Jobsheet 04 - Class Relations

I. Competence

After studying this subject, students are able to:

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

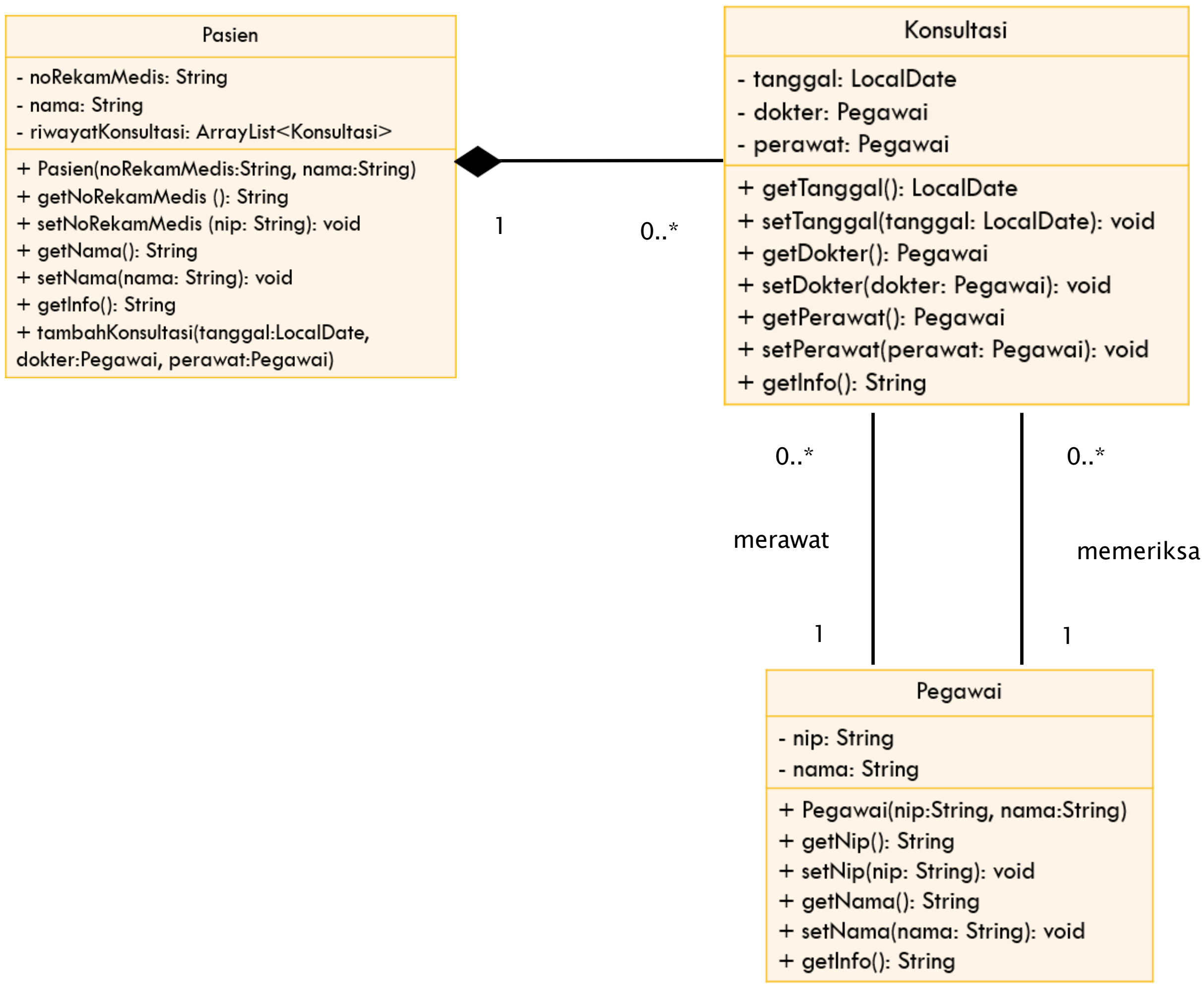
II. Introduction

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 :


Pasien
- noRekamMedis: String
- nama: String
- riwayatKonsultasi: ArrayList<Konsultasi>
+ Pasien(noRekamMedis:String, nama:String)
+ getNoRekamMedis (): String
+ setNoRekamMedis (nip: String): void
+ getNama(): String
+ setNama(nama: String): void
+ getInfo(): String
+ tambahKonsultasi(tanggal:LocalDate, dokter:Pegawai, perawat:Pegawai)
1
0..*
Konsultasi
- tanggal: LocalDate
- dokter: Pegawai
- perawat: Pegawai
+ getTanggal(): LocalDate
+ setTanggal(tanggal: LocalDate): void
+ getDokter(): Pegawai
+ setDokter(dokter: Pegawai): void
+ getPerawat(): Pegawai
+ setPerawat(perawat: Pegawai): void
+ getInfo(): String
0..*
merawat
1
0..*
memeriksa
1
Pegawai
- nip: String
- nama: String
+ Pegawai(nip:String, nama:String)
+ getNip(): String
+ setNip(nip: String): void
+ getNama(): String
+ setNama(nama: String): void
+ getInfo(): String

