

Command Pattern: Assignment-2

Course ID: SWE-4502

Course Name: Design Pattern Lab

Name: Farzana Tabassum

ID: 180042119

Department: CSE (SWE)

Semester: Winter (5th semester)

Date: 13/08/2021

GitLink: <https://github.com/farzana-snigdha/dpatterna/tree/command-pattern>

Code implementation

command-lightChanges.ts

```
Page.svelte TS command-provider.ts TS command-lightChange.ts 1 X TS command-lightChange.test.ts
dpatterna > src > patterns > command > TS command-lightChange.ts > ...
1 // Light class| act as a request
2 export class Light {
3   public on(): string {
4     return "on";
5   }
6   public off(): string {
7     return "off";
8   }
9
10  increase: number = 0;
11  redOn(): string {
12    this.increase = 0;
13    return "red0";
14  }
15
16  //increase red light intensity on each click, after reaching maximum intensity,
17  //it does not change even if the intensity is increased
18  increaseRedLight(): string {
19    if (this.increase < 3) {
20      this.increase++;
21      return `red${this.increase}`;
22    } else {
23      return `red3`;
24    }
25  }
26
27  //decrease red light intensity on each click, after reaching minimum intensity,
28  //it turned of the light
29  decreaseRedLight(): string {
30    this.increase--;
31    if (this.increase < 0) {
32      return "off";
33    } else {
34      return `red${this.increase}`;
35    }
36  }
37 }
38
```

```

35     }
36 }
37 }
38
39 //Command interface encapsulates a request by binding together a set of actions
40 //on a specific receiver. It does so by exposing just one method setCommand()
41 // that causes some actions to be invoked on the receiver.
42 export interface Command {
43     setCommand(): string;
44 }
45
46 //Light classe's corresponding command classes
47 export class TurnOnLightCommand implements Command {
48     light: Light;
49
50     // The constructor is passed the light it is going to control.
51     constructor(light: Light) {
52         this.light = light;
53     }
54     setCommand(): string {
55         return this.light.on();
56     }
57 }
58
59 export class TurnOffLightCommand implements Command {
60     light: Light;
61
62     constructor(light: Light) {
63         this.light = light;
64     }
65     setCommand(): string {
66         return this.light.off();
67     }
68 }
69

```

```

68     }
69
70     //RedLight classe's corresponding command classes
71     export class TurnOnRedLightCommand implements Command {
72         light: Light;
73         constructor(light: Light) {
74             this.light = light;
75         }
76         setCommand(): string {
77             return this.light.redOn();
78         }
79     }
80
81     export class IncreaseRedLightCommand implements Command {
82         light: Light;
83         constructor(light: Light) {
84             this.light = light;
85         }
86         setCommand(): string {
87             return this.light.increaseRedLight();
88         }
89     }
90
91     export class DecreaseRedLightCommand implements Command {
92         light: Light;
93         constructor(light: Light) {
94             this.light = light;
95         }
96         setCommand(): string {
97             return this.light.decreaseRedLight();
98         }
99     }
100
101     //invoker class
102     export class RemoteControl {

```

```
100
101 //invoker class
102 export class RemoteControl {
103     command: Command;
104
105     // set the command the remote will execute
106     execute(command: Command) {
107         this.command = command;
108     }
109
110     public executeCommand() {
111         return this.command.setCommand();
112     }
113 }
114
```

command-provider.ts

```
1  //this class uses RemoteControl class to demonstrate command pattern
2  import {
3      Command,
4      DecreaseRedLightCommand,
5      IncreaseRedLightCommand,
6      Light,
7
8      RemoteControl,
9      TurnOffLightCommand,
10     TurnOnLightCommand,
11     TurnOnRedLightCommand,
12 } from "patterns/command/command-lightChange";
13
14 let light;
15 let redLight;
16 let turnOnRedLight: Command;
17 let turnOnLight: Command;
18 let turnOffLight: Command;
19 let remoteControl: RemoteControl;
20 let increaseRedLight: Command;
21 let decreaseRedLight: Command;
22
23 light = new Light();
24
25 turnOnRedLight = new TurnOnRedLightCommand(light);
26 turnOnLight = new TurnOnLightCommand(light);
27 turnOffLight = new TurnOffLightCommand(light);
28 increaseRedLight = new IncreaseRedLightCommand(light);
29 decreaseRedLight = new DecreaseRedLightCommand(light);
30
31 var isRedLightOn = false;
32
33 //Use the RemoteControl class to take and execute commands.
34 remoteControl = new RemoteControl();
35
36 //setLightCommands() is called in Page.svelte and there it takes 2 values
37 //to determine action type. here we can change command dynamically. it calls
```



```
35
36 //setLightCommands() is called in Page.svelte and there it takes 2 values
37 //to determine action type. here we can change command dynamically. it calls
38 //execute() from RemoteControl class and executes user commands and
39 // returns light value using executeCommand()
40 export function setLightCommands(command: string, count: number) {
41     console.log(" count>0 ", count);
42     if (command == "on") {
43         isRedLightOn = false;
44         remoteControl.execute(turnOnLight);
45     } else if (command == "off") {
46         isRedLightOn = false;
47         remoteControl.execute(turnOffLight);
48     } else if (command == "redLight") {
49         isRedLightOn = true;
50         remoteControl.execute(turnOnRedLight);
51         console.log("provider ", command);
52     } else if (command == "increase" && count > 0) {
53         remoteControl.execute(increaseRedLight);
54     } else if (command == "decrease" && count > 0) {
55         remoteControl.execute(decreaseRedLight);
56     }
57
58     return remoteControl.executeCommand();
59 }
60
```

