Jobsheet 04 - Class Relations

# Competence

After studying this subject, students are able to:

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

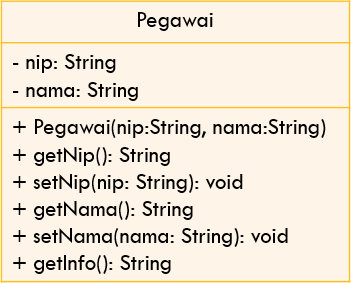
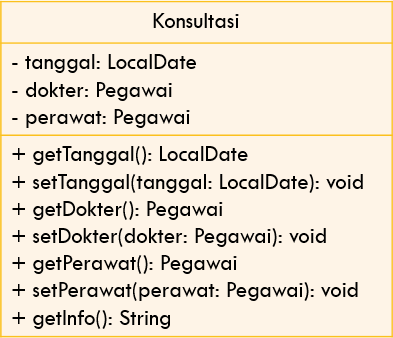
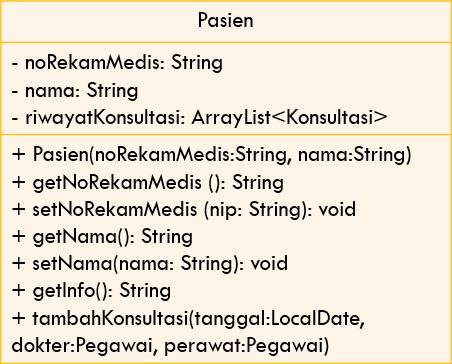
# Introduction

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..\*

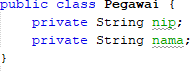
merawat

memeriksa

1

1

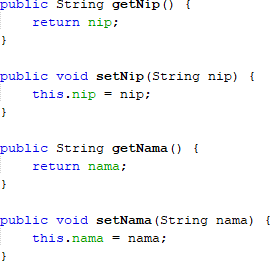
1. Create a new folder with the name Hospital.
2. Create an Employee class. Add nip and name attributes to Employee class with private modifier access



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



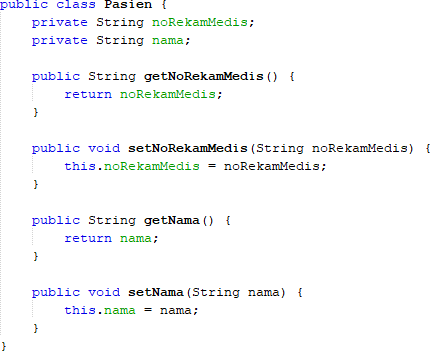
1. Implement **setters** and **getters** for the Employee class.



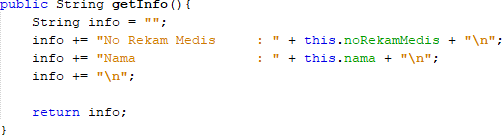
1. Implement *the getInfo()* method as follows:



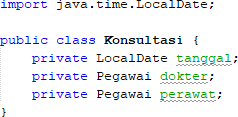
1. Next, create a Patient class then add the noReReRecordMedical attribute and name to the Patient class with a private access level modifier. Also provide setters and getters for these two attributes.
2. Create a constructor for the Patient class with the parameter noReReMedical , and the name



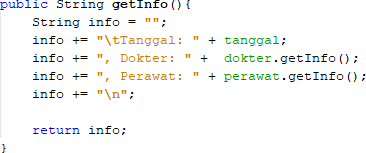
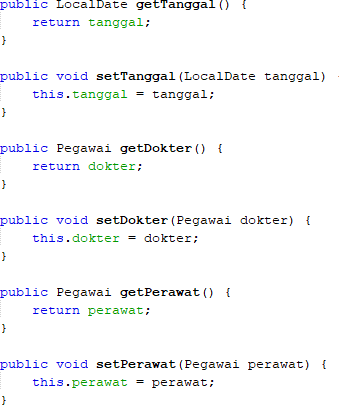
1. Implement *the getInfo()* method as follows:



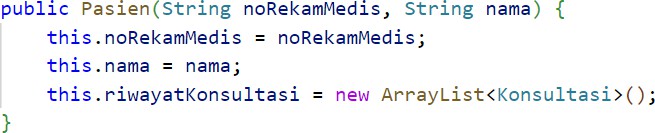
1. This system will store data on every consultation that the patient conducts. Patients can have a consultation more than once. Therefore, the consultation data will be stored in the form of an ArrayList of objects of type Consultation.
2. Create a class called Consultation with date attributes of type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



