Universidad Tecnológica del Perú

Docente:

Dilian Anabel HURTADO PONCE

Ejercicio 1
PC 3

Semana: 15

Integrantes:

Roberto Agustín Mejía Collazos

21/11/24

Una Tienda de Libros

Command.java (Interface)

```java
package roberto.ejercicio1;

// Interfaz Command
interface Command {
    void execute();
}
```

AddToCardCommand.java

```java
package roberto.ejercicio1;

// Comando para agregar libro
class AddToCartCommand implements Command {
    private CarroCompras carrito;
    private Libro libro;
    private int cantidad;

    public AddToCartCommand(CarroCompras carrito, Libro libro, int cantidad) {
        this.carrito = carrito;
        this.libro = libro;
        this.cantidad = cantidad;
    }

    @Override
    public void execute() {
        carrito.agregarLibro(libro, cantidad);
    }
}
```

RemoveFromCartCommand

```java
package roberto.ejercicio1;

// Comando para eliminar libro
class RemoveFromCartCommand implements Command {
    private CarroCompras carrito;
    private String isbn;

    public RemoveFromCartCommand(CarroCompras carrito, String isbn) {
        this.carrito = carrito;
        this.isbn = isbn;
    }

    @Override
    public void execute() {
        carrito.eliminarLibro(isbn);
    }
}
```

Libro.java

```java
package roberto.ejercicio1;

// 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

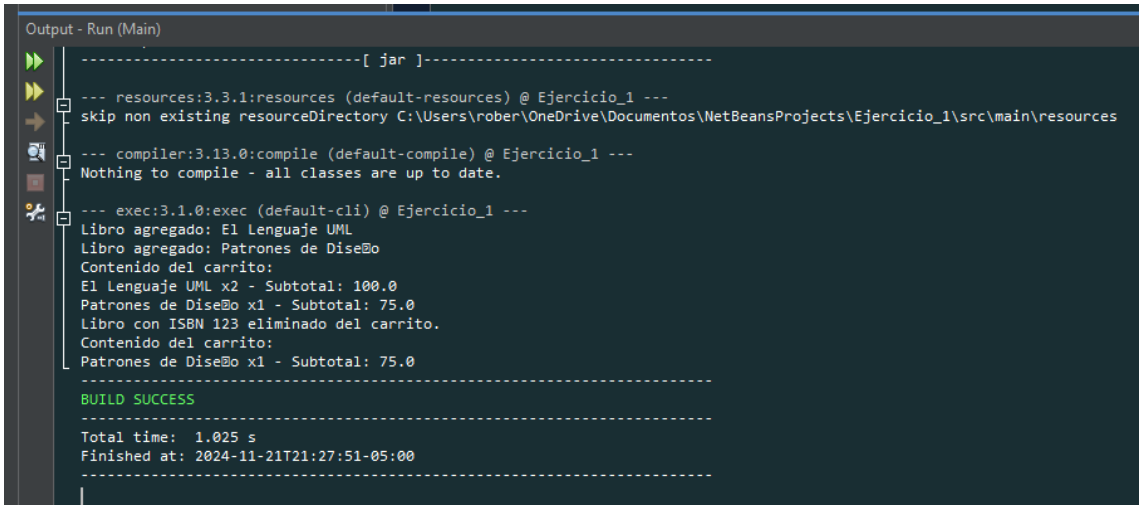
```java
package roberto.ejercicio1;

// 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);
    }
}
```

CarroCompra.java

```java
package roberto.ejercicio1;

// Clase CarroCompras

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

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

    public void agregarLibro(Libro libro, int cantidad) {
        items.add(new ItemCompra(libro, cantidad));
        System.out.println("Libro agregado: " + libro.getTitulo());
    }

    public void eliminarLibro(String isbn) {
        items.removeIf(item -> item.getLibro().getIsbn().equals(isbn));
        System.out.println("Libro con ISBN " + isbn + " eliminado del carrito.");
    }

    public void mostrarCarrito() {
        System.out.println("Contenido del carrito:");
        for (ItemCompra item : items) {
            System.out.println(item);
        }
    }
}
```

Main.java

```java
package roberto.ejercicio1;

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

public class Main {
    public static void main(String[] args) {
        // Crear catálogo y carrito
        CarroCompras carrito = new CarroCompras();
        Libro libro1 = new Libro("123", "El Lenguaje UML", 50.0);
        Libro libro2 = new Libro("456", "Patrones de Diseño", 75.0);

        // Crear comandos
        Command agregarUML = new AddToCartCommand(carrito, libro1, 2);
        Command agregarPatrones = new AddToCartCommand(carrito, libro2, 1);
        Command eliminarLibro = new RemoveFromCartCommand(carrito, "123");

        // Ejecutar comandos
        agregarUML.execute();
        agregarPatrones.execute();
        carrito.mostrarCarrito();

