Teori Tugas 3

Disusun untuk memenuhi tugas mata kuliah Pemrograman Berorientasi Objek (Teori)

Disusun Oleh:

Dwika Ali Ramdhan (231511042)

2B – D3

Jurusan Teknik Komputer dan Informatika

Politeknik Negeri Bandung

2024

DAFTAR ISI

PEMBAHASAN

1.1. Setup Software Engine

```
C:\Users\bushi>java -version
java version "22.0.2" 2024-07-16
Java(TM) SE Runtime Environment (build 22.0.2+9-70)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.2+9-70, mixed mode, sharing)
```

2.

1.2. Dependency

1.2.1. public class Dependency {

```
// Kelas Engine yang merepresentasikan mesin mobil
static class Engine {
    public void start() {
        System.out.println("Engine is starting...");
    }

    public void stop() {
        System.out.println("Engine is stopping...");
    }
}

// Kelas Wheel yang merepresentasikan roda mobil
static class Wheel {
    private String position;

    public Wheel(String position) {
        this.position = position;
    }

    public void rotate() {
        System.out.println(position + " wheel is rotating.");
    }

    public void stopRotation() {
        System.out.println(position + " wheel has stopped rotating.");
    }
}

// Kelas Car yang bergantung langsung pada Engine dan Wheel
static class Car {
    private Engine engine;  // Dependency pada Engine
    private Wheel frontLeftWheel;  // Dependency pada Wheel
    private Wheel frontRightWheel; // Dependency pada Wheel
    private Wheel rearLeftWheel;   // Dependency pada Wheel
    private Wheel rearRightWheel;  // Dependency pada Wheel

    public Car() {
        // Membuat dependency di dalam class Car
        engine = new Engine();
        frontLeftWheel = new Wheel("Front Left");
        frontRightWheel = new Wheel("Front Right");
        rearLeftWheel = new Wheel("Rear Left");
        rearRightWheel = new Wheel("Rear Right");
    }

    public void startCar() {
        System.out.println("Car is starting...");
        engine.start();
        frontLeftWheel.rotate();
```

```
            frontRightWheel.rotate();
            rearLeftWheel.rotate();
            rearRightWheel.rotate();
            System.out.println("Car has started.\n");
        }

        public void stopCar() {
            System.out.println("Car is stopping...");
            frontLeftWheel.stopRotation();
            frontRightWheel.stopRotation();
            rearLeftWheel.stopRotation();
            rearRightWheel.stopRotation();
            engine.stop();
            System.out.println("Car has stopped.\n");
        }
    }

    // Program utama untuk menjalankan simulasi
    public static void main(String[] args) {
        // Membuat objek Car
        Car myCar = new Car();

        // Memulai mobil
        myCar.startCar();

        // Menghentikan mobil
        myCar.stopCar();
    }
}
```

Dependency dalam pemrograman adalah hubungan antara dua kelas atau objek di mana satu kelas memerlukan kelas lain untuk melakukan tugas tertentu. Artinya, satu objek tidak dapat bekerja sendiri tanpa bantuan objek lain. Di sini, Car tidak dapat memulai atau berhenti tanpa menggunakan metode dari Engine dan Wheel. Oleh karena itu, kelas Car memiliki dependency pada Engine dan Wheel karena tanpa mereka, Car tidak bisa berfungsi.

Dependency (ketergantungan) muncul di dalam kelas Car yang bergantung pada objek-objek dari kelas Engine dan Wheel untuk berfungsi. Kelas Car tidak bisa beroperasi tanpa keberadaan Engine dan Wheel, karena Car memerlukan mesin untuk menjalankan fungsinya dan roda untuk bergerak. Kelas Car bergantung pada objek Engine untuk memulai dan menghentikan mobil, serta bergantung pada empat objek Wheel untuk memutar dan menghentikan rotasi roda-roda.