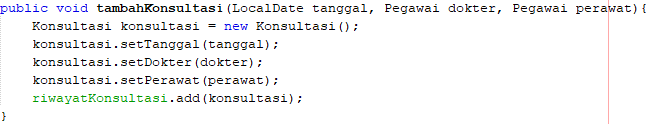
1. Implement the getInfo() method as follows:



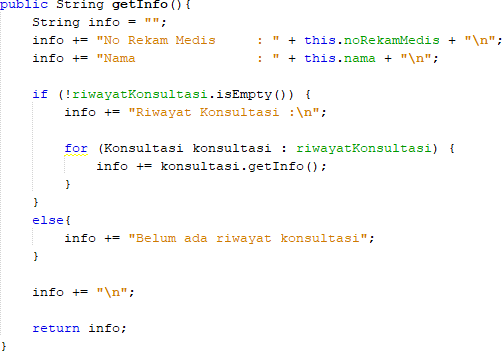
1. To store patient consultation history data, add the Consultation history attribute to the Patient class with the arrayList<Consultation> type. This attribute will store a series of objects of type Consultation. Import java.util.ArrayList in order to declare an attribute of type ArrayList of object.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

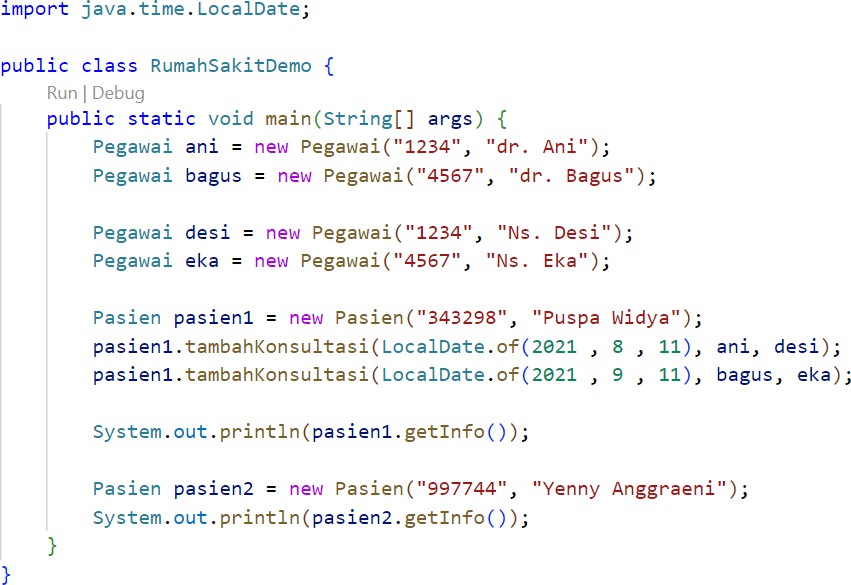


1. Modify the getInfo() method to return patient info and a list of consultations that have been done

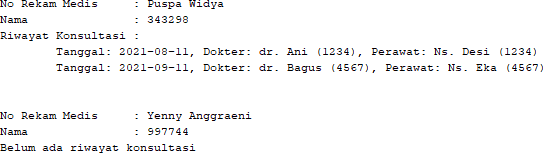


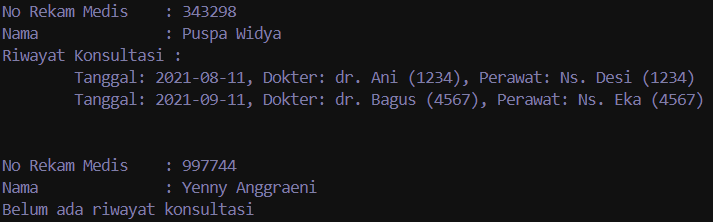
1. Import java.time.LocalDate in order to declare a date attribute of type LocalDate in the

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, while **getters** are used to read the value. They help protect the data by controlling how attributes are accessed and modified.

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?**

Yes, the Konsultasi class has a constructor. If none is defined, Java automatically provides a default no-argument constructor, allowing you to create instances like `new Konsultasi()`.

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

The attributes of the Konsultasi class, dokter and perawat, both of type Pegawai, are references to objects of the Pegawai class, making them objects rather than primitive types like int or boolean.

1. **Pay attention to *the Consultation* class, on which line does it show that the *Consultation* class has a relationship with the *Employee* class?**

**Line 8:** private Pegawai dokter;

**Line 9:** private Pegawai perawat;

Both of these lines declare attributes (dokter and perawat) as being of type Pegawai, which is a custom object class.

