Slip 6

6.1

// Command interface

interface Command {

void execute();

}

// Concrete Command classes

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOn();

}

}

class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

light.turnOff();

}

}

// Receiver class

class Light {

public void turnOn() {

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

}

public void turnOff() {

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

}

}

// Invoker class

class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

// Client class to test the Remote Control with Command Pattern

public class RemoteControlTest {

public static void main(String[] args) {

// Creating the Light and the corresponding Command objects

Light livingRoomLight = new Light();

LightOnCommand livingRoomLightOn = new LightOnCommand(livingRoomLight);

LightOffCommand livingRoomLightOff = new LightOffCommand(livingRoomLight);

// Creating the Remote Control

RemoteControl remoteControl = new RemoteControl();

// Setting the command for the remote control

remoteControl.setCommand(livingRoomLightOn);

// Pressing the button on the remote control

remoteControl.pressButton();

// Changing the command for the remote control

remoteControl.setCommand(livingRoomLightOff);

// Pressing the button again on the remote control

remoteControl.pressButton();

}

}

6.2

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset=pd.read\_csv('Position\_Salaries.csv')

x=dataset.iloc[:,1:-1].values

y=dataset.iloc[:,-1].values

print(dataset.head(5))

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

p\_r=PolynomialFeatures(degree=4)

x\_poly=p\_r.fit\_transform(x)

lin\_reg=LinearRegression()

lin\_reg.fit(x\_poly,y)

LinearRegression()

y\_pred=lin\_reg.predict(x\_poly)

df=pd.DataFrame({'Real Values':y,'Predicted Values':y\_pred})

print(df)

x\_grid=np.arange(min(x),max(x),0.1)

x\_grid=x\_grid.reshape((len(x\_grid),1))

plt.scatter(x,y,color='yellow')

plt.scatter(x,y\_pred,color='red')

plt.plot(x\_grid,lin\_reg.predict(p\_r.fit\_transform(x\_grid)),color='black')

plt.title('Polynomial Regression')

plt.xlabel('position level')

plt.ylabel('Salary')

plt.show()

6.3

const http = require('http');

const fs = require('fs');

const path = require('path');

const server = http.createServer((req, res) => {

// Extract the requested file path from the URL

const filePath = path.join(\_\_dirname, req.url);

// Read the file and send its content to the client

fs.readFile(filePath, 'utf8', (err, data) => {

if (err) {

// Handle 404 error if the file is not found

if (err.code === 'ENOENT') {

res.writeHead(404, { 'Content-Type': 'text/plain' });

res.end('404 Not Found');

} else {

// Handle other errors

res.writeHead(500, { 'Content-Type': 'text/plain' });

res.end('500 Internal Server Error');

}

} else {

// Send the file content to the client

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end(data);

}

});

});

const PORT = 3000;

server.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});