Jobsheet 04 - Class Relations

Name : Evan Diantha Fafian

Class : SIB 2G

Absent : 09

NIM : 2341760163

# Competence

After studying this subject, students are able to:

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

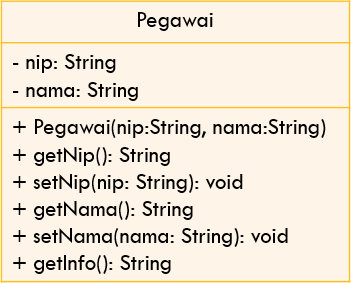
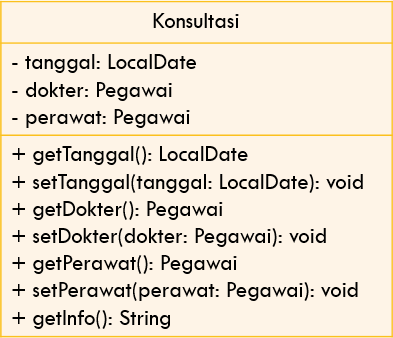
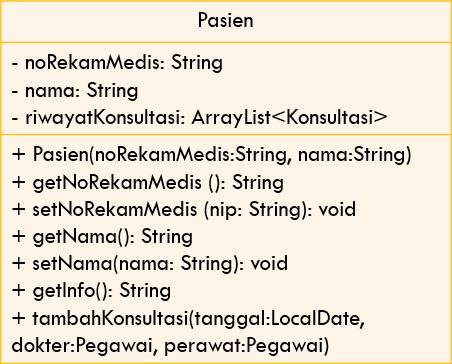
# Introduction

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.

# Practicum

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

Consider the following class diagram :



1

0..\*

0..\*

0..\*

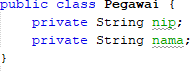
merawat

memeriksa

1

1

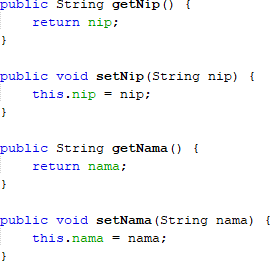
1. Create a new folder with the name Hospital.
2. Create an Employee class. Add nip and name attributes to Employee class with private modifier access



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



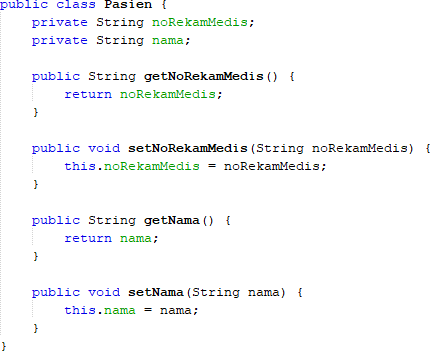
1. Implement **setters** and **getters** for the Employee class.



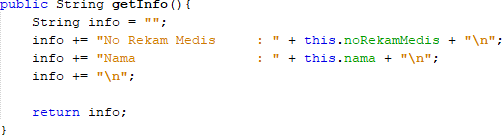
1. Implement *the getInfo()* method as follows:



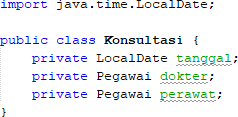
1. Next, create a Patient class then add the noReReRecordMedical attribute and name to the Patient class with a private access level modifier. Also provide setters and getters for these two attributes.
2. Create a constructor for the Patient class with the parameter noReReMedical , and the name



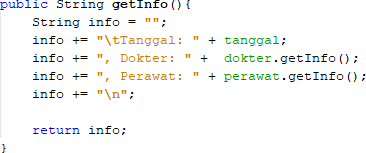
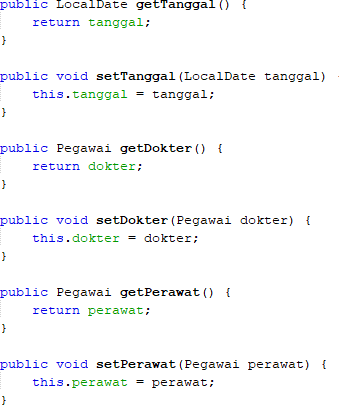
1. Implement *the getInfo()* method as follows:



