Q.1 :Can we overload or overide static methods?

Answer: 'Yes'. We can have two or more static methods with the same name, but differences in input parameters.No. Static methods in Java cannot be overridden. This is because static methods are not associated with the instance of a class, but with the class itself.Method hiding can be defined as, "if a subclass defines a static method with the same signature as a static method in the super class, in such a case, the method in the subclass hides the one in the superclass." The mechanism is known as method hiding.

Q.2

final methods can be overloaded but they cannot be overridden. It means a class can have more than one final methods of the same name.

Q.3

inheritance is about creating a hierarchy of classes where subclasses inherit properties and behaviors from superclasses, while polymorphism is about allowing objects of different classes to be used interchangeably.Java disallows multiple inheritance to avoid the complexity and ambiguity associated with it, particularly the "diamond problem,"

Q.4

The answer is 40.

Q.5

Add the super(I) in the class B default constructor.

Coding Question Q.1

import java.util.ArrayList;

class Passenger {

private String name;

private int age;

private String destination;

public Passenger(String n, int a, String d) {

name = n;

age = a;

destination = d;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

public String getDestination() {

return destination;

}

}

abstract class Bike {

protected String make;

protected String model;

protected int year;

protected int passengerCount;

protected ArrayList<Passenger> passengers;

public Bike(String mk, String mdl, int yr) {

make = mk;

model = mdl;

year = yr;

passengerCount = 0;

passengers = new ArrayList<>();

}

public String getMake() {

return make;

}

public String getModel() {

return model;

}

public int getYear() {

return year;

}

public int getPassengerCount() {

return passengerCount;

}

public abstract void addPassenger(Passenger p);

public abstract int calculateFare(double distance, boolean isPeakHour);

public void printPassengers() {

System.out.println("Passengers: ");

for (int i = 0; i < passengerCount; i++) {

System.out.println(i + 1 + " " + passengers.get(i).getName() + " is dropped to " + passengers.get(i).getDestination());

}

System.out.println();

}

}

class Yango extends Bike {

public Yango(String mk, String mdl, int yr) {

super(mk, mdl, yr);

}

@Override

public void addPassenger(Passenger p) {

if (passengerCount < 2) {

passengers.add(p);

passengerCount++;

} else {

System.out.println("Sorry, this car is already full.");

}

}

@Override

public int calculateFare(double distance, boolean isPeakHour) {

int baseFare = 50;

double distanceRate = 5.5;

double peakHourRate = 1.2;

int fare = (int) (baseFare + (distanceRate \* distance));

if (isPeakHour) {

fare = (int) (fare \* peakHourRate);

}

return fare;

}

}

class Bykea extends Bike {

public Bykea(String mk, String mdl, int yr) {

super(mk, mdl, yr);

}

@Override

public void addPassenger(Passenger p) {

if (passengerCount < 3) {

passengers.add(p);

passengerCount++;

} else {

System.out.println("Sorry, this car is already full.");

}

}

@Override

public int calculateFare(double distance, boolean isPeakHour) {

int baseFare = 40;

double distanceRate = 6.0;

double peakHourRate = 1.3;

int fare = (int) (baseFare + (distanceRate \* distance));

if (isPeakHour) {

fare = (int) (fare \* peakHourRate);

}

return fare;

}

}

public class Main {

public static void main(String[] args) {

// create some passengers

Passenger p1 = new Passenger("Alice", 25, "Airport");

Passenger p2 = new Passenger("Bob", 35, "Downtown");

Passenger p3 = new Passenger("Charlie", 20, "Shopping Mall");

// create some Bikes

Yango u1 = new Yango("Toyota", "Corolla", 2018);

Bykea c1 = new Bykea("Honda", "City", 2019);

// add passengers to the Bikes

u1.addPassenger(p1);

u1.addPassenger(p2);

c1.addPassenger(p2);

c1.addPassenger(p3);

