Zadanie 3

First Name: Illia
Second name: Ponomarov
Mail: xponomarov@stuba.sk
ID: 113047

Project name: Best HS Manager

(HS - home store)

Project description:

My project will be to simulate the purchase of the Customer with the Manager. The project will simulate a purchase process or a ban on goods from the Home Store (For example IKEA store).

The goal of the project

Goal of Customer:

The main goal of the Customer is for the Customer to find the product he needs, and so that he meets his criteria.

Goal of Manager:

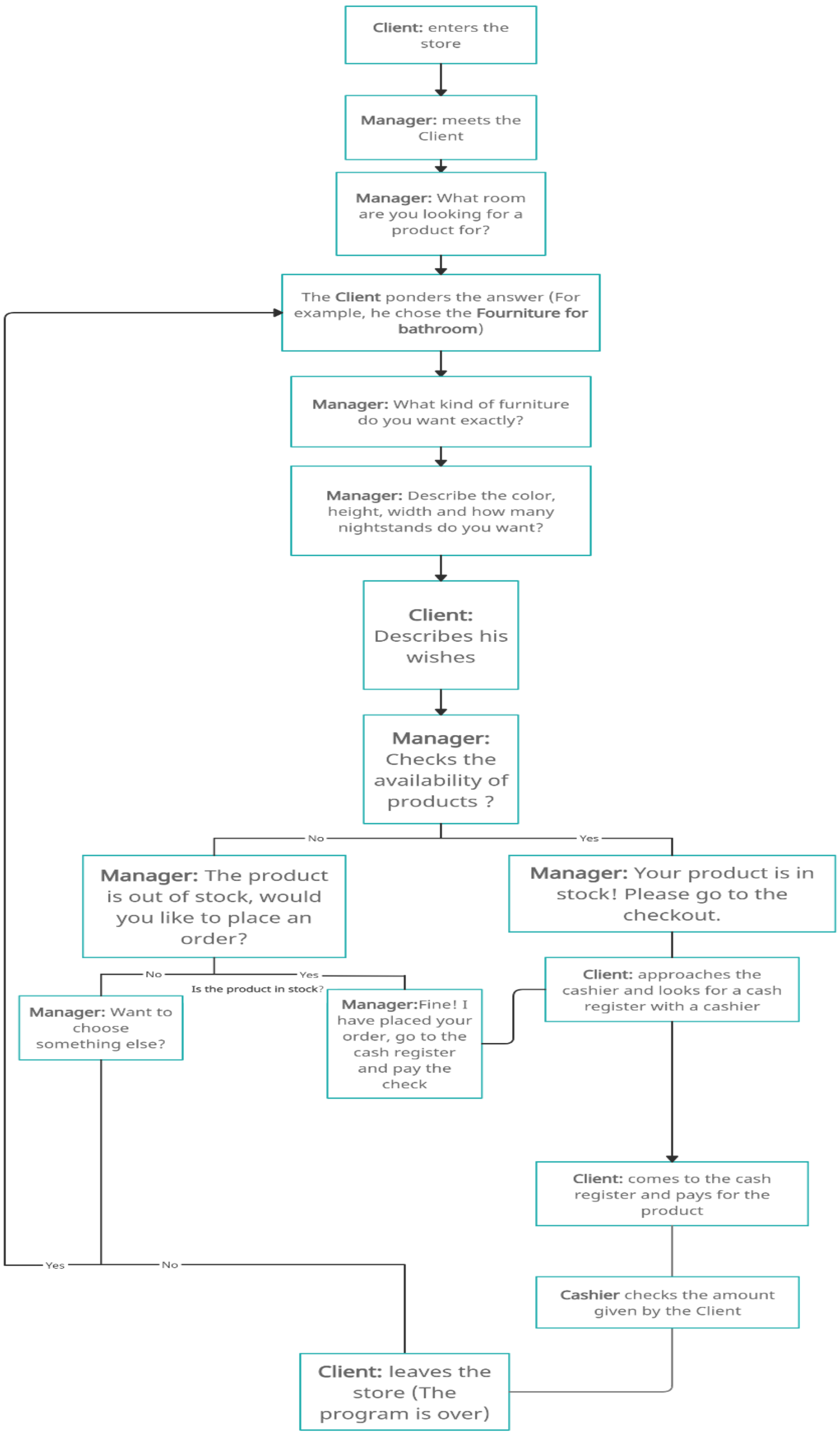
The manager must help the customer find the right product using the information he has about rooms and products.