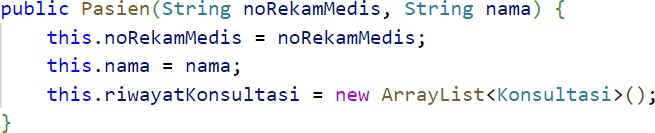
1. This system will store data on every consultation that the patient conducts. Patients can have a consultation more than once. Therefore, the consultation data will be stored in the form of an ArrayList of objects of type Consultation.
2. Create a class called Consultation with date attributes of type 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.
3. Provide setters and getters for each attribute in the Consult class



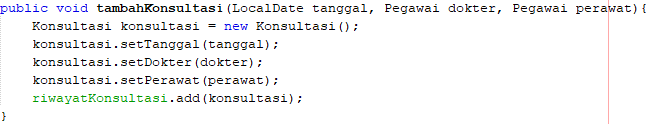
1. Implement the getInfo() method as follows:



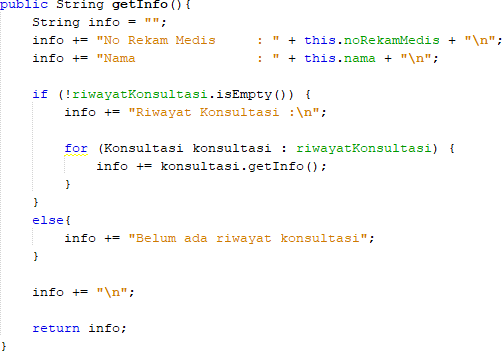
1. To store patient consultation history data, add the Consultation history attribute to the Patient class with the arrayList<Consultation> type. This attribute will store a series of objects of type Consultation. Import java.util.ArrayList in order to declare an attribute of type ArrayList of object.
2. 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;



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

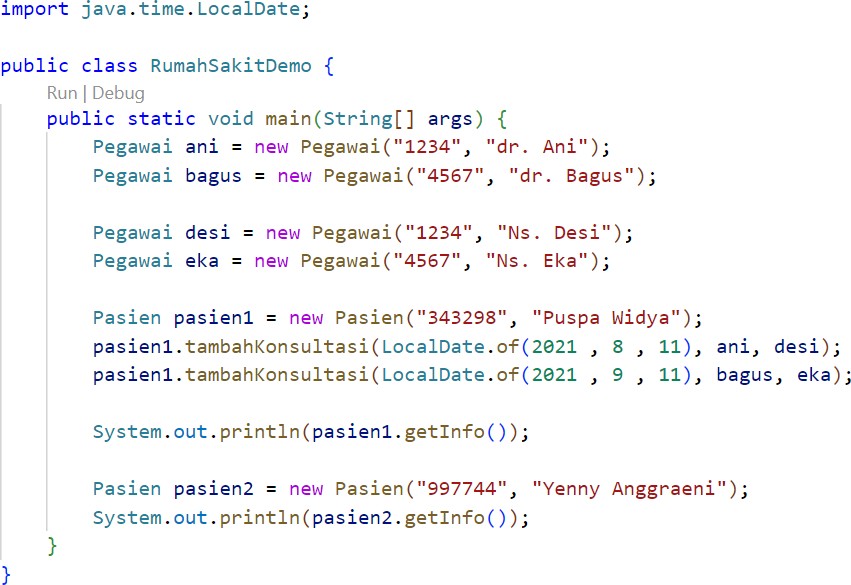


1. Modify the getInfo() method to return patient info and a list of consultations that have been done

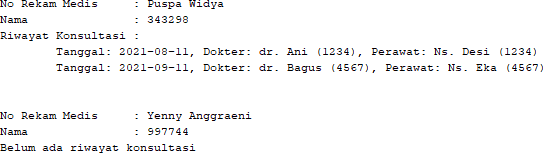


1. Import java.time.LocalDate in order to declare a date attribute of type LocalDate in the

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:



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



# 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.

1. 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.

1. Notice the *Consult* class, which attributes are of type *object*?

* dokter
* perawat

1. Pay attention to *the Consultation* class, on which line does it show that the

*Consultation* class has a relationship with the *Employee* class?

* 

1. 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.

1. 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.

