**SAFRIZAL RAHMAN SIB 2G**

[**https://github.com/safrizalrahman46/PBO\_SAFRIZ\_ THEVIGILANTE**](https://github.com/safrizalrahman46/PBO_SAFRIZ_THEVIGILANTE)

**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.

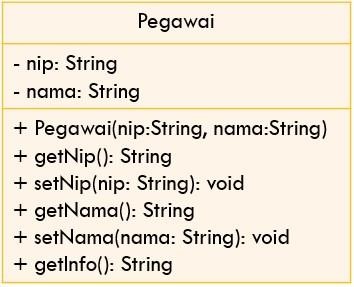
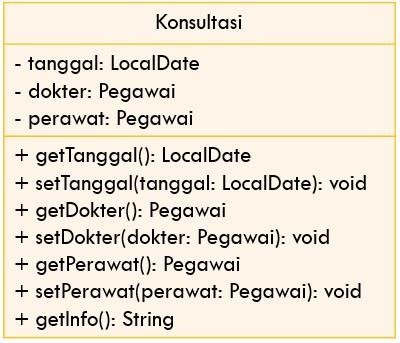
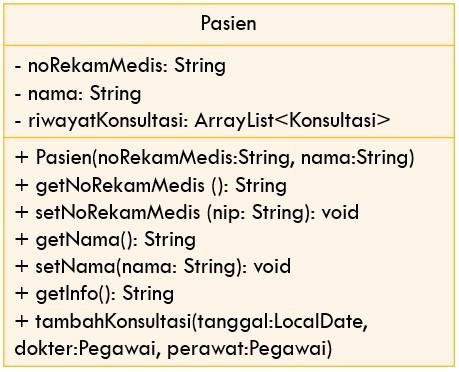
# Introduction

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 :



merawat



memeriksa



0..\*



0..\*



1



1



0..\*

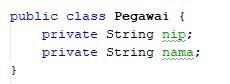


1

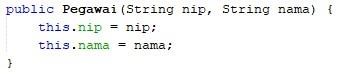


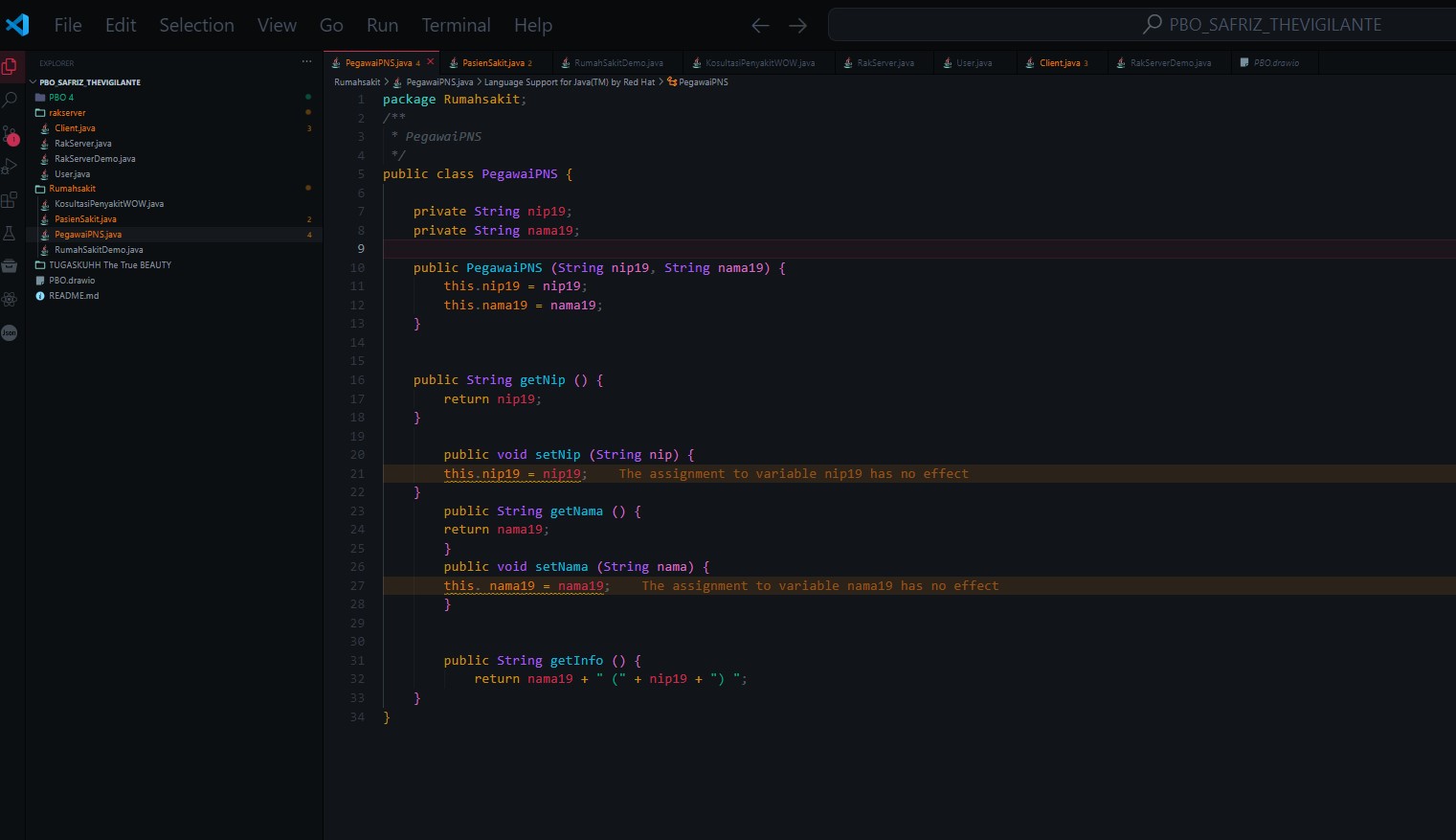
1. Create a new folder with the name Hospital.

