Desarrollo de Software

Departamento de Lenguajes y Sistemas Informáticos

Grado en Informática

UNIVERSIDAD DE GRANADA

Práctica 3

Cristóbal Merino Sáez, Carlos Manuel Perales Gómez, Adrián Jaén Fuentes, Enrique Balmaseda García

Índice

1. Ejercicio 3: Pruebas en Flutter/Dart

1.1. Introducción

El objetivo de esta práctica ha sido desarrollar un sistema bancario en Dart/Flutter y validar su funcionamiento mediante pruebas unitarias y de componentes. El sistema simula operaciones básicas como la creación de cuentas, transacciones y gestión de usuarios.

Esta práctica se centra en aplicar buenas prácticas de testing para asegurar la calidad del software desde etapas tempranas del desarrollo. Se ha buscado diseñar clases bien encapsuladas, fácilmente testeables y con responsabilidades claras, lo que permite detectar errores de manera temprana y mejorar la mantenibilidad.

1.2. Diseño del Sistema

El sistema se estructura en torno a tres entidades principales:

- **Usuario**: Representa a una persona que puede poseer múltiples cuentas.
- **Cuenta**: Cada cuenta tiene un saldo y permite operaciones como depósitos y retiros.
- **Transacción**: Registra operaciones realizadas entre cuentas.

Se ha seguido una arquitectura modular, con separación entre la lógica de negocio y la presentación, lo que ha facilitado el diseño de pruebas automatizadas.

1.3. Pruebas Unitarias

Las pruebas unitarias validan el comportamiento individual de cada clase. Para ello, se ha utilizado el paquete `test` de Dart. Se han cubierto los siguientes aspectos:

Gestión de cuentas

- Creación de cuentas con saldo inicial correcto.
- Depósitos y retiros modifican el saldo adecuadamente.
- Control de errores al intentar retirar más dinero del disponible.

Transacciones

- Validación de transacciones entre cuentas.

- Comprobación del saldo final en origen y destino.
- Verificación de registros en el historial.

Gestión de usuarios

- Asociación correcta entre usuarios y cuentas.
- Comprobación de saldos totales de un usuario.

