**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

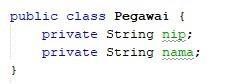
In this practicum, a hospital information system will be developed that stores patient consultation history data.

Consider the following class diagram :

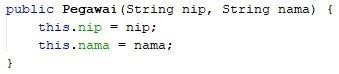


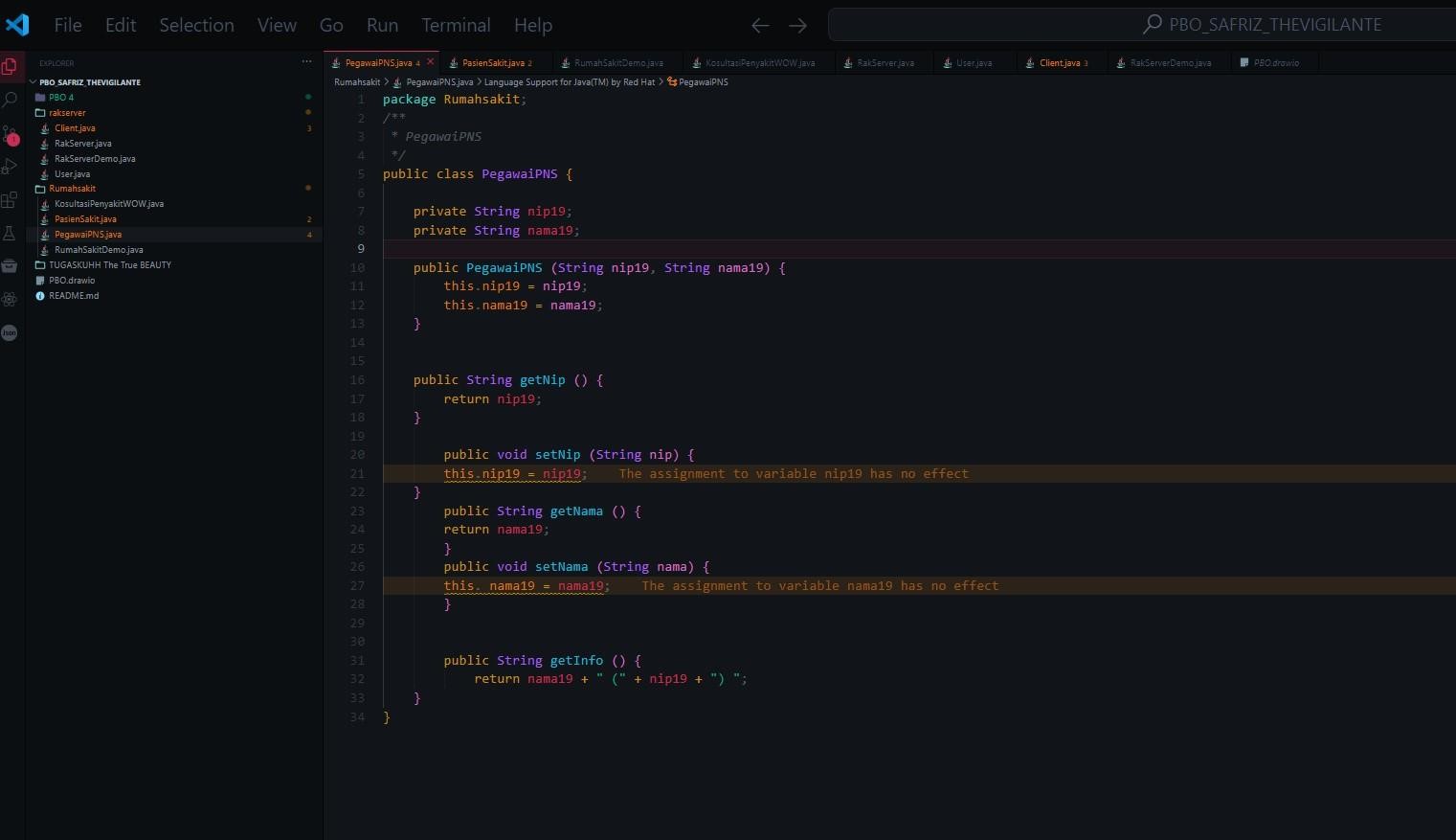
1. Create a new folder with the name Hospital.

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.

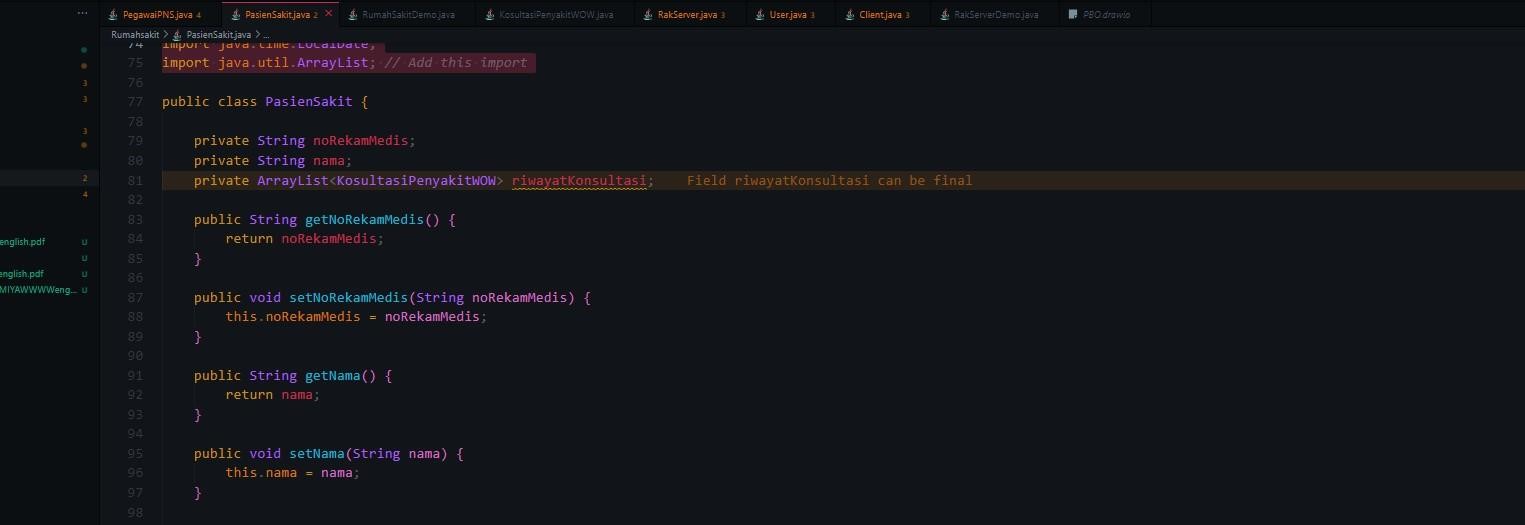
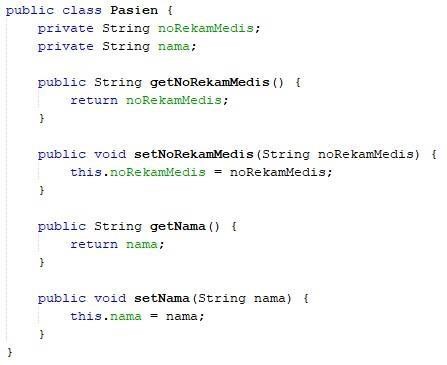




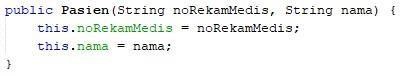
1. Implement **setters** and **getters** for the Employee class.