1. 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.



e.



Implement



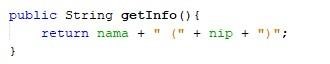
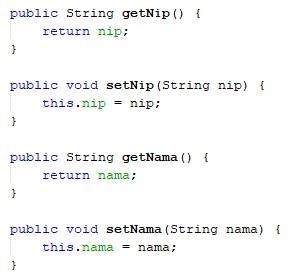
*the getInfo()*



method

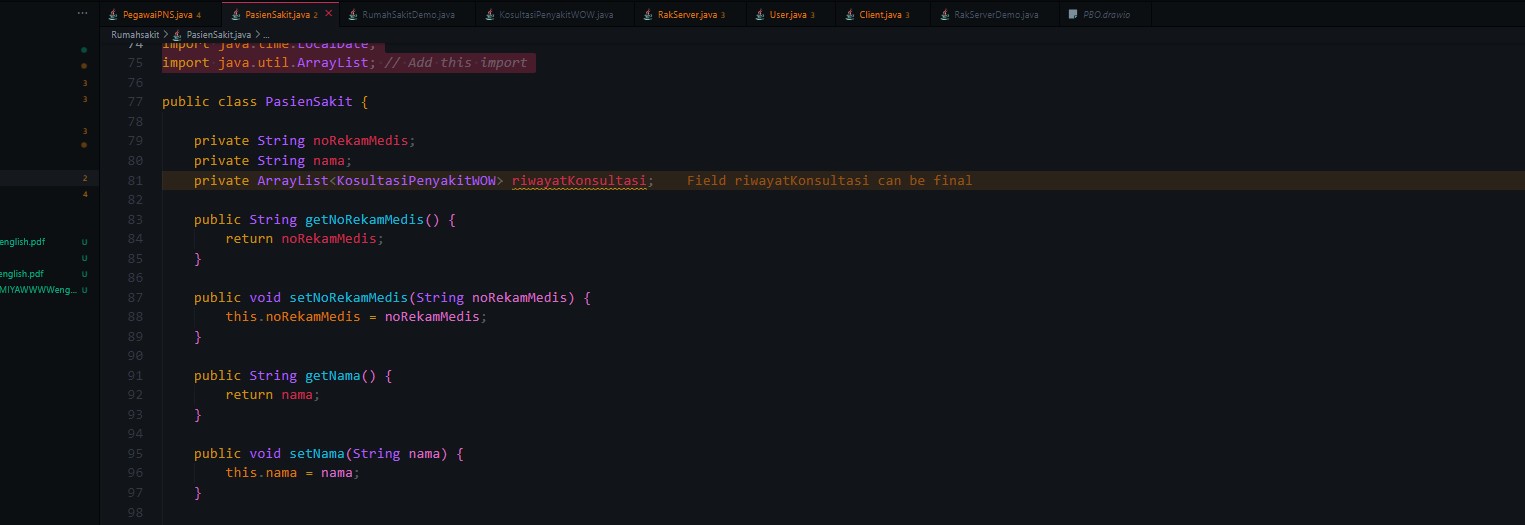


as follows:



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.

