

# Desarrollo de Software - Práctica 3



Fernando Cuesta Bueno  
Carlos Fernández Arrabal  
Antonio Manuel García Mesa  
18/05/2025

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Ejercicio 3.1</b>	<b>2</b>
2.1. Clase Transaction . . . . .	2
2.2. Clase DepositTransaction . . . . .	2
2.3. Clase WithdrawalTransaction . . . . .	2
2.4. Clase TransferTransaction . . . . .	3
2.5. Clase Account . . . . .	3
2.6. Clase BankService . . . . .	3
<b>3. Ejercicio 3.2</b>	<b>4</b>
<b>4. Interfaz gráfica</b>	<b>5</b>
4.1. Pantalla de inicio . . . . .	5
4.2. Detalles de cuenta . . . . .	6
4.3. Test de la interfaz gráfica . . . . .	7
<b>5. Repositorio</b>	<b>7</b>
<b>6. Cómo probar los test</b>	<b>7</b>

# 1. Introducción

Esta práctica tiene como objetivo consolidar los conocimientos adquiridos en clase sobre el diseño orientado a objetos, pruebas de software y desarrollo en Dart. Para ello, se propone implementar un sistema bancario. Para su desarrollo se parte de un diagrama UML proporcionado, que sirve como base para diseñar e implementar las clases que actúen como cuentas bancarias y transacciones.

A parte del desarrollo de esta funcionalidad, también se busca realizar un análisis de las pruebas necesarias para garantizar el buen funcionamiento del sistema. Para ello, se pide implementar un conjunto de pruebas unitarias organizadas en tres grupos: *Account*, *Transaction* y *BankService*. Estas pruebas permiten verificar que las operaciones básicas del sistema funcionan correctamente y detectar errores automáticamente.

Finalmente, se ha creado una interfaz gráfica en Flutter que permite interactuar con el sistema bancario. Esta interfaz proporciona una forma sencilla y amigable de realizar operaciones bancarias, como crear cuentas, realizar depósitos, retiradas y transferencias. La interfaz está diseñada para ser intuitiva y fácil de usar, lo que facilita la interacción del usuario con el sistema.

## 2. Ejercicio 3.1

Para la implementación del sistema bancario, se han desarrollado las clases especificadas en el enunciado de la práctica. A continuación, se detalla la funcionalidad de cada una de ellas:

### 2.1. Clase Transaction

Esta clase es abstracta y define el contrato común para todas las operaciones sobre cuentas bancarias. Contiene los atributos `transactionId` (identificador para cada cuenta) y `amount` (importe de cada operación). Además, incluye la definición del método abstracto `apply(Account account)`, que debe ser implementado por cada subclase para definir la lógica de cada tipo de operación.

### 2.2. Clase DepositTransaction

Es una subclase concreta de la clase `Transaction`. Su objetivo es incrementar el saldo de la cuenta destino al aplicar una transacción. Implementa el método `apply(Account account)` realizando una llamada al método `deposit` de la cuenta.

### 2.3. Clase WithdrawalTransaction

Es otra subclase concreta de la clase `Transaction`. Su objetivo es retirar una cantidad de dinero de la cuenta, siempre que haya un saldo suficiente. Implementa el método `apply(Account account)`, realizando una llamada al método `withdraw`, donde se valida que el importe no supere el saldo total disponible.

## 2.4. Clase **TransferTransaction**

Es otra subclase concreta de la clase **Transaction**. Permite mover dinero entre dos cuentas. Implementa el método `apply(Account account)`, donde se realiza una retirada de la cuenta origen seguido de un depósito en la cuenta destino. Para realizar esto, se hace una llamada al método `deposit` y `withdraw`.

## 2.5. Clase **Account**

Es una clase que representa una cuenta bancaria. Contiene un identificador único (`accountId`) y un atributo para el saldo `_saldoCuenta`. Se implementan los métodos `deposit(double amount)` y `withdraw(double amount)`. El primero se encarga de depositar dinero en una cuenta y aumentar su saldo. El segundo método se encarga de retirar dinero de la cuenta, validando que la cantidad a retirar no sea mayor que el saldo de la cuenta.