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.

```
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

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

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

```
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;
}
```

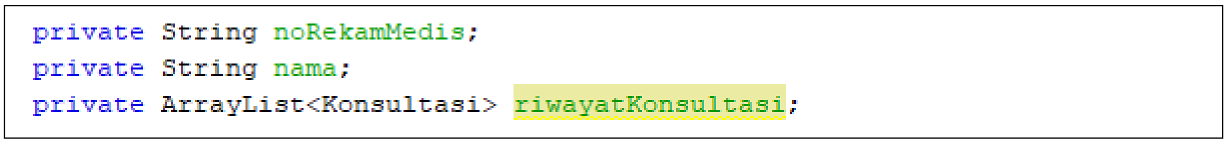
k. Provide setters and getters for each attribute in the Consult class

```
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:

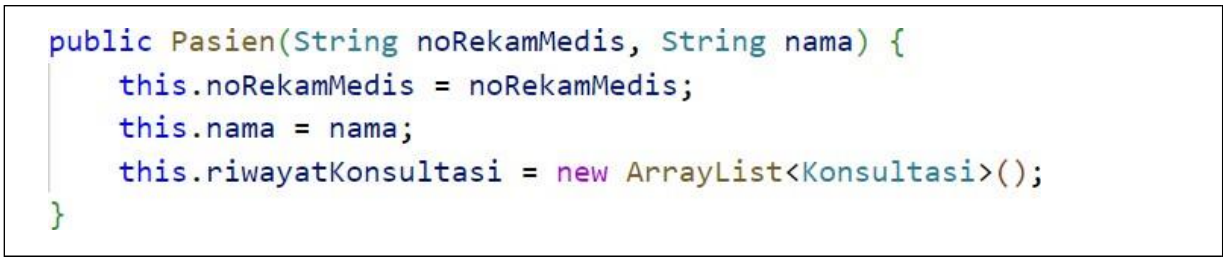
```
public String getInfo(){
    String info = "";
    info += "\tTanggal: " + tanggal;
    info += ", Dokter: " +  dokter.getInfo();
    info += ", Perawat: " + perawat.getInfo();
    info += "\n";

    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.



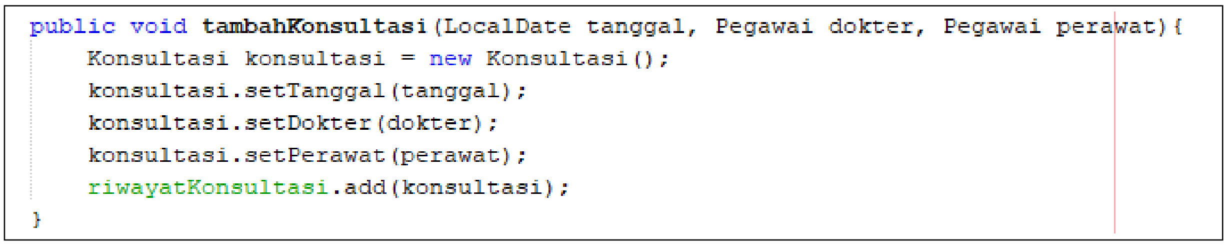
```java
private String noRekamMedis;
private String nama;
private ArrayList<Konsultasi> riwayatKonsultasi;
```

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;



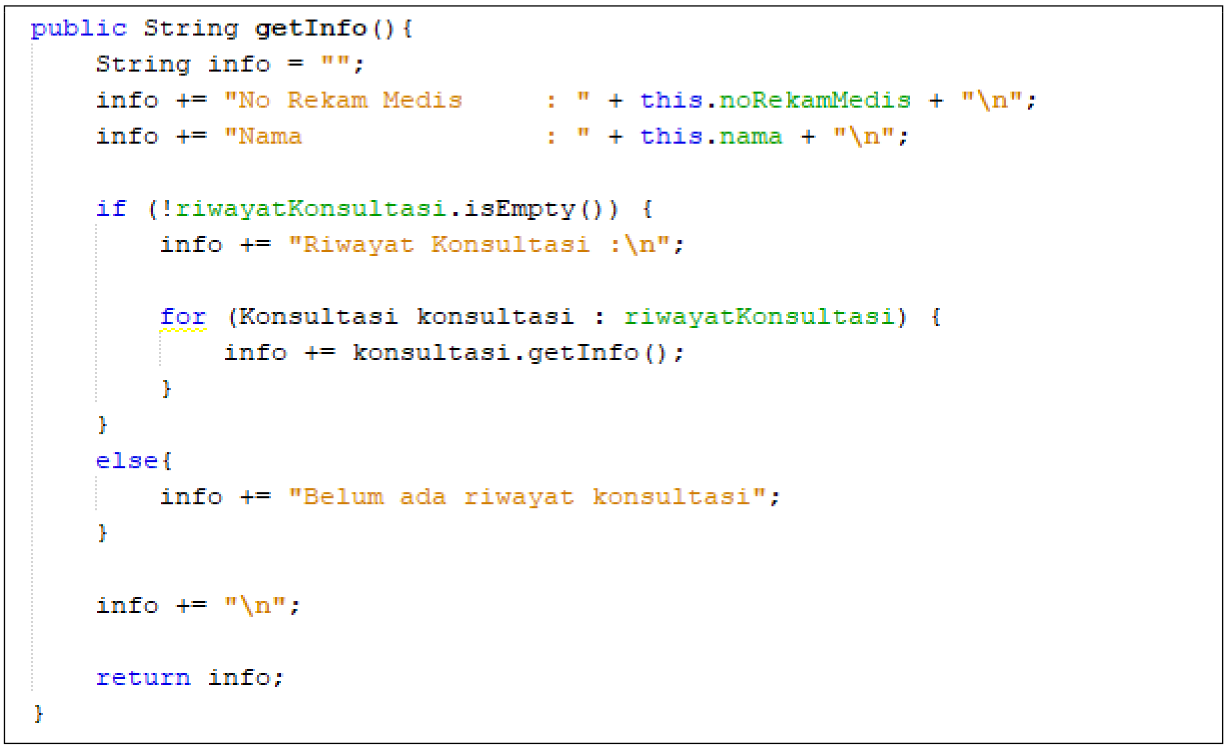
```java
public Pasien(String noRekamMedis, String nama) {
    this.noRekamMedis = noRekamMedis;
    this.nama = nama;
    this.riwayatKonsultasi = new ArrayList<Konsultasi>();
}
```

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