|  |  |
| --- | --- |
|  | |
|  |  |



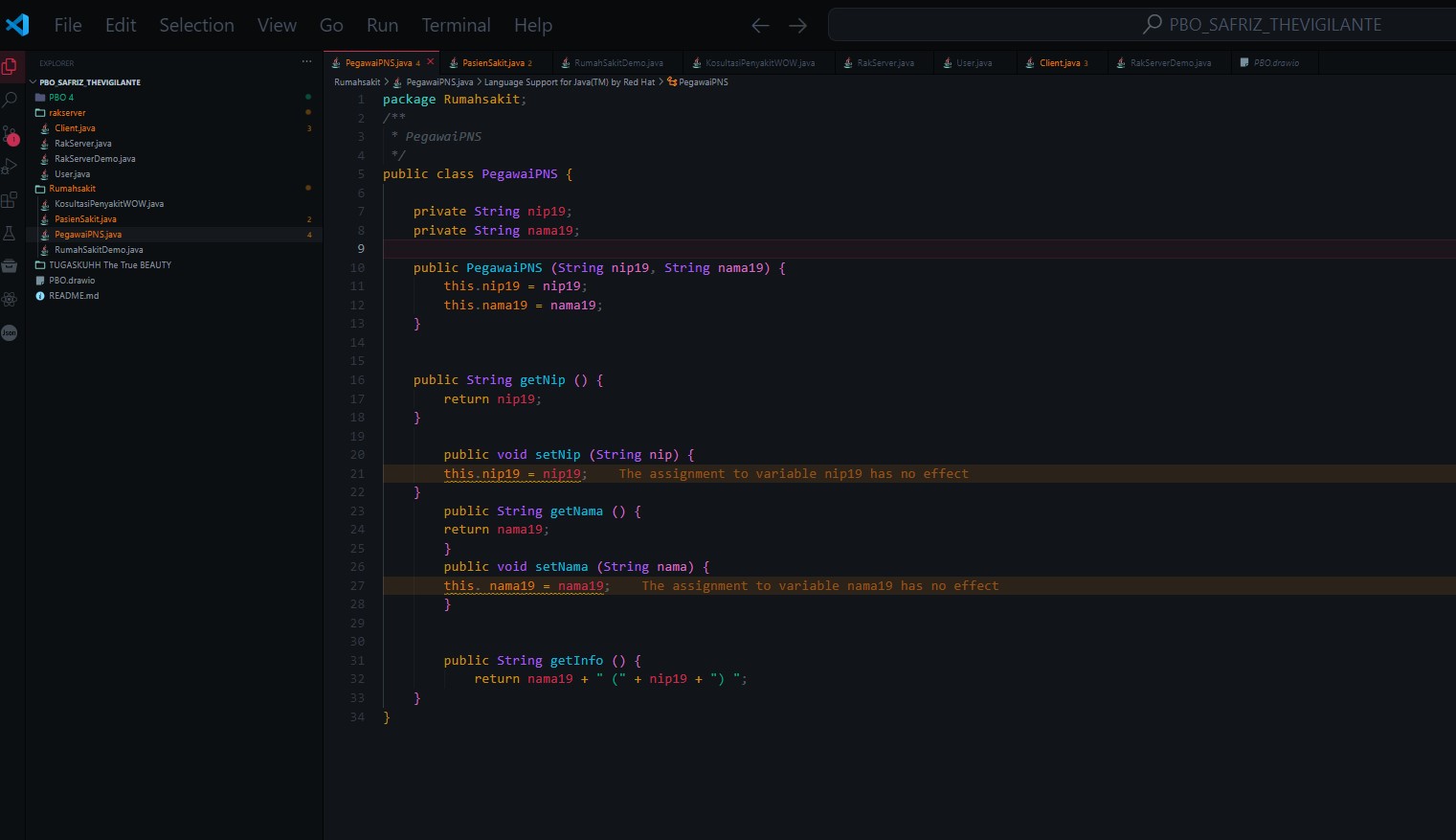
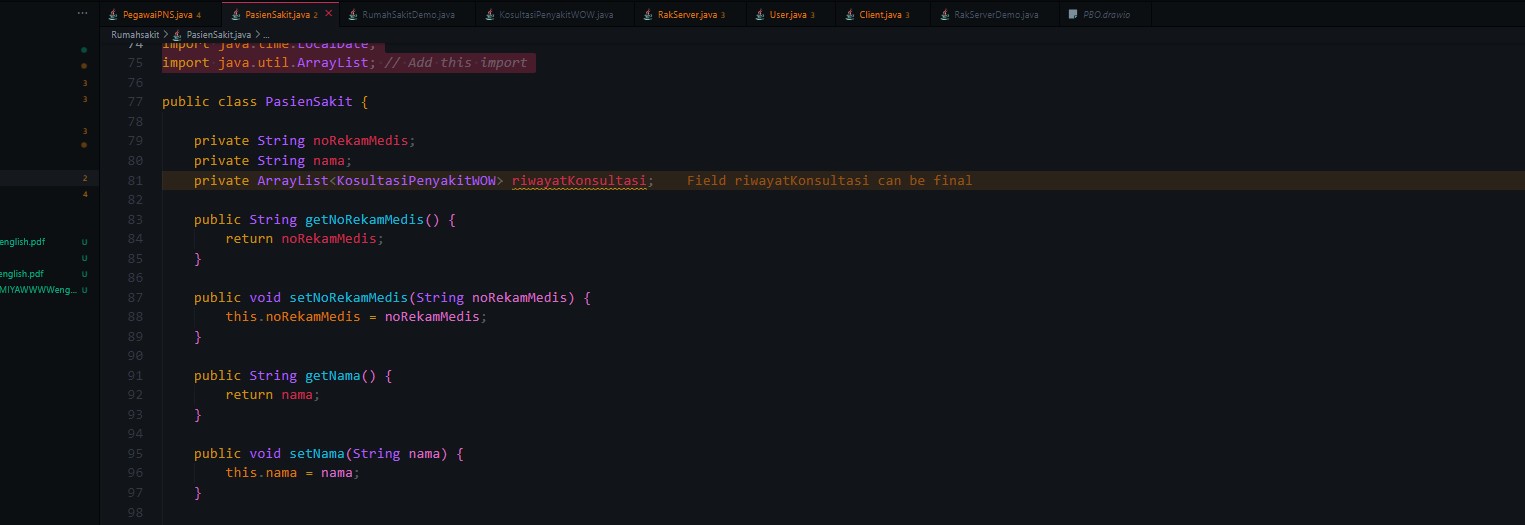
1. Create a constructor for the Patient class with the parameter noReReMedical , and the name

|  |  |
| --- | --- |
|  | |
|  |  |

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.