Goal of Cashier:

The cashier must check the availability of the check and the amount of the Client's money. Then punch through the goods.

Logic diagram:

Before starting to write the Technical part of the project (Classes, etc.), we need to draw up a logical diagram

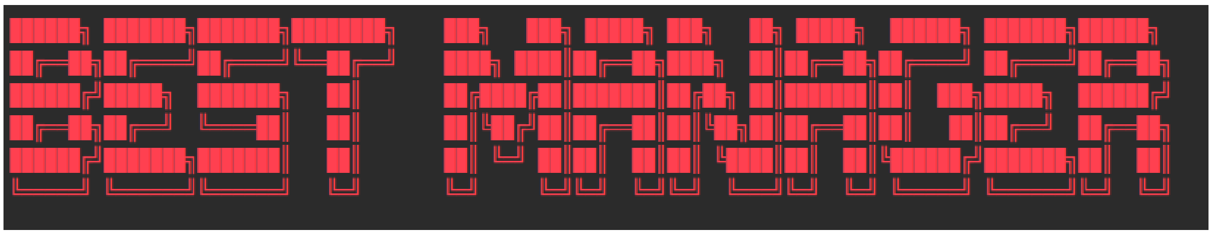


Client: enters the store
Manager: meets the Client
Manager: What room are you looking for a product for?
The Client ponders the answer (For example, he chose the Fourniture for bathroom)
Manager: What kind of furniture do you want exactly?
Manager: Describe the color, height, width and how many nightstands do you want?
Client: Describes his wishes
Manager: Checks the availability of products ?
No
Yes
Manager: The product is out of stock, would you like to place an order?
Manager: Your product is in stock! Please go to the checkout.
No
Yes
Is the product in stock?
Manager: Want to choose something else?
Manager:Fine! I have placed your order, go to the cash register and pay the check
Client: approaches the cashier and looks for a cash register with a cashier
Client: comes to the cash register and pays for the product
Yes
No
Cashier checks the amount given by the Client
Client: leaves the store (The program is over)

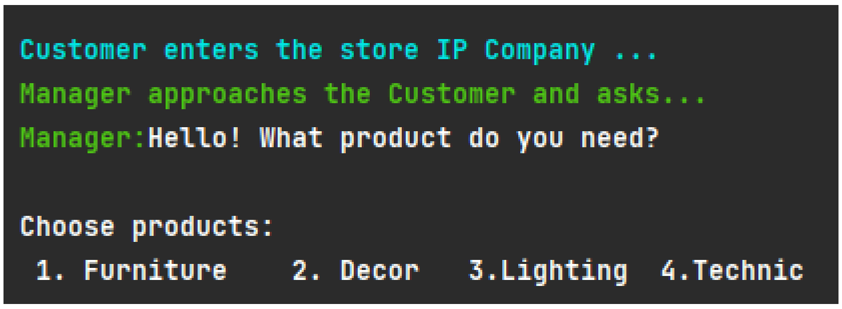
Program interface:

In this part, you will see how the entire interface of my program will look, from start to finish.