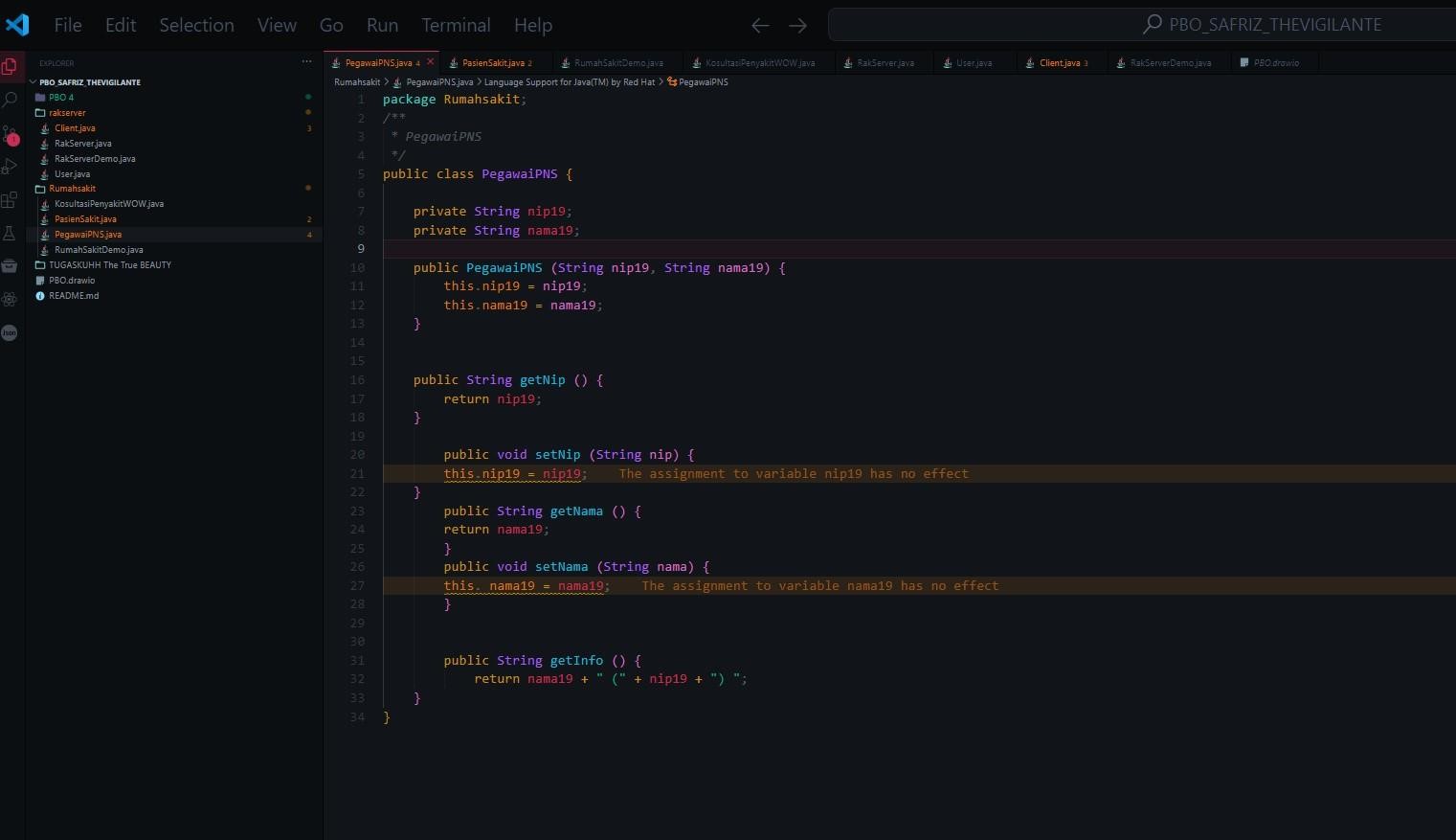
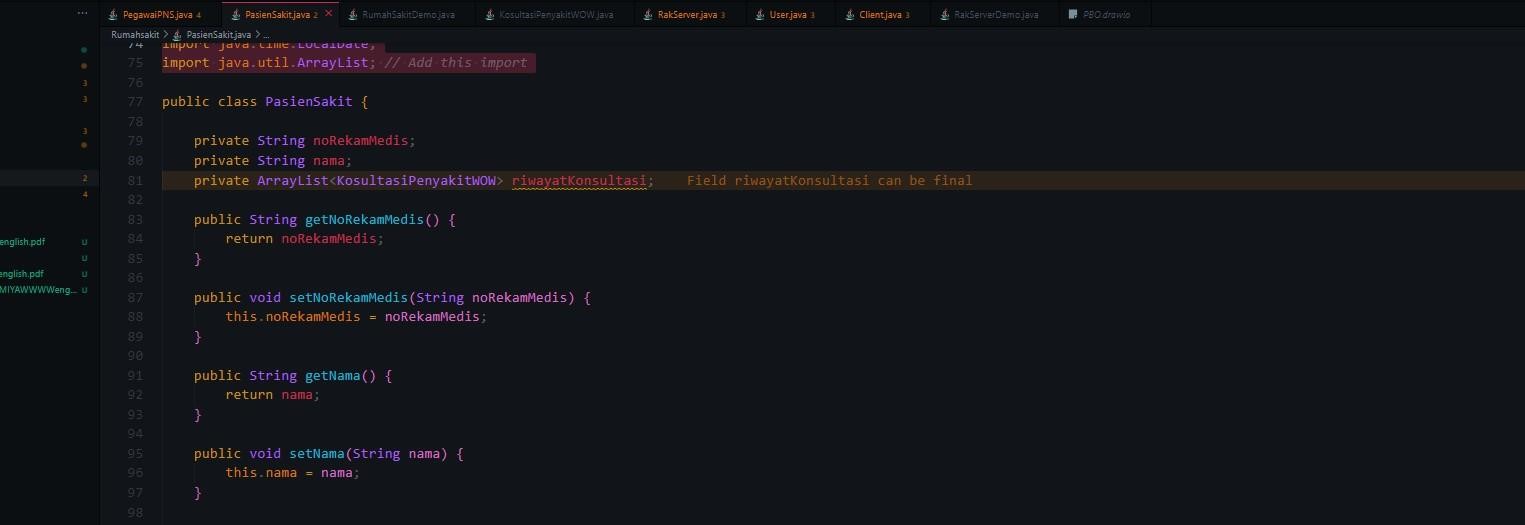
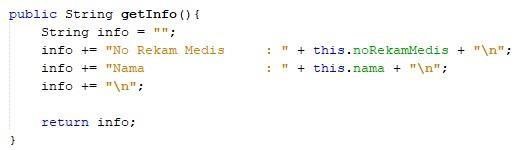
1. Next, create a Patient class then add the noReReRecordMedical attribute and name to the Patient class with a private access level modifier. Also provide setters and getters for these two attributes.



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

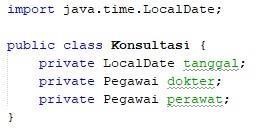


1. Implement *the getInfo()* method as follows:



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.
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.



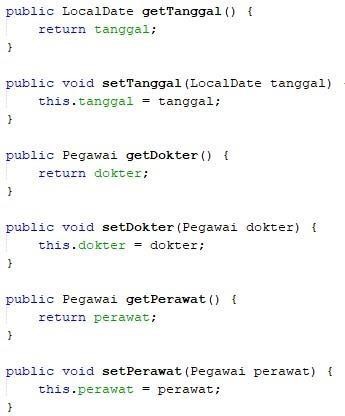
1. Provide setters and getters for each attribute in the Consult class