// calculate fares for the Bikes

double u1\_distance = 10.5; // kilometers

boolean u1\_is\_peak\_hour = true;

int u1\_fare = u1.calculateFare(u1\_distance, u1\_is\_peak\_hour);

System.out.println("Yango fare for " + u1\_distance + " km distance: $" + u1\_fare);

double c1\_distance = 8.2; // kilometers

boolean c1\_is\_peak\_hour = false;

int c1\_fare = c1.calculateFare(c1\_distance, c1\_is\_peak\_hour);

System.out.println("Bykea fare for " + c1\_distance + " km distance: $" + c1\_fare);

// print the passengers in the Bikes

u1.printPassengers();

c1.printPassengers();

}

}

Q.2 Coding Solution

class Pastry {

protected String name;

protected String ingredients;

protected double productionCost;

public Pastry(String pastryName, String pastryIngredients, double cost) {

name = pastryName;

ingredients = pastryIngredients;

productionCost = cost;

}

public double calculateTotalCost() {

return productionCost \* 1.10; // 10% markup for labor and expenses

}

public double calculateRetailPrice() {

return calculateTotalCost(); // No tax by default

}

public double calculateProfit() {

double retailPrice = calculateRetailPrice();

double profit = (retailPrice - calculateTotalCost()) \* 0.70; // Shop retains 70% profit

return profit;

}

}

class SweetPastry extends Pastry {

private double salesTaxRate; // Tax rate for sweet pastries

public SweetPastry(String pastryName, String pastryIngredients, double cost, double taxRate) {

super(pastryName, pastryIngredients, cost);

salesTaxRate = taxRate;

}

@Override

public double calculateRetailPrice() {

double totalCost = calculateTotalCost();

double retailPrice = totalCost \* (1 + salesTaxRate);

return retailPrice;

}

}

class SavoryPastry extends Pastry {

private double salesTaxRate; // Tax rate for savory pastries

public SavoryPastry(String pastryName, String pastryIngredients, double cost, double taxRate) {

super(pastryName, pastryIngredients, cost);

salesTaxRate = taxRate;

}

@Override

public double calculateRetailPrice() {

double totalCost = calculateTotalCost();

double retailPrice = totalCost \* (1 + salesTaxRate);

return retailPrice;

}

}

protected class PastryReport {

private static double totalSales = 0.0;

private static double totalProfit = 0.0;

public static void addSale(double salesAmount, double profitAmount) {

totalSales += salesAmount;

totalProfit += profitAmount;

}

public static void generateReport() {

System.out.println("Total Sales: " + totalSales);

System.out.println("Total Profit: " + totalProfit);

}

}

class PastryShop {

private Pastry[] pastries = new Pastry[2]; // Assuming only 2 pastries for simplicity

public void addPastry(Pastry pastry, int index) {

if (index >= 0 && index < 2) {

pastries[index] = pastry;

}

}

protected void calculateTotalProfit() {

for (int i = 0; i < 2; ++i) {

if (pastries[i] != null) {

double profit = pastries[i].calculateProfit();

double salesAmount = pastries[i].calculateRetailPrice();

PastryReport.addSale(salesAmount, profit);

}

}

}

public void generateSalesReport() {

PastryReport.generateReport();

}

}

public class Main {

public static void main(String[] args) {

// Create sweet pastry

SweetPastry sweetPastry = new SweetPastry("Chocolate Cake", "Flour, Sugar, Cocoa Powder", 20.0, 0.10);

// Create savory pastry

SavoryPastry savoryPastry = new SavoryPastry("Spinach Pie", "Spinach, Cheese, Dough", 15.0, 0.15);

// Create pastry shop

PastryShop pastryShop = new PastryShop();

// Add pastries to the pastry shop

pastryShop.addPastry(sweetPastry, 0);

pastryShop.addPastry(savoryPastry, 1);

// Calculate total profit

pastryShop.calculateTotalProfit();

// Generate sales report

pastryShop.generateSalesReport();

}

}