        // Eliminar un libro
        eliminarLibro.execute();
        carrito.mostrarCarrito();
    }
}
```

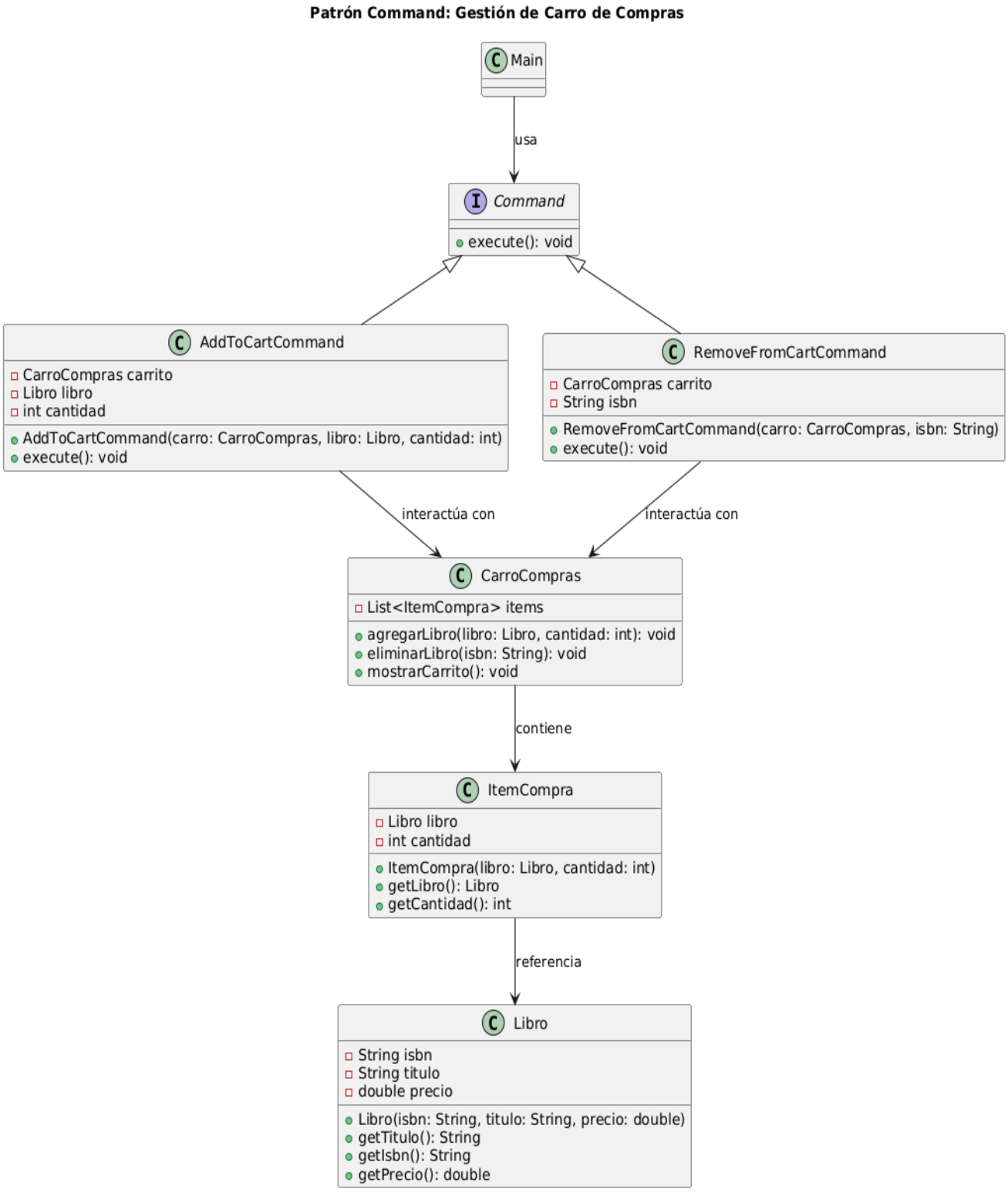


Explicación del Patrón Command Aplicado

El patrón Command encapsula las solicitudes como objetos, permitiendo parametrizar acciones, registrar su historial y proporcionar funcionalidad de deshacer/rehacer. En el caso presentado:

1. Actores principales:
    - Cliente (Main): Solicita la ejecución de comandos.
    - Receptor (CarroCompras): Contiene la lógica principal para gestionar libros.
    - Comandos concretos (AddToCartCommand, RemoveFromCartCommand): Encapsulan las acciones que pueden realizarse sobre el receptor.
2. Ventajas del uso del patrón Command en este caso:
    - Flexibilidad: Cada acción está encapsulada, facilitando la adición de nuevas operaciones.
    - Historial: Permite llevar un registro de los comandos ejecutados.
    - Extensibilidad: Podemos agregar comandos como "vaciar carrito" o "mostrar totales" sin modificar la lógica existente.
3. Flujo del programa:
    - El cliente (Main) crea instancias de comandos con parámetros específicos (como el carrito y los libros).
    - Los comandos son ejecutados, invocando métodos en el receptor (CarroCompras).

Diagrama UML



Explicación del UML

1. Relaciones principales:
    - Command es una interfaz que define el contrato común para los comandos.
    - AddToCartCommand y RemoveFromCartCommand implementan la interfaz Command.
    - El cliente (Main) utiliza los comandos para interactuar con el receptor (CarroCompras).
2. Dependencias:
    - Los comandos (AddToCartCommand, RemoveFromCartCommand) dependen del receptor (CarroCompras).
    - CarroCompras gestiona los elementos del carrito como una lista de ItemCompra.
    - Cada ItemCompra tiene una referencia a un Libro.
3. Beneficio visual:
    - El diagrama organiza claramente los roles de cada clase, mostrando cómo encapsulan sus responsabilidades y colaboran para lograr la funcionalidad.