## 2.6. Clase **BankService**

Esta clase actúa como fachada del sistema. Su principal función es simplificar el acceso a las operaciones bancarias, como la creación de cuentas, la realización de depósitos, retiradas y transferencias. A parte de esto, también se encarga de:

- Generar identificadores únicos para las cuentas y transacciones.
- Almacenar las cuentas activas en un mapa.
- Delegar la ejecución de transacciones según el tipo solicitado (depósito, retirada o transferencia).
- Implementar métodos públicos para listar cuentas, realizar operaciones y consultar transacciones de una cuenta.

En cuanto a los métodos implementados, tenemos los siguientes:

- `Account createAccount()`: Genera un nuevo número de cuenta único, crea una instancia de tipo `Account` con ese número y la guarda en un map. A modo de resumen, es el método encargado de registrar nuevas cuentas bancarias.
- `void deposit(String accountNumber, double amount)`: Realiza un depósito de dinero en la cuenta indicada. El método obtiene la cuenta, crea una instancia de la clase `DepositTransaction`, llama al método `apply` y realiza la transacción y la guarda en el historial de transacciones.
- `void withdraw(String accountNumber, double amount)`: Realiza una retirada de dinero sobre una cuenta bancaria. Al igual que el método anterior, se crea una instancia de la clase `WithdrawalTransaction`, llama al método `apply` y realiza la transacción si hay saldo suficiente y la guarda en el historial de transacciones.
- `void transfer(String fromAccount, String toAccount, double amount)`: Permite transferir dinero de una cuenta a otra. Crea una instancia de `TransferTransaction`, se obtienen los IDs de ambas cuentas y se aplica la transacción sobre la cuenta origen y la almacena en el historial.

- `Account _getAccount(String accountNumber)` (privado): Método auxiliar para obtener una cuenta a partir de su identificador. Si no existe, lanza un error. Es utilizado para validar que las operaciones se apliquen sobre cuentas existentes.
- `List<Account>listAccounts()`: Devuelve una lista de todas las cuentas registradas en el sistema bancario.
- `List<Transaction>listTransactions()`: Devuelve una lista de todas las transacciones realizadas en el sistema bancario.

### 3. Ejercicio 3.2

Respecto a los test, hemos implementado todos los test que se piden para la práctica, para ello hemos realizado tres groups distintos; uno `Account`, otro `Transaction` y otro `BankService` y en cada uno hemos implementado los test su respectivo grupo:

- Group `Account`.
  - Test para comprobar que el balance inicial de una cuenta debe ser cero. Para ello se comprueba que el atributo `saldoCuenta` sea cero.
  - Test para comprobar que no se puedan depositar cantidades negativas o cero. Para ello ejecuto el método `deposit` con una cantidad 0 y otro con una cantidad -10 para comprobar que no puede ser negativa.
  - Test para comprobar que no se puedan retirar cantidades negativas o cero. Para ello ejecuto el método `withdraw` con una cantidad 0 y otro con una cantidad -5 para comprobar que no puede ser negativa.
- Group `Transaction`.
  - Test para comprobar si `DepositTransaction.apply` aumenta el saldo correctamente. Para ello deposito en una cuenta 100 de saldo y compruebo que el `saldoCuenta` sea de 100.
  - Test para comprobar si `WithdrawalTransaction.apply` lanza `StateError` cuando no hay fondos suficientes. Para ello creo una cuenta con saldo 0 e intento retirarle 50.
  - Test para comprobar si `TransferTransaction.apply` mueve fondos entre cuentas de forma correcta. Para ello creo una cuenta en la que deposito 200 y luego creo otra con saldo 0 que va a ser la cuenta en la que transfiera el dinero. Una vez realizada la transacción comprueba que el saldo de la primera cuenta sea 50 y el de la segunda 150.
- Group `BankService`.
  - Test para comprobar si la lista inicial de cuentas está vacía. Para ello creo un `BankService` y compruebo si el método `listAccounts()` es true.

