Universidad Tecnológica del Perú

Docente:

Dilian Anabel HURTADO PONCE

Ejercicio 2
PC 3

Semana: 15

Integrantes:

Roberto Agustín Mejía Collazos

21/11/24

Una Tienda de Libros

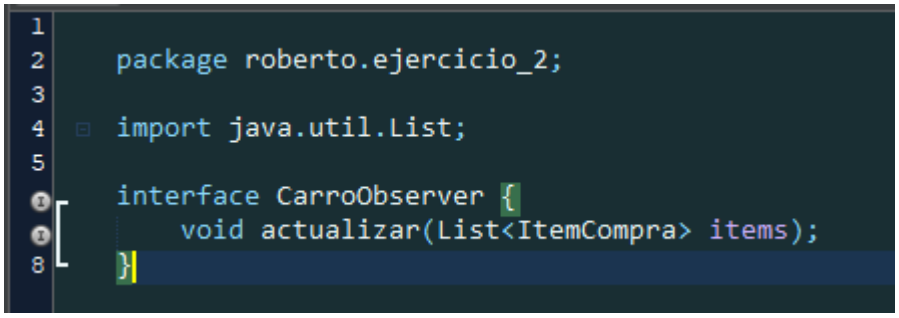
Libro.java

```java
package roberto.ejercicio_2;
// Clase Libro
class Libro {
    private String isbn;
    private String titulo;
    private double precio;

    public Libro(String isbn, String titulo, double precio) {
        this.isbn = isbn;
        this.titulo = titulo;
        this.precio = precio;
    }

    public String getTitulo() {
        return titulo;
    }

    public String getIsbn() {
        return isbn;
    }

    public double getPrecio() {
        return precio;
    }

    @Override
    public String toString() {
        return titulo + " (ISBN: " + isbn + ", Precio: " + precio + ")";
    }
}
```

ItemCompra.java

```java
package roberto.ejercicio_2;

// Clase ItemCompra
class ItemCompra {
    private Libro libro;
    private int cantidad;

    public ItemCompra(Libro libro, int cantidad) {
        this.libro = libro;
        this.cantidad = cantidad;
    }

    public Libro getLibro() {
        return libro;
    }

    public int getCantidad() {
        return cantidad;
    }

    @Override
    public String toString() {
        return libro.getTitulo() + " x" + cantidad + " - Subtotal: " + (libro.getPrecio() * cantidad);
    }
}
```

CarroObserver.java (Interface)

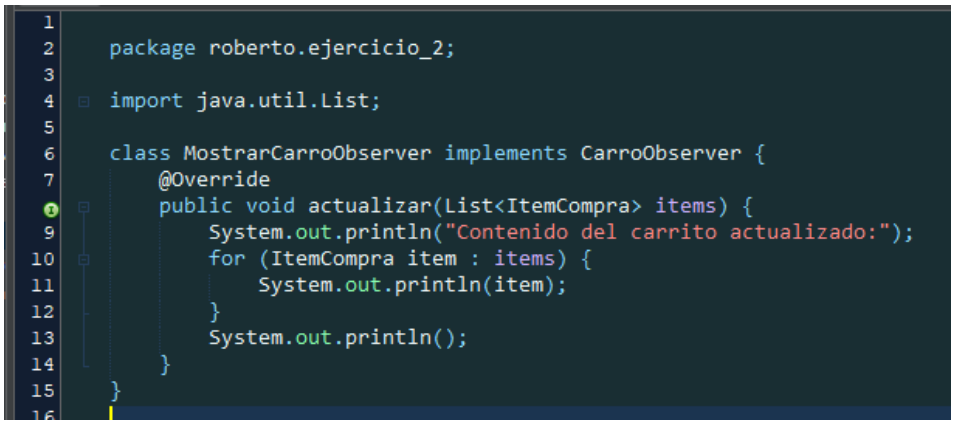


```java
package roberto.ejercicio_2;

import java.util.List;

interface CarroObserver {
    void actualizar(List<ItemCompra> items);
}
```