1. 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.

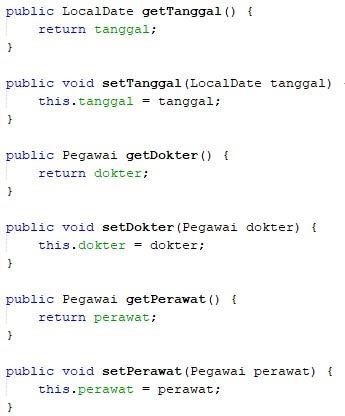
Import java.time.LocalDate to declare a date attribute of type LocalDate.

|  |  |
| --- | --- |
|  | |
|  |  |

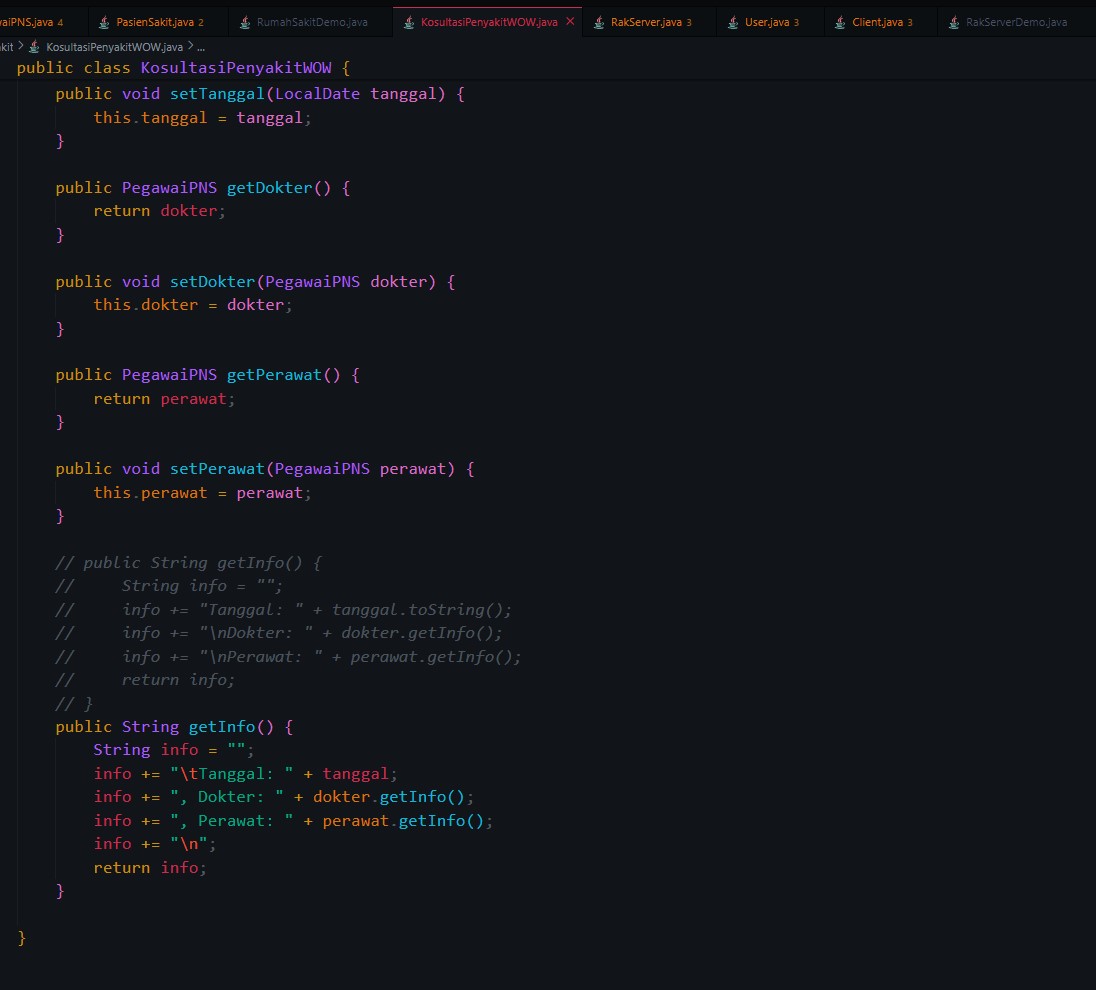
1. Provide setters and getters for each attribute in the Consult class