```java
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);
}
```

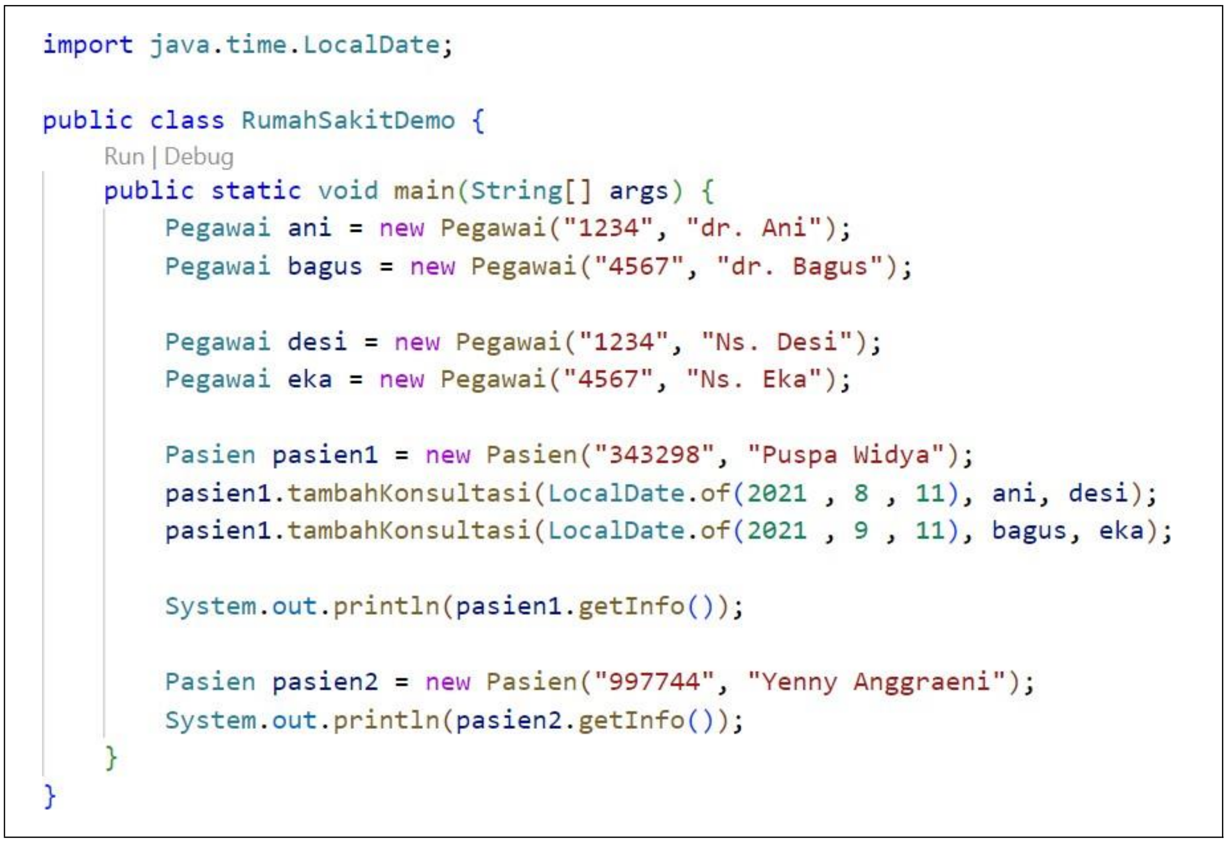
p. Modify the getInfo() method to return patient info and a list of consultations that have been done



