Jobsheet 04 - Class Relations

Name : Evan Diantha Fafian
Class : SIB 2G
Absent : 09
NIM : 2341760163

I. Competence

After studying this subject, students are able to:

1. Understand the concept of class relations;
2. Implement association relations into the program.

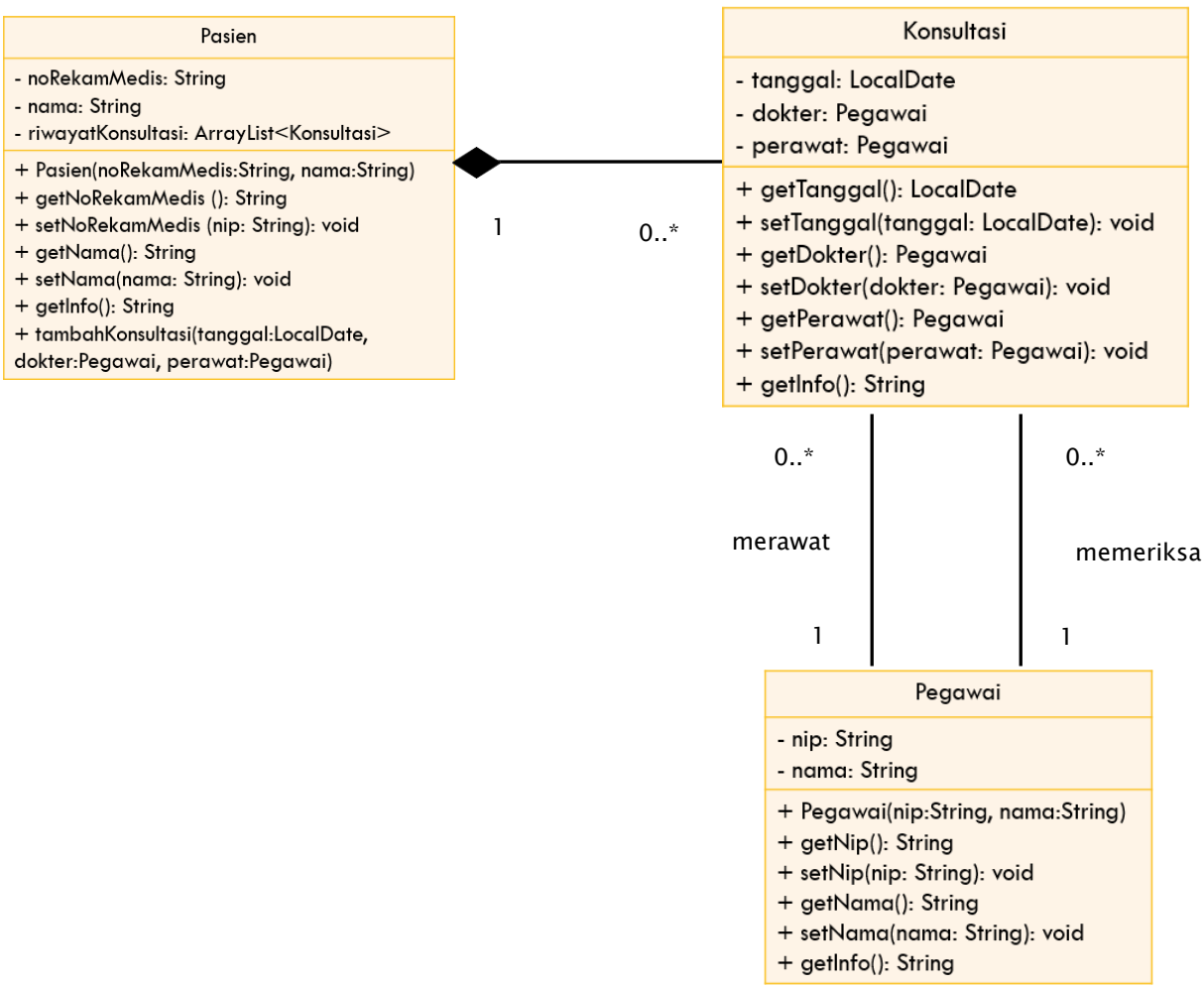
II. Introduction

In more complex cases, in a system there will be more than one *class* that is related to each other. In previous experiments, the majority of cases that have been worked on have only focused on one *class*. In this jobsheet, an experiment will be carried out involving several classes that are related to each other.

