destruye algun objeto...”);
    }

}
```

{desafío} latam_

Cierre

{desafío}
latam_

¿Existe algún concepto que no hayas comprendido?

Volvamos a revisar los conceptos que más te hayan costado antes de seguir adelante

Reflexionemos

{desafío} latam_

{desafío} latam_

Academia de talentos digitales