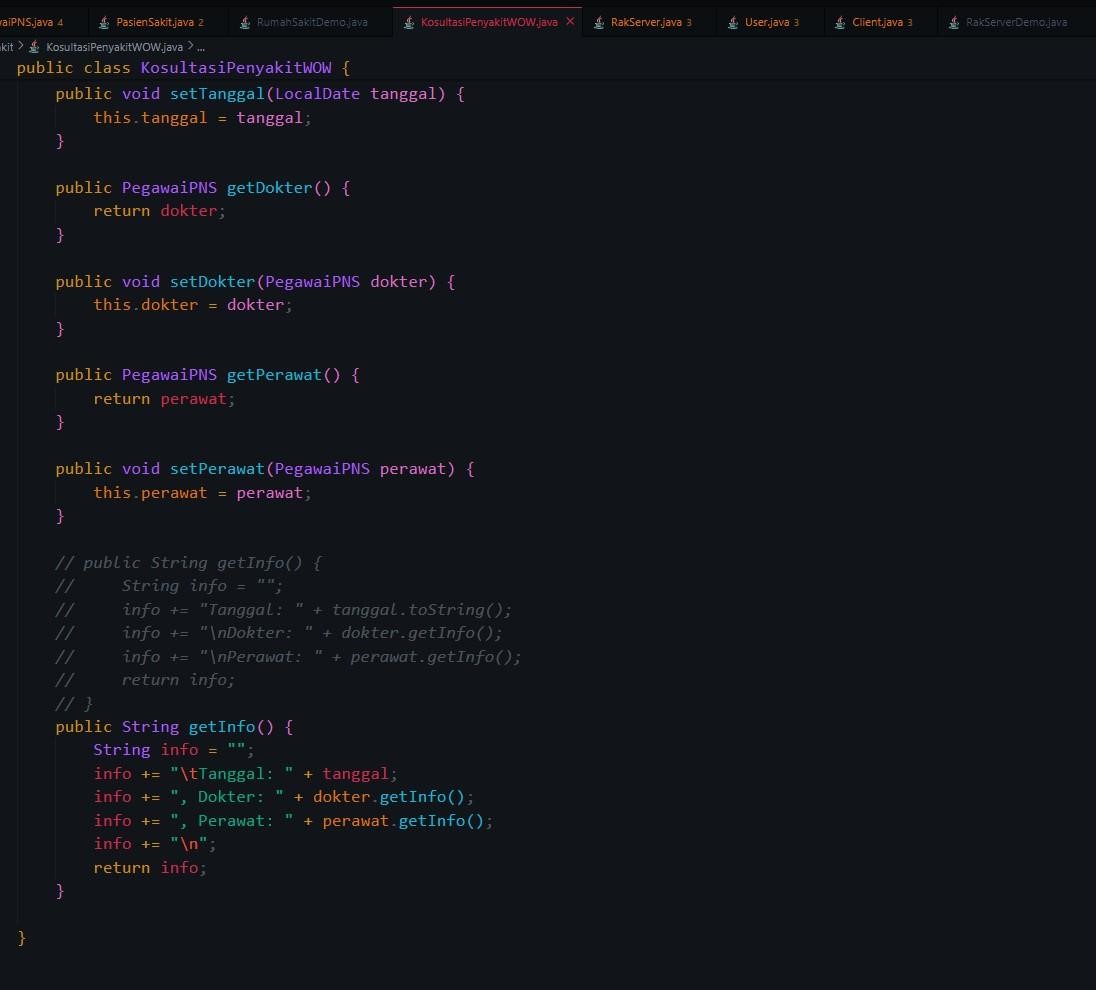


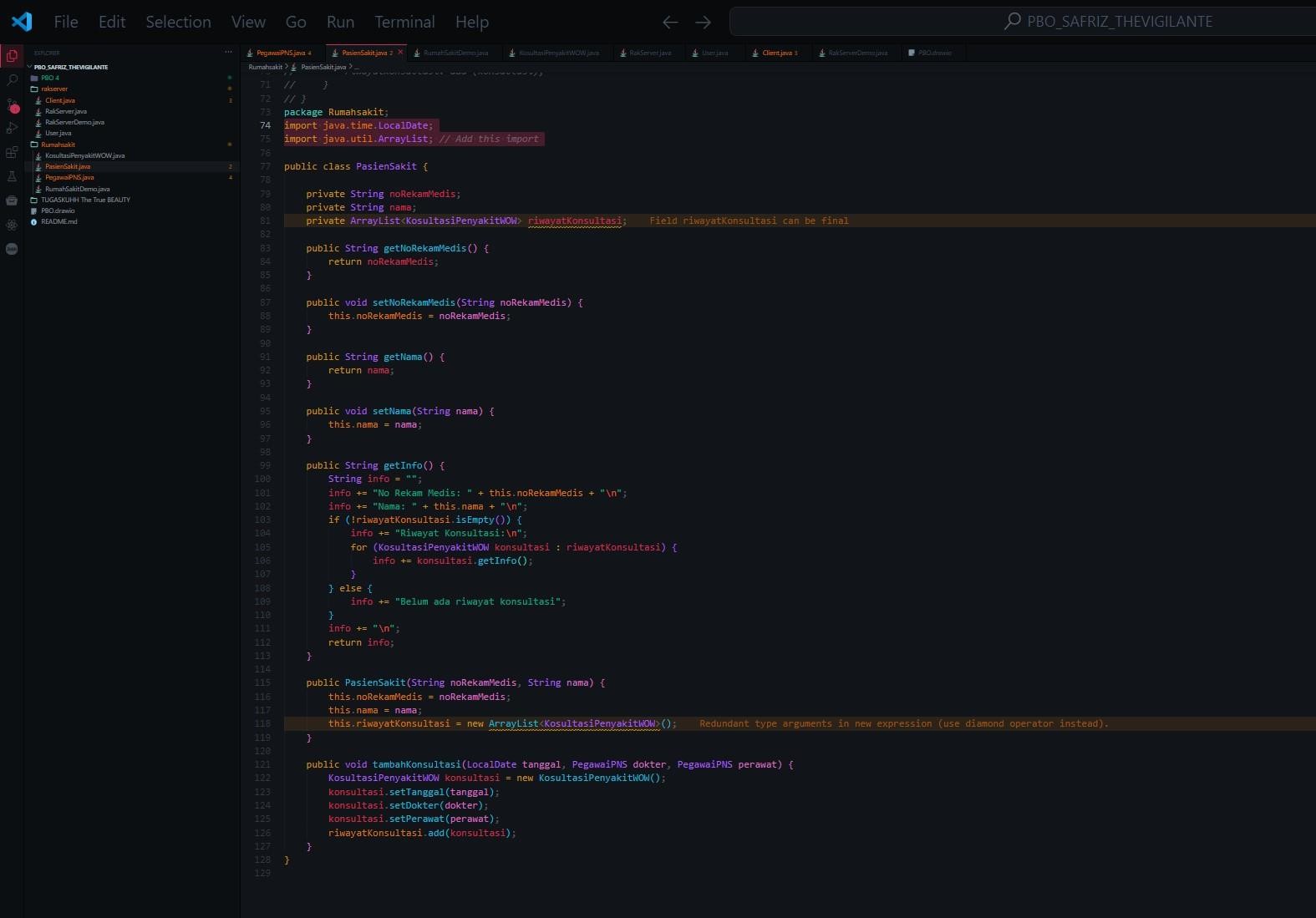
l.



Implement the getInfo() method as follows:





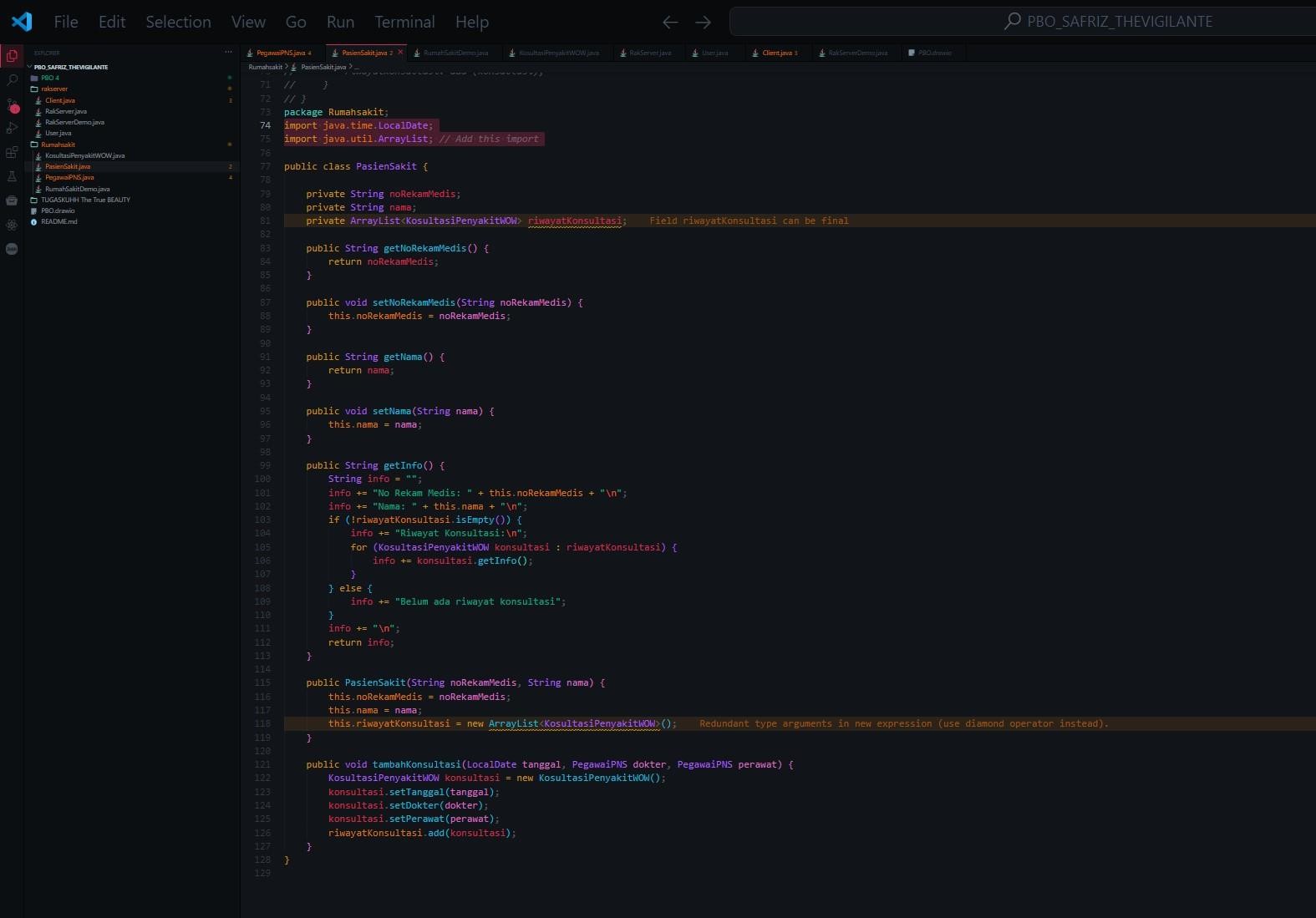
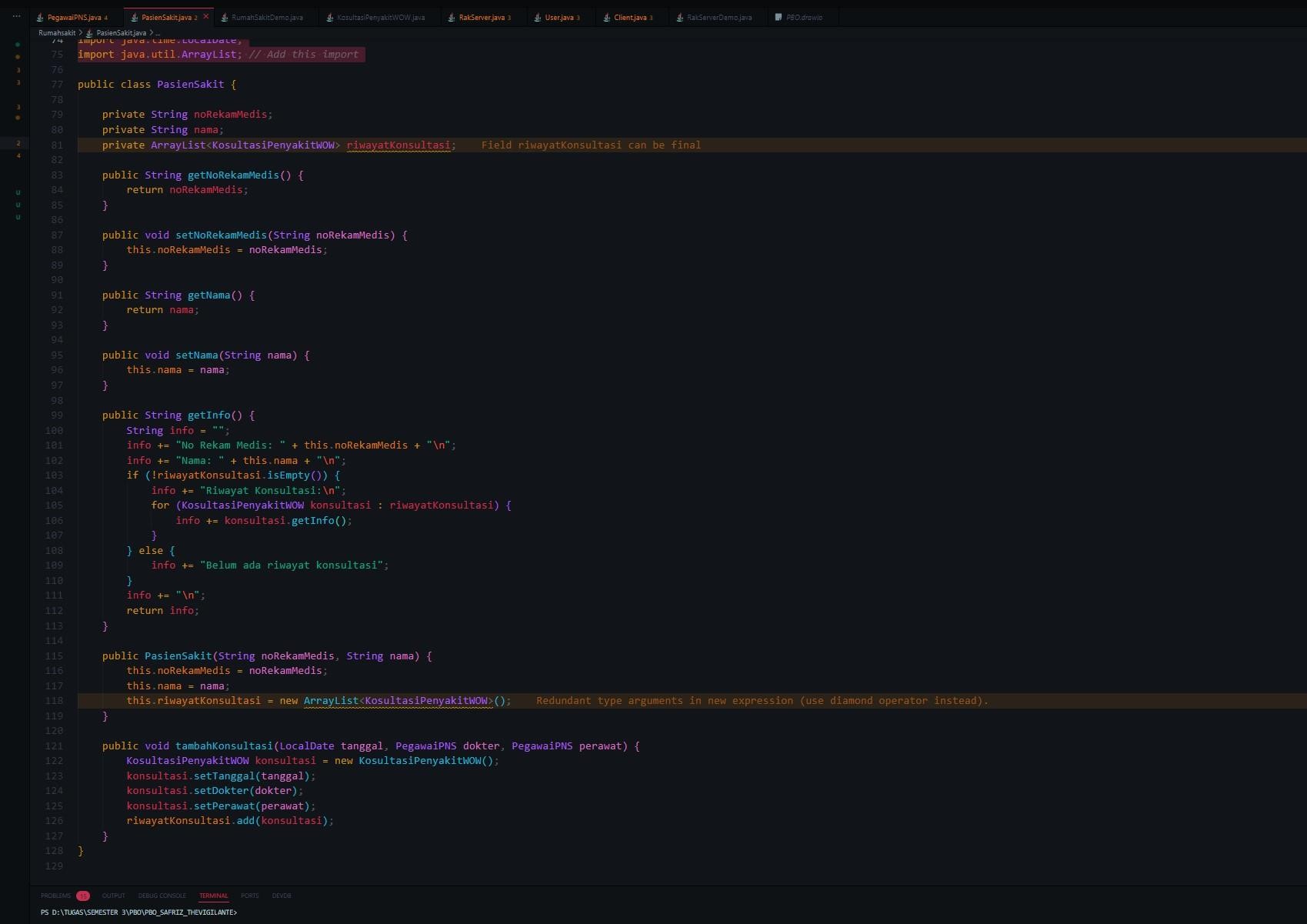
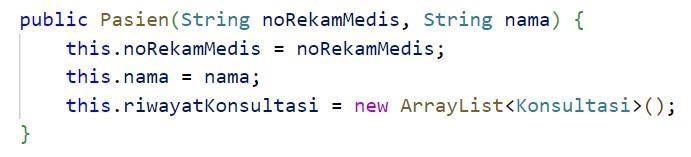


1. To store patient consultation history data, add the Consultation history attribute to the Patient class with the arrayList<Consultation> type. This attribute will store a series of objects of type Consultation. Import java.util.ArrayList in order to declare an attribute of type ArrayList of object.