III. Practicum

In this practicum, a hospital information system will be developed that stores patient consultation history data.

Consider the following class diagram :



a. Create a new folder with the name Hospital.

b. Create an Employee class. Add nip and name attributes to Employee class with private modifier access

```
public class Pegawai {
    private String nip;
    private String nama;
}
```

c. Create a *constructor* for the Officer class with the nip and name parameters.

```
public Pegawai(String nip, String nama) {
    this.nip = nip;
    this.nama = nama;
}
```

d. Implement **setters** and **getters** for the Employee class.

```
public String getNip() {
    return nip;
}

public void setNip(String nip) {
    this.nip = nip;
}

public String getNama() {
    return nama;
}

public void setNama(String nama) {
    this.nama = nama;
}
```

e. Implement *the getInfo()* method as follows:

```
public String getInfo(){
    return nama + " (" + nip + ")";
}
```

f. Next, create a Patient class then add the noReReRecordMedical attribute and name to the Patient class with a private access level modifier. Also provide setters and getters for these two attributes.

```java
public class Pasien {
    private String noRekamMedis;
    private String nama;

    public String getNoRekamMedis() {
        return noRekamMedis;
    }

    public void setNoRekamMedis(String noRekamMedis) {
        this.noRekamMedis = noRekamMedis;
    }

    public String getNama() {
        return nama;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }
}
```

g. Create a constructor for the Patient class with the parameter noReReMedical , and the name

```java
public Pasien(String noRekamMedis, String nama) {
    this.noRekamMedis = noRekamMedis;
    this.nama = nama;
}
```

h. Implement *the getInfo()* method as follows:

```java
public String getInfo(){
    String info = "";
    info += "No Rekam Medis      : " + this.noRekamMedis + "\n";
    info += "Nama                : " + this.nama + "\n";
    info += "\n";

    return info;
}
```

i. This system will store data on every consultation that the patient conducts. Patients can have a consultation more than once. Therefore, the consultation data will be stored in the form of an ArrayList of objects of type Consultation.

j. Create a class called Consultation with date attributes of type LocalDate, doctor type employee, and nurse type employee. Set private access level modifiers for all attributes. Import `java.time.LocalDate` to declare a date attribute of type LocalDate.

```
import java.time.LocalDate;

public class Konsultasi {
    private LocalDate tanggal;
    private Pegawai dokter;
    private Pegawai perawat;
}
```

k. Provide setters and getters for each attribute in the Consult class

```
public LocalDate getTanggal() {
    return tanggal;
}

public void setTanggal(LocalDate tanggal)
    this.tanggal = tanggal;
}

public Pegawai getDokter() {
    return dokter;
}

public void setDokter(Pegawai dokter) {
    this.dokter = dokter;
}

public Pegawai getPerawat() {
    return perawat;
}

public void setPerawat(Pegawai perawat) {
    this.perawat = perawat;
}
```

l. Implement the getInfo() method as follows:

```
public String getInfo(){
    String info = "";
    info += "\tTanggal: " + tanggal;
    info += ", Dokter: " +  dokter.getInfo();
    info += ", Perawat: " + perawat.getInfo();
    info += "\n";

    return info;
}
```

m. To store patient consultation history data, add the Consultation history attribute to the Patient class with the arrayList<Consultation> type. This attribute will store a series of objects of type Consultation. Import java.util.ArrayList in order to declare an attribute of type ArrayList of object.

```
private String noRekamMedis;
private String nama;
private ArrayList<Konsultasi> riwayatKonsultasi;
```

n. Create a parameterized constructor for the Patient class. Initiation of the value of the noReRecordMedical attribute and the name based on the name attribute. Instantiate/create a new ArrayList for the Consultation history attribute;

```
public Pasien(String noRekamMedis, String nama) {
    this.noRekamMedis = noRekamMedis;
    this.nama = nama;
    this.riwayatKonsultasi = new ArrayList<Konsultasi>();
}
```

o. Import java.time.LocalDate to declare a date attribute of type LocalDate in the Patient class. Next, implement the method addConsultation() as follows:

```
public void tambahKonsultasi(LocalDate tanggal, Pegawai dokter, Pegawai perawat){
    Konsultasi konsultasi = new Konsultasi();
    konsultasi.setTanggal(tanggal);
    konsultasi.setDokter(dokter);
    konsultasi.setPerawat(perawat);
    riwayatKonsultasi.add(konsultasi);
}
```