page.svelte

```
dpatterna > src > pages > hello-command > 🐞 Page.svelte > 📄 script
1  <script>
2    import { setLightCommands } from "../command-provider";
3
4    let commands = {
5      on: "on",
6      off: "off",
7      red0: "red/0",
8      red1: "red/1",
9      red2: "red/2",
10     red3: "red/3",
11   };
12   let commandType = "off";
13   let src = `./images/light-receiver/${commands[commandType]}.png`;
14
15   //set intensity for +/- buttons. initially 0; means red light is turned off. after
16   // clicking red button i.e. turn on red light, count value will
17   // increase and then +/- will increase/decrease red light intensity accordingly
18   let count = 0;
19
20   //after clicking on a particular button, this function will execute.
21   //it takes 2 params: 1st on for the type of action user wants to see
22   //and the 2nd one is count value to check whether red light is turned on or not
23   function executeCommand(command, counts) {
24     console.log("src ", command);
25
26     //commandType determines which type of light should be visible to the user
27     commandType = setLightCommands(command, counts);
28     src = `./images/light-receiver/${commands[commandType]}.png`;
29     console.log(commandType);
30   }
31
32 </script>
33
34 <h1>Command buttons</h1>
35
```



```

32 <h1>Command buttons</h1>
33
34 <div class="btn-group">
35   <button class="on" on:click={() => executeCommand('on', count)}>On</button>
36   <button class="off" on:click={() => executeCommand('off', count)}>Off</button>
37
38   <button
39     class="increase-lum"
40     on:click={() => executeCommand('increase', count)}
41   >+</button>
42   <button
43     class="decrease-lum"
44     on:click={() => executeCommand('decrease', count)}
45   >-</button>
46
47   <button
48     class="red-light"
49     on:click={() => executeCommand('redLight', count++)}
50   >Red</button>
51 </div>
52
53 <div class="portrait"><img {src} alt={src} /></div>
54
55 <style>
56   .btn-group button {
57     padding: 10px 24px;
58     cursor: pointer;
59     width: 20%;
60     display: block;
61   }
62
63   .btn-group button:not(:last-child) {
64     border-bottom: none; /* Prevent double borders */
65   }
66

```

```
66
67  /* Add a background color on hover */
68  .btn-group button:hover {
69    | filter: brightness(85%);
70  }
71
72  .decrease-lum {
73    | background-color: #a9c1c9;
74  }
75  .red-light {
76    | background-color: #f44336;
77  }
78  .increase-lum {
79    | background-color: #8bb19b;
80  }
81  .on {
82    | background-color: #e7e7e7;
83    | color: black;
84  }
85  .off {
86    | background-color: #555555;
87    | color: white;
88  }
89  img {
90    | max-width: 100%;
91    | max-height: 100%;
92  }
93  .portrait {
94    | height: 300px;
95    | width: 500px;
96  }
97
98 </style>
99
```

Unit tests

```
dpatterna > src > __tests__ > TS command-lightChange.test.ts > describe("light command pattern") callback >
1  import { setLightCommands } from "pages/hello-command/command-provider";
2  import {
3    Light,
4
5    TurnOnLightCommand,
6    TurnOffLightCommand,
7    TurnOnRedLightCommand,
8    IncreaseRedLightCommand,
9    DecreaseRedLightCommand,
10 } from "patterns/command/command-lightChange";
11
12  describe("light command pattern", () => {
13    let light = new Light();
14
15    let count = 0;
16
17    test("light on", () => {
18      let expectation = setLightCommands("on", count);
19      let reality = new TurnOnLightCommand(light);
20      expect(expectation).toEqual(reality.setCommand());
21    });
22
23    test("light off", () => {
24      let expectation = setLightCommands("off", count);
25      let reality = new TurnOffLightCommand(light);
26      expect(expectation).toEqual(reality.setCommand());
27    });
28
29    test("red light on", () => {
30      let expectation = setLightCommands("redLight", count);
31      let reality = new TurnOnRedLightCommand(light);
32      expect(expectation).toEqual(reality.setCommand());
33    });
34  });
```

Run | Debug

```
29 test("red light on", () => {
30     let expectation = setLightCommands("redLight", count);
31     let reality = new TurnOnRedLightCommand(light);
32     expect(expectation).toEqual(reality.setCommand());
33 });
34
```

Run | Debug

```
35 test("increase red light", () => {
36     count++;
37     let expectation = setLightCommands("increase", count);
38     let reality = new IncreaseRedLightCommand(light);
39     expect(expectation).toEqual(reality.setCommand());
40 });
41
```

Run | Debug

```
42 test("decrease red light", () => {
43     count++;
44     let expectation = setLightCommands("decrease", count);
45     let reality = new DecreaseRedLightCommand(light);
46     expect(expectation).toEqual(reality.setCommand());
47 });
48 });
49
```

UML diagram

