Department of Computer Science and Engineering

Islamic University of Technology (IUT)

A subsidiary organ of OIC

Lab Task

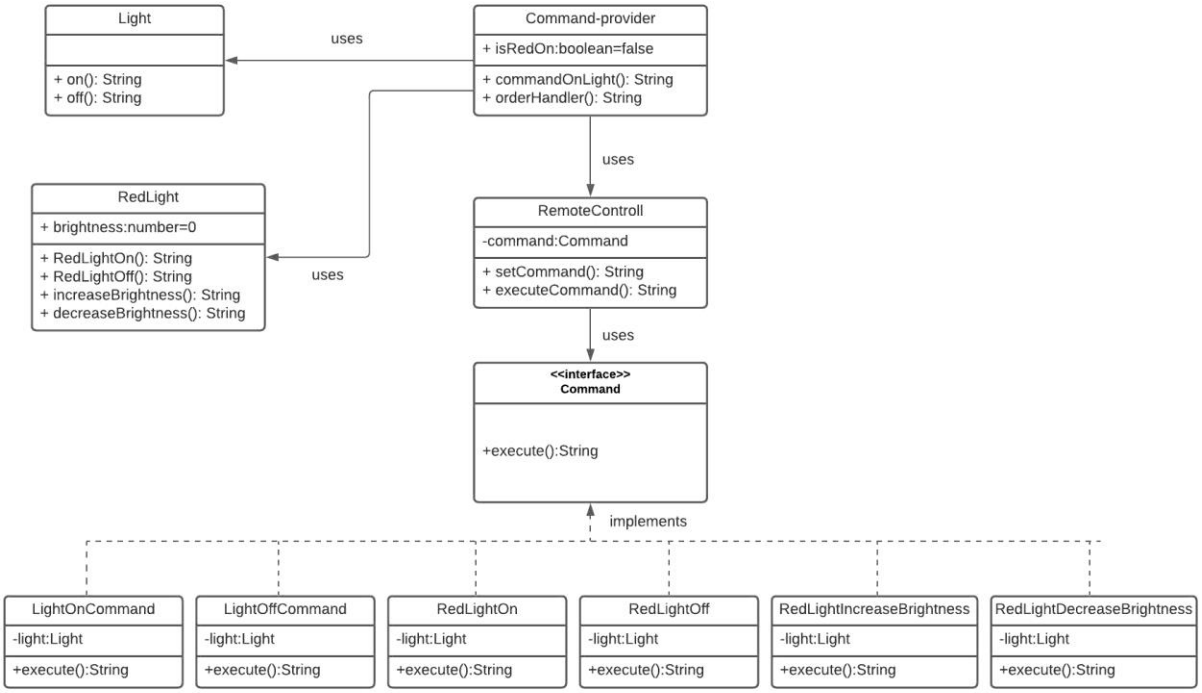
On

Command Pattern

SWE 4502

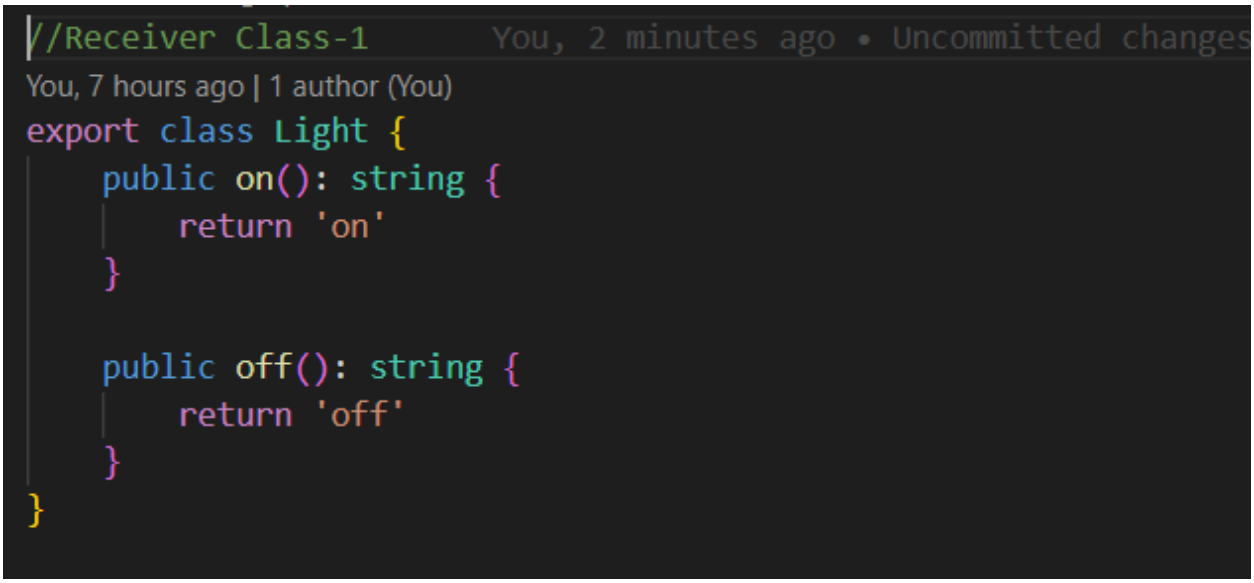
Name: SIANA RIZWAN
Student ID: 180042105
Semester: Winter semester
Academic Year: 2018-1019
Date of Submission: 19/08/2021