1. **Notice in the *Patient* class, what does the consultation code.getInfo() do?**

The getInfo() method gives you a summary of a consultation. It shows the date of the consultation and the names and IDs of the doctor and nurse involved.

1. **In the getInfo() method in the Patient class, there is a line of code:**

**if (!historyConsultation.isEmpty())**

**What does the line do?**

This line checks if there are any consultations recorded. If there are, it will show the list of consultations. If there are none, it will say there are no consultations yet.

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

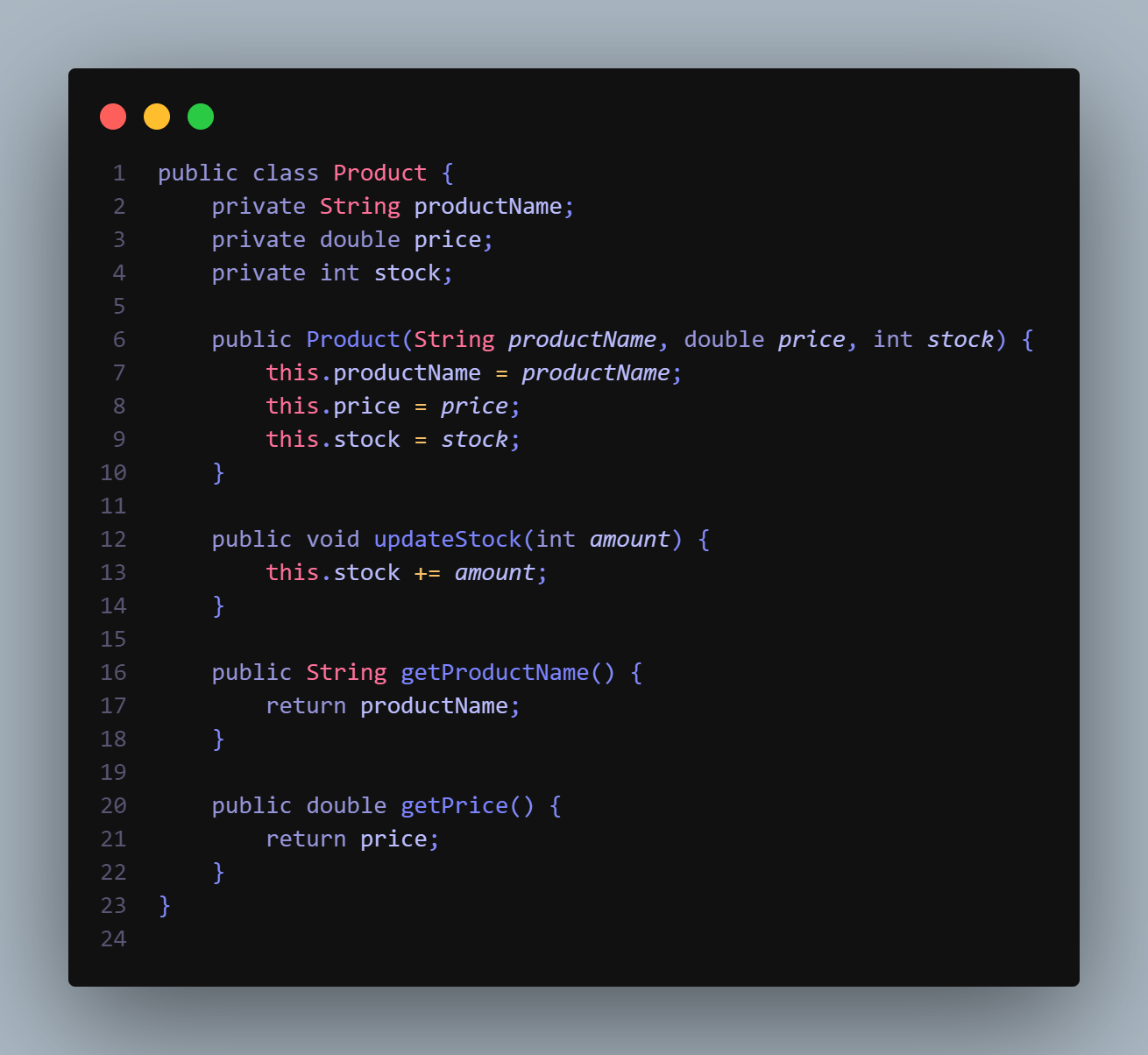
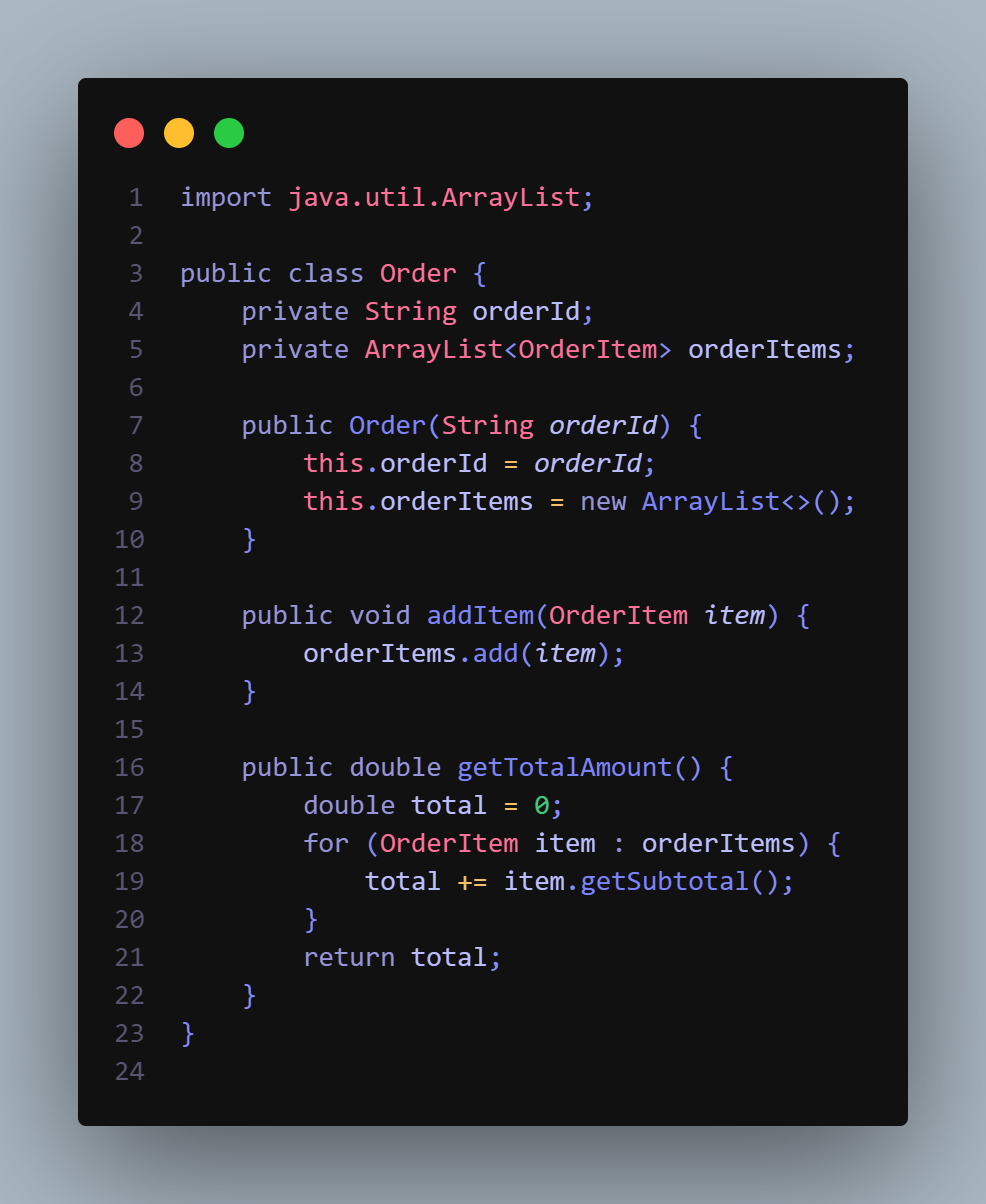
**What does the line do? What happens if the line is omitted?**

This line creates a new list to hold the patient's consultation history. If you leave this line out, the list will not be created, and trying to add consultations or check if the list is empty will cause an error (because you can't use a list that doesn't exist).

# Assignment

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

The program started by creating a market called “Traditional Market”. Within this market, there are two vendors: “John's Vegetables” and ‘Sarah's Fruits’, each with a list of products they sell. Each seller adds their products, such as tomatoes, cucumbers, apples, and bananas. The program then prints out the list of sellers and products sold by each seller in the market.



**Result :**