CarroCompra.java

```java
package roberto.ejercicio_2;

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

class CarroCompras {
    private List<ItemCompra> items = new ArrayList<>();
    private List<CarroObserver> observadores = new ArrayList<>();

    public void agregarLibro(Libro libro, int cantidad) {
        items.add(new ItemCompra(libro, cantidad));
        notificarObservadores();
    }

    public void eliminarLibro(String isbn) {
        items.removeIf(item -> item.getLibro().getIsbn().equals(isbn));
        notificarObservadores();
    }

    public void registrarObservador(CarroObserver observador) {
        observadores.add(observador);
    }

    public void eliminarObservador(CarroObserver observador) {
        observadores.remove(observador);
    }

    private void notificarObservadores() {
        for (CarroObserver observador : observadores) {
            observador.actualizar(items);
        }
    }
}
```

MostrarCarroObserver.java



```java
package roberto.ejercicio_2;

import java.util.List;

class MostrarCarroObserver implements CarroObserver {
    @Override
    public void actualizar(List<ItemCompra> items) {
        System.out.println("Contenido del carrito actualizado:");
        for (ItemCompra item : items) {
            System.out.println(item);
        }
        System.out.println();
    }
}
```

MostrarTotalObserver.java

```java
package roberto.ejercicio_2;

import java.util.List;

class MostrarTotalObserver implements CarroObserver {
    @Override
    public void actualizar(List<ItemCompra> items) {
        double total = items.stream()
                .mapToDouble(item -> item.getLibro().getPrecio() * item.getCantidad())
                .sum();
        System.out.println("Total de la compra: $" + total);
    }
}
```

Main.java

```java
package roberto.ejercicio_2;

public class Main {
    public static void main(String[] args) {
        // Crear carrito y observadores
        CarroCompras carrito = new CarroCompras();
        CarroObserver mostrarCarro = new MostrarCarroObserver();
        CarroObserver mostrarTotal = new MostrarTotalObserver();

        // Registrar observadores
        carrito.registrarObservador(mostrarCarro);
        carrito.registrarObservador(mostrarTotal);

        // Crear libros
        Libro libro1 = new Libro("123", "El Lenguaje UML", 50.0);
        Libro libro2 = new Libro("456", "Patrones de Diseño", 75.0);

        // Realizar operaciones en el carrito
        carrito.agregarLibro(libro1, 2); // Agregar libro UML
        carrito.agregarLibro(libro2, 1); // Agregar libro de patrones
        carrito.eliminarLibro("123");    // Eliminar libro UML
    }
}
```

Ejecutando el código

```
--- exec:3.1.0:exec (default-cli) @ Ejercicio_2 ---
Contenido del carrito actualizado:
El Lenguaje UML x2 - Subtotal: 100.0

Total de la compra: $100.0
Contenido del carrito actualizado:
El Lenguaje UML x2 - Subtotal: 100.0
Patrones de Dise□o x1 - Subtotal: 75.0

Total de la compra: $175.0
Contenido del carrito actualizado:
Patrones de Dise□o x1 - Subtotal: 75.0

Total de la compra: $75.0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.710 s
Finished at: 2024-11-21T21:48:50-05:00
------------------------------------------------------------------------
```