- Test para comprobar si el método deposit aumenta el saldo de la cuenta. Para ello creo un BankService y una cuenta. Posteriormente ingreso en la cuenta a través del BankService 100 de saldo de la cuenta y compruebo si el saldo de la cuenta es de 100.
- Test para comprobar si el método withdraw lanza StateError cuando el saldo es insuficiente. Para ello creo un BankService y una cuenta vacía e intento retirar de la cuenta 50 de saldo.
- Test para comprobar si el método transfer mueve fondos correctamente. Para ello creo un BankService y dos accounts vacías. Una vez está todo creado deposito en la primera cuenta 200 de saldo y le transfiero 100 a la segunda. Finalmente compruebo que ambas cuentas tengan 100 de saldo.
- Test para comprobar que el método transfer lanza StateError cuando los fondos son insuficientes. Para ello creo un BankService y dos accounts vacías e intento transferir 100 de saldo de una cuenta a otra y lanzo el throwError.
- Test para comprobar si txId genera identificadores únicos. Para ello creo un BankService y una account y realizo en la cuenta un depósito y una retirada usando los métodos deposit y withdraw de BankService. Una vez he realizado las dos transacciones, obtengo la lista de transacciones, verifico que hay dos transacciones y compruebo que las dos id son distintas.

## 4. Interfaz gráfica

La interfaz gráfica ha sido desarrollada utilizando Flutter, un framework de desarrollo de aplicaciones móviles. La interfaz permite a los usuarios interactuar con el sistema bancario de manera sencilla y amigable. A continuación, se describen las principales características de la interfaz:

### 4.1. Pantalla de inicio

La pantalla de inicio muestra las cuentas bancarias registradas en el sistema. Cada cuenta se presenta con su número de cuenta y saldo actual. Además, se proporciona un botón para crear una nueva cuenta.

En la parte superior de la pantalla, se encuentra un botón para alternar entre el modo claro y oscuro de la aplicación. Esto permite a los usuarios personalizar la apariencia de la interfaz según sus preferencias.

En las figuras 1 y 2 se muestran ejemplos de la pantalla de inicio en modo claro y oscuro, respectivamente.

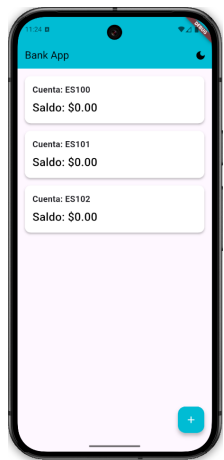


Figura 1: Pantalla de inicio (modo claro)

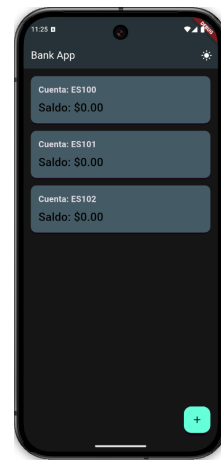


Figura 2: Pantalla de inicio (modo oscuro)

## 4.2. Detalles de cuenta

Al seleccionar una cuenta de la lista, se accede a la pantalla de detalles de la cuenta. En esta pantalla, se muestra información detallada sobre la cuenta seleccionada, incluyendo su número de cuenta, el saldo actual y un historial de transacciones realizadas.

Además, se proporcionan botones para realizar operaciones bancarias, como depósitos, retiradas y transferencias. Los usuarios pueden ingresar la cantidad deseada y confirmar la operación.

En la figura 3 se muestran los detalles de una cuenta y en la figura 4 se puede ver como realizar un depósito.

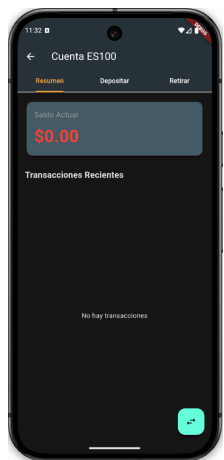


Figura 3: Detalles de cuenta

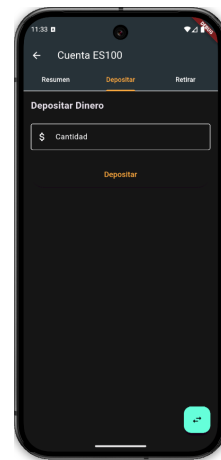


Figura 4: Detalles de cuenta (depósito)

Por otro lado, en la figura 5 se puede ver como realizar un retiro y en la figura 6 se puede ver como realizar una transferencia. En este menu, tendremos una lista desplegable donde podremos seleccionar la cuenta a la que queremos transferir el dinero.