1.2.2. Output

```
Car is starting...
Engine is starting...
Front Left wheel is rotating.
Front Right wheel is rotating.
Rear Left wheel is rotating.
Rear Right wheel is rotating.
Car has started.

Car is stopping...
Front Left wheel has stopped rotating.
Front Right wheel has stopped rotating.
Rear Left wheel has stopped rotating.
Rear Right wheel has stopped rotating.
Engine is stopping...
Car has stopped.
```

1.3. Aggregasion

1.3.1. class aggregation {

```
String date;
String candy;
int consumption;

aggregation(String date, String candy, int consumption) {
    this.date = date;
    this.candy = candy;
    this.consumption = consumption;
}

public String toString() {
    StringBuffer str = new StringBuffer();
    str.append(date);
    str.append("\t\t\t\t");
    str.append(String.valueOf(candy));
    str.append("\t\t\t\t");
    str.append(String.format("%20s", String.valueOf(consumption)));
    return str.toString();
}

public static void main(String[] args) {
    aggregation[] cc = new aggregation[9];
    cc[0] = new aggregation("27-08-2022", "skittles", 20);
    cc[1] = new aggregation("27-08-2022", "Kitkat", 10);
    cc[2] = new aggregation("27-08-2022", "Alpenliebe", 20);
    cc[3] = new aggregation("28-08-2022", "Kitkat", 30);
    cc[4] = new aggregation("28-08-2022", "Hershey's", 25);
    cc[5] = new aggregation("29-08-2022", "Kitkat", 30);
    cc[6] = new aggregation("29-08-2022", "skittles", 15);
    cc[7] = new aggregation("29-08-2022", "Alpenliebe", 20);
    cc[8] = new aggregation("29-08-2022", "Cadbury", 45);

    // Before Aggregation
    System.out.println("Date\t\t\t\tCandy\t\t\t\tConsumption");
    for (int i = 0; i < cc.length; i++) {
        System.out.println(cc[i]);
```

```
    }

    System.out.println();
    System.out.println();
    System.out.println("After Aggregation");
    System.out.println();

    // After aggregation
    aggregate(cc);
}

public static void aggregate(aggregation[] cc) {
    HashMap<String, HashMap<String, Integer>> map = new HashMap<>();

    // An arraylist to store unique dates
    ArrayList<String> dates = new ArrayList<>();

    HashMap<String, Integer> consumptionDatewise = new HashMap<>();

    HashMap<String, Integer> consumptionCandywise = new HashMap<>();

    // Populate map HashMap
    for (int i = 0; i < cc.length; i++) {
        String date = cc[i].date;
        String candy = cc[i].candy;
        int consumption = cc[i].consumption;

        if (!map.containsKey(candy)) {
            map.put(candy, new HashMap<>());
        }

        map.get(candy).put(date, consumption);

        if (!dates.contains(date)) {
            dates.add(date);
        }

        if (!consumptionDatewise.containsKey(date)) {
            consumptionDatewise.put(date, 0);
        }

        consumptionDatewise.put(date, consumptionDatewise.getOrDefault(date, 0) + consumption);
    }

    for (String candy : map.keySet()) {
        HashMap<String, Integer> candyVal = map.get(candy);
        int total = 0;
        for (String date : candyVal.keySet()) {
            total += candyVal.get(date);
        }
        consumptionCandywise.put(candy, total);
    }

    System.out.print(String.format("%-15s", "Candy/Date"));
    for (String date : dates) {
        System.out.print(date + "\t");
    }
    System.out.println("Total");

    // Printing the rest of the table
    for (String candy : map.keySet()) {
        // System.out.printf("%-4s", candy);
        System.out.print(String.format("%-15s", candy));
        HashMap<String, Integer> candyVal = map.get(candy);
        for (int i = 0; i < dates.size(); i++) {
            if (!candyVal.containsKey(dates.get(i)))
```

```
                System.out.print("0" + "\t\t");
            else
                System.out.print(candyVal.get(dates.get(i)) + "\t\t");
        }

        // Finally printing the total candywise
        System.out.println(consumptionCandywise.get(candy));
    }

    System.out.print(String.format("%-15s", "Total"));
    int total = 0;
    for (int i = 0; i < dates.size(); i++) {
        int candiesOnDate = consumptionDatewise.get(dates.get(i));
        total += candiesOnDate;
        System.out.print(candiesOnDate + "\t\t");
    }
    System.out.println(total);
  }
}
```

Agregasi adalah proses mengumpulkan data dan menggabungkannya untuk mendapatkan informasi yang lebih ringkas dan terorganisir, sering kali dengan menghitung total atau rata-rata. Di sini, agregasi dilakukan dengan mengelompokkan jumlah konsumsi permen pada setiap tanggal untuk setiap jenis permen, serta menghitung total konsumsi berdasarkan tanggal dan jenis permen.