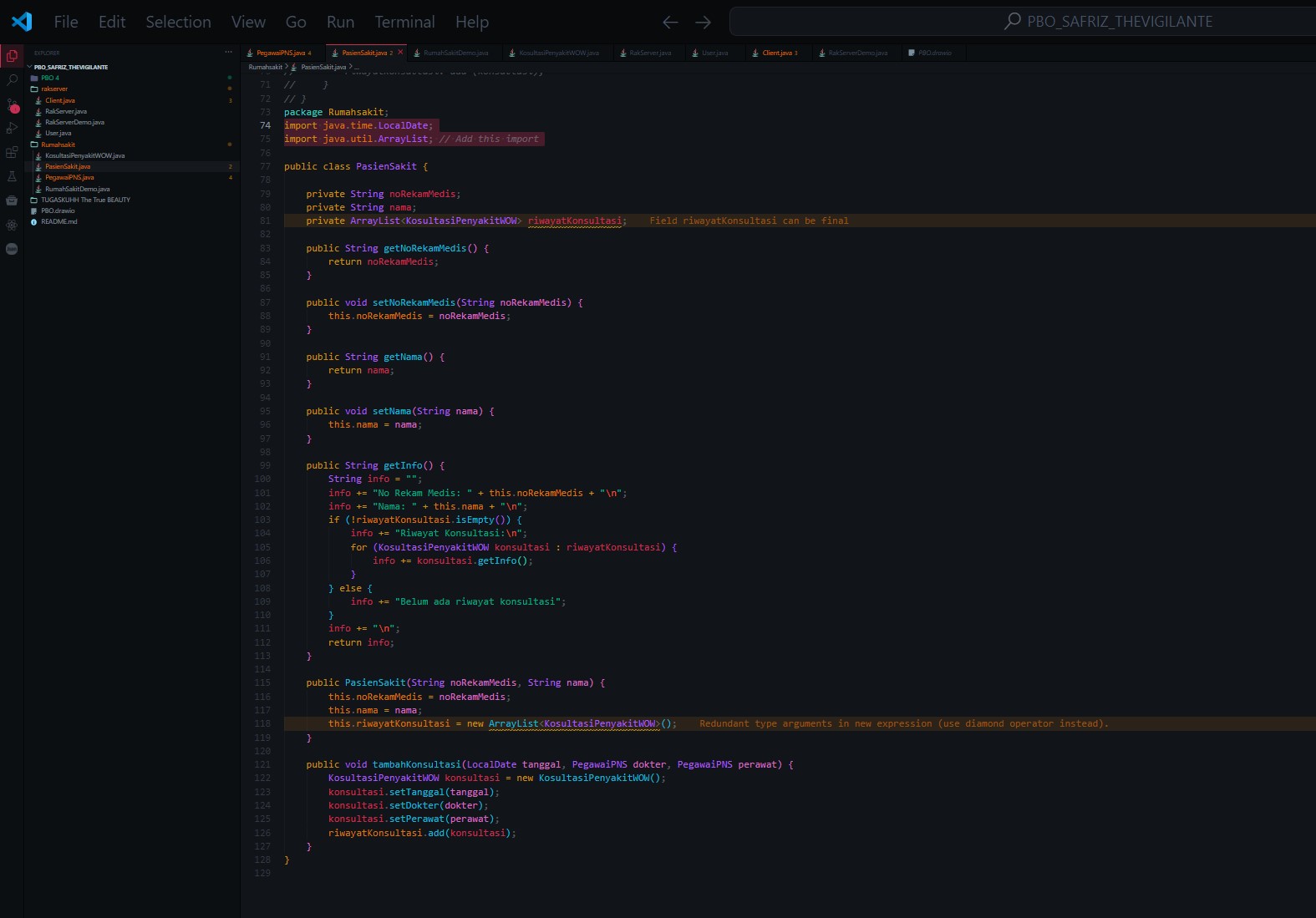
l.

Implement the getInfo() method as follows:



|  |  |
| --- | --- |
|  | |
|  |  |





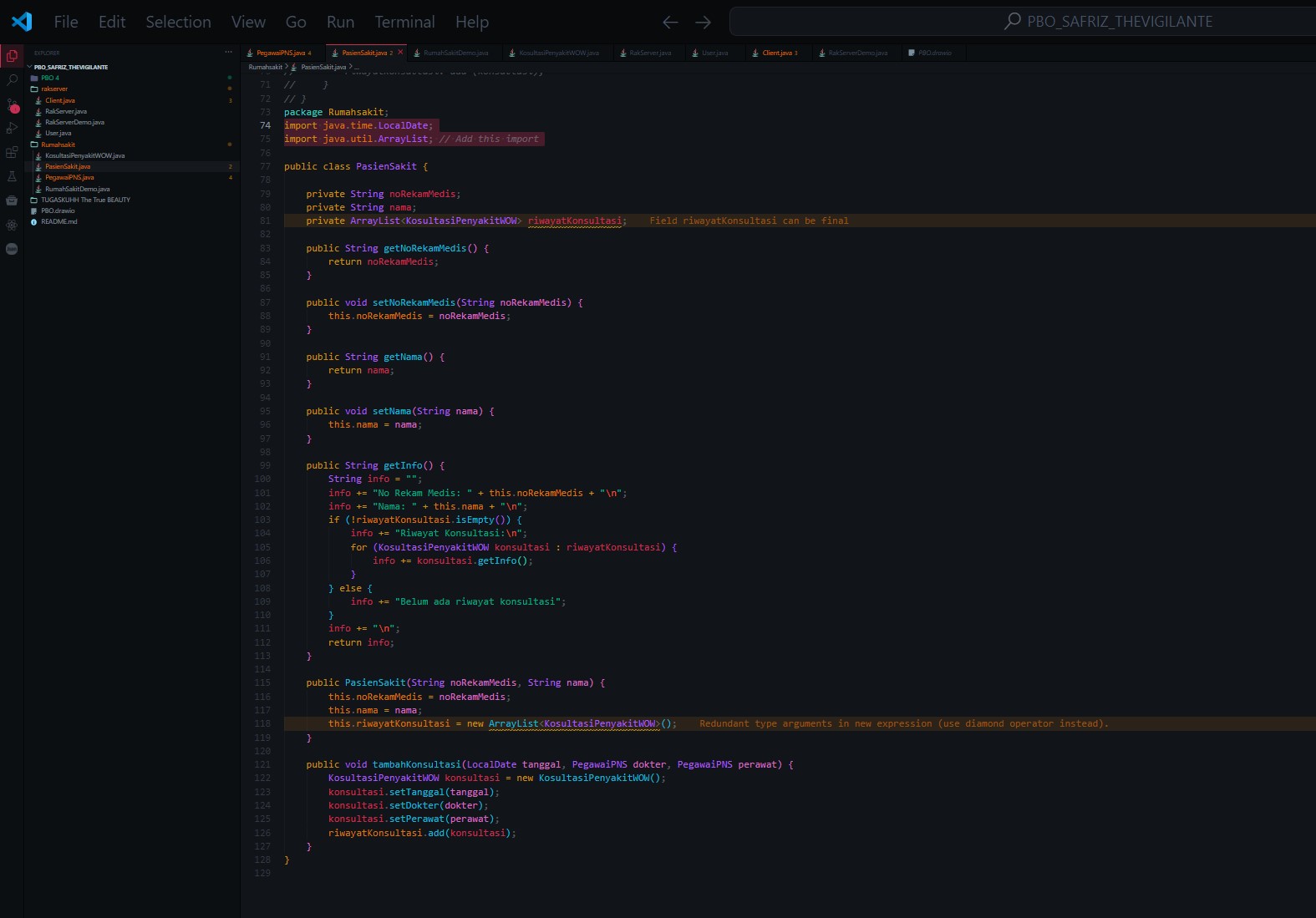
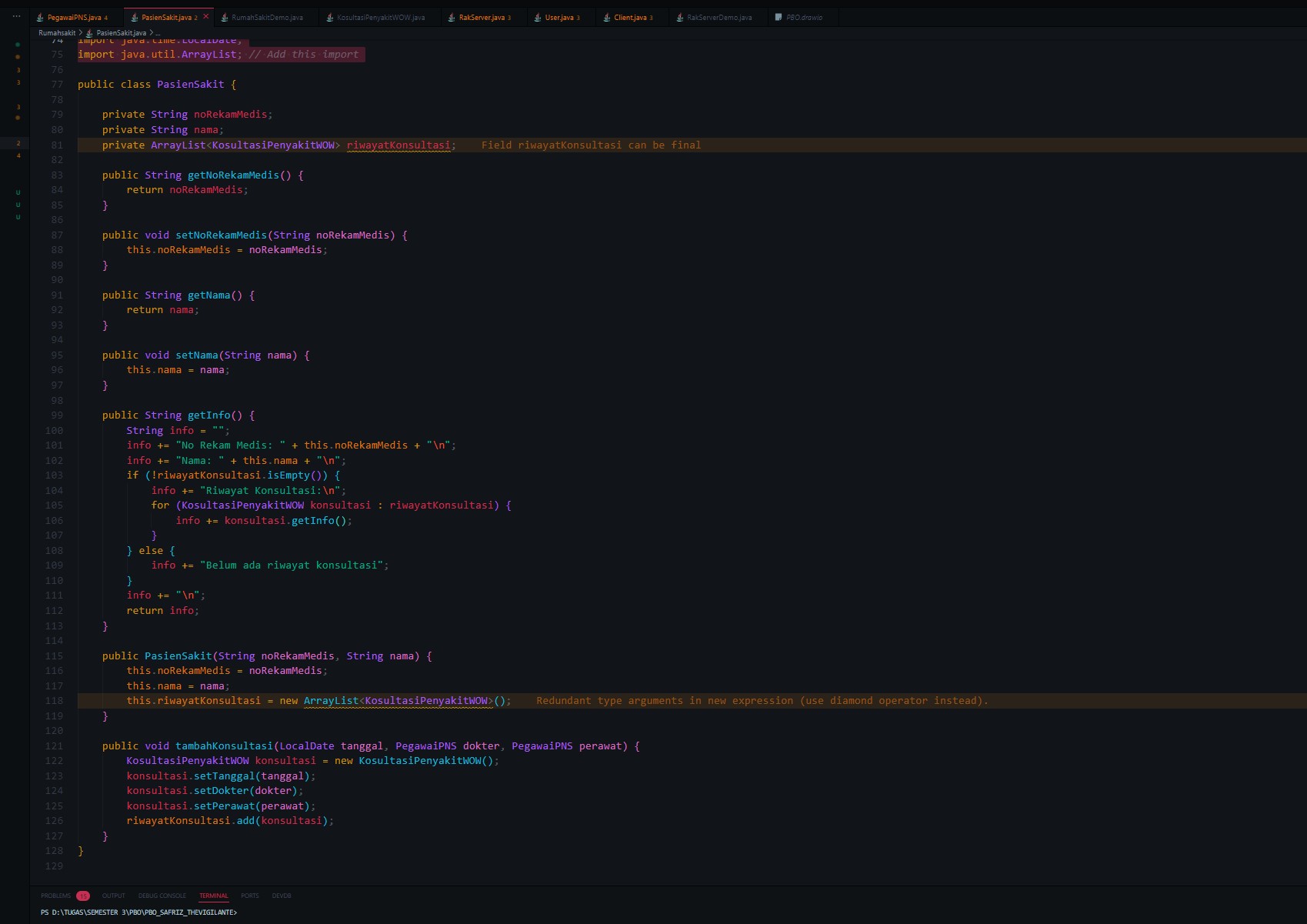
1. To store patient consultation history data, add the Consultation history attribute to the Patient class with the arrayList<Consultation> type. This attribute will store a series of objects of type Consultation. Import java.util.ArrayList in order to declare an attribute of type ArrayList of object.