p. Modify the getInfo() method to return patient info and a list of consultations that have been done

```
public String getInfo(){
    String info = "";
    info += "No Rekam Medis      : " + this.noRekamMedis + "\n";
    info += "Nama                : " + this.nama + "\n";

    if (!riwayatKonsultasi.isEmpty()) {
        info += "Riwayat Konsultasi :\n";

        for (Konsultasi konsultasi : riwayatKonsultasi) {
            info += konsultasi.getInfo();
        }
    }
    else{
        info += "Belum ada riwayat konsultasi";
    }

    info += "\n";

    return info;
}
```

q. Import java.time.LocalDate in order to declare a date attribute of type LocalDate in the

HospitalDemo class. Test the program that has been created by creating objects in the RumahSakit Demo class. The new object instance of type Employee with the name ani uses the Employee constructor (String nip, String name) with the value of the argument nip "1234" and the name "dr. Ani". Continue the object instantiation as follows:

```
import java.time.LocalDate;

public class RumahSakitDemo {
    Run | Debug
    public static void main(String[] args) {
        Pegawai ani = new Pegawai("1234", "dr. Ani");
        Pegawai bagus = new Pegawai("4567", "dr. Bagus");

        Pegawai desi = new Pegawai("1234", "Ns. Desi");
        Pegawai eka = new Pegawai("4567", "Ns. Eka");

        Pasien pasien1 = new Pasien("343298", "Puspa Widya");
        pasien1.tambahKonsultasi(LocalDate.of(2021 , 8 , 11), ani, desi);
        pasien1.tambahKonsultasi(LocalDate.of(2021 , 9 , 11), bagus, eka);

        System.out.println(pasien1.getInfo());

        Pasien pasien2 = new Pasien("997744", "Yenny Anggraeni");
        System.out.println(pasien2.getInfo());
    }
}
```

r. *Compile* then *run* RumahSakitDemo and get the following results:

```
No Rekam Medis      : Puspa Widya
Nama                : 343298
Riwayat Konsultasi :
        Tanggal: 2021-08-11, Dokter: dr. Ani (1234), Perawat: Ns. Desi (1234)
        Tanggal: 2021-09-11, Dokter: dr. Bagus (4567), Perawat: Ns. Eka (4567)


No Rekam Medis      : Yenny Anggraeni
Nama                : 997744
Belum ada riwayat konsultasi
```

Question

Based on experiment 1, answer the related questions:

1. In the *Employee, Patient,* and Consultation classes, there are method *setters* and *getters* for each of their attributes. What is the use of *the setter and getter* methods ?
   - Setters are used to change the value of an attribute. For example, if there is a name attribute, the setter method setName() is used to set (or change) the value of that attribute.
   - Getters are used to retrieve (access) the value of an attribute. If we want to know the

value of the name attribute, then we use the getName() method.

2. In the *Consult* class there is not explicitly a constructor with parameters. Does this mean that the Consult class doesn't have a constructor?
   - Java automatically provides a default constructor without parameters, even though the "Consult" class does not have a special constructor. This constructor allows the creation of a "Consult" object, but attributes such as "date", "doctor", and "nurse" start with default values, namely "null" for objects and "0" for primitive data types. We can write a constructor with clear parameters to initialize the attributes when the object is created if desired. Therefore, classes still have constructors even though they cannot be seen directly in the code.
3. Notice the *Consult* class, which attributes are of type *object*?
   - dokter
   - perawat
4. Pay attention to *the Consultation* class, on which line does it show that the *Consultation* class has a relationship with the *Employee* class?
   -
   ```
   private pegawai dokter;
   private pegawai perawat;
   ```
5. Notice in the *Patient* class, what does the `consultation code.getInfo()` do?
   - In the Patient class, the code consult.getInfo() is used to retrieve information from each Consultation object in the Consultationhistory list and combine it into a string containing the patient's consultation history.
6. In the `getInfo() method` in the Patient class, there is a line of code:
   `if (!historyConsultation.isEmpty())`
   What does the line do?
   - The if (!Consultation history.isEmpty()) code line in the getInfo() method in the Patient class functions to check whether the Consultation history list (which is an ArrayList of Consultation objects) is empty or not.
7. In the Patient constructor class, there is a line of code:
   this.historyConsultation = new ArrayList<>();
   What does the line do? What happens if the line is omitted?
   - Code line this.Consultationhistory = new ArrayList<>(); in the constructor of the Patient class aims to initialize the Consultation history attribute as an empty ArrayList object. This line ensures that when the Patient object is created, theConsultationhistory attribute is ready to be used to store the patient's consultation history.