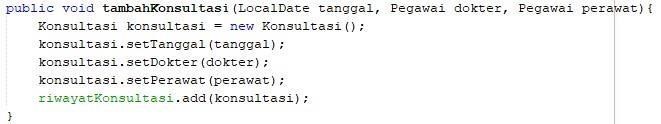


1. Create a parameterized constructor for the Patient class. Initiation of the value of the noReRecordMedical attribute and the name based on the name attribute.

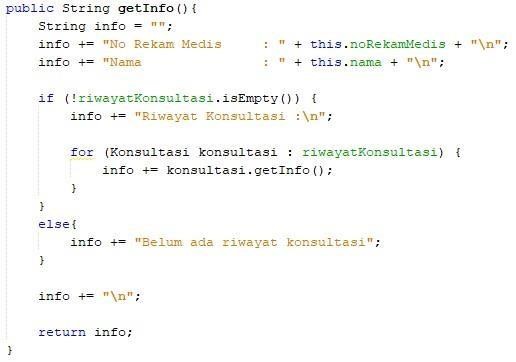
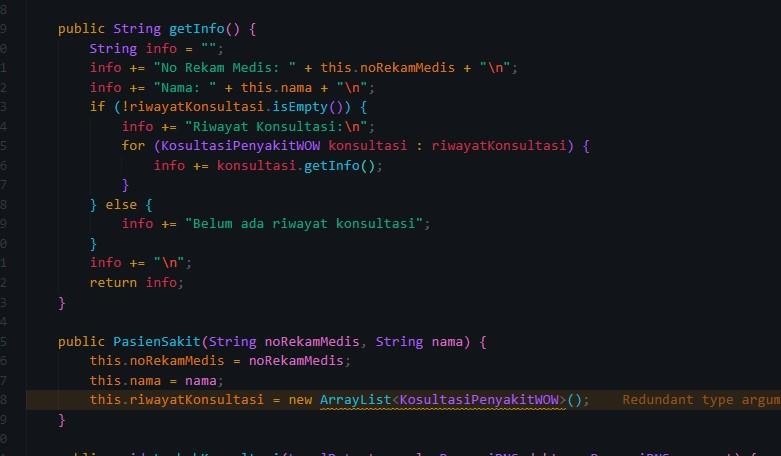
Instantiate/create a new ArrayList for the Consultation history attribute;



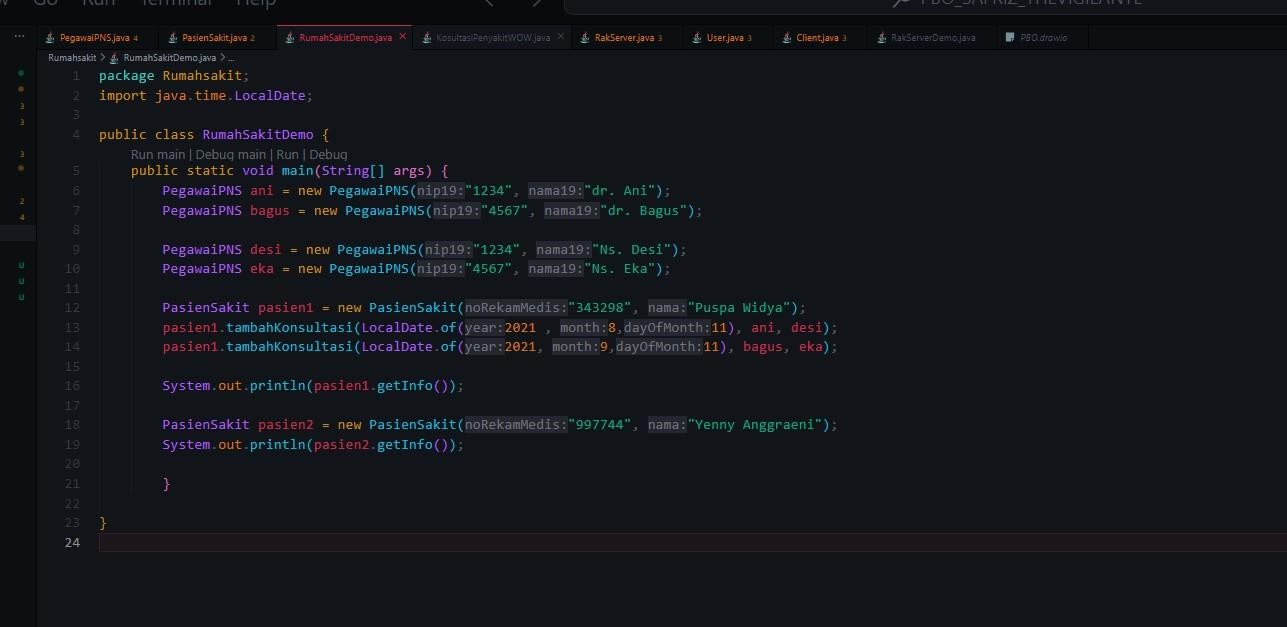
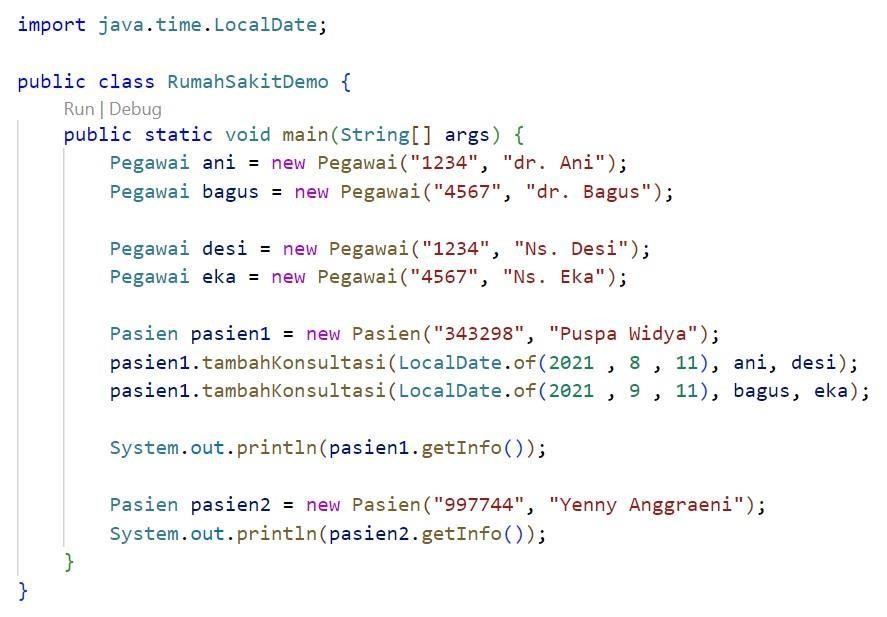
1. Import java.time.LocalDate to declare a date attribute of type LocalDate in the Patient class. Next, implement the method addConsultation() as follows:



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())

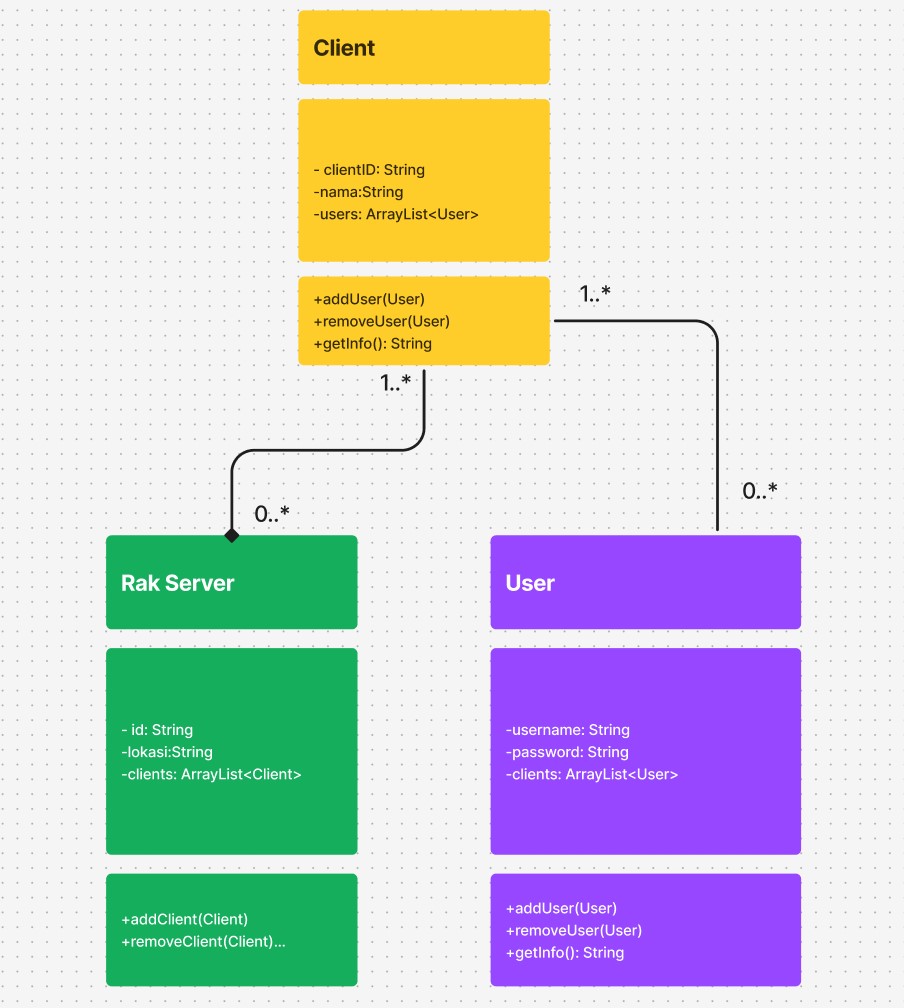
What does the line do?