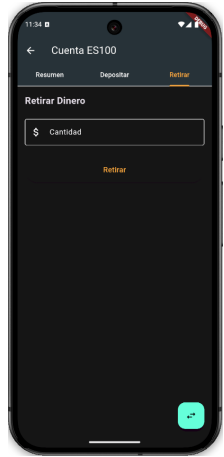


Figura 5: Detalles de cuenta (retiro)

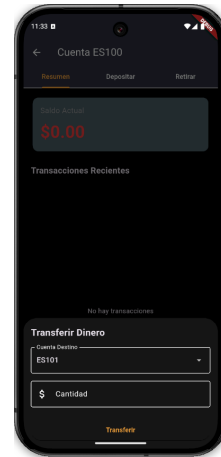


Figura 6: Detalles de cuenta (transferencia)

### 4.3. Test de la interfaz gráfica

Para garantizar el correcto funcionamiento de las pantallas y los widgets de la interfaz gráfica, se han implementado pruebas unitarias utilizando el paquete `flutter_test`. Estas pruebas verifican que los widgets se construyan correctamente y que las interacciones del usuario generen los resultados esperados.

En el archivo `account_card_test.dart` se encuentran las pruebas relacionadas con la tarjeta de cuenta. En particular, una de las pruebas principales verifica que el widget `AccountCard` muestre correctamente la información de la cuenta. Esta prueba crea una cuenta de prueba con ID 'TEST123' y un saldo de 150.0. Después, comprueba que el widget muestre correctamente el ID de la cuenta como "Cuenta: TEST123" y el saldo como "Saldo: \$150.00". Además, se comprueba que al pulsar sobre la tarjeta se active correctamente el callback `onTap`, lo que permitiría la navegación a la pantalla de detalles de la cuenta en la aplicación real.

## 5. Repositorio

El código fuente de la práctica, incluyendo la implementación del sistema bancario y la interfaz gráfica, se encuentra disponible en el siguiente repositorio de GitHub:

[https://github.com/YTTREW/DS\\_FAC/tree/main](https://github.com/YTTREW/DS_FAC/tree/main)

## 6. Cómo probar los test

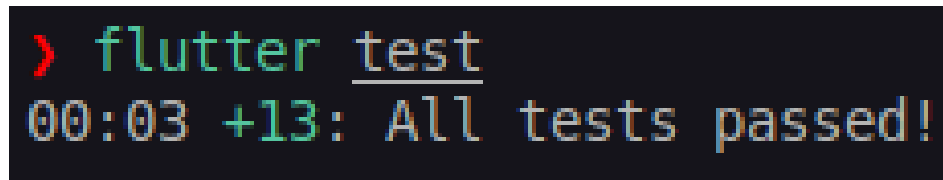
Para probar los test de la práctica, basta con clonar el repositorio y abrirlo en un editor de código compatible con Dart y Flutter, como Visual Studio Code o Android Studio. En la terminal, navega hasta la carpeta del proyecto y ejecuta el siguiente comando:

```
flutter test
```

El resultado debe mostrar un resumen de los test ejecutados, indicando si han pasado o fallado. Si todos los test pasan, significa que el sistema bancario y la interfaz gráfica



funcionan correctamente. En la imagen 7 se puede ver un ejemplo de la salida de los test.

A terminal window with a dark background. The first line shows a red prompt character followed by the command 'flutter test' in green, with the 'test' part underlined. The second line shows the output '00:03 +13: All tests passed!' in green, where '00:03' is the duration, '+13' is the number of tests passed, and 'All tests passed!' is the status message.

```
> flutter test  
00:03 +13: All tests passed!
```

Figura 7: Resultado de los test