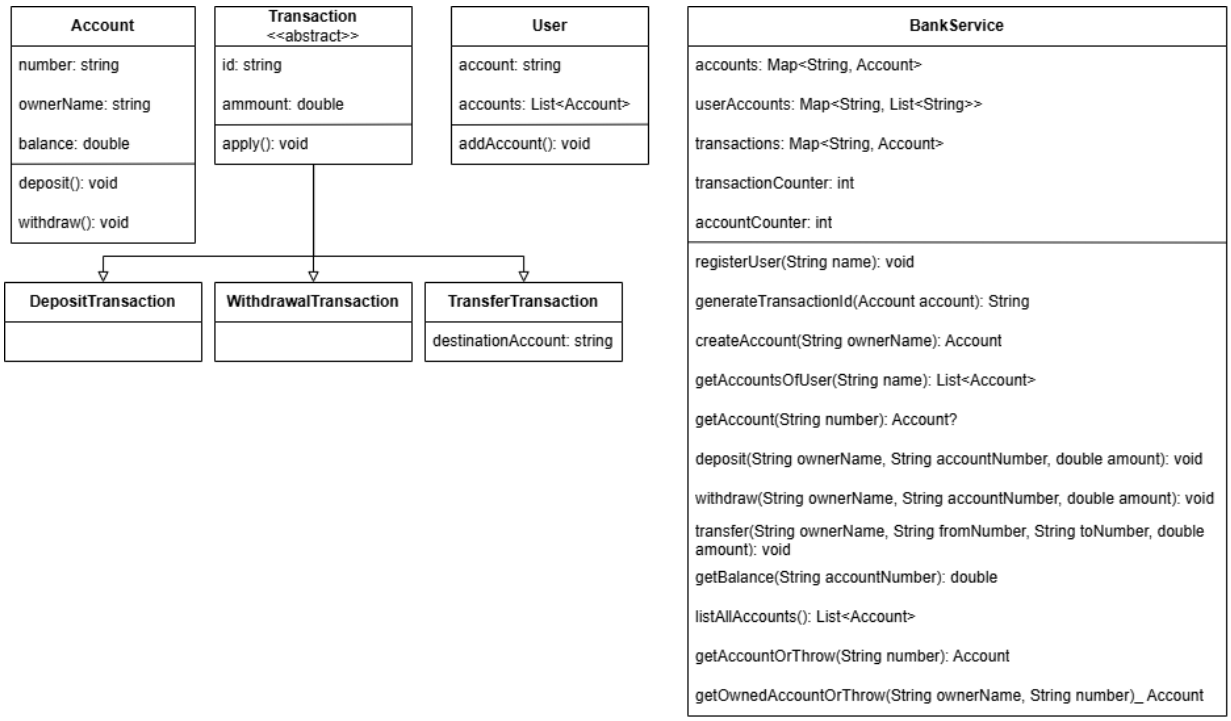

Figura 1: Diagrama UML del servicio bancario

Ejemplo de Prueba Unitaria

```
test('Retiro superior al saldo lanza excepción', () {
  final cuenta = Cuenta(saldoInicial: 100);
  expect(() => cuenta.retirar(200), throwsA(isA<Exception>()));
});
```

Esta prueba garantiza que no se puede retirar un monto superior al saldo actual de la cuenta, evitando inconsistencias en los datos financieros.

Resumen de pruebas implementadas

A continuación se detallan los principales tests desarrollados para validar el comportamiento del sistema bancario:

- El balance inicial de una cuenta debe ser cero.
- No se permite depositar cantidades negativas o cero.
- No se permite retirar cantidades negativas o cero.
- La lista inicial de cuentas está vacía.
- `DepositTransaction.apply` aumenta el saldo correctamente.
- `WithdrawalTransaction.apply` lanza `StateError` cuando no hay fondos suficientes.
- `TransferTransaction.apply` mueve fondos entre cuentas de forma correcta.
- `withdraw` lanza `StateError` cuando el saldo es insuficiente.
- `transfer` lanza `StateError` cuando los fondos son insuficientes.
- `deposit` aumenta el saldo de la cuenta.
- `transfer` mueve fondos correctamente.
- `txId` genera identificadores únicos.

Cada prueba se ha diseñado con el objetivo de cubrir un caso de uso relevante, validar los requisitos funcionales del sistema y prevenir errores lógicos o de integridad de datos.

1.4. Pruebas de Componentes

Además de las pruebas unitarias, se han implementado pruebas de componentes para verificar el funcionamiento de widgets específicos de la interfaz, utilizando el paquete `flutter_test`.

Validación de entradas Se ha verificado que los campos de texto acepten únicamente valores válidos (como montos numéricos positivos) y que los botones estén habilitados solo cuando el formulario esté correctamente completado.

Actualización de saldos Tras realizar una transacción desde la interfaz, se comprueba que los valores del saldo se actualicen correctamente en pantalla.

1.5. Cobertura de Código

Se ha utilizado la herramienta de cobertura de Dart para asegurar que las pruebas cubren la mayor parte del código. El informe generado mostró un **índice de cobertura superior al 90 %**, lo que indica una validación exhaustiva del sistema.

1.6. Conclusiones

Esta práctica ha permitido aplicar de manera efectiva principios de testing en un proyecto realista. La separación entre lógica y presentación ha facilitado el diseño de pruebas robustas, mientras que el uso de pruebas de componente ha permitido validar la experiencia de usuario.

Gracias a este enfoque, se ha conseguido un sistema fiable, mantenible y con una alta cobertura de código, preparado para crecer y adaptarse a nuevos requisitos funcionales.