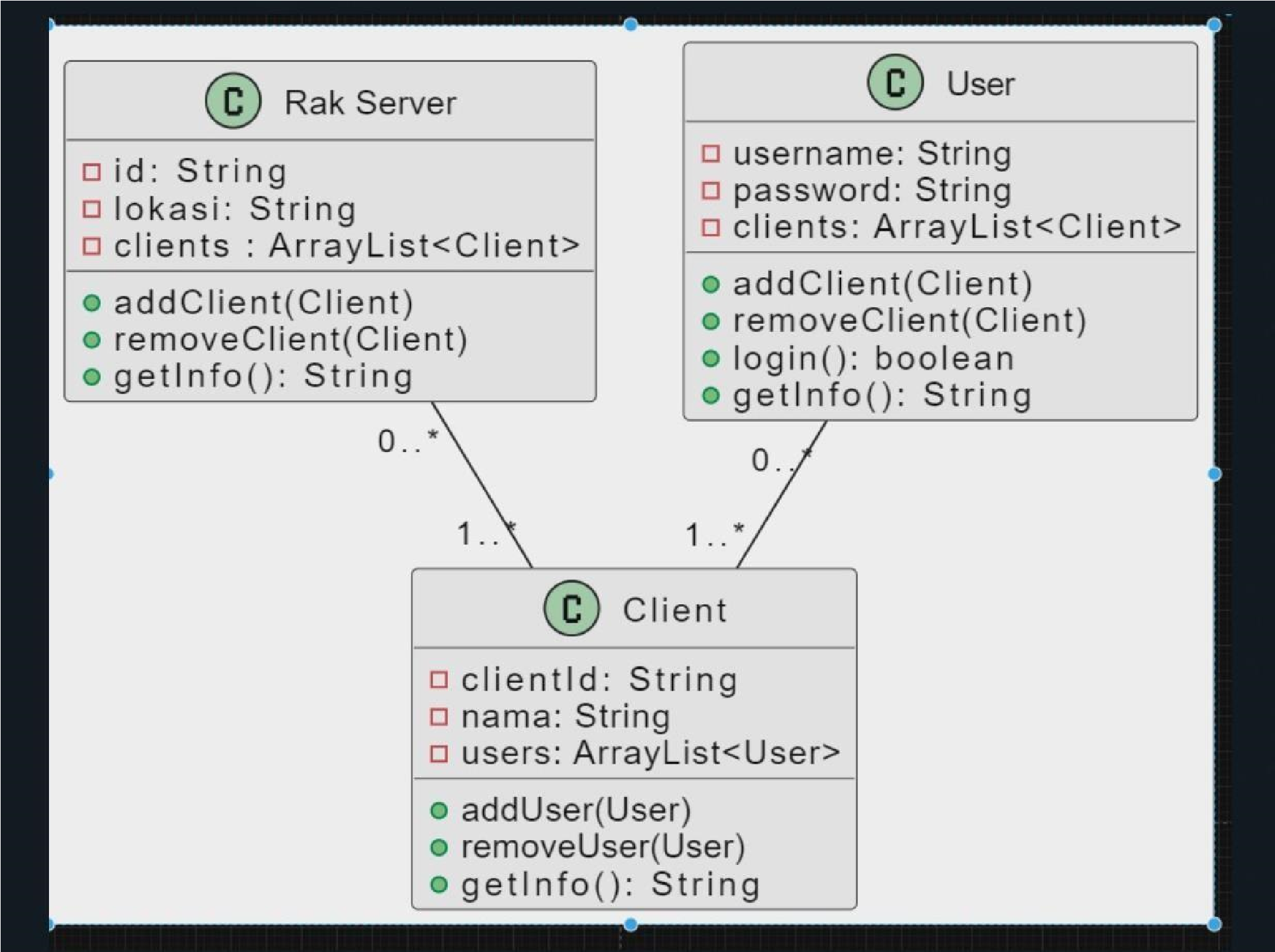
1. In the Patient constructor class, there is a line of code: this.historyConsultation = new ArrayList<>();

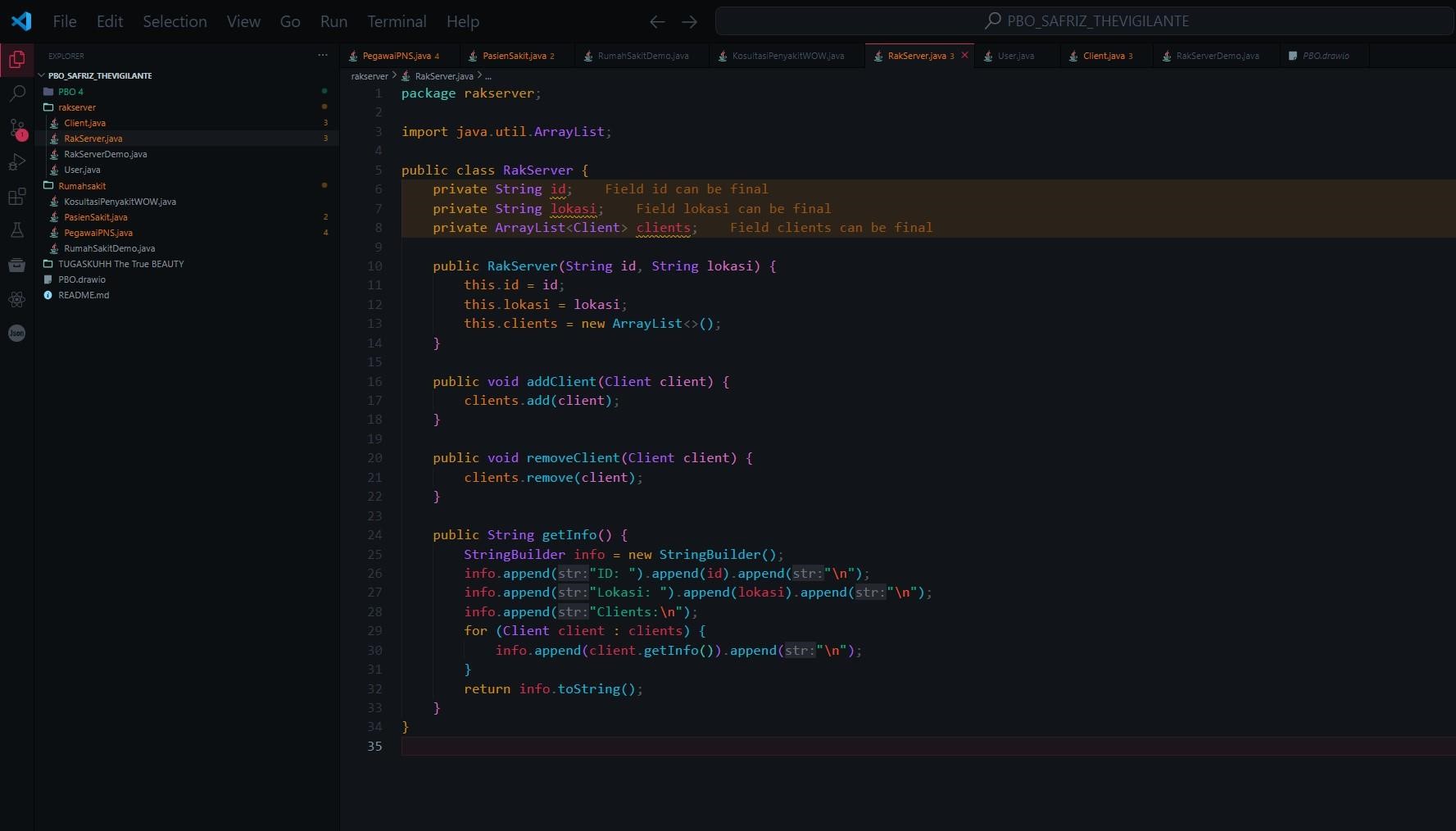
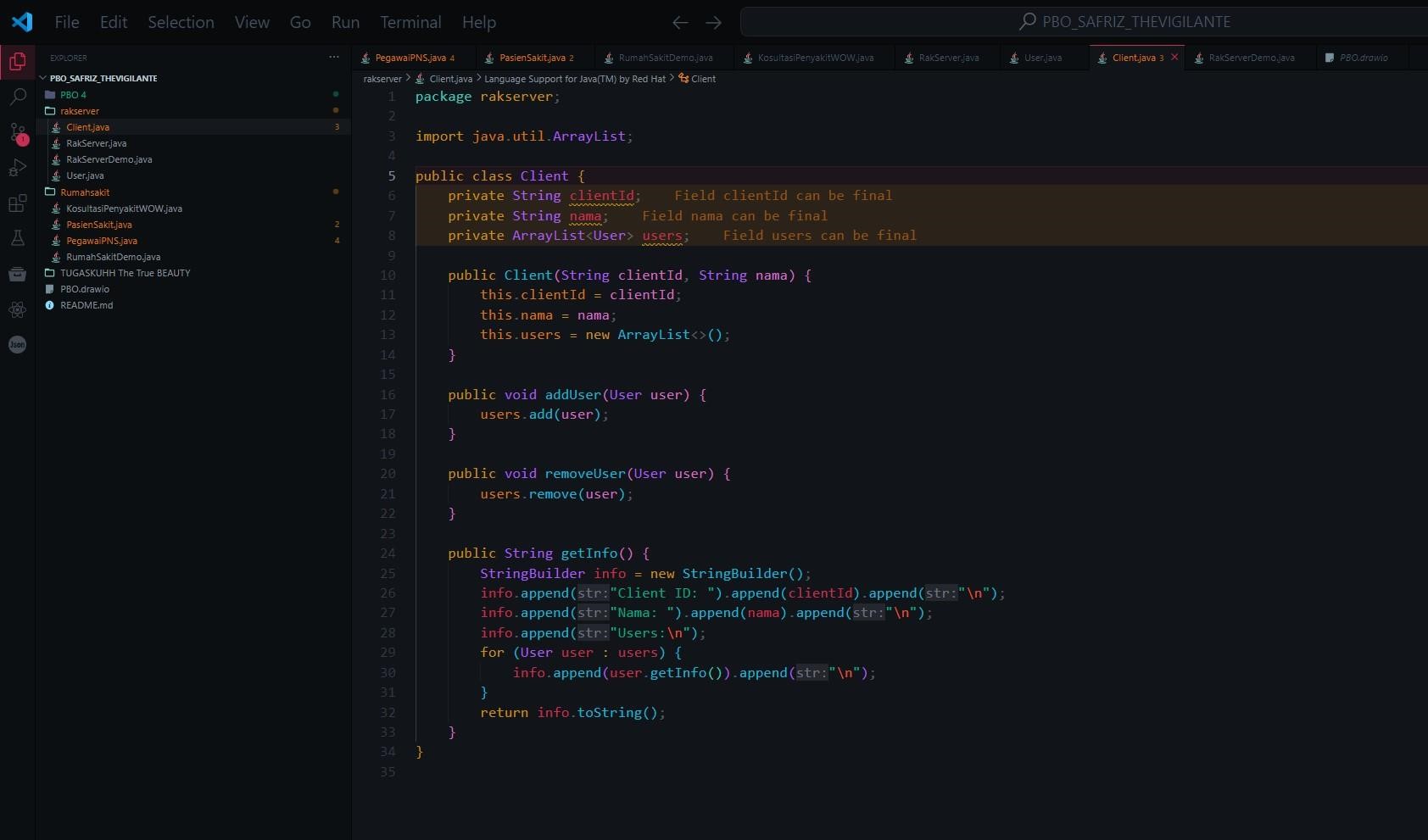
What does the line do? What happens if the line is omitted?