```java
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

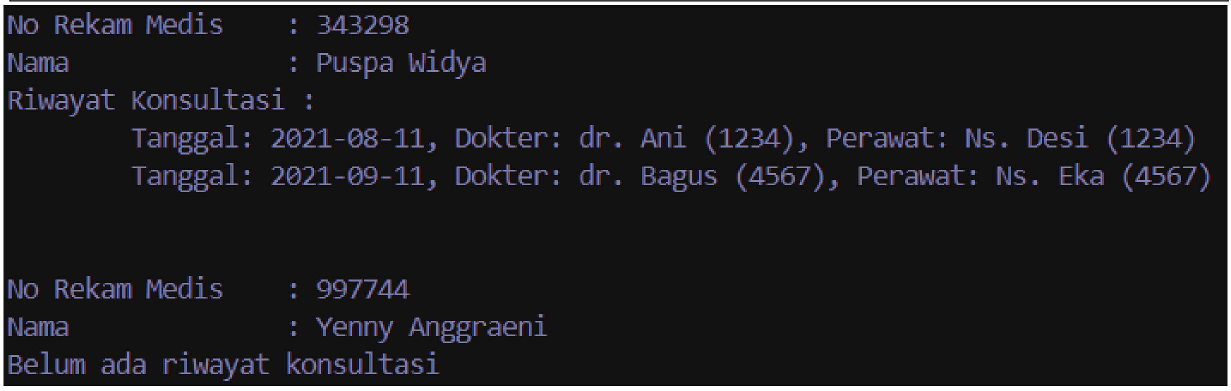
HospitalDemo class. Test the program that has been created by creating objects in the RumahSakit Demo class. The new object instance of type Employee with the name ani uses the Employee constructor (String nip, String name) with the value of the argument nip "1234" and the name "dr. Ani". Continue the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



```java
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
```



```
No Rekam Medis     : 343298
Nama               : Puspa Widya
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     : 997744
Nama               : Yenny Anggraeni