UML Diagram


Light
+ on(): String
+ off(): String
uses
Command-provider
+ isRedOn:boolean=false
+ commandOnLight(): String
+ orderHandler(): String
uses
RedLight
+ brightness:number=0
+ RedLightOn(): String
+ RedLightOff(): String
+ increaseBrightness(): String
+ decreaseBrightness(): String
uses
RemoteControll
-command:Command
+ setCommand(): String
+ executeCommand(): String
uses
<<interface>>
Command
+execute():String
implements
LightOnCommand
-light:Light
+execute():String
LightOffCommand
-light:Light
+execute():String
RedLightOn
-light:Light
+execute():String
RedLightOff
-light:Light
+execute():String
RedLightIncreaseBrightness
-light:Light
+execute():String
RedLightDecreaseBrightness
-light:Light
+execute():String

Code Implementation

1. The command pattern design has been implemented in the following way in the file command-RemoteControll.ts-



Two receiver classes for normal light on off and red light on off have been created

```
var brightness:number=0
//Receiver Class-2
You, 2 minutes ago | 1 author (You)
export class RedLight {
    public RedLightOn(): string {
        brightness=0;
        return `red${brightness}`
    }

    public RedLightOff(): string {
        return "off"
    }

    public increaseBrightness(): string {
        brightness++;
        return `red${brightness}`
    }

    public decreaseBrightness(): string {
        brightness--;
        return `red${brightness}`

    }
}
```

Interface that will bind the commands with receiver class have been created

```
//interface that passes specific request to specific receiver
You, 7 hours ago | 1 author (You)
export interface Command {
    execute(): string
}
```

Next classes for specific commands have been created

```
//implementing the Command interface for specific commands
//that needs to be passed to specific receiver
You, 2 minutes ago | 1 author (You)
export class LightOnCommand implements Command {

    private light: Light

    //constructor takes the object of the specific receiver class that it will controll
    constructor(light: Light) {
        this.light = light;
    }

    execute(): string {
        return this.light.on();
    }
```

```
//implementing the Command interface for specific commands
//that needs to be passed to specific receiver
You, 2 minutes ago | 1 author (You)
export class LightOffCommand implements Command {
    private light: Light

    //constructor takes the object of the specific receiver class that it will controll
    constructor(light: Light) {
        this.light = light
    }

    execute(): string {
        return this.light.off()
    }

}
```

```
//implementing the Command interface for specific commands
//that needs to be passed to specific receiver
You, 3 minutes ago | 1 author (You)
export class RedLightOn implements Command {
    light: RedLight;

    //constructor takes the object of the specific receiver class that it will controll
    constructor(light: RedLight) {
        this.light = light;
    }

    execute(): string {
        return this.light.RedLightOn()
    }

}
```

```
//implementing the Command interface for specific commands
//that needs to be passed to specific receiver
You, 3 minutes ago | 1 author (You)
export class RedLightOff implements Command {
    light: RedLight;

    //constructor takes the object of the specific receiver class that it will controll
    constructor(light: RedLight) {
        this.light = light;
    }

    execute(): string {
        return this.light.RedLightOff()
    }

}
```

```
//implementing the Command interface for specific commands
//that needs to be passed to specific receiver
You, 3 minutes ago | 1 author (You)
export class RedLightIncreaseBrightness implements Command {
    light: RedLight;

    //constructor takes the object of the specific receiver class that it will controll
    constructor(light: RedLight) {
        this.light = light;
    }

    execute(): string {
        return this.light.increaseBrightness()
    }

}
```

```
//implementing the Command interface for specific commands
//that needs to be passed to specific receiver
You, 3 minutes ago | 1 author (You)
export class RedLightDecreaseBrightness implements Command {
    light: RedLight;

    //constructor takes the object of the specific receiver class that it will controll
    constructor(light: RedLight) {
        this.light = light;
    }

    execute(): string {
        return this.light.decreaseBrightness()
    }

}
```