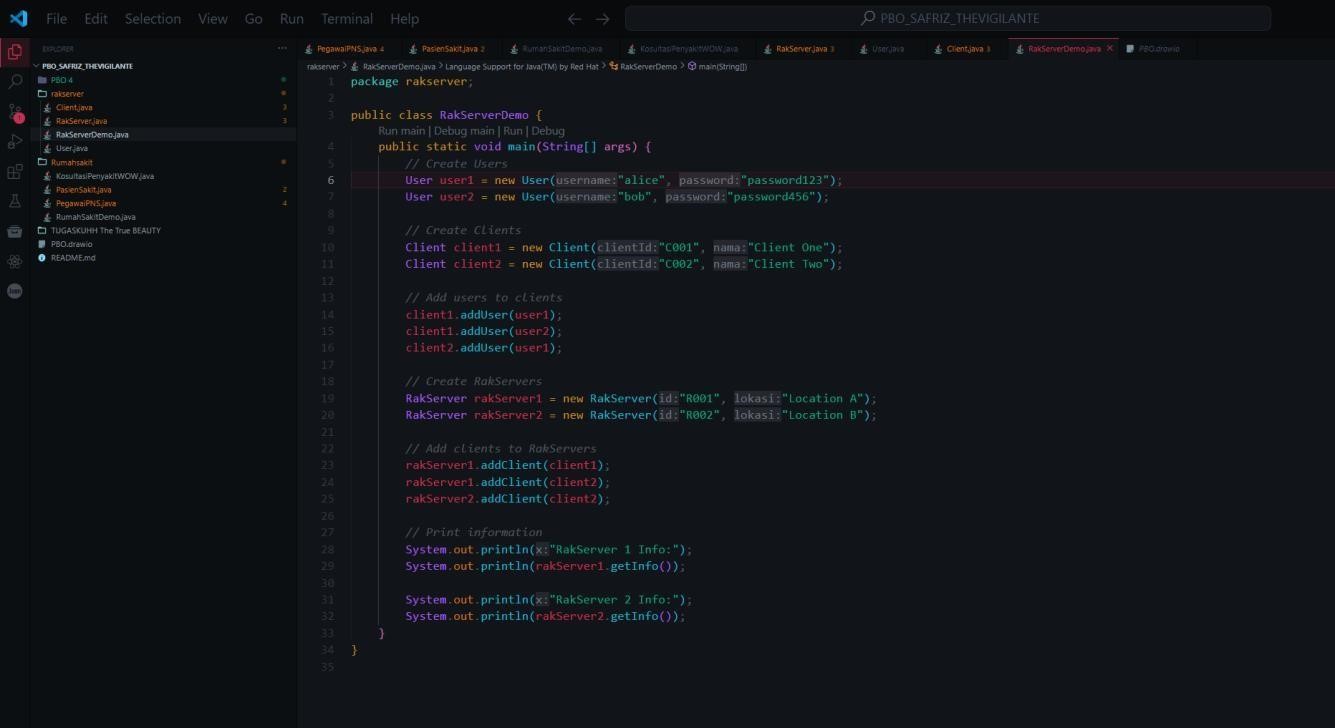
# Assignment

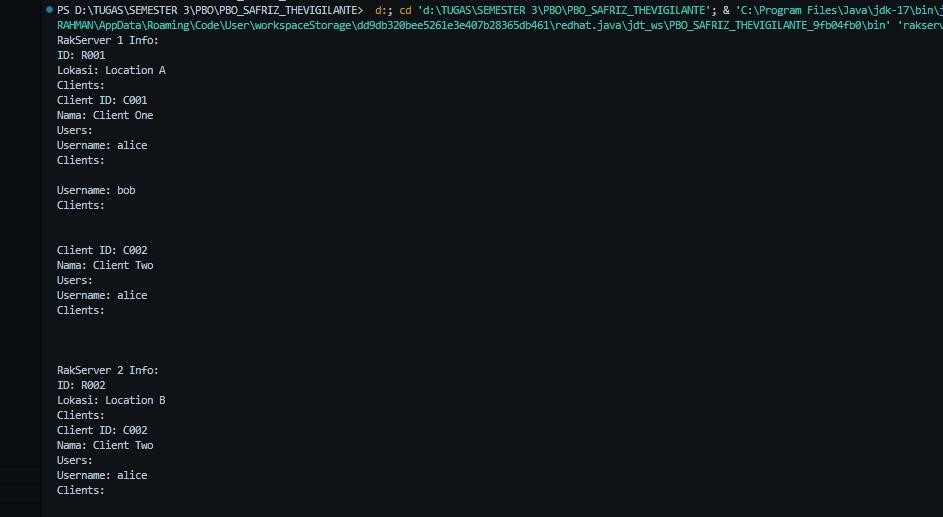
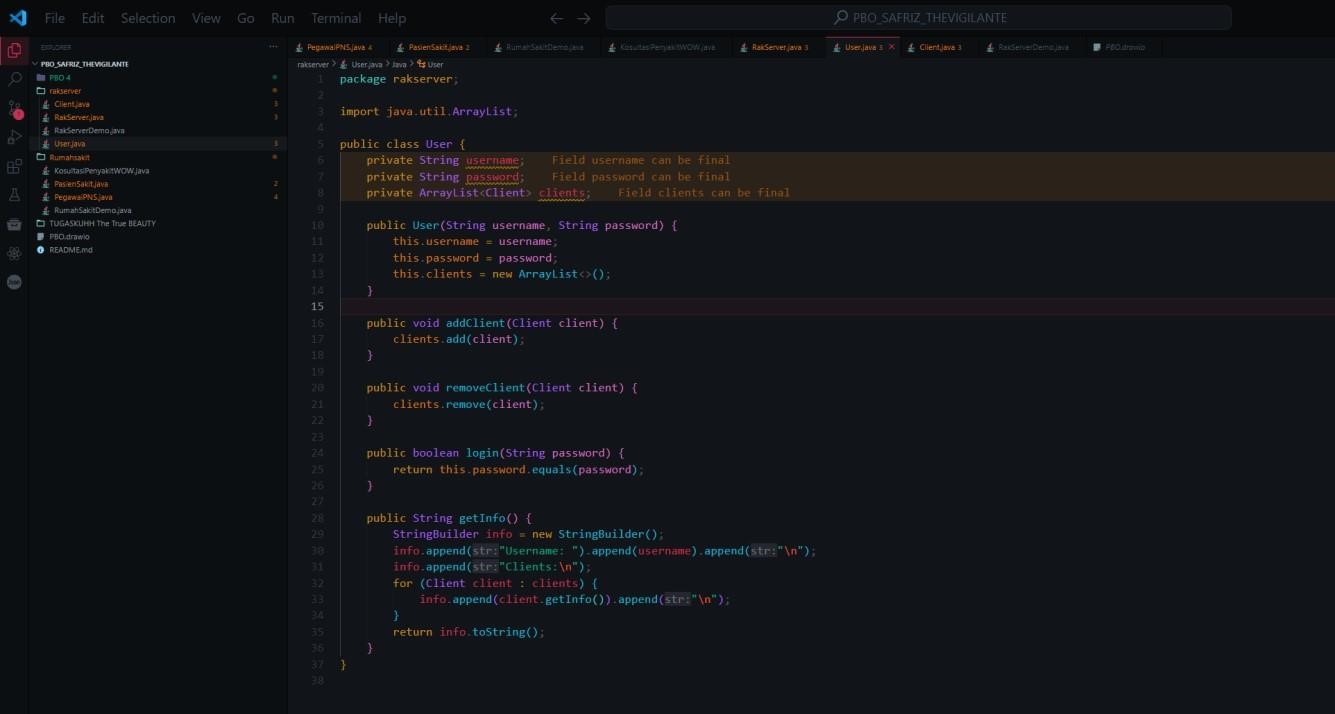
Implement the case studies that have been made on the Theory PBO assignment into the program.











