Как и когда использовать поведенческий шаблон проектирования «Команда»

Содержание

Знакомство с кодом проекта

1. Ознакомтесь с кодом задания 1 (src/task1) и задания 2 (src/task2)
2. Запустите проект на выполнение
3. Раскомментируйте код в src/app.ts
4. Посмотрите результаты в консоли браузера

Задание 1. Реализация команд и макрокоманд

1. Создайте папку **src/task1/commands**.
2. Создайте файл **src/task1/commands/command.interface.ts** и добавьте в него следующий фрагмент кода:

```
export interface Command {
    execute(): void;
    stop(): void;
}
```

3. Создайте команды **AdvertizeCommand, CodeCommand, TestCommand, MacroCommand** в виде классов, которые реализуют интерфейс **Command.** Для этого создайте 4 файла
    a. commands/advertize-command.class.ts
    b. commands/code-command.class.ts
    c. commands/test-command.class.ts
    d. commands/macro-command.class.ts

    и разместите в них следующий код:

commands/advertize-command.class.ts

```
import { MarketingExpert } from '../receivers-workers';
import { Command } from './command.interface';

export class AdvertizeCommand implements Command {
    constructor(private marketingExpert: MarketingExpert) {
    }

    execute(): void {
        this.marketingExpert.startAdvertizing();
    }

    stop(): void {
        this.marketingExpert.stopAdvertizing();
    }
}
```

commands/code-command.class.ts

```
import { SoftwareEngineer } from '../receivers-workers';
import { Command } from './command.interface';

export class CodeCommand implements Command {
    constructor(private softwareEngineer: SoftwareEngineer) {
    }
```

```
    execute(): void {
        this.softwareEngineer.startCodding();
    }

    stop(): void {
        this.softwareEngineer.stopCodding();
    }
}
```

commands/test-command.class.ts

```
import { SoftwareTestingEngineer } from '../receivers-workers';
import { Command } from './command.interface';

export class TestCommand implements Command {
    constructor(private softwareTestingEngineer: SoftwareTestingEngineer) {
    }

    execute(): void {
        this.softwareTestingEngineer.startTesting();
    }

    stop(): void {
        this.softwareTestingEngineer.stopTesting();
    }
}
```

commands/macro-command.class.ts

```
import { Command } from './command.interface';

export class MacroCommand implements Command {
    constructor(private commands: Array<Command>) {
    }

    execute(): void {
        this.commands.forEach(command => command.execute());
    }

    stop(): void {
        this.commands.forEach(command => command.stop());
    }
}
```

4. Создайте файл **commands/index.ts**, используя следующий фрагмент кода

```
export * from './advertize-command.class';
export * from './code-command.class';
export * from './command.interface';
export * from './macro-command.class';
export * from './test-command.class';
```

5. Создайте файл **src/invoker/manager.ts**, используя следующий фрагмент кода

```
import { Command } from "../commands";
```

```
export class Manager {
    command!: Command;

    setCommand(c: Command): void {
        this.command = c;
    }

    start(): this {
        this.command?.execute();
        return this;
    }

    stop(): this {
        this.command?.stop();
        return this;
    }
}
```

6. Внесите изменения в файл invoker/index.ts, используя следующий фрагмент кода

```
export * from './manager.class';
```

7. Добавьте в приложение (файл **app.ts**) следующий фрагмент кода, импортируйте недостающий классы:

```
const softwareEngineer1 = new SoftwareEngineer();
    const softwareEngineer2 = new SoftwareEngineer();
    const softwareTestingEngineer = new SoftwareTestingEngineer();
    const marketingExpert = new MarketingExpert();

    const commands: Array<Command> = [
        new CodeCommand(softwareEngineer1),
        new CodeCommand(softwareEngineer2),
        new TestCommand(softwareTestingEngineer),
        new AdvertizeCommand(marketingExpert)
    ];

    // case 1
    const manager = new Manager();
    manager.setCommand(new MacroCommand(commands));
    manager.start();
    manager.stop();


    // case 2
    const codeCommand1 = new CodeCommand(softwareEngineer1);
    const codeCommand2 = new CodeCommand(softwareEngineer2);
    manager.setCommand(codeCommand1);
    manager.start();
    manager.stop();
    manager.setCommand(codeCommand2);
    manager.start();
```

```
manager.stop();
```

8. Запустите проект, посмотрите результат работы приложения в консоли браузера

Задание 2. Реализация хранения и отмены команд

1. Создайте папку **src/task2/commands**.
2. Создайте файл **src/task2/commands/command.interface.ts** и добавьте в него следующий фрагмент кода:

```
export interface Command {
    execute(): void;
    undo(): void;
}
```

3. Создайте команды **NoCommand, TVCommand, VolumeCommand,** в виде классов, которые реализуют интерфейс **Command.** Для этого создайте 3 файла
    a. commands/no-command.class.ts
    b. commands/tv-command.class.ts
    c. commands/volume-command.class.ts

    и разместите в них следующий код:

commands/no-command.class.ts

```
import { Command } from './command.interface';

export class NoCommand implements Command {
    execute(): void { }
    undo(): void {}
}
```

commands/tv-command.class.ts

```
import { TV } from '../receiver-workers';
import { Command } from './command.interface';

export class TVCommand implements Command {

    constructor(private tv: TV) { }

    execute(): void {
        this.tv.on();
    }

    undo(): void {
        this.tv.off();
    }
}
```

commands/volume-command.class.ts

```
import { Volume } from '../receiver-workers';
import { Command } from './command.interface';

export class VolumeCommand implements Command {

    constructor(private volume: Volume) {}

    execute(): void {
        this.volume.raiseLevel();
    }

    undo(): void {
        this.volume.dropLevel();
    }
}
```

4. Создайте файл **commands/index.ts**, используя следующий фрагмент кода

```
export * from './command.interface';
export * from './no-command.class';
export * from './tv-command';
export * from './volume-command.class';
```

5. Создайте файл **src/invoker/pult.ts**, используя следующий фрагмент кода

```
import { Command, NoCommand } from './../commands';

export class Pult {
    buttons: Command[] = [];
    commandsHistory: Array<Command> = [];

    constructor() {
        for (let i = 0; i < 2; i++) {
            this.buttons[i] = new NoCommand();
        }
    }

    setCommand(i: number, command: Command) {
        this.buttons[i] = command;
    }

    pressButton(i: number): void {
        this.buttons[i].execute();
        this.commandsHistory.push(this.buttons[i]);
    }

    pressUndoButton() {
        if (this.commandsHistory.length > 0) {
            const undoCommand = this.commandsHistory.pop();
            undoCommand?.undo();
        }
    }

}
```

6. Внесите изменения в файл **invoker/index.ts**, используя следующий фрагмент кода

```
export * from './pult.class';
```

7. Добавьте в приложение (файл **app.ts**) следующий фрагмент кода, импортируйте недостающий классы:

```
const tv = new TV();
    const volume = new Volume();
    const pult = new Pult();

    pult.setCommand(0, new TVCommand(tv));
    pult.setCommand(1, new VolumeCommand(volume));

    pult.pressButton(0);    // включить телевизор
    pult.pressButton(1);    // увеличить громкость
    pult.pressButton(1);    // увеличить громкость
    pult.pressButton(1);    // увеличить громкость

    pult.pressUndoButton(); // уменьшить громкость
    pult.pressUndoButton(); // уменьшить громкость
    pult.pressUndoButton(); // уменьшить громкость
    pult.pressUndoButton(); // выключить телевизор
```

8. Запустите проект, посмотрите результат работы приложения в консоли браузера