Slip 9

9.1

// Strategy interface for flying behavior

interface FlyBehavior {

void fly();

}

// Concrete implementations of flying behavior

class FlyWithWings implements FlyBehavior {

@Override

public void fly() {

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

}

}

class FlyNoWay implements FlyBehavior {

@Override

public void fly() {

System.out.println("Unable to fly");

}

}

// Strategy interface for quacking behavior

interface QuackBehavior {

void quack();

}

// Concrete implementations of quacking behavior

class Quack implements QuackBehavior {

@Override

public void quack() {

System.out.println("Quack");

}

}

class MuteQuack implements QuackBehavior {

@Override

public void quack() {

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

}

}

// Context class (Duck)

class Duck {

private FlyBehavior flyBehavior;

private QuackBehavior quackBehavior;

public Duck(FlyBehavior flyBehavior, QuackBehavior quackBehavior) {

this.flyBehavior = flyBehavior;

this.quackBehavior = quackBehavior;

}

public void performFly() {

flyBehavior.fly();

}

public void performQuack() {

quackBehavior.quack();

}

public void swim() {

System.out.println("All ducks float, even decoys!");

}

// Setter methods to change behaviors dynamically

public void setFlyBehavior(FlyBehavior flyBehavior) {

this.flyBehavior = flyBehavior;

}

public void setQuackBehavior(QuackBehavior quackBehavior) {

this.quackBehavior = quackBehavior;

}

}

// Test program

public class DuckBehaviorTest {

public static void main(String[] args) {

// Create a Mallard Duck with flying and quacking behaviors

Duck mallardDuck = new Duck(new FlyWithWings(), new Quack());

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

mallardDuck.performFly();

mallardDuck.performQuack();

mallardDuck.swim();

// Change Mallard Duck's flying behavior dynamically

mallardDuck.setFlyBehavior(new FlyNoWay());

System.out.println("Mallard Duck (after changing flying behavior):");

mallardDuck.performFly();

mallardDuck.performQuack();

mallardDuck.swim();

// Create a Rubber Duck with different flying and quacking behaviors

Duck rubberDuck = new Duck(new FlyNoWay(), new MuteQuack());

System.out.println("\nRubber Duck:");

rubberDuck.performFly();

rubberDuck.performQuack();

rubberDuck.swim();

}

}

9.2

# Write a python Program to prepare scatter plot for iris dataset

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

iris = pd.read\_csv("iris.csv")

print(iris.head(20))

plt.plot(iris.Id,iris["sepal.length"],"r--")

plt.show

iris.plot(kind = "scatter", x='sepal.length', y ='petal.length')

plt.show()

9.3

const mysql = require('mysql');

// MySQL connection configuration

const connection = mysql.createConnection({

host: 'localhost',

user: 'your\_username',

password: 'your\_password',

database: 'your\_database\_name',

});

// Connect to MySQL server

connection.connect((err) => {

if (err) {

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

return;

}

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

// Insert multiple records into the "student" table

const students = [

{ name: 'John Doe', age: 20, grade: 'A' },

{ name: 'Jane Smith', age: 22, grade: 'B' },

{ name: 'Bob Johnson', age: 21, grade: 'C' },

];

const insertQuery = 'INSERT INTO student (name, age, grade) VALUES ?';

connection.query(insertQuery, [students.map(student => [student.name, student.age, student.grade])], (err, result) => {

if (err) {

console.error('Error inserting records:', err.message);

connection.end();

return;

}

console.log(`${result.affectedRows} records inserted into the "student" table`);

console.log('Result object:', result);

// Close the MySQL connection

connection.end((err) => {

if (err) {

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

} else {

console.log('Connection closed');

}

});

});

});