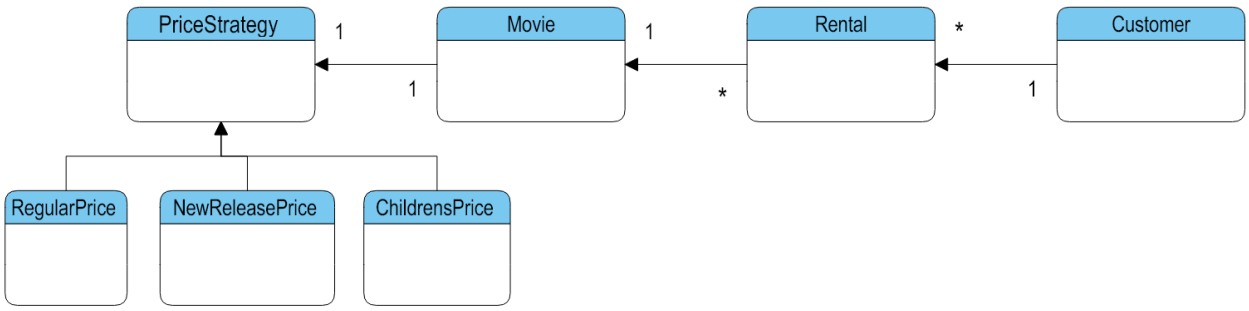
Práctica 2 Testing
Alquiler de Películas

ACG
BRIAN VALIENTE RÓDENAS

Tabla de contenido

0. Diagrama



1. Creación de una batería de pruebas con JUnit

Antes de modificar el código, es fundamental asegurarse de que la funcionalidad actual no cambie. Para ello, creamos una clase de prueba (por ejemplo, RentalTest.java) en la que comprobamos:

- **Películas:** Que se crean correctamente (título y código de precio).
- **Clientes:** Que se asigna el nombre correctamente.
- **Alquileres:** Que se asocian correctamente las películas y se registran los días.
- **Método statement():** Que el string devuelto coincide con el esperado.
- **Método getCharge():** Que devuelve el valor correcto según las reglas del negocio.
- **Método getFrequentRenterPoint():** Que se calcula correctamente el número de puntos.

Archivo **ReltalTest.java** en el paquete **PrincipalTest**. Este conjunto de pruebas cubre la creación de objetos y la validación de los métodos principales, lo que te ayudará a verificar que los cambios de refactorización no introducen errores.

2. Refactorización del código

El enunciado propone refactorizar el método statement() para extraer la lógica de cálculo en métodos específicos y trasladar responsabilidades a la clase adecuada. De esta forma se podría aprovechar más código y favorecer el factor de calidad de SW visto en clase, es decir, la Mantenibilidad.

Extraer el cálculo del importe de cada alquiler y los puntos

Extraemos el código que calcula el importe (originalmente se calculaba todo en el método statement) y lo llamamos getCharge(). Este método lo movemos de la clase Customer a la clase Rental, ya que el cálculo depende de los datos de un alquiler (película y días). Hacemos

lo mismo con un nuevo método llamado getFrequentRenterPoint() para calcular los puntos y lo movemos a Rental. Con esta modificación, la responsabilidad del cálculo se centraliza en la clase Rental, lo que mejora la cohesión y reduce el acoplamiento.

Este cambio permite que el método statement() sea más claro y se centre en la composición del resultado, delegando el cálculo en las clases correspondientes.

Extracción de métodos para calcular totales

Actualmente, el método statement() se encarga de calcular tanto el total del importe a pagar como el total de puntos de fidelidad. Podemos extraer esta lógica en métodos separados, lo que hará que el código sea más modular y fácil de entender. Una posible implementación en la clase Customer sería:

- **getTotalCharge()**: Recorre la lista de alquileres y suma el importe de cada uno.
- **getTotalFrequentRenterPoints()**: Recorre los alquileres y acumula los puntos obtenidos.

Esta separación permite que cada método se encargue de una responsabilidad única, siguiendo el principio de responsabilidad única.

Uso de polimorfismo en lugar de estructuras condicionales

El método getCharge() de la clase Rental utiliza un switch para determinar el precio basado en el tipo de película. Esto se puede mejorar utilizando polimorfismo. La idea es crear una jerarquía de clases en la que cada subclase de Movie implemente su propia lógica para calcular el precio.

Clase abstracta PriceStrategy, con un método getCharge(int daysRented).

Subclases concretas para cada tipo de película: RegularPrice, NewReleasePrice y ChildrensPrice que implementen la interfaz y contengan la lógica específica para cada tipo. En lugar de almacenar simplemente el tipo de precio, la clase Movie tendría una instancia de PriceStrategy y el método getCharge() de Rental llamaría a esa estrategia.

Este enfoque elimina la necesidad de un switch y facilita la extensión o modificación de la lógica de precios, ya que solo tendrás que modificar o agregar nuevas implementaciones de PriceStrategy sin tocar la estructura de clases existente.