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, while **getters** are used to read the value. They help protect the data by controlling how attributes are accessed and modified.

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?**
   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()`.

3. **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.

4. **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.

5. **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.

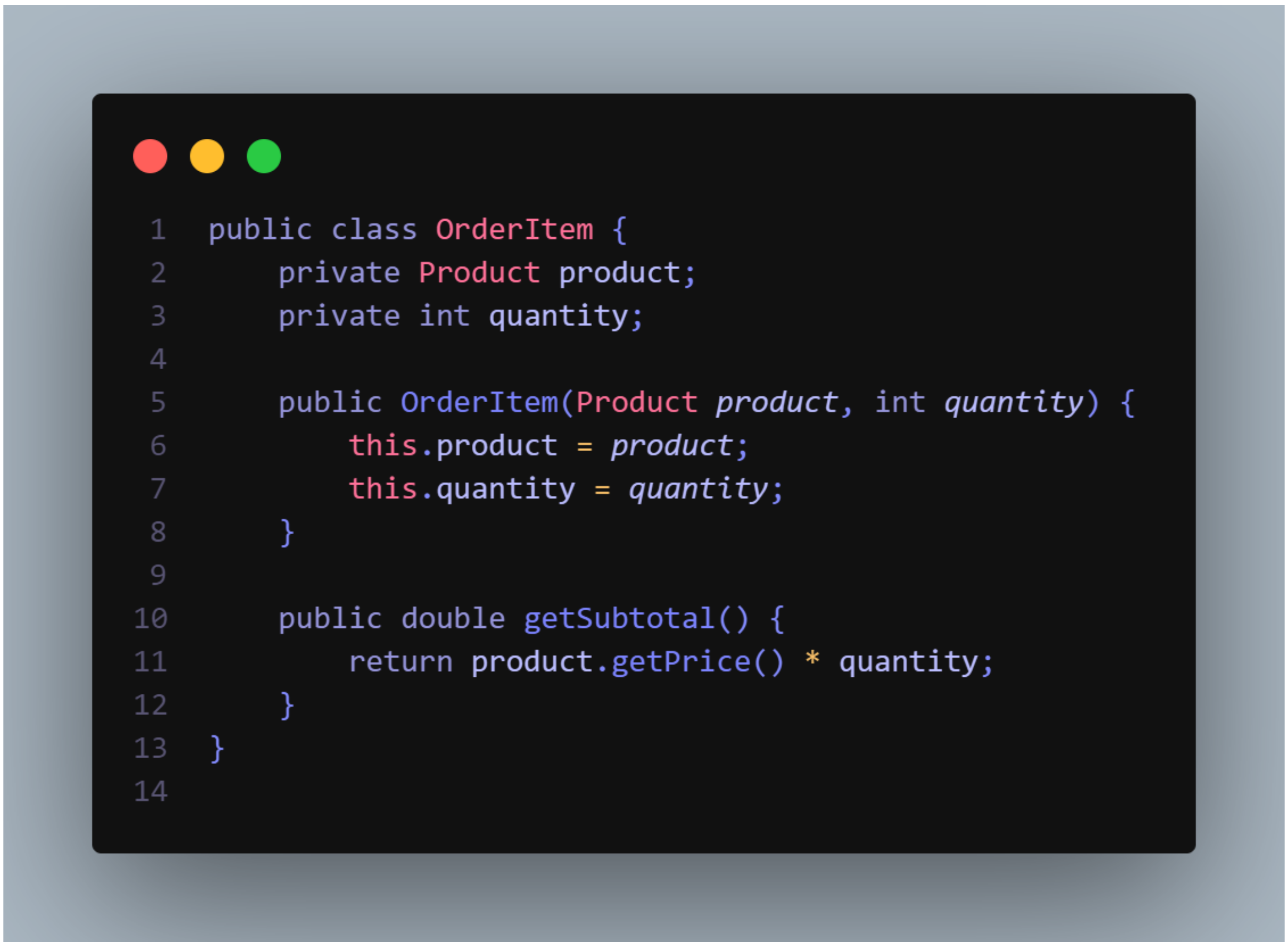
6. **In the `getInfo() method` in the Patient class, there is a line of code:**
   `if (!historyConsultation.isEmpty())`
   What does the line do?
   This line checks if there are any consultations recorded. If there are, it will show the list of consultations. If there are none, it will say there are no consultations yet.