Lastly the invoker class has been created that will be used to invoke the specific command-

```
//Invoker class
You, 3 minutes ago | 1 author (You)
export class RemoteControll {
    command!: Command;
    constructor() {
    }
    //sets the command that the invoker(the remote) will execute
    setCommand(command: Command) {
        this.command = command;
    }

    //executes the command
    executeCommand() {
        return this.command.execute()
    }


}
```

2. Next in the command-provider.ts file, commands have been set and defined according to the ui manipulation

```
import {
    Light,
    LightOnCommand,
    LightOffCommand,
    RedLight,
    RedLightOn,
    RedLightOff,
    RedLightIncreaseBrightness ,
    RedLightDecreaseBrightness,
    RemoteControll,
    Command
} from "../../patterns/command/command-RemoteControl";

//function that will take and implement the commands through the remotecontroll
export function commandOnLight(command: Command): string {
    const remoteControll = new RemoteControll();
    remoteControll.setCommand(command)
    return remoteControll.executeCommand()

}
```

```
let IsRedOn: boolean = false
let IsLightOn: boolean = false
//this function is also called in page.svelte that gets and defines commands from ui
//and sets for which command what action will be taken by passing the command
//to the above created function
export function orderHandler(command: string): string {
    let action: string;
    switch (command) {
        case "on":
            IsLightOn=true
            action = IsLightOn ? commandOnLight(new LightOnCommand(new Light())) : commandOnLight(new LightOnCommand(new Light()))
            break;

        case "off":
            IsRedOn = false
            IsLightOn=false
            action = IsRedOn || IsLightOn ? commandOnLight(new RedLightOff(new RedLight())) : commandOnLight(new LightOffCommand(new Light()))
            break

        case "red":
            IsRedOn = true
            action = IsRedOn && IsLightOn ? commandOnLight(new RedLightOn(new RedLight())):commandOnLight(new LightOffCommand(new Light()))
            break

        case "increase":
            action = IsRedOn ? commandOnLight(new RedLightIncreaseBrightness(new RedLight())):commandOnLight(new LightOnCommand(new Light()))
            break

        case "decrease":
            action = IsRedOn ? commandOnLight(new RedLightDecreaseBrightness(new RedLight())) : commandOnLight(new LightOnCommand(new Light()))
            break

        default:      You, 8 hours ago • command pattern lab task done
    }
    // @ts-ignore
    return action;
}
```

3. Lastly in Page. Svelte the actions in the UI have been set in the following way-

```
<script>

  import {orderHandler} from "./command-provider"
  let commands = {
    on: "on",
    red0:"red/0",
    red1: "red/1",
    red2: "red/2",
    red3: "red/3",
    off: "off",
  };

//clicking each button , this function will be called
//that will set the command in the command-provider class for the commands to be executed
    let parameters = "off"
    function changeCommand(a){
        $: parameters=orderHandler(a);
        $: src=`./images/light-receiver/${commands[parameters]}.png`;
    }
    let src = `./images/light-receiver/${commands[parameters]}.png`;


</script>
```

