Ejercicio 13

Tu tarea es desarrollar una aplicación informática utilizando la técnica TDD para gestionar una cuenta bancaria. La aplicación debe permitir a los usuarios abrir una cuenta, realizar depósitos, hacer retiros y transferir fondos entre cuentas. A continuación, se detallan las etapas de desarrollo utilizando TDD:

Etapa 1: Especificación y prueba inicial

1. **Especifica los requisitos básicos del sistema y las funcionalidades clave, como la apertura de cuenta, depósito de fondos, retiro de fondos y transferencia de fondos.**
    - La aplicación debe permitir a los usuarios abrir una nueva cuenta bancaria, solicitando detalles como el nombre del titular, el tipo de cuenta (por ejemplo, ahorros o corriente) y estableciendo un saldo inicial, que podría ser cero.

    - Los usuarios deben poder depositar dinero en su cuenta. Para ello, se debe especificar la cantidad a depositar y actualizar el saldo de la cuenta en consecuencia. La aplicación debe asegurarse de que la cantidad a depositar sea un número positivo.

    - Los usuarios deben poder retirar dinero de su cuenta. Esto requiere especificar la cantidad a retirar y actualizar el saldo de la cuenta en consecuencia. La aplicación debe verificar que la cantidad a retirar sea un número positivo y que haya suficientes fondos en la cuenta para cubrir el retiro.

    - Los usuarios deben poder transferir dinero de una cuenta a otra. Esto implica especificar la cuenta de origen, la cuenta de destino y la cantidad a transferir. La aplicación debe comprobar que ambas cuentas existan, que la cuenta de origen tenga fondos suficientes para la transferencia y que la cantidad a transferir sea un número positivo. Luego, la aplicación debe debitar la cantidad de la cuenta de origen y acreditarla en la cuenta de destino.

2. **Escribe una prueba inicial que verifique si el sistema puede crear una instancia de una cuenta bancaria y obtener su saldo inicial.**
    cuenta = nueva Cuenta() // Crea una nueva instancia de Cuenta

    confirmar_que(cuenta.saldo == 0) // Verificar que el saldo inicial es 0

Etapa 2: Desarrollo de las funcionalidades básicas

3. **Implementa la funcionalidad para abrir una cuenta bancaria, asegurándote de que se cumplan los requisitos especificados. Ejecuta la prueba y verifica que pase correctamente.**

```
Clase Cuenta {
  // Constructor.
  Constructor(nombreTitular, tipoCuenta) {
    this.nombreTitular = nombreTitular
    this.tipoCuenta = tipoCuenta
    this.saldo = 0
  }
  // Método para obtener el saldo de la cuenta.
  obtenerSaldo() {
    return this.saldo
  }
}
// Verificar que se puede abrir una cuenta bancaria.
prueba "abrir cuenta bancaria" {
  // Crear una nueva cuenta
  cuenta = nueva Cuenta("Ernesto Moretti", "Corriente")

  // Verificar que el saldo inicial es 0
  afirmar_que(cuenta.obtenerSaldo() == 0)
}
```

4. **Implementa la funcionalidad para realizar depósitos en una cuenta bancaria. Ejecuta las pruebas y verifica que pasen correctamente.**

```
// Método para depositar fondos en la cuenta.
  depositarFondos(cantidad) {
    // Verificar que la cantidad es un número positivo.
    if (cantidad > 0) {
      this.saldo += cantidad
    } else {
      lanzar nueva Excepcion("La cantidad a depositar debe ser un número positivo.")
    }
  }
}
// Verificar el proceso de depositar fondos en una cuenta bancaria
prueba "depositar fondos en cuenta bancaria" {
  // Crear una nueva cuenta
  cuenta = nueva Cuenta("Ernesto Moretti", "Corriente")

  // Depositar fondos en la cuenta
  cuenta.depositarFondos(15000)

  // Verificar que el saldo de la cuenta es ahora 15000
  afirmar_que(cuenta.obtenerSaldo() == 15000)
}
```