1. Create a parameterized constructor for the Patient class. Initiation of the value of the noReRecordMedical attribute and the name based on the name attribute.

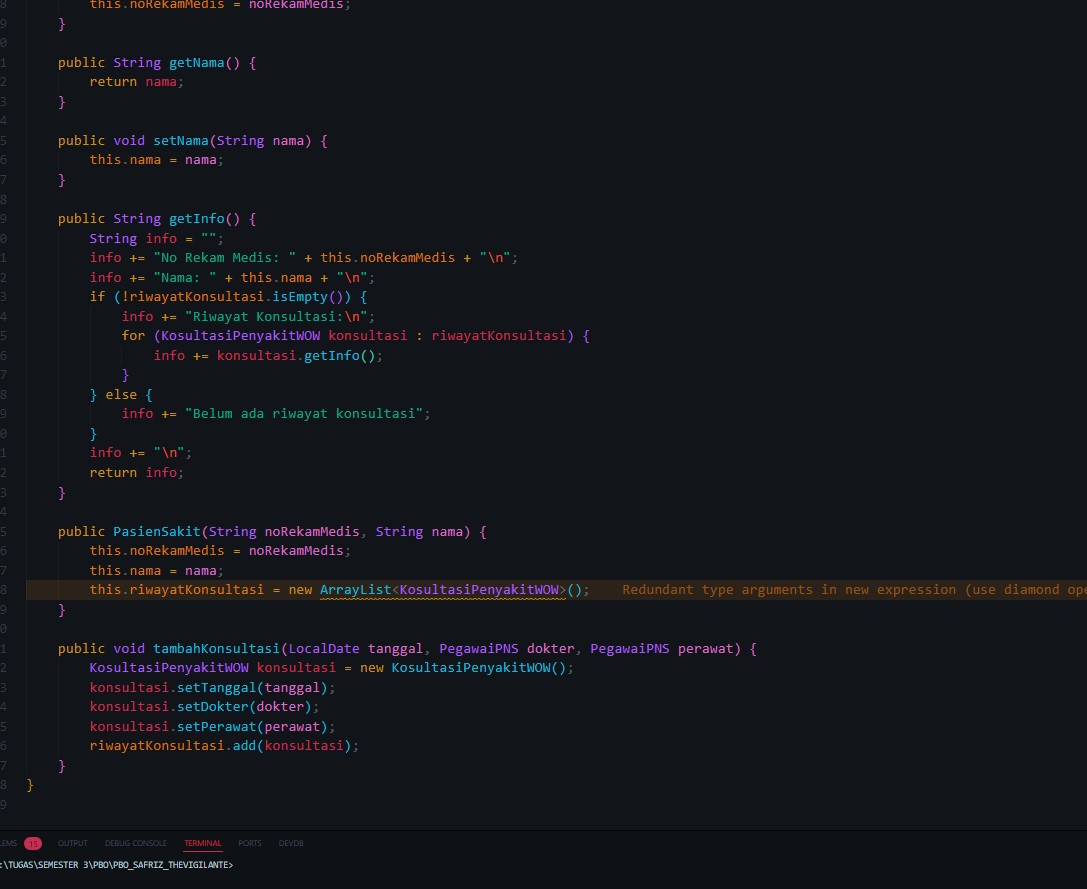
Instantiate/create a new ArrayList for the Consultation history attribute;

|  |  |
| --- | --- |
|  | |
|  |  |

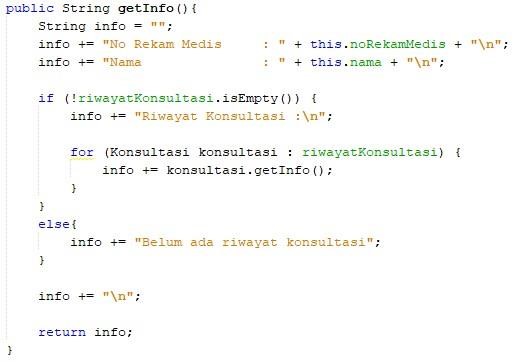


1. Import java.time.LocalDate to declare a date attribute of type LocalDate in the Patient class. Next, implement the method addConsultation() as follows:

|  |  |
| --- | --- |
|  | |
|  |  |

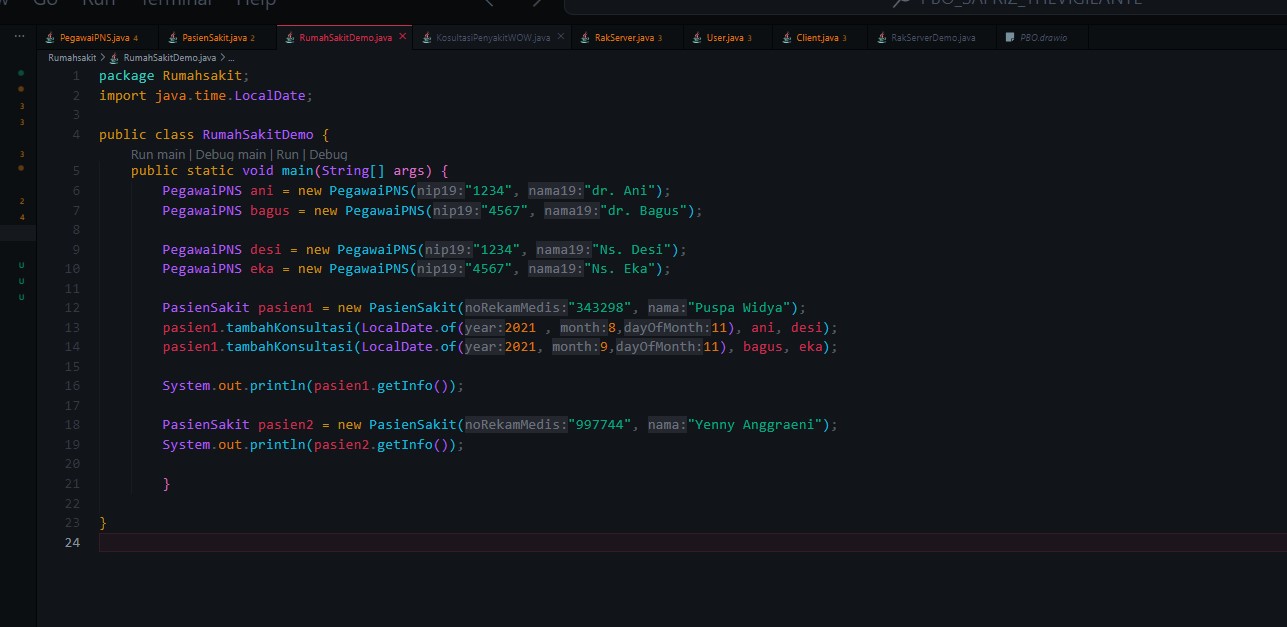


1. Modify the getInfo() method to return patient info and a list of consultations that have been done



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

ANSWER

1. Purpose of Setter and Getter Methods:

* Setter: Used to set or change the value of an object attribute. This is useful for validating or manipulating data before it is stored.
* Getter: Used to retrieve the value of an object attribute. It helps in retrieving data from an object without directly accessing its attributes, supporting encapsulation.

2. Constructors in Consultation Class:

* If the `ConsultingWOWDisease` class does not explicitly define a constructor with parameters, Java automatically provides a default constructor without parameters. Therefore, even if no constructor is defined with parameters, the class still has a default constructor. Object Type Attributes of the Consultation Class:
* In class `KonsultationPainWOW`, the attributes of the object type are:
* doctor (of type CivilServant)
* nurse (of type CivilServantEmployee) Employee Class Relationship:
* The relationship between the ConsultationPainWOW class and the Civil ServantEmployee class can be seen in the following lines in the ConsultationPainWOW class:
* doctor Private civil servant;
* private civil servant nurse;
* This shows that ConsultationDiseaseWOW has two attributes that are objects of the Civil Servant class, indicating that the ConsultationDiseaseWOW class is related to the Civil Servant class. The function of the consultation.getInfo() code in the Patient class:
* The consultation.getInfo() code in the getInfo method of the PatientSick class calls the getInfo() method of the ConsultationDiseaseWOW object, which returns information related to the consultation, including the date, doctor, and nurse. This is used to display the patient's consultation details.

6. The function of the if(!historyConsultation.isEmpty()) line:

- This line checks whether the historyConsultation list is empty or not. If it is not empty, there is a consultation history stored, and the information will be displayed. If it is empty, display a message indicating that there is no consultation history.

7. This line.historyConsultation = new ArrayList(); serves to initialize the historyConsultation as a new ArrayList when creating the PatientSick object. This ensures that historyConsultation is ready to store consultations. If this line is deleted, historyConsultation will become null and will generate a NullPointerException when trying to add a consultation.

Translated with DeepL.com (free version)

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 ?
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?
3. Notice the *Consult* class, which attributes are of type *object*?
4. 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?
2. In the getInfo() method in the Patient class, there is a line of code: if (!historyConsultation.isEmpty())

What does the line do?

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?

# Assignment

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

