# **Command Pattern: Assignment-2**

Course ID: SWE-4502

Course Name: Design Pattern Lab

Name: Farzana Tabassum

**ID:** 180042119

**Department:** CSE (SWE)

**Semester:** Winter (5<sup>th</sup> semester)

**Date:** 13/08/2021

GitLink: <a href="https://github.com/farzana-">https://github.com/farzana-</a>

snigdha/dpatterna/tree/command-pattern

## **Code implementation**

#### command-lightChanges.ts

```
TS command-provider.ts
                                         TS command-lightChange.ts 1 X TS command-lightChange.test.ts
dpatterna > src > patterns > command > TS command-lightChange.ts > ...
       // Light class act as a request
       export class Light {
         public on(): string {
          return "on";
         public off(): string {
         increase: number = 0;
         redOn(): string {
          this.increase = 0;
          return "red0";
         //increase red light intensity on each click, after reaching maximum intesity,
         increaseRedLight(): string {
           if (this.increase < 3) {</pre>
             this.increase++;
             return red${this.increase};
           } else {
         //decrease red light intensity on each click, after reaching minimum intesity,
         decreaseRedLight(): string {
           this.increase--;
           if (this.increase < 0) {</pre>
           } else {
             return `red${this.increase}`;
```

```
//Command interface encapsulates a request by binding together a set of actions
     //on a specific receiver. It does so by exposing just one method setCommand()
     // that causes some actions to be invoked on the receiver.
     export interface Command {
       setCommand(): string;
     //Light classe's corresponding command classes
     export class TurnOnLightCommand implements Command {
       light: Light;
       // The constructor is passed the light it is going to control.
       constructor(light: Light) {
         this.light = light;
       setCommand(): string {
         return this.light.on();
     export class TurnOffLightCommand implements Command {
       light: Light;
       constructor(light: Light) {
         this.light = light;
       setCommand(): string {
         return this.light.off();
     }
68
```

```
68
     //RedLight classe's corresponding command classes
70
71
     export class TurnOnRedLightCommand implements Command {
       light: Light;
72
       constructor(light: Light) {
73
         this.light = light;
75
       setCommand(): string {
76
         return this.light.redOn();
78
79
     export class IncreaseRedLightCommand implements Command {
       light: Light;
82
       constructor(light: Light) {
         this.light = light;
       setCommand(): string {
87
         return this.light.increaseRedLight();
       }
90
     export class DecreaseRedLightCommand implements Command {
       light: Light;
       constructor(light: Light) {
         this.light = light;
96
       setCommand(): string {
         return this.light.decreaseRedLight();
       }
```

```
101
      //invoker class
      export class RemoteControl {
102
        command: Command;
103
104
        // set the command the remote will execute
105
        execute(command: Command) {
106
          this.command = command;
107
        }
108
109
        public executeCommand() {
110
          return this.command.setCommand();
111
       }
112
      }
113
114
```

#### **command-provider.ts**

```
import {
 Command,
 DecreaseRedLightCommand,
  IncreaseRedLightCommand,
 Light,
 RemoteControl,
 TurnOffLightCommand,
 TurnOnLightCommand,
 TurnOnRedLightCommand,
} from "patterns/command/command-lightChange";
let light;
let redLight;
let turnOnRedLight: Command;
let turnOnLight: Command;
let turnOffLight: Command;
let remoteControl: RemoteControl;
let increaseRedLight: Command;
let decreaseRedLight: Command;
light = new Light();
turnOnRedLight = new TurnOnRedLightCommand(light);
turnOnLight = new TurnOnLightCommand(light);
turnOffLight = new TurnOffLightCommand(light);
increaseRedLight = new IncreaseRedLightCommand(light);
decreaseRedLight = new DecreaseRedLightCommand(light);
var isRedLightOn = false;
remoteControl = new RemoteControl();
//setLightCommands() is called in Page.svelte and there it takes 2 values
//to determine action type. here we can change command dynamically. it calls
```

```
//setLightCommands() is called in Page.svelte and there it takes 2 values
     //to determine action type. here we can change command dynamically. it calls
37
     // returns light value using executeCommand()
     export function setLightCommands(command: string, count: number) {
41
       console.log(" count>0 ", count);
       if (command == "on") {
         isRedLightOn = false;
         remoteControl.execute(turnOnLight);
       } else if (command == "off") {
46
         isRedLightOn = false;
         remoteControl.execute(turnOffLight);
       } else if (command == "redLight") {
         isRedLightOn = true;
         remoteControl.execute(turnOnRedLight);
         console.log("provider ", command);
       } else if (command == "increase" && count > 0) {
         remoteControl.execute(increaseRedLight);
53
       } else if (command == "decrease" && count > 0) {
        remoteControl.execute(decreaseRedLight);
       return remoteControl.executeCommand();
60
```