1. In the Patient constructor class, there is a line of code: this.historyConsultation = new ArrayList<>();

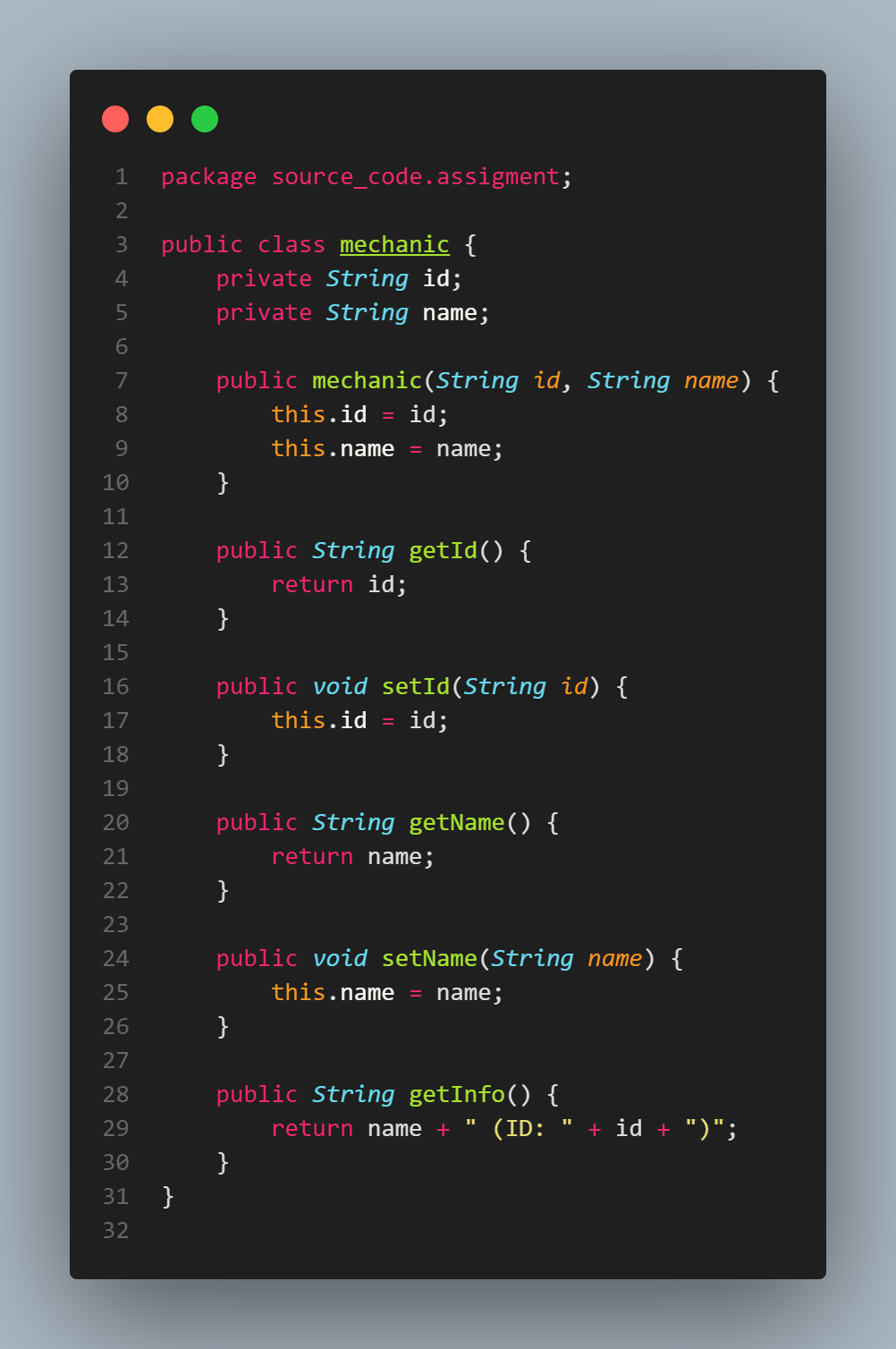
What does the line do? What happens if the line is omitted?

* 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.

# Assignment

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

* mechanic class



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

* service class



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



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



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

* Output

