**Exercise 9: Implementing the Command Pattern**

**Command.java**

**package** mypackage;

**public** **interface** Command {

**void** execute();

}

**Implement Receiver Class**

**Light.java**

**package** mypackage;

**public** **class** Light {

**public** **void** turnOn() {

System.***out***.println("Light is ON");

}

**public** **void** turnOff() {

System.***out***.println("Light is OFF");

}

}

**Implement Concrete Commands**

**LightOnCommand.java**

**package** mypackage;

**public** **class** LightOnCommand **implements** Command {

**private** Light light;

**public** LightOnCommand(Light light) {

**this**.light = light;

}

**public** **void** execute() {

light.turnOn();

}

}

**LightOffCommand.java**

**package** mypackage;

**public** **class** LightOffCommand **implements** Command {

**private** Light light;

**public** LightOffCommand(Light light) {

**this**.light = light;

}

**public** **void** execute() {

light.turnOff();

}

}

**Implement Invoker Class**

**RemoteControl.java**

**package** mypackage;

**public** **class** RemoteControl {

**private** Command command;

**public** **void** setCommand(Command command) {

**this**.command = command;

}

**public** **void** pressButton() {

**if** (command != **null**) {

command.execute();

} **else** {

System.***out***.println("No command assigned.");

}

}

}

**Test the Command Implementation**

**TestCommandPattern.java**

**package** mypackage;

**import** java.util.Scanner;

**public** **class** TestCommandPattern {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

Light light = **new** Light();

RemoteControl remote = **new** RemoteControl();

System.***out***.println("Home Automation - Command Pattern");

System.***out***.println("1. Turn ON Light");

System.***out***.println("2. Turn OFF Light");

System.***out***.print("Enter your choice: ");

**int** choice = sc.nextInt();

**if** (choice == 1) {

remote.setCommand(**new** LightOnCommand(light));

} **else** **if** (choice == 2) {

remote.setCommand(**new** LightOffCommand(light));

} **else** {

System.***out***.println("Invalid choice.");

sc.close();

**return**;

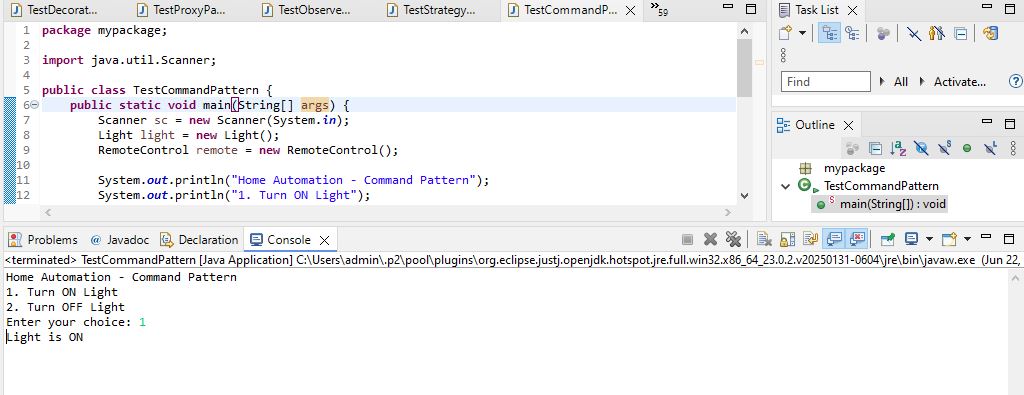
}

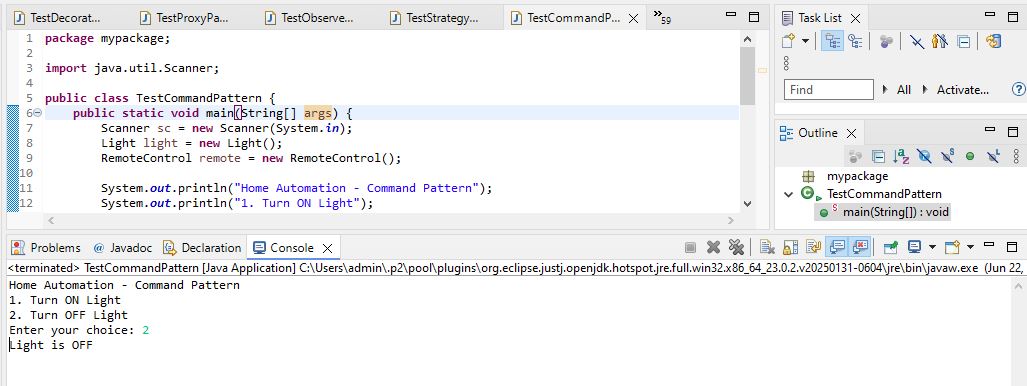
remote.pressButton();

sc.close();

}

}





We are developing a home automation system where commands can be issued to turn devices on or off. We use the Command Pattern to encapsulate requests as objects.

**1. Components**

* **Command Interface:** Declares execute() method.
* **Concrete Commands:** LightOnCommand and LightOffCommand implement Command and call the corresponding methods on Light.
* **Receiver:** The Light class performs actual operations (turnOn() and turnOff()).
* **Invoker:** RemoteControl stores and executes commands.
* **Client:** The main method creates commands and assigns them to the invoker.

**2. Advantages of Command Pattern**

* Decouples sender (Invoker) from receiver.
* Supports undo/redo functionality.
* Commands can be queued, logged, or scheduled.
* Easy to add new commands without modifying existing code (Open-Closed Principle).

**3. Time Complexity**

* Setting command: O(1)
* Executing command: O(1)

**4. Real-Life Applications**

* Home automation systems.
* Transaction management.
* Task scheduling systems.
* GUI button actions (like Undo, Redo).