5. **Implementa la funcionalidad para realizar retiros de una cuenta bancaria. Ejecuta las pruebas y verifica que pasen correctamente.**

```
retirarFondos(cantidad) {
    // Verificar que la cantidad es un número positivo y que hay suficientes fondos
    if (cantidad > 0 && this.saldo >= cantidad) {
      this.saldo -= cantidad
    } else {
      lanzar nueva Excepcion("La cantidad a retirar debe ser un número positivo y no puede exceder el saldo de la cuenta")
    }
  }
}
// Verificar que se puede retirar fondos de una cuenta bancaria
prueba "retirar fondos de cuenta bancaria" {
  // Crear una nueva cuenta
  cuenta = nueva Cuenta("Ernesto Moretti", "Corriente")

  // Depositar fondos en la cuenta
  cuenta.depositarFondos(15000)

  // Retirar fondos de la cuenta
  cuenta.retirarFondos(7500)

  // Verificar que el saldo de la cuenta es ahora 7500
  afirmar_que(cuenta.obtenerSaldo() == 7500)
}
```

6. **Implementa la funcionalidad para transferir fondos entre cuentas bancarias. Ejecuta las pruebas y verifica que pasen correctamente.**

```
// Método para transferir fondos a otra cuenta
  transferirFondos(cuentaDestino, cantidad) {
    // Verificar que la cantidad es un número positivo y que hay suficientes fondos
    if (cantidad > 0 && this.saldo >= cantidad) {
      this.saldo -= cantidad
      cuentaDestino.depositarFondos(cantidad)
    } else {
      lanzar nueva Excepcion("La cantidad a transferir debe ser un número positivo y no puede exceder el saldo de la cuenta")
    }
  }
}

// Verificar que se pueden transferir fondos entre cuentas bancarias
prueba "transferir fondos entre cuentas bancarias" {
  // Crear dos nuevas cuentas
  cuenta1 = nueva Cuenta("Ernesto Moretti", "Corriente")
  cuenta2 = nueva Cuenta("Bianca Moretti", "Ahorro")

  // Depositar fondos en la primera cuenta
  cuenta1.depositarFondos(100)

  // Transferir fondos de la primera a la segunda cuenta
  cuenta1.transferirFondos(cuenta2, 50)

  // Verificar que el saldo de la primera cuenta es ahora 50
  afirmar_que(cuenta1.obtenerSaldo() == 50)

  // Verificar que el saldo de la segunda cuenta es ahora 50
  afirmar_que(cuenta2.obtenerSaldo() == 50)
}
```

Etapa 3: Pruebas adicionales y mejoras

7. **Escribe pruebas adicionales para cubrir casos de prueba específicos, como intentar retirar más dinero del disponible en una cuenta o transferir fondos a una cuenta inexistente.**

```
// Verificar que no se puede retirar más dinero del disponible en una cuenta
prueba "retirar más dinero del disponible en una cuenta" {
  // Crear una nueva cuenta
  cuenta = nueva Cuenta("Ernesto Moretti", "Corriente")

  // Depositar fondos en la cuenta
  cuenta.depositarFondos(100)

  // Intentar retirar más dinero del disponible en la cuenta
  try {
    cuenta.retirarFondos(200)
    afirmar_falso("Debería haber lanzado una excepción")
  } catch (excepcion) {
    afirmar_verdadero(excepcion.mensaje == "La cantidad a retirar debe ser un número positivo y no puede exceder el saldo de la cuenta")
  }
}

// Prueba para verificar que no se puede transferir fondos a una cuenta inexistente
prueba "transferir fondos a una cuenta inexistente" {
  // Crear una nueva cuenta
  cuenta = nueva CuentaBancaria("Juan Perez", "ahorros")

  // Depositar fondos en la cuenta
  cuenta.depositarFondos(100)

  // Intentar transferir fondos a una cuenta inexistente
  try {
    cuenta.transferirFondos(null, 50)
    afirmar_falso("Debería haber lanzado una excepción")
  } catch (excepcion) {
    afirmar_verdadero(excepcion.mensaje == "La cuenta destino no puede ser nula")
  }
}
```

8. **Ejecuta todas las pruebas y verifica que pasen correctamente.**
   Todo correcto.