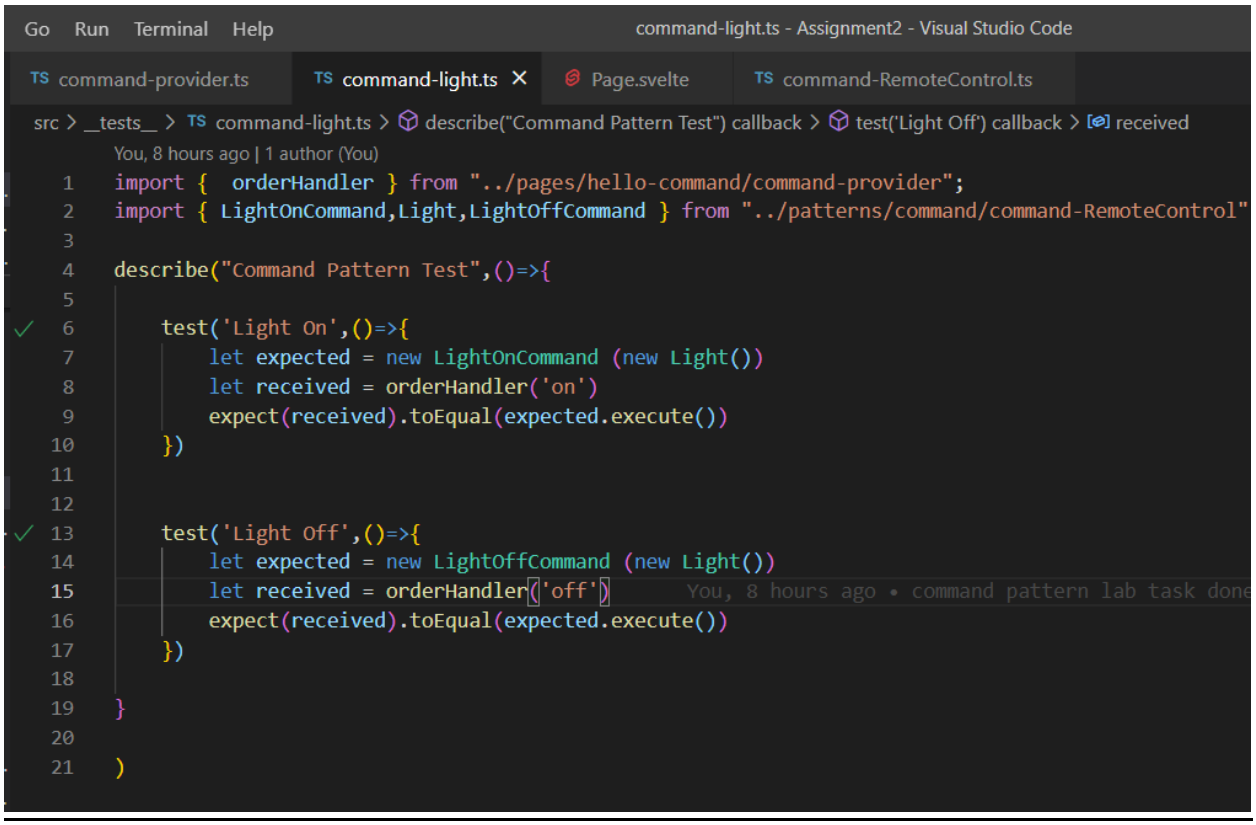
```
<h1>Command buttons</h1>
      You, 8 hours ago • command pattern lab task done
<div class="btn-group">
  <button class="on" on:click={()=>{changeCommand('on')}}>On</button>
  <button class="off" on:click={()=>{changeCommand('off')}}>Off</button>

  <button class="increase-lum" on:click={()=>{changeCommand('increase')}}>+</button>
  <button class="decrease-lum" on:click={()=>{changeCommand('decrease')}}>-</button>

  <button class="red-light" on:click={()=>{changeCommand('red')}}>Red</button>
</div>

<div class="portrait"><img {src} alt={src} /></div>
```

Unit Tests


Go Run Terminal Help
command-light.ts - Assignment2 - Visual Studio Code
command-provider.ts
command-light.ts
Page.svelte
command-RemoteControl.ts
src > __tests__ > command-light.ts > describe("Command Pattern Test") callback > test('Light Off') callback > received
You, 8 hours ago | 1 author (You)
import { orderHandler } from "../pages/hello-command/command-provider";
import { LightOnCommand,Light,LightOffCommand } from "../patterns/command/command-RemoteControl"
describe("Command Pattern Test",()=>{
test('Light On',()=>{
let expected = new LightOnCommand (new Light())
let received = orderHandler('on')
expect(received).toEqual(expected.execute())
})
test('Light Off',()=>{
let expected = new LightOffCommand (new Light())
let received = orderHandler('off')
You, 8 hours ago • command pattern lab task done
expect(received).toEqual(expected.execute())
})
}
)