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?
   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).

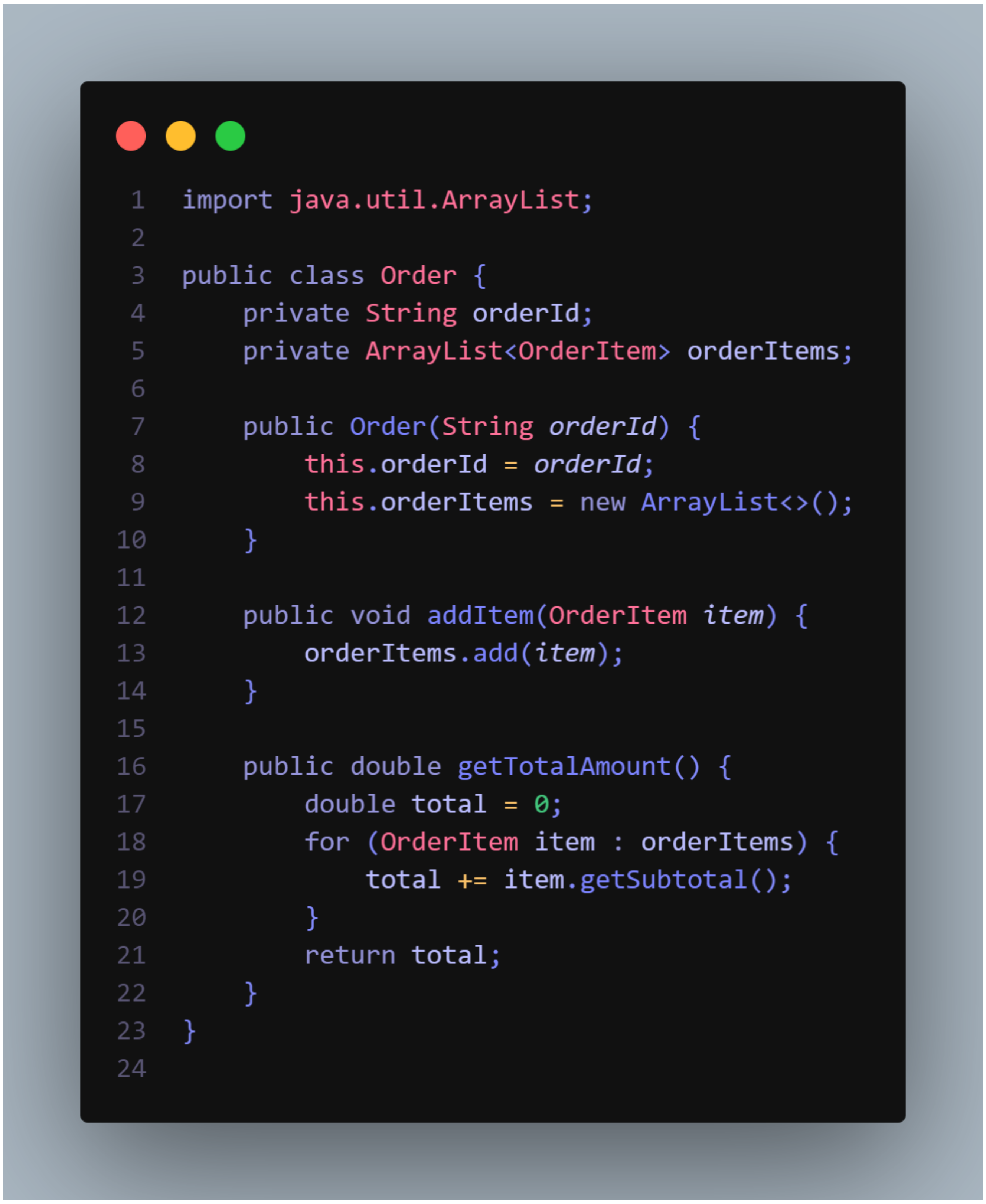
IV. Assignment

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

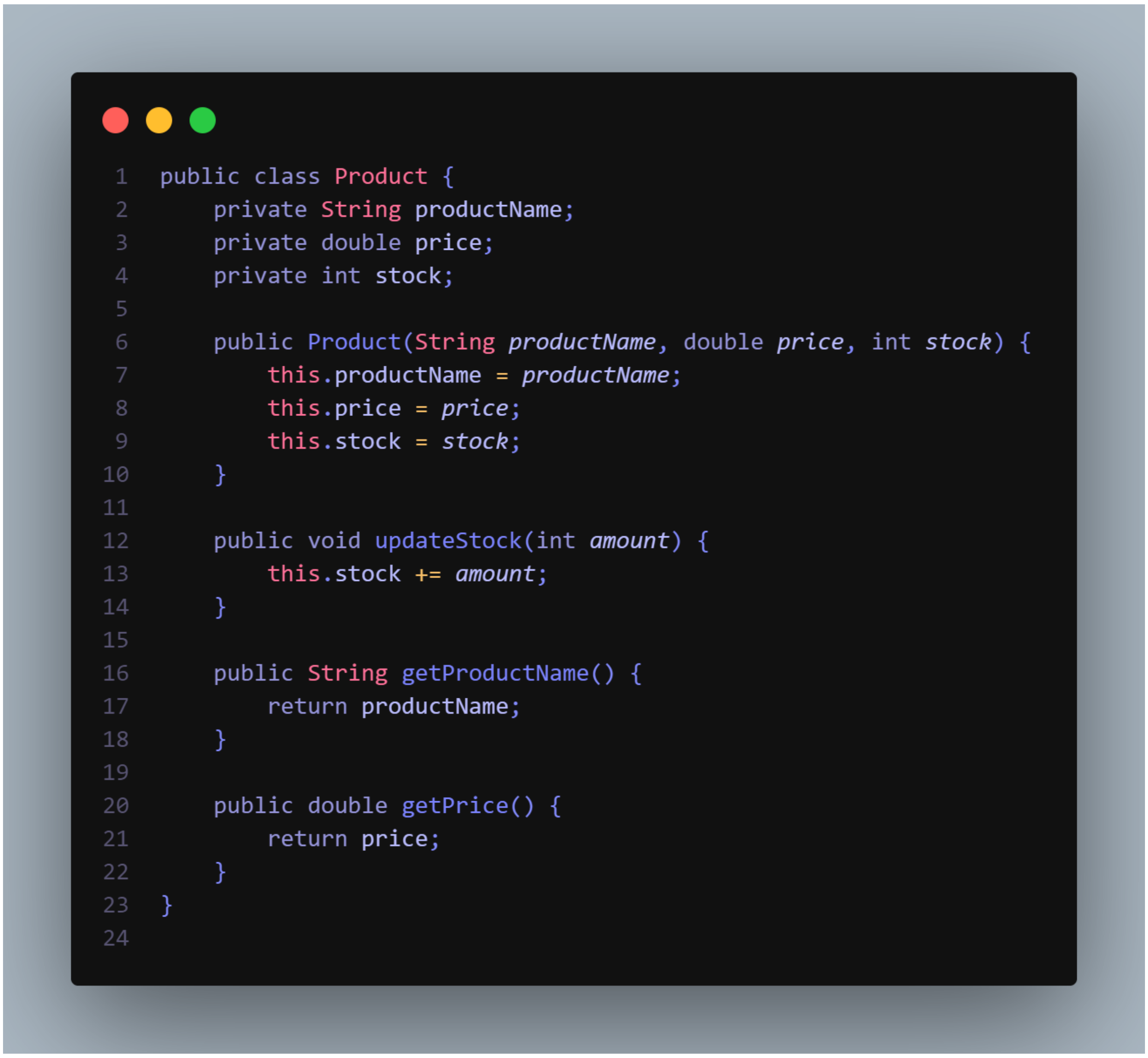
The program started by creating a market called “Traditional Market”. Within this market, there are two vendors: “John's Vegetables” and ‘Sarah's Fruits’, each with a list of products they sell. Each seller adds their products, such as tomatoes, cucumbers, apples, and bananas. The program then prints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



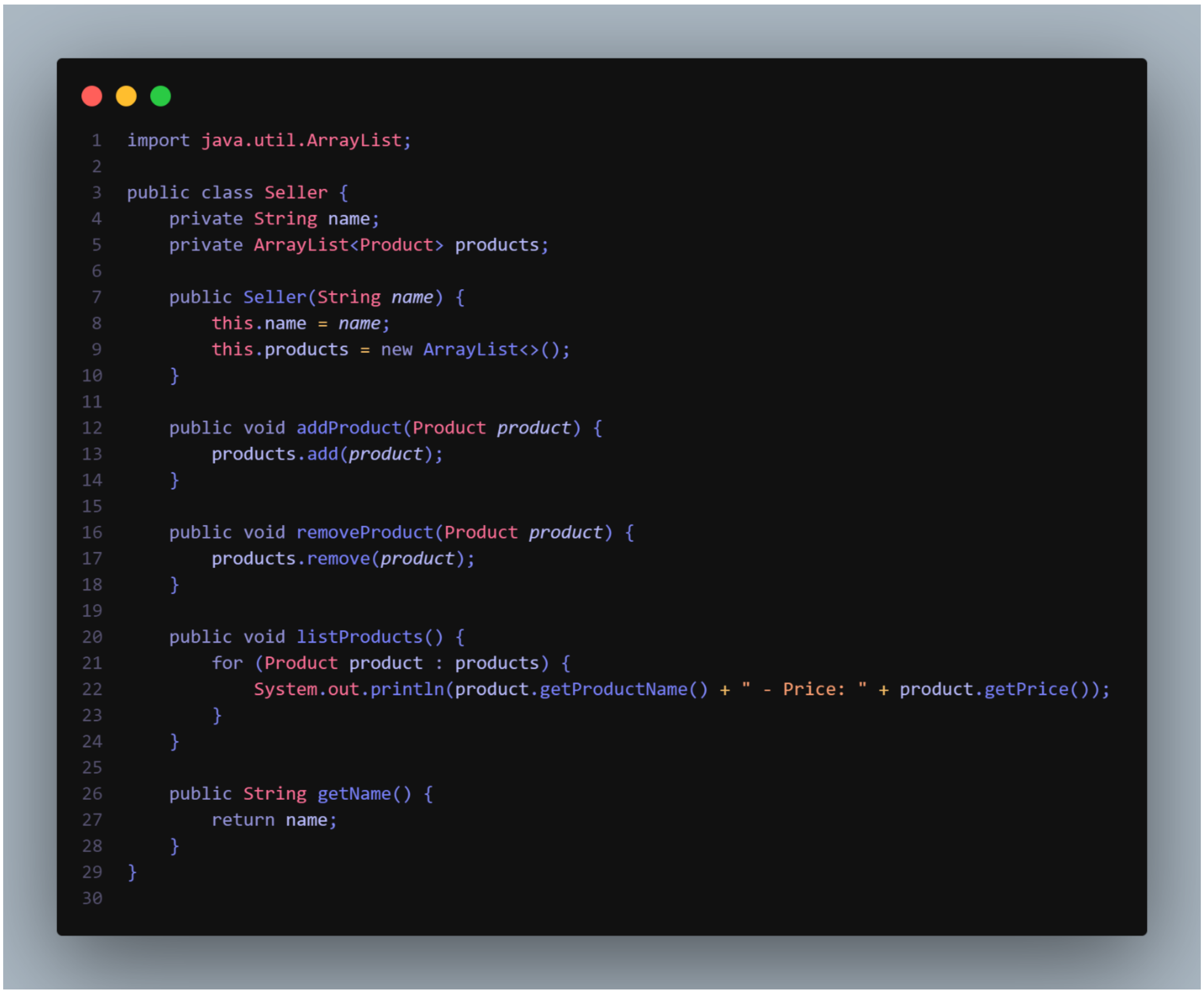
```java
public class OrderItem {
    private Product product;
    private int quantity;