#### page.svelte

```
dpatterna > src > pages > hello-command > ∅ Page.svelte > ❤ script
         import { setLightCommands } from "./command-provider";
         let commands = {
           red0: "red/0",
           red1: "red/1",
           red2: "red/2",
           red3: "red/3",
         };
         let commandType = "off";
         let src = `./images/light-receiver/${commands[commandType]}.png`;
         let count = 0;
         //it takes 2 params: 1st on for the type of action user wants to see
 21
         function executeCommand(command, counts) {
           console.log("src ", command);
           //commandType determines which type of light should be visible to the user
           commandType = setLightCommands(command, counts);
           src = `./images/light-receiver/${commands[commandType]}.png`;
           console.log(commandType);
       <h1>Command buttons</h1>
```

```
<h1>Command buttons</h1>
<div class="btn-group">
 <button class="on" on:click={() => executeCommand('on', count)}>On</button>
  <button class="off" on:click={() => executeCommand('off', count)}>Off</button>
   on:click={() => executeCommand('increase', count)}
   on:click={() => executeCommand('decrease', count)}
   class="red-light"
   on:click={() => executeCommand('redLight', count++)}
 >Red</button>
<div class="portrait"><img {src} alt={src} /></div>
.btn-group button {
    padding: 10px 24px;
    cursor: pointer;
   width: 20%;
   display: block;
  .btn-group button:not(:last-child) {
    border-bottom: none; /* Prevent double borders */
```

```
/* Add a background color on hover */
  .btn-group button:hover {
 filter: brightness(85%);
  .decrease-lum {
   background-color: ■#a9c1c9;
  .red-light {
   background-color: #f44336;
  .increase-lum {
   background-color: ■#8bb19b;
 .on {
   background-color: ■#e7e7e7;
   color: □black;
  .off {
   background-color: □#555555;
   color: ☐ white;
 img {
   max-width: 100%;
   max-height: 100%;
 .portrait {
   height: 300px;
   width: 500px;
</style>
```

### **Unit tests**

```
dpatterna > src > _tests_ > TS command-lightChange.test.ts > ♥ describe("light command pattern") callback >
       import { setLightCommands } from "pages/hello-command/command-provider";
       import {
         Light,
         TurnOnLightCommand,
         TurnOffLightCommand,
         TurnOnRedLightCommand,
         IncreaseRedLightCommand,
         DecreaseRedLightCommand,
       } from "patterns/command/command-lightChange";
       🏃 | 🏨 | Run | Debug
       describe("light command pattern", () => {
         let light = new Light();
         let count = 0;
         🏃 | 🐞 | Run | Debug
         test("light on", () => {
           let expectation = setLightCommands("on", count);
           let reality = new TurnOnLightCommand(light);
           expect(expectation).toEqual(reality.setCommand());
         });
         🏃 | 🐞 | Run | Debug
         test("light off", () => {
           let expectation = setLightCommands("off", count);
 24
           let reality = new TurnOffLightCommand(light);
           expect(expectation).toEqual(reality.setCommand());
         });
         🏃 | 🀞 | Run | Debug
         test("red light on", () => {
           let expectation = setLightCommands("redLight", count);
           let reality = new TurnOnRedLightCommand(light);
           expect(expectation).toEqual(reality.setCommand());
         });
```

```
🏃 | 🀞 | Run | Debug
       test("red light on", () => {
         let expectation = setLightCommands("redLight", count);
         let reality = new TurnOnRedLightCommand(light);
         expect(expectation).toEqual(reality.setCommand());
       });
        🔭 | 🐞 | Run | Debug
       test("increase red light", () => {
35
         count++;
         let expectation = setLightCommands("increase", count);
         let reality = new IncreaseRedLightCommand(light);
         expect(expectation).toEqual(reality.setCommand());
       });
41
       🔭 | 🐞 | Run | Debug
       test("decrease red light", () => {
42
         count++;
         let expectation = setLightCommands("decrease", count);
         let reality = new DecreaseRedLightCommand(light);
         expect(expectation).toEqual(reality.setCommand());
       });
     });
```

## **UML diagram**