Finalmente, el código queda tal que así:

CUSTOMER

```
//Customer.java

package Principal;

import java.util.ArrayList;
import java.util.List;
```

```
public class Customer {

    private final String name;
    private final List<Rental> rentals = new ArrayList<>();

    public Customer(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void addRental(Rental rental) {
        rentals.add(rental);
    }

    public double getTotalCharge() {
        double total = 0;
        for (Rental rental : rentals) {
            total += rental.getCharge();
        }
        return total;
    }

    public int getTotalFrequentRenterPoints() {
        int total = 0;
        for (Rental rental : rentals) {
            total += rental.getFrequentRenterPoint();
        }
        return total;
    }

    public String statement() {
        StringBuilder result = new StringBuilder("Rental record for " +
name + "\n");
        for (Rental rental : rentals) {
            result.append("\t")
                  .append(rental.getMovie().getTitle())
                  .append("\t")
                  .append(rental.getCharge())
                  .append("\n");
        }
        result.append("Amount owed is
").append(getTotalCharge()).append("\n");
        result.append("You earned
").append(getTotalFrequentRenterPoints()).append(" frequent renter
points.");
```

```
        return result.toString();
    }

}
```

RENTAL

```java
//Rental.java

package Principal;

public class Rental {
    private final Movie movie;
    private final int daysRented;

    public Rental(Movie movie, int daysRented) {
        this.movie = movie;
        this.daysRented = daysRented;
    }

    public Movie getMovie() {
        return movie;
    }

    public int getDaysRented() {
        return daysRented;
    }

    public double getCharge() {
        return movie.getCharge(daysRented);
    }

    public int getFrequentRenterPoint() {
        return movie.getFrequentRenterPoint(daysRented);
    }
}
```

MOVIE

```java
//Movie.java

package Principal;

public class Movie {
    private final String title;
    private final PriceStrategy priceStrategy;

    public Movie(String title, PriceStrategy priceStrategy) {
        this.title = title;
        this.priceStrategy = priceStrategy;
    }

    public String getTitle() {
        return title;
    }

    public double getCharge(int daysRented) {
        return priceStrategy.getCharge(daysRented);
    }

    public int getFrequentRenterPoint(int daysRented) {
        if (priceStrategy instanceof NewReleasePrice && daysRented > 1)
            return 2;
        return 1;
    }

}
```

TESTING

```
package PrincipalTest;

import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import Principal.*;

public class RentalTest {
    private Movie regularMovie;
    private Movie newReleaseMovie;
    private Movie childrensMovie;
    private Rental regularRental;
    private Rental newReleaseRental;
    private Rental childrensRental;
    private Customer customer;

    @Before
    public void setUp() {
         regularMovie = new Movie("Regular Movie", new RegularPrice());
         newReleaseMovie = new Movie("New Release Movie", new
NewReleasePrice());
         childrensMovie = new Movie("Childrens Movie", new
ChildrensPrice());

         // Para una película Regular: 2 + (días - 2) * 1.5. Con 3 días → 2
+ 1.5 = 3.5
         regularRental = new Rental(regularMovie, 3);
         // Para una película NewRelease: días * 3. Con 2 días → 6.0
         newReleaseRental = new Rental(newReleaseMovie, 2);
         // Para una película Childrens: según el código, con 4 días → 1.5
(ya que se sobreescribe)
         childrensRental = new Rental(childrensMovie, 4);

         customer = new Customer("John Doe");
    }

    @Test
    public void testMovies() {
         assertEquals("Regular Movie", regularMovie.getTitle());
         assertEquals(3.5, regularMovie.getCharge(3), 0.001);
         assertEquals("New Release Movie", newReleaseMovie.getTitle());
         assertEquals(6.0, newReleaseMovie.getCharge(2), 0.001);
         assertEquals("Childrens Movie", childrensMovie.getTitle());
         assertEquals(1.5, childrensMovie.getCharge(4), 0.001);
    }
```

```java
    @Test
    public void testCustomers() {
         assertEquals("John Doe", customer.getName());
    }

    @Test
    public void testRentals() {
         assertEquals(regularMovie, regularRental.getMovie());
         assertEquals(3, regularRental.getDaysRented());
    }

    @Test
    public void testGetCharge() {
         // Comprobamos los cálculos:
         assertEquals(3.5, regularRental.getCharge(), 0.001); // el 0.001
es el margen de error (delta)
         assertEquals(6.0, newReleaseRental.getCharge(), 0.001); // se pone
porque los decimales no son exactos
         assertEquals(1.5, childrensRental.getCharge(), 0.001); // y puede
haber errores de redondeo en memoria
    }

    @Test
    public void testGetFrequentRenterPoint() {
         // Para películas que no sean NewRelease, se devuelve 1 punto.
         assertEquals(1, regularRental.getFrequentRenterPoint());
         assertEquals(1, childrensRental.getFrequentRenterPoint());
         // Para NewRelease con más de 1 día, se devuelve 2 puntos.
         assertEquals(2, newReleaseRental.getFrequentRenterPoint());
    }

    @Test
    public void testStatement() {
         // Añadimos los alquileres al cliente
         customer.addRental(regularRental);
         customer.addRental(newReleaseRental);
         customer.addRental(childrensRental);

         String expected = "Rental record for John Doe\n";
         expected += "\tRegular Movie\t3.5\n";
         expected += "\tNew Release Movie\t6.0\n";
         expected += "\tChildrens Movie\t1.5\n";
         expected += "Amount owed is 11.0\n";
         expected += "You earned 4 frequent renter points.";

         assertEquals(expected, customer.statement());
    }
}
```