Proses agregasi terjadi dalam metode aggregate() , yang melakukan beberapa langkah untuk mengelompokkan dan menghitung data:

HashMap map digunakan untuk mengelompokkan konsumsi permen di mana key adalah nama permen, dan value adalah HashMap lain yang mengaitkan tanggal dengan jumlah konsumsi pada tanggal tersebut.

HashMap consumptionDatewise digunakan untuk menghitung total konsumsi permen pada setiap tanggal, di mana key adalah tanggal dan value adalah total jumlah permen yang dikonsumsi pada tanggal tersebut. Ini membantu dalam mendapatkan total konsumsi untuk setiap tanggal secara keseluruhan.

HashMap consumptionCandywise digunakan untuk menghitung total konsumsi setiap jenis permen, tanpa memperhatikan tanggal. Ini berarti kita bisa mengetahui berapa total konsumsi dari setiap jenis permen secara keseluruhan.

Setelah data di- process , hasil agregasi ditampilkan dalam bentuk tabel yang mengelompokkan data konsumsi permen pada setiap tanggal dan menunjukkan total konsumsi untuk setiap jenis permen, serta total keseluruhan konsumsi per tanggal.

1.3.2. Output

```
Date                    Candy               Consumption
27-08-2022              skittles                        20
27-08-2022              Kitkat                  10
27-08-2022              Alpenliebe                      20
28-08-2022              Kitkat                  30
28-08-2022              Hershey's                       25
29-08-2022              Kitkat                  30
29-08-2022              skittles                        15
29-08-2022              Alpenliebe                      20
29-08-2022              Cadbury                 45


After Aggregation

Candy/Date      27-08-2022    28-08-2022    29-08-2022    Total
Kitkat          10            30            30            70
Cadbury         0             0             45            45
Alpenliebe      20            0             20            40
Hershey's       0             25            0             25
skittles        20            0             15            35
Total           50            55            110           215
```

1.4. Inheritance

1.4.1. class Person1 {

```
    int id;
    String name;

    void set_Person(int id, String name)
    {
        try {
            this.id = id;
            this.name = name;
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    void disp_Person()
    {
        System.out.print(id + "\t" + name + "\t");
    }
}

class Employee1 extends Person1 {
    int sal;
    String desgn;
    void set_Emp(int id, String name, String desgn, int sal)
    {
        try {
            set_Person(id, name);
            this.desgn = desgn;
            this.sal = sal;
        }
        catch (Exception ex) {
```

```
            ex.printStackTrace();
        }
    }
    void disp_Emp()
    {
        disp_Person();
        System.out.print(desgn + "\t" + sal);
    }

    public static void main(String args[])
    {

        Employee1 e1 = new Employee1();
        e1.set_Emp(1001, "Manjeet", "AP", 20000);
        e1.disp_Emp();
    }
}
```

Inheritance atau pewarisan dalam pemrograman berorientasi objek adalah mekanisme di mana sebuah kelas (kelas anak) dapat mewarisi properti (variabel) dan perilaku (metode) dari kelas lain (kelas induk). Ini memungkinkan penggunaan kembali kode yang sudah ada dan meningkatkan organisasi kode dengan membuat hierarki antar kelas. Kelas anak dapat menambahkan fungsionalitas baru atau memodifikasi yang sudah ada, seperti yang terjadi dalam kelas Employee1 yang menambahkan properti sal dan desgn , serta metode untuk mengatur dan menampilkan informasi tersebut, selain yang diwarisi dari Person1 .

Inheritance atau pewarisan terjadi ketika kelas Employee1 (kelas anak) mewarisi atribut dan metode dari kelas Person1 (kelas induk). Employee1 mewarisi dua variabel (id dan name) serta dua metode (set_Person() dan disp_Person()) dari Person1 . Kelas Employee1 juga menambahkan variabel dan metode tambahan, yaitu sal (gaji) dan desgn (jabatan), serta metode set_Emp() dan disp_Emp() untuk menangani data tambahan tersebut.

1.4.2. Output

```
1001    Manjeet AP  20000
1002    Brick   Dev 15000
```