**CATATAN**

Kode yang kamu lampirkan merupakan contoh dari pemrograman berorientasi objek (OOP) di Java. Saya akan menjawab setiap pertanyaanmu secara detail satu per satu berdasarkan kode yang diberikan.

1. Apakah Memiliki Pemrograman Struktural?

Pemrograman struktural melibatkan penggunaan variabel, langkah-langkah logis, dan fungsi yang terstruktur.

- Variabel yang digunakan:

- Di kelas `RakServer`: `id`, `lokasi`, dan `clients`.

- Di kelas `Client`: `clientId`, `nama`, dan `users`.

- Di kelas `User`: `username`, `password`, dan `clients`.

- Langkah-langkah:

1. Membuat objek `User`.

2. Membuat objek `Client`.

3. Menambahkan pengguna (`User`) ke dalam `Client`.

4. Membuat `RakServer` dan menambahkan `Client` ke dalamnya.

- Fungsi yang digunakan:

- `addUser`, `removeUser`, `addClient`, `removeClient`, dan `getInfo` di berbagai kelas. Fungsi-fungsi ini digunakan untuk manipulasi data dan menampilkan informasi.

2. Apakah Termasuk Objek Oriented?

Ya, kode ini merupakan pemrograman berorientasi objek (OOP). Ini terlihat dari:

- Penggunaan kelas (`RakServer`, `Client`, `User`).

- Penggunaan enkapsulasi (akses variabel melalui metode).

- Penggunaan objek dan interaksi antar-objek.

3. Apakah Termasuk OOP?

Ya, kode ini menggunakan konsep OOP. Beberapa elemen OOP yang ada adalah:

- Enkapsulasi: Melalui penggunaan modifier akses seperti `private` dan `public`.

- Inheritance: Tidak diterapkan di kode ini.

- Polimorfisme: Tidak diterapkan secara eksplisit di sini.

- Abstraksi: Dicapai melalui pengkapsulan data dalam kelas `User`, `Client`, dan `RakServer`.

4. Objeknya, Atribut, dan Fungsi (Metode) dari Objek

- Objek-objek:

1. `User`: Memiliki `username` dan `password`, bisa menambahkan/menghapus `Client`.

2. `Client`: Memiliki `clientId` dan `nama`, bisa menambahkan/menghapus `User`.

3. `RakServer`: Memiliki `id` dan `lokasi`, bisa menambahkan/menghapus `Client`.

- Fungsi (Metode):

- `User`: `addClient`, `removeClient`, `login`, `getInfo`.

- `Client`: `addUser`, `removeUser`, `getInfo`.

- `RakServer`: `addClient`, `removeClient`, `getInfo`.

5. Constructor, Atribut, dan Metode

- Constructor:

- `RakServer(String id, String lokasi)`.

- `Client(String clientId, String nama)`.

- `User(String username, String password)`.

- Atribut dan Deklarasinya:

- `RakServer`: `String id`, `String lokasi`, `ArrayList<Client> clients`.

- `Client`: `String clientId`, `String nama`, `ArrayList<User> users`.

- `User`: `String username`, `String password`, `ArrayList<Client> clients`.

- Metode dan Deklarasinya:

- `addUser(User user)`, `removeUser(User user)`, `addClient(Client client)`, `removeClient(Client client)`, `getInfo()`.

- Bagian yang Memerlukan Return:

- `getInfo()` pada semua kelas (`User`, `Client`, `RakServer`) mengembalikan `String` yang berisi informasi dari objek terkait.

6. Enkapsulasi

- Modifikasi Akses:

- `private`: Digunakan untuk atribut seperti `clientId`, `nama`, `username`, `password`, dan `users`, yang berarti atribut ini hanya bisa diakses melalui metode getter atau setter.

- `public`: Digunakan untuk metode seperti `getInfo`, `addUser`, `removeUser`, `addClient`, yang berarti metode ini bisa diakses dari luar kelas.

- Fungsi `this`: `this` digunakan untuk merujuk pada atribut kelas saat ada konflik dengan nama parameter (misalnya pada constructor `Client(String clientId, String nama)`).

7. Relasi Kelas

- Association:

- Ada hubungan asosiasi antara `RakServer` dan `Client`, dimana `RakServer` memiliki daftar klien.

- Ada hubungan asosiasi antara `Client` dan `User`, dimana `Client` memiliki daftar pengguna.

- Aggregation:

- `Client` dan `User` memiliki hubungan agregasi karena `User` dapat ditambahkan ke beberapa `Client`, namun tidak ada pemilikan yang kuat (penghapusan `Client` tidak akan menghapus `User`).

- Composition:

- Tidak ada hubungan komposisi yang eksplisit, karena semua hubungan menunjukkan bahwa objek dapat hidup secara independen.

Penjelasan Kode Baris per Baris

1. `package rakserver;`: Menentukan bahwa kelas ini merupakan bagian dari package `rakserver`.

2. `import java.util.ArrayList;`: Mengimpor kelas `ArrayList` untuk penggunaan daftar dinamis.

3. `public class Client {`: Deklarasi kelas `Client` yang public.

4. `private String clientId;`: Mendeklarasikan atribut `clientId` dengan akses `private`.

5. `public Client(String clientId, String nama) {`: Constructor untuk menginisialisasi objek `Client`.

6. `public void addUser(User user) {`: Metode untuk menambahkan pengguna ke dalam daftar `users`.

7. `public String getInfo() {`: Metode untuk mendapatkan informasi objek `Client`.

8. `public class RakServer {`: Deklarasi kelas `RakServer`.

9. `private String id;`: Mendeklarasikan atribut `id` dari `RakServer`.

10. `public void addClient(Client client) {`: Metode untuk menambahkan klien ke dalam server.

11. `public String getInfo() {`: Metode untuk mendapatkan informasi objek `RakServer`.

12. `public class User {`: Deklarasi kelas `User`.

13. `public boolean login(String password) {`: Metode untuk verifikasi login pengguna.

Penutup

Secara keseluruhan, kode ini menggunakan OOP dengan baik, meskipun tidak ada polimorfisme dan inheritance. Relasi antar kelas seperti asosiasi dan agregasi telah dijelaskan dengan baik, dan enkapsulasi diterapkan melalui penggunaan modifier akses.

Berikut adalah kode Java lengkap berdasarkan penjelasan konsep OOP yang telah dijelaskan sebelumnya. Saya juga akan memberikan sedikit perbaikan dan penambahan agar konsep yang dijelaskan lebih lengkap dan jelas dalam implementasi kode.

Kode Java Lengkap

```java

package rakserver;

import java.util.ArrayList;

// Class User yang merepresentasikan pengguna di sistem

public class User {

private String username;

private String password;

private ArrayList<Client> clients; // Relasi dengan Client (Agregasi)

// Constructor untuk inisialisasi User

public User(String username, String password) {

this.username = username;

this.password = password;

this.clients = new ArrayList<>();

}

// Menambahkan Client ke daftar Client yang dihubungkan dengan User ini

public void addClient(Client client) {

clients.add(client);

}

// Menghapus Client dari daftar Client User ini

public void removeClient(Client client) {

clients.remove(client);

}

// Metode login untuk memverifikasi password

public boolean login(String password) {

return this.password.equals(password);

}

// Mengembalikan informasi tentang User ini

public String getInfo() {

StringBuilder info = new StringBuilder();

info.append("Username: ").append(username).append("\n");

info.append("Clients:\n");

for (Client client : clients) {

info.append(client.getClientId()).append("\n"); // Hanya menampilkan ID client

}

return info.toString();

}

}

// Class Client yang merepresentasikan sebuah Client di dalam sistem

public class Client {

private String clientId;

private String nama;

private ArrayList<User> users; // Relasi dengan User (Agregasi)

// Constructor untuk inisialisasi Client

public Client(String clientId, String nama) {

this.clientId = clientId;

this.nama = nama;

this.users = new ArrayList<>();

}

// Menambahkan User ke Client ini

public void addUser(User user) {

users.add(user);

}

// Menghapus User dari Client ini

public void removeUser(User user) {

users.remove(user);

}

// Mengembalikan informasi tentang Client

public String getInfo() {

StringBuilder info = new StringBuilder();

info.append("Client ID: ").append(clientId).append("\n");

info.append("Nama: ").append(nama).append("\n");

info.append("Users:\n");

for (User user : users) {

info.append(user.getInfo()).append("\n");

}

return info.toString();

}

// Getter untuk mendapatkan ID Client

public String getClientId() {

return clientId;

}

}

// Class RakServer yang merepresentasikan sebuah server di dalam sistem

public class RakServer {

private String id;

private String lokasi;

private ArrayList<Client> clients; // Relasi dengan Client (Agregasi)

// Constructor untuk inisialisasi RakServer

public RakServer(String id, String lokasi) {

this.id = id;

this.lokasi = lokasi;

this.clients = new ArrayList<>();

}

// Menambahkan Client ke RakServer

public void addClient(Client client) {

clients.add(client);

}

// Menghapus Client dari RakServer

public void removeClient(Client client) {

clients.remove(client);

}

// Mengembalikan informasi tentang RakServer

public String getInfo() {

StringBuilder info = new StringBuilder();

info.append("ID: ").append(id).append("\n");

info.append("Lokasi: ").append(lokasi).append("\n");

info.append("Clients:\n");

for (Client client : clients) {

info.append(client.getInfo()).append("\n");

}

return info.toString();

}

}

// Class RakServerDemo untuk mendemonstrasikan program

public class RakServerDemo {

public static void main(String[] args) {

// Membuat pengguna (User)

User user1 = new User("alice", "password123");

User user2 = new User("bob", "password456");

// Membuat klien (Client)

Client client1 = new Client("C001", "Client One");

Client client2 = new Client("C002", "Client Two");

// Menambahkan pengguna ke klien

client1.addUser(user1);

client1.addUser(user2);

client2.addUser(user1);

// Membuat server (RakServer)

RakServer rakServer1 = new RakServer("R001", "Location A");

RakServer rakServer2 = new RakServer("R002", "Location B");

// Menambahkan klien ke server

rakServer1.addClient(client1);

rakServer1.addClient(client2);

rakServer2.addClient(client2);

// Menampilkan informasi RakServer

System.out.println("RakServer 1 Info:");

System.out.println(rakServer1.getInfo());

System.out.println("RakServer 2 Info:");

System.out.println(rakServer2.getInfo());

}

}

