ACTIVIDAD
PRUEBAS UNITARIAS

Aprendizaje esperado

Implementar una suite de pruebas unitarias en lenguaje Java utilizando JUnit para asegurar el buen funcionamiento de una pieza de software.

Actividad

Contexto:

Tu equipo de desarrollo ha sido contratado por MercaExpress, una plataforma emergente de comercio electrónico, para asegurar la calidad del software que gestiona los pedidos realizados por los usuarios. El sistema está creciendo rápidamente, y la empresa ha tenido problemas en producción debido a errores no detectados en sus servicios básicos.
El desafío es aplicar una estrategia de pruebas unitarias estructurada utilizando JUnit, que permita cubrir de forma automatizada los componentes esenciales del sistema. Además, se espera que se construya la suite de pruebas siguiendo principios de Desarrollo Dirigido por Pruebas (TDD).

Objetivo de Aprendizaje
Implementar una suite de pruebas unitarias en Java utilizando JUnit, incluyendo fixtures, mocks y estructuras de test agrupadas, para validar el correcto funcionamiento de una pieza de software de baja a mediana complejidad, aplicando fundamentos de pruebas unitarias y buenas prácticas de diseño de test.

Requerimientos Específicos

1. Software base a testear
Desarrollar una pieza de software que represente parte del sistema de pedidos. Debe contener:

- Clase Producto con atributos: id, nombre, precio, stockDisponible.
- Clase Pedido con una lista de productos y métodos como:
    - agregarProducto(Producto producto, int cantidad)
    - calcularTotal()
    - confirmarPedido()

- Clase GestorPedidos con lógica de negocio:
    - realizarPedido(Pedido pedido)
    - cancelarPedido(Pedido pedido)
    - validarStock(Pedido pedido)

2. Creación de pruebas unitarias con JUnit

- Integrar JUnit 5 en el proyecto (puede ser en Eclipse o IntelliJ).
- Crear clases de prueba separadas para cada clase del sistema (ProductoTest, PedidoTest, GestorPedidosTest).
- Usar anotaciones estándar de JUnit (@Test, @BeforeEach, @AfterEach, etc.).
- Incluir casos de prueba con aserciones múltiples (assertEquals, assertThrows, etc.).

3. Uso de Fixtures

- Preparar objetos comunes (productos de prueba, pedidos predefinidos) usando @BeforeEach para evitar duplicación de código.

4. Uso de Mocks

- Simular el comportamiento de un repositorio o fuente de datos usando un mock (puede ser con Mockito o usando una clase auxiliar).
- Verificar interacciones entre clases (verify, when...thenReturn).

5. Creación de una Suite de Pruebas

- Organizar todos los casos en una suite de pruebas unificada usando @Suite y @SelectClasses.

6. Aplicación de TDD (Test Driven Development)

- Documentar al menos un componente de la solución que haya sido desarrollado siguiendo TDD: primero escribir el test que falla, luego el código que lo hace pasar, luego refactorizar.
- Incluir evidencia (capturas o comentarios en el código) del ciclo Red – Green – Refactor.

7. Reflexión sobre pruebas unitarias

- Incluir una reflexión escrita que responda:
  - ¿Qué ventajas observaron al implementar pruebas unitarias?
  - ¿Qué limitaciones detectaron?
  - ¿Qué beneficios ofreció aplicar TDD?
  - ¿Cómo podrían escalar la solución para incluir pruebas de integración?

Entregables

- **Proyecto Java completo**, que incluya:
  - Código funcional del sistema de pedidos.
  - Carpeta /test con todas las clases de prueba implementadas.
  - Fixture configurado correctamente.
  - Uso de mocks verificado en al menos una clase.
  - Suite de pruebas ejecutable.
- **Documento PDF** con:
  - Explicación de la estructura de pruebas creada.
  - Justificación del uso de fixtures y mocks.
  - Evidencia del uso de TDD.
  - Capturas de pantallas con resultados de test.
  - Reflexión final sobre la experiencia con JUnit.

Actividad 3: Desarrollo Dirigido por Pruebas (TDD) e Integración de JUnit en Eclipse

Objetivo: Aplicar el enfoque de **Desarrollo Dirigido por Pruebas (TDD)** y gestionar una suite de pruebas en JUnit con múltiples casos de prueba.

Instrucciones:

1. **Teoría**: Explicar el ciclo de TDD (Red, Green, Refactor), donde se escribe una prueba antes de escribir el código funcional. Describir cómo JUnit se puede integrar en el ciclo de TDD.
2. **Práctica**:
    - Crear un proyecto en Eclipse y realizar varias iteraciones de TDD para una clase OperacionesMatematicas que implemente operaciones como multiplicación y división.
    - Al principio, escribir una prueba unitaria que falle (fase roja).
    - Escribir el código necesario para que pase la prueba (fase verde).
    - Refactorizar el código para mejorar la calidad (fase de refactorización).
3. **Entregables**:
    - Proyecto Java con las clases necesarias (OperacionesMatematicas y su clase de prueba) que implementen las pruebas según el ciclo TDD, en formato .java.
    - Documento explicativo que describa el ciclo de TDD, cómo se aplicó en la actividad y las ventajas de este enfoque en el desarrollo de software.