Slip 26

26.1

// Define the Duck interface

interface Duck {

void display();

void performFly();

void performQuack();

void setFlyBehavior(FlyBehavior flyBehavior);

void setQuackBehavior(QuackBehavior quackBehavior);

}

// Define the FlyBehavior interface

interface FlyBehavior {

void fly();

}

// Define the QuackBehavior interface

interface QuackBehavior {

void quack();

}

// Concrete implementation of FlyBehavior for flying

class FlyWithWings implements FlyBehavior {

@Override

public void fly() {

System.out.println("Flying with wings");

}

}

// Concrete implementation of FlyBehavior for not flying

class FlyNoWay implements FlyBehavior {

@Override

public void fly() {

System.out.println("I can't fly");

}

}

// Concrete implementation of QuackBehavior for quacking

class Quack implements QuackBehavior {

@Override

public void quack() {

System.out.println("Quack");

}

}

// Concrete implementation of QuackBehavior for not quacking

class MuteQuack implements QuackBehavior {

@Override

public void quack() {

System.out.println("<< Silence >>");

}

}

// Concrete implementation of Duck

class MallardDuck implements Duck {

private FlyBehavior flyBehavior;

private QuackBehavior quackBehavior;

public MallardDuck() {

// Default behaviors

this.flyBehavior = new FlyWithWings();

this.quackBehavior = new Quack();

}

@Override

public void display() {

System.out.println("Displaying Mallard Duck");

}

@Override

public void performFly() {

flyBehavior.fly();

}

@Override

public void performQuack() {

quackBehavior.quack();

}

@Override

public void setFlyBehavior(FlyBehavior flyBehavior) {

this.flyBehavior = flyBehavior;

}

@Override

public void setQuackBehavior(QuackBehavior quackBehavior) {

this.quackBehavior = quackBehavior;

}

}

public class DuckSimulator {

public static void main(String[] args) {

Duck mallardDuck = new MallardDuck();

mallardDuck.display();

mallardDuck.performFly();

mallardDuck.performQuack();

// Change fly behavior dynamically

mallardDuck.setFlyBehavior(new FlyNoWay());

mallardDuck.performFly();

}

}

26.2

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset=pd.read\_csv('50\_Startups.csv')

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

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

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[3])

],remainder='passthrough')

x=np.array(ct.fit\_transform(x))

print(x)

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2)

from sklearn.linear\_model import LinearRegression

regressor=LinearRegression()

regressor.fit(x\_train,y\_train)

LinearRegression()

y\_pred=regressor.predict(x\_test)

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

print(df)

26.3

const mysql = require('mysql2');

// Create a connection to the MySQL server

const connection = mysql.createConnection({

host: 'localhost', // Change this to your MySQL server host

user: 'root', // Change this to your MySQL username

password: 'password', // Change this to your MySQL password

});

// Connect to MySQL server

connection.connect((err) => {

if (err) {

console.error('Error connecting to MySQL server: ', err);

return;

}

console.log('Connected to MySQL server');

// Create a new database

connection.query('CREATE DATABASE IF NOT EXISTS mydatabase', (createDbErr) => {

if (createDbErr) {

console.error('Error creating database: ', createDbErr);

connection.end(); // Close the connection

return;

}

console.log('Database created or already exists');

// Use the newly created database

connection.query('USE mydatabase', (useDbErr) => {

if (useDbErr) {

console.error('Error selecting database: ', useDbErr);

connection.end(); // Close the connection

return;

}

console.log('Using database: mydatabase');

// Create a new table

const createTableQuery = `

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL

)

`;

connection.query(createTableQuery, (createTableErr) => {

if (createTableErr) {

console.error('Error creating table: ', createTableErr);

} else {

console.log('Table "users" created or already exists');

}

// Close the connection

connection.end((endErr) => {

if (endErr) {

console.error('Error closing connection: ', endErr);

} else {

console.log('Connection closed');

}

});

});

});

});

});