    public OrderItem(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }

    public double getSubtotal() {
        return product.getPrice() * quantity;
    }
}
```



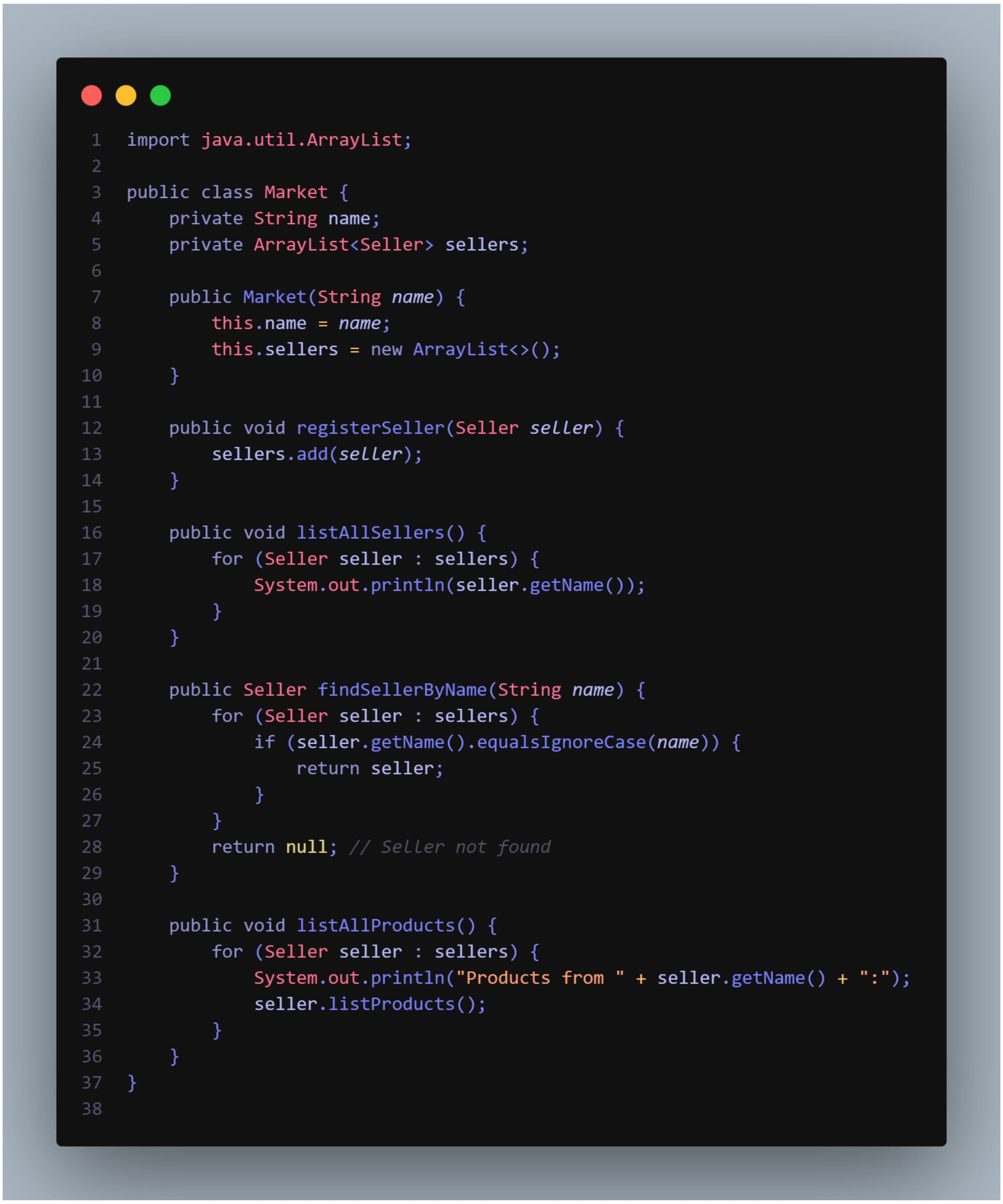
```java
import java.util.ArrayList;