IV. Assignment

Implement the case studies that have been made on the Theory PBO assignment into the program.

- mechanic class

```java
package source_code.assigment;

public class mechanic {
    private String id;
    private String name;

    public mechanic(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getInfo() {
        return name + " (ID: " + id + ")";
    }
}

```

Stores information about mechanics, such as their ID and name.

- service class

```java
package source_code.assigment;

import java.time.LocalDate;

public class service {
    private LocalDate date;
    private mechanic mechanic;
    private String description;

    public LocalDate getDate() {
        return date;
    }

    public void setDate(LocalDate date) {
        this.date = date;
    }

    public mechanic getMechanic() {
        return mechanic;
    }

    public void setMechanic(mechanic mechanic) {
        this.mechanic = mechanic;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getInfo() {
        String info = "";
        info += "\tDate: " + date;
        info += ", Mechanic: " + mechanic.getInfo();
        info += ", Service: " + description + "\n";

        return info;
    }
}

```

Session to service the car. It stores the date, the mechanic involved, and a description of the service (such as an oil change or tire rotation and others).

- car class

```java
package source_code.assigment;

import java.time.LocalDate;
import java.util.ArrayList;

public class car {
    private String numberPlate;
    private String owner;
    private ArrayList<service> serviceHistory;

    public car(String numberPlate, String owner) {
        this.numberPlate = numberPlate;
        this.owner = owner;
        this.serviceHistory = new ArrayList<>();
    }

    public String getNumberPlate() {
        return numberPlate;
    }

    public void setNumberPlate(String numberPlate) {
        this.numberPlate = numberPlate;
    }

    public String getOwner() {
        return owner;
    }

    public void setOwner(String owner) {
        this.owner = owner;
    }

    public String getInfo() {
        String info = "";
        info += "Number Plate      : " + this.numberPlate + "\n";
        info += "Owner             : " + this.owner + "\n";
        info += "\n";

        if (!serviceHistory.isEmpty()) {
            info += "Service History   :\n";

            for (service service : serviceHistory) {
                info += service.getInfo();
            }
        } else {
            info += "No service history available.";
        }

        info += "\n";

        return info;
    }

    public void addService(LocalDate date, mechanic mechanic, String description) {
        service service09 = new service();
        service09.setDate(date);
        service09.setMechanic(mechanic);
        service09.setDescription(description);
        serviceHistory.add(service09);
    }
}

```

To represent a car. It stores the license plate number and owner's name. It also has a list (serviceHistory) of all the service sessions the car has had.

- 'carServiceDemo' main class

```java
package source_code.assigment;

import java.time.LocalDate;

public class carServiceDemo {

    public static void main(String[] args) {
        mechanic adi = new mechanic("001", "Adi");
        mechanic edi = new mechanic("002", "Edi");

        car car1 = new car("N 5435 NAC", "Sumargo");
        car1.addService(LocalDate.of(2024, 8, 4), adi, "Oil change");
        car1.addService(LocalDate.of(2024, 9, 17), edi, "Tire rotation");

        System.out.println(car1.getInfo());

        car car2 = new car("N 7547 ZX", "Nurdi");
        System.out.println(car2.getInfo());
    }
}

```

To run how to use the class by creating a car object, adding a service record, and printing the information.

- Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                    Debug: carServiceDemo

PS E:\KULIAH POLINEMA\Semester 3\Pemrograman Berbasis Objek\Praktikum\week_4>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:63032' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Evan Diantha F\AppData\Roaming\Code\User\workspaceStorage\b17bf3b76386e2b67d4794bb31174090\redhat.java\jdt_ws\week_4_61250abe\bin' 'source_code.assigment.carServiceDemo'
Number Plate      : N 5435 NAC
Owner             : Sumargo

Service History   :
        Date: 2024-08-04, Mechanic: Adi (ID: 001), Service: Oil change
        Date: 2024-09-17, Mechanic: Edi (ID: 002), Service: Tire rotation


Number Plate      : N 7547 ZX
Owner             : Nurdi

No service history available.

PS E:\KULIAH POLINEMA\Semester 3\Pemrograman Berbasis Objek\Praktikum\week_4>
```