```

Penjelasan Kode Baris per Baris

1. `package rakserver;`: Menentukan bahwa semua kelas berada dalam package `rakserver`.

2. `import java.util.ArrayList;`: Mengimpor `ArrayList` untuk menyimpan daftar dinamis dari `User` dan `Client`.

3. Kelas `User`:

- Atribut: `username`, `password`, dan `clients`.

- Constructor: Inisialisasi `username`, `password`, dan menginisialisasi list `clients`.

- Metode:

- `addClient` untuk menambahkan klien.

- `removeClient` untuk menghapus klien.

- `login` untuk verifikasi login berdasarkan password.

- `getInfo` untuk mendapatkan detail pengguna.

4. Kelas `Client`:

- Atribut: `clientId`, `nama`, dan `users`.

- Constructor: Inisialisasi `clientId`, `nama`, dan menginisialisasi list `users`.

- Metode:

- `addUser` dan `removeUser` untuk menambah dan menghapus pengguna dari klien.

- `getInfo` untuk mendapatkan detail tentang klien, termasuk daftar pengguna.

5. Kelas `RakServer`:

- Atribut: `id`, `lokasi`, dan `clients`.

- Constructor: Inisialisasi `id`, `lokasi`, dan menginisialisasi list `clients`.

- Metode:

- `addClient` dan `removeClient` untuk menambah atau menghapus klien dari server.

- `getInfo` untuk mendapatkan informasi tentang server dan klien yang ada di dalamnya.

6. Kelas `RakServerDemo`:

- Digunakan untuk menguji dan mendemonstrasikan implementasi dari kelas `User`, `Client`, dan `RakServer`.

Konsep OOP

1. Encapsulation: Atribut dalam setiap kelas bersifat private, hanya bisa diakses melalui method public.

2. Association: Kelas `User` dan `Client` saling terkait. `Client` juga berhubungan dengan `RakServer`.

3. Aggregation: `RakServer` mengandung `Client`, dan `Client` mengandung `User`. Relasinya bersifat agregasi karena `Client` dan `User` bisa ada terpisah dari objek induk mereka.

Penutup

Kode ini menunjukkan bagaimana OOP diterapkan dalam Java dengan konsep enkapsulasi, agregasi, dan asosiasi yang jelas di antara kelas-kelas. Jika kamu membutuhkan penjelasan lebih lanjut, jangan ragu untuk bertanya!  
  
  
  
  
  
  
  
  
  
  
  
  
  
 1. Object

- Object adalah instansiasi dari suatu class, contoh objek dalam kode ini adalah:

- `User user1 = new User("alice", "password123");`

- `Client client1 = new Client("C001", "Client One");`

- `RakServer rakServer1 = new RakServer("R001", "Location A");`

Ini semua adalah objek dari kelas `User`, `Client`, dan `RakServer`.

2. Fungsi (Method)

- Method adalah blok kode dalam suatu class yang melakukan tugas tertentu.

- Method yang terdapat di class `User`:

- `addClient(Client client)`: Menambahkan client ke daftar client user.

- `removeClient(Client client)`: Menghapus client dari daftar client user.

- `login(String password)`: Memverifikasi apakah password yang diberikan cocok.

- `getInfo()`: Mengembalikan informasi dari objek user.

- Method di class `Client`:

- `addUser(User user)`: Menambahkan user ke dalam daftar user client.

- `removeUser(User user)`: Menghapus user dari daftar user client.

- `getInfo()`: Mengembalikan informasi dari objek client.

- Method di class `RakServer`:

- `addClient(Client client)`: Menambahkan client ke daftar client server.

- `removeClient(Client client)`: Menghapus client dari daftar client server.

- `getInfo()`: Mengembalikan informasi dari objek server.

3. Constructor

- Constructor adalah metode khusus yang digunakan untuk menginisialisasi objek saat instansiasi kelas.

- Constructor di class `User`:

```java

public User(String username, String password) {

this.username = username;

this.password = password;

this.clients = new ArrayList<>();

}

```

- Constructor di class `Client`:

```java

public Client(String clientId, String nama) {

this.clientId = clientId;

this.nama = nama;

this.users = new ArrayList<>();

}

```

- Constructor di class `RakServer`:

```java

public RakServer(String id, String lokasi) {

this.id = id;

this.lokasi = lokasi;

this.clients = new ArrayList<>();

}

```

4. Class

- Class adalah cetak biru atau template untuk membuat objek. Kelas yang ada di dalam kode ini:

- `User`

- `Client`

- `RakServer`

- `RakServerDemo` (Digunakan untuk mendemonstrasikan penggunaan class lain).

5. Atribut

- Atribut adalah variabel yang menyimpan data di dalam class. Atribut di dalam setiap class:

- User:

- `private String username;`

- `private String password;`

- `private ArrayList<Client> clients;`

- Client:

- `private String clientId;`

- `private String nama;`

- `private ArrayList<User> users;`

- RakServer:

- `private String id;`

- `private String lokasi;`

- `private ArrayList<Client> clients;`

6. Deklarasi Atribut

- Setiap atribut di dalam class harus dideklarasikan dengan tipe data dan modifier akses. Contohnya:

```java

private String username; // Deklarasi atribut dengan tipe data String dan akses private

```

7. Method dan Deklarasi Method

- Method harus dideklarasikan dengan tipe return, nama method, dan parameter jika ada.

Contoh deklarasi method:

```java

public String getInfo() {

StringBuilder info = new StringBuilder();

info.append("Username: ").append(username).append("\n");

info.append("Clients:\n");

for (Client client : clients) {

info.append(client.getClientId()).append("\n");

}

return info.toString();

}

```

8. Bagian yang Memerlukan Return

- Method yang memerlukan return:

- `public String getInfo()` di semua class (`User`, `Client`, `RakServer`) mengembalikan `String` yang berisi informasi objek.

- `public boolean login(String password)` mengembalikan `boolean` untuk menentukan apakah login berhasil.

9. Object Instantiation (Pembuatan Objek)

- Instansiasi objek dilakukan dengan menggunakan keyword `new` diikuti oleh nama class dan parameternya.

Contoh:

```java

User user1 = new User("alice", "password123"); // Membuat objek user dengan constructor

```

10. Constructor dengan Parameter

- Constructor dapat memiliki parameter untuk menerima nilai saat objek dibuat. Contohnya:

```java

public Client(String clientId, String nama) {

this.clientId = clientId;

this.nama = nama;

this.users = new ArrayList<>();

}

```

11. Enkapsulasi

- Enkapsulasi adalah konsep OOP yang membatasi akses langsung ke atribut kelas dan mengharuskan penggunaan method getter dan setter untuk manipulasi data.

- Modifier akses dalam Java:

- `private`: Atribut hanya bisa diakses dari dalam class itu sendiri. Digunakan untuk menjaga data dari manipulasi langsung. Contoh: `private String username;`

- `public`: Metode atau atribut yang dapat diakses dari mana saja. Contoh: `public void addClient(Client client)`.

- No modifier (default): Jika tidak ada modifier akses yang ditentukan, maka akses hanya diizinkan dalam package yang sama.

- `protected`: Dapat diakses oleh class dalam package yang sama atau subclass. Modifier ini tidak digunakan dalam kode ini.

12. Fungsi `this`

- `this` digunakan untuk merujuk pada instance variabel dari class saat ada konflik antara nama parameter dengan nama atribut. Ini digunakan untuk membedakan variabel instance dengan parameter dalam constructor.

Contoh:

```java

public User(String username, String password) {

this.username = username; // 'this' merujuk pada variabel instance

this.password = password;

}

```

Kesimpulan

Kode di atas menerapkan konsep OOP (Object-Oriented Programming) dengan baik, menggunakan enkapsulasi, instansiasi objek, constructor, dan modifier akses untuk memastikan keamanan dan modularitas kode.

**RakServer 1 Info:**

* **ID**: R001
* **Lokasi**: Location A
* **Clients**:
  + **Client ID**: C001
    - **Nama**: Client One
    - **Users**:
      * **Username**: alice
      * **Username**: bob
  + **Client ID**: C002
    - **Nama**: Client Two
    - **Users**: Tidak ada user terdaftar di bawah client ini.
* **Client One** dengan ID C001 yang memiliki dua pengguna: alice dan bob.
* **Client Two** dengan ID C002 yang tidak memiliki pengguna.

**RakServer 2 Info:**

* **ID**: R002
* **Lokasi**: Location B
* **Clients**: Tidak ada client terdaftar di bawah server ini.