Explicación del Código del Patrón Observer

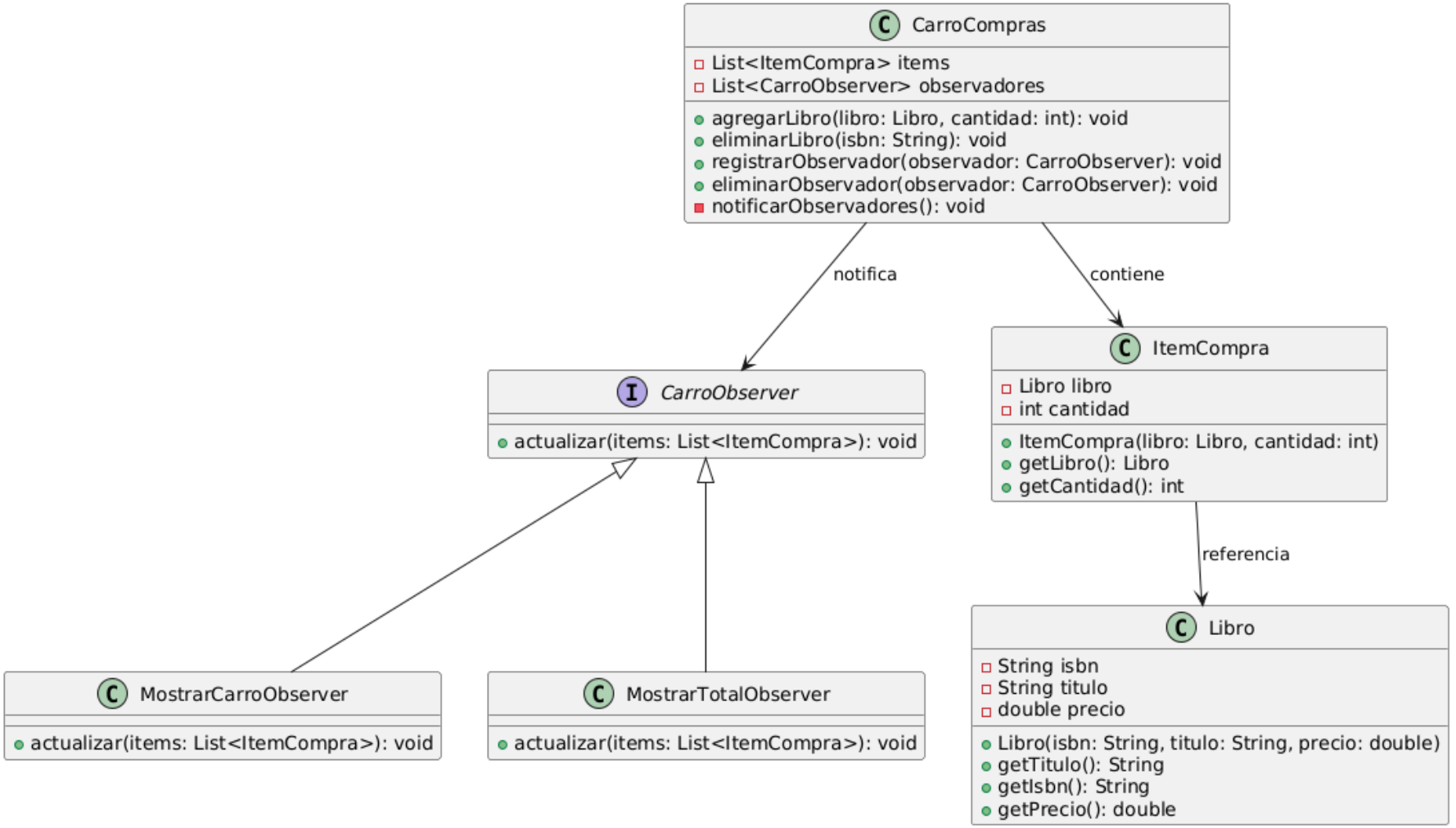
El patrón Observer permite que el objeto sujeto (en este caso, el CarroCompras) notifique automáticamente a múltiples observadores cada vez que cambie su estado.

Componentes Clave del Código

1. Interfaz CarroObserver: Define el contrato para los observadores. Cualquier clase que implemente esta interfaz puede registrarse para recibir notificaciones del carrito.
2. Clase CarroCompras (Sujeto):
    - Gestiona la lista de observadores que quieren ser notificados.
    - Tiene métodos para agregar y eliminar libros del carrito.
    - Cada vez que cambia su estado, llama al método actualizar de todos los observadores registrados.
3. Observadores Concretos:
    - MostrarCarroObserver: Muestra el contenido del carrito actualizado.
    - MostrarTotalObserver: Calcula y muestra el total de la compra.
4. Clase Principal (Main):
    - Crea el carrito y registra los observadores.
    - Simula operaciones como agregar y eliminar libros, desencadenando las actualizaciones automáticas.

Diagrama UML

Patrón Observer: Gestión de Carrito de Compras



Explicación del UML

1. Relaciones Principales:

   CarroObserver es la interfaz que define el método actualizar.

   MostrarCarroObserver y MostrarTotalObserver implementan CarroObserver como observadores concretos.

   CarroCompras mantiene una lista de observadores (CarroObserver) que son notificados mediante el método notificarObservadores().

2. Flujo General:

   Cuando cambia el estado del carrito (por ejemplo, se agrega o elimina un libro), el carrito invoca notificarObservadores().

   Este método recorre la lista de observadores y llama a su método actualizar, pasando la lista actualizada de items.

3. Ventajas Visuales:

   El diagrama muestra cómo los observadores (MostrarCarroObserver, MostrarTotalObserver) están desacoplados del CarroCompras.

   Facilita la extensión del sistema agregando nuevos observadores sin modificar la lógica del carrito.