INTERFAZ de las DIFERENTES PELIS

```java
package Principal;

public interface PriceStrategy {
    double getCharge(int daysRented);
}
```

```java
package Principal;

public class RegularPrice implements PriceStrategy {
    @Override
    public double getCharge(int daysRented) {
        double result = 2;
        if (daysRented > 2) {
            result += (daysRented - 2) * 1.5;
        }
        return result;
    }
}
```

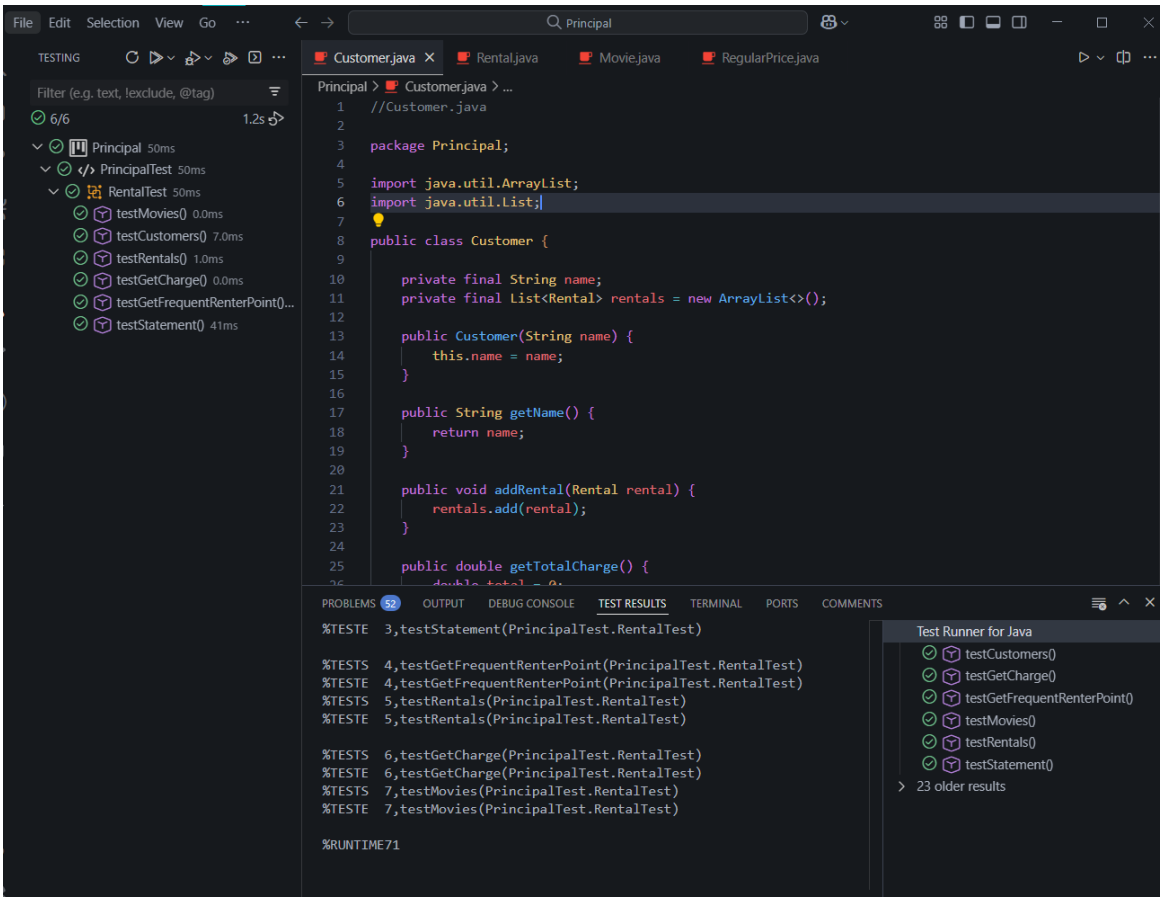
```java
package Principal;

public class NewReleasePrice implements PriceStrategy {
    @Override
    public double getCharge(int daysRented) {
        return daysRented * 3;
    }
}
```

```java
package Principal;

public class implements PriceStrategy {
    @Override
    public double getCharge(int daysRented) {
        double result = 1.5;
        if (daysRented > 3) {
            result = (daysRented - 3) * 1.5;
        }
        return result;
    }
}
```

RESULTADO DE LAS PRUEBAS (testing)



✅ Todo correcto!