public class Order {
    private String orderId;
    private ArrayList<OrderItem> orderItems;

    public Order(String orderId) {
        this.orderId = orderId;
        this.orderItems = new ArrayList<>();
    }

    public void addItem(OrderItem item) {
        orderItems.add(item);
    }

    public double getTotalAmount() {
        double total = 0;
        for (OrderItem item : orderItems) {
            total += item.getSubtotal();
        }
        return total;
    }
}

```



```java
public class Product {
    private String productName;
    private double price;
    private int stock;

    public Product(String productName, double price, int stock) {
        this.productName = productName;
        this.price = price;
        this.stock = stock;
    }

    public void updateStock(int amount) {
        this.stock += amount;
    }

    public String getProductName() {
        return productName;
    }

    public double getPrice() {
        return price;
    }
}

```

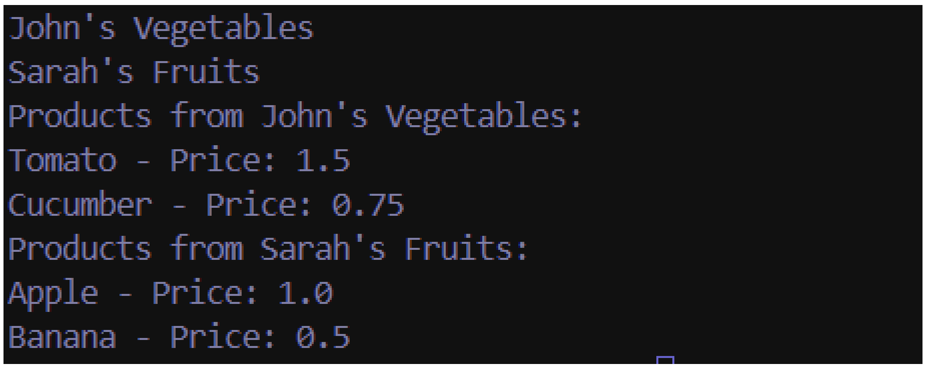


```java
import java.util.ArrayList;

public class Seller {
    private String name;
    private ArrayList<Product> products;

    public Seller(String name) {
        this.name = name;
        this.products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        products.add(product);
    }

    public void removeProduct(Product product) {
        products.remove(product);
    }

    public void listProducts() {
        for (Product product : products) {
            System.out.println(product.getProductName() + " - Price: " + product.getPrice());
        }
    }

    public String getName() {
        return name;
    }
}

```



```java
import java.util.ArrayList;

public class Market {
    private String name;
    private ArrayList<Seller> sellers;

    public Market(String name) {
        this.name = name;
        this.sellers = new ArrayList<>();
    }

    public void registerSeller(Seller seller) {
        sellers.add(seller);
    }

    public void listAllSellers() {
        for (Seller seller : sellers) {
            System.out.println(seller.getName());
        }
    }

    public Seller findSellerByName(String name) {
        for (Seller seller : sellers) {
            if (seller.getName().equalsIgnoreCase(name)) {
                return seller;
            }
        }
        return null; // Seller not found
    }

    public void listAllProducts() {
        for (Seller seller : sellers) {
            System.out.println("Products from " + seller.getName() + ":");
            seller.listProducts();
        }
    }
}

```

Result :



```
John's Vegetables
Sarah's Fruits
Products from John's Vegetables:
Tomato - Price: 1.5
Cucumber - Price: 0.75
Products from Sarah's Fruits:
Apple - Price: 1.0
Banana - Price: 0.5
```