9. **Refactoriza tu código si es necesario para mejorar su estructura, legibilidad y eficiencia.**

```
Clase Cuenta {
  // Constructor
  Constructor(nombreTitular, tipoCuenta) {
    this.nombreTitular = nombreTitular
    this.tipoCuenta = tipoCuenta
    this.saldo = 0
  }
  // Método para obtener el saldo de la cuenta
  obtenerSaldo() {
    return this.saldo
  }
  // Método para depositar fondos en la cuenta
  depositarFondos(cantidad) {
    verificarCantidadPositiva(cantidad)
    this.saldo += cantidad
  }
  // Método para retirar fondos de la cuenta
  retirarFondos(cantidad) {
    verificarCantidadPositiva(cantidad)
    verificarSaldoSuficiente(cantidad)
    this.saldo -= cantidad
  }
  // Método para transferir fondos a otra cuenta
  transferirFondos(cuentaDestino, cantidad) {
    verificarCuentaExistente(cuentaDestino)
    verificarCantidadPositiva(cantidad)
    verificarSaldoSuficiente(cantidad)
    this.saldo -= cantidad
    cuentaDestino.depositarFondos(cantidad)
  }
  // Método para verificar que la cantidad es un número positivo
  verificarCantidadPositiva(cantidad) {
    if (cantidad <= 0) { lanzar nueva Excepcion("La cantidad debe ser un número positivo") }
  }
  // Método para verificar que hay suficientes fondos
  verificarSaldoSuficiente(cantidad) {
    if (this.saldo < cantidad) { lanzar nueva Excepcion("No hay suficientes fondos en la
cuenta") }
  }
  // Método para verificar que la cuenta existe
  verificarCuentaExistente(cuenta) {
    if (cuenta == null) { lanzar nueva Excepcion("La cuenta destino no puede ser nula") }
  }
}
```

10. **Ejecuta todas las pruebas nuevamente para asegurarte de que el código refactorizado no haya introducido errores.**

    Todo correcto.

Etapa 4: Cobertura completa de pruebas

11. **Asegúrate de que todas las funcionalidades del sistema estén cubiertas por pruebas automatizadas.**
12. **Examina los casos límite y situaciones excepcionales para garantizar que el sistema se comporte correctamente en todos los escenarios.**

```
// Verificar que no se puede depositar una cantidad negativa
prueba "depositar cantidad negativa" {
  // Crear una nueva cuenta
  cuenta = nueva Cuenta("Ernesto Moretti", "Corriente")
  // Intentar depositar una cantidad negativa
  try {
    cuenta.depositarFondos(-100)
    afirmar_falso("Debería haber lanzado una excepción")
  } catch (excepcion) {
    afirmar_verdadero(excepcion.mensaje == "La cantidad a depositar debe ser un número positivo")
  }
}
// Verificar que no se puede retirar una cantidad negativa
prueba "retirar cantidad negativa" {
  // Crear una nueva cuenta
  cuenta = nueva Cuenta("Ernesto Moretti", "Corriente")
  // Intentar retirar una cantidad negativa
  try {
    cuenta.retirarFondos(-50)
    afirmar_falso("Debería haber lanzado una excepción")
  } catch (excepcion) {
    afirmar_verdadero(excepcion.mensaje == "La cantidad a retirar debe ser un número positivo y no puede exceder el saldo de la cuenta")
  }
}
// Verificar que no se puede transferir una cantidad negativa
prueba "transferir cantidad negativa" {
  // Crear dos nuevas cuentas
  cuenta1 = nueva Cuenta("Ernesto Moretti", "Corriente")
  cuenta2 = nueva Cuenta("Bianca Moretti", "Ahorro")
  // Depositar fondos en la primera cuenta
  cuenta1.depositarFondos(100)
  // Intentar transferir una cantidad negativa
  try {
    cuenta1.transferirFondos(cuenta2, -50)
    afirmar_falso("Debería haber lanzado una excepción")
  } catch (excepcion) {
    afirmar_verdadero(excepcion.mensaje == "La cantidad a transferir debe ser un número positivo y no puede exceder el saldo de la cuenta")
  }
}
```

13. **Ejecuta todas las pruebas y verifica que pasen correctamente.**
    Todo correcto.

Recuerda seguir el enfoque TDD, donde agregarás una prueba antes de implementar cada funcionalidad y verificarás que todas las pruebas pasen antes de pasar a la siguiente etapa. Esto te ayudará a desarrollar una aplicación confiable, mantenible y que cumpla con los requisitos establecidos.