1. When you run the program, you will see the logo of my program.

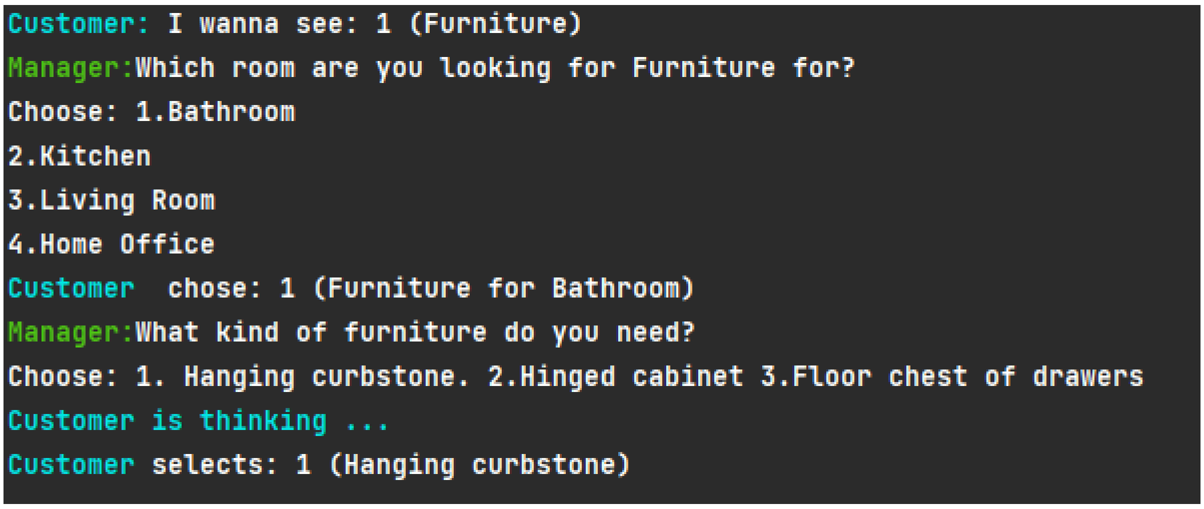


2. Then the program starts working.
   The customer enters the store and meets the manager. The manager asks what type of product you want to buy.



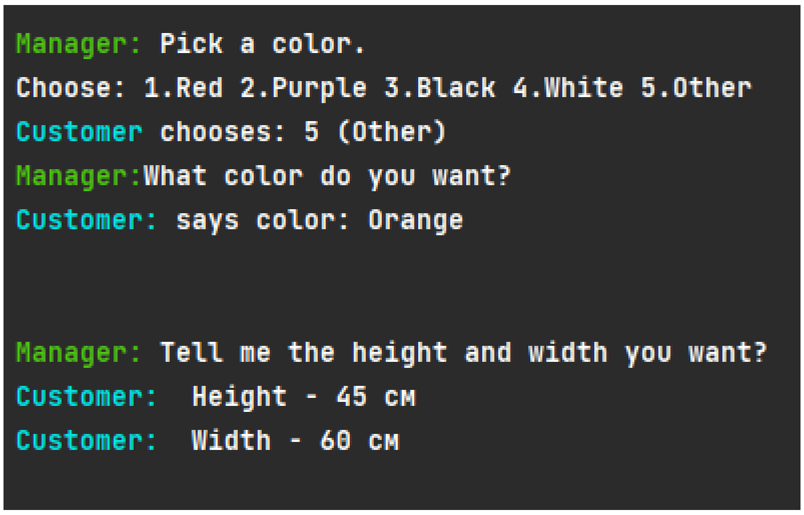
```
Customer enters the store IP Company ...
Manager approaches the Customer and asks...
Manager:Hello! What product do you need?

Choose products:
 1. Furniture    2. Decor   3.Lighting  4.Technic
```

3. In the third step, you choose for which room you need your product so that the manager can pick up what you are looking for.

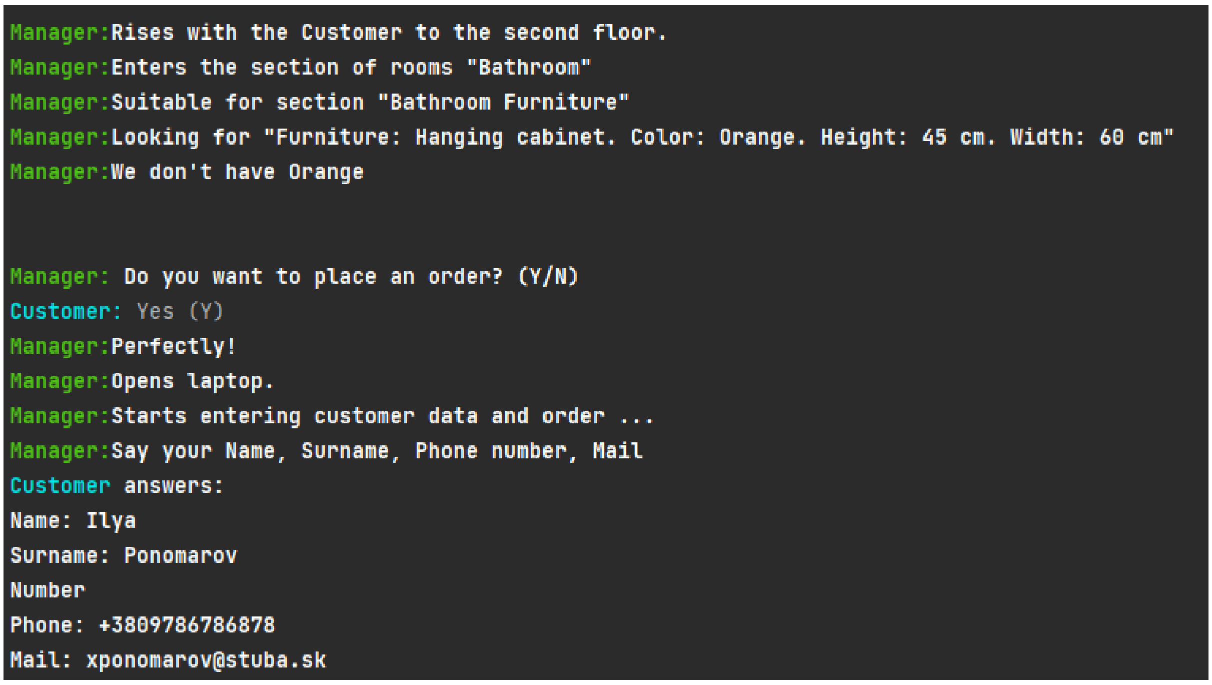


```
Customer: I wanna see: 1 (Furniture)
Manager:Which room are you looking for Furniture for?
Choose: 1.Bathroom
2.Kitchen
3.Living Room
4.Home Office
Customer  chose: 1 (Furniture for Bathroom)
Manager:What kind of furniture do you need?
Choose: 1. Hanging curbstone. 2.Hinged cabinet 3.Floor chest of drawers
Customer is thinking ...
Customer selects: 1 (Hanging curbstone)
```

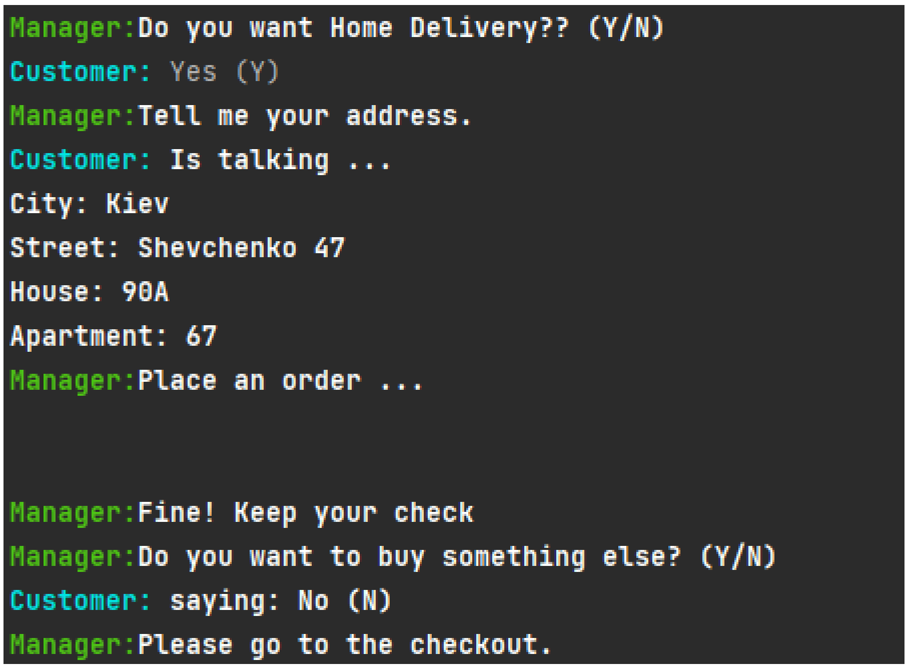
4. In the 4th step, you choose what width, height and color you need the product (width and height is needed if you are an object that needs this data).


```
Manager: Pick a color.
Choose: 1.Red 2.Purple 3.Black 4.White 5.Other
Customer chooses: 5 (Other)
Manager:What color do you want?
Customer: says color: Orange


Manager: Tell me the height and width you want?
Customer:  Height - 45 cm
Customer:  Width - 60 cm
```

5. In the 5th step, the manager starts looking for your product. First he will go to the **desired section of rooms -> find the type of furniture for this room -> look for furniture according to your characteristics**. If he does not find it, he offers to order a product.

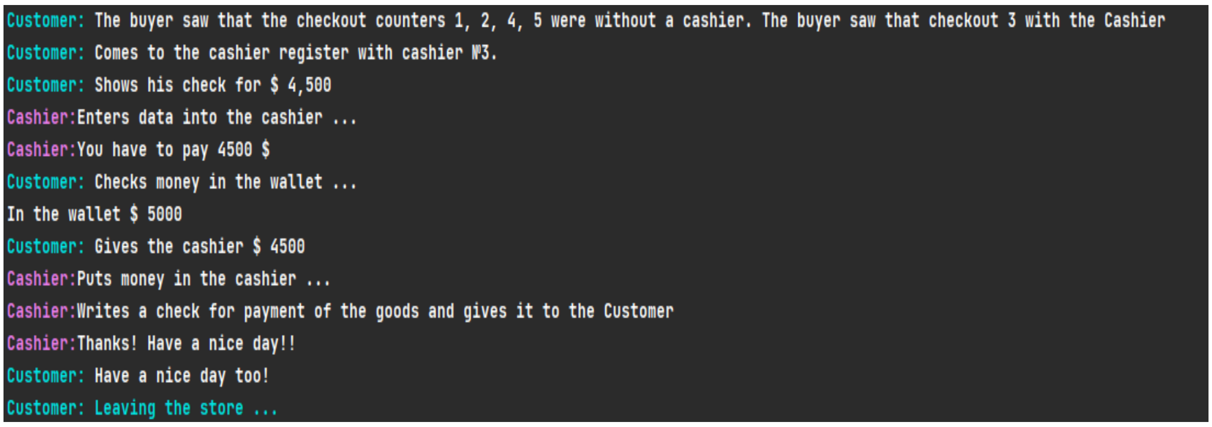

```
Manager:Rises with the Customer to the second floor.
Manager:Enters the section of rooms "Bathroom"
Manager:Suitable for section "Bathroom Furniture"
Manager:Looking for "Furniture: Hanging cabinet. Color: Orange. Height: 45 cm. Width: 60 cm"
Manager:We don't have Orange


Manager: Do you want to place an order? (Y/N)
Customer: Yes (Y)
Manager:Perfectly!
Manager:Opens laptop.
Manager:Starts entering customer data and order ...
Manager:Say your Name, Surname, Phone number, Mail
Customer answers:
Name: Ilya
Surname: Ponomarov
Number
Phone: +3809786786878
Mail: xponomarov@stuba.sk
```

6. Then the manager will ask if you want home delivery, if so, he will ask for your address. After registration, he will offer to look at something else. If you refuse, he will give you a check.



```
Manager:Do you want Home Delivery?? (Y/N)
Customer: Yes (Y)
Manager:Tell me your address.
Customer: Is talking ...
City: Kiev
Street: Shevchenko 47
House: 90A
Apartment: 67
Manager:Place an order ...


Manager:Fine! Keep your check
Manager:Do you want to buy something else? (Y/N)
Customer: saying: No (N)
Manager:Please go to the checkout.
```

7. After you receive a check from the manager, you go down and search for a free cashier, when the Customer finds a cashier, he will come to it. The cashier will check your check and ask you to pay. The customer will check wallet, and if the required amount.



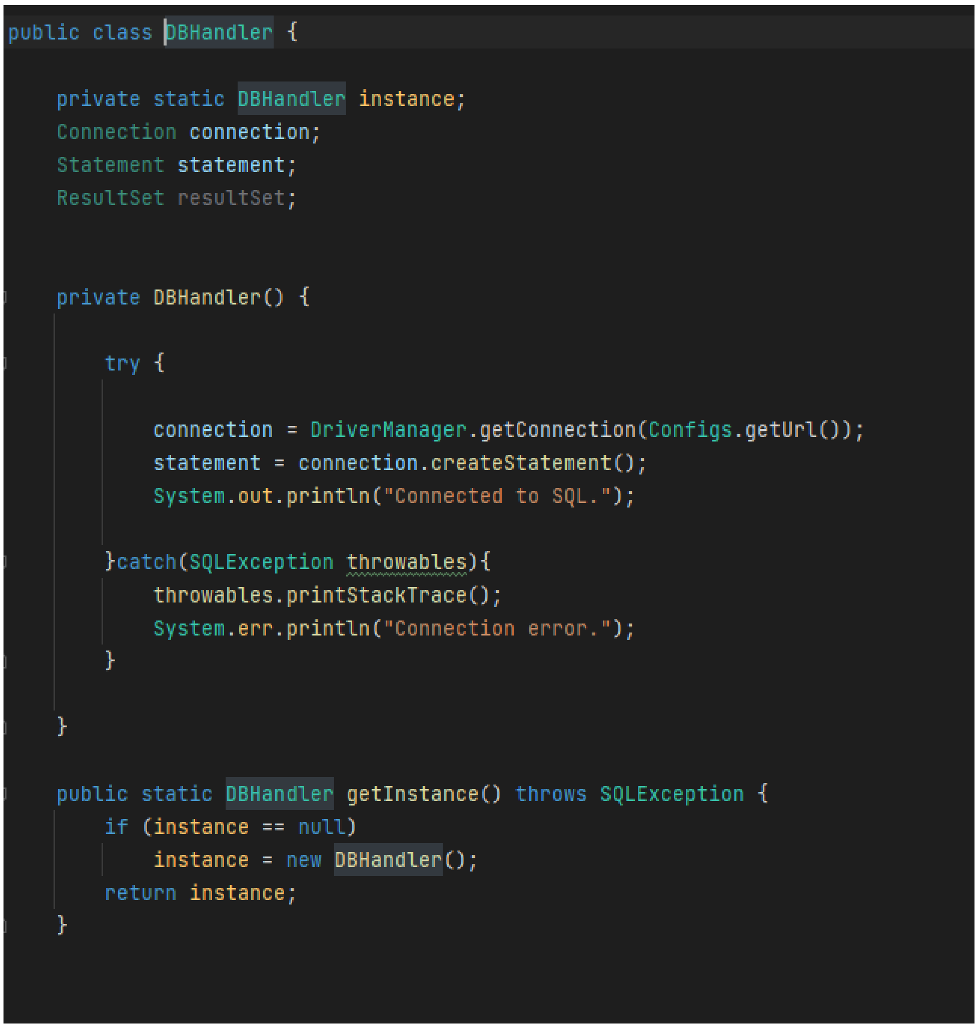
```
Customer: The buyer saw that the checkout counters 1, 2, 4, 5 were without a cashier. The buyer saw that checkout 3 with the Cashier
Customer: Comes to the cashier register with cashier №3.
Customer: Shows his check for $ 4,500
Cashier:Enters data into the cashier ...
Cashier:You have to pay 4500 $
Customer: Checks money in the wallet ...
In the wallet $ 5000
Customer: Gives the cashier $ 4500
Cashier:Puts money in the cashier ...
Cashier:Writes a check for payment of the goods and gives it to the Customer
Cashier:Thanks! Have a nice day!!
Customer: Have a nice day too!
Customer: Leaving the store ...
```

Technical part:

This part will describe what my program will consist of (classes, methods). I will describe in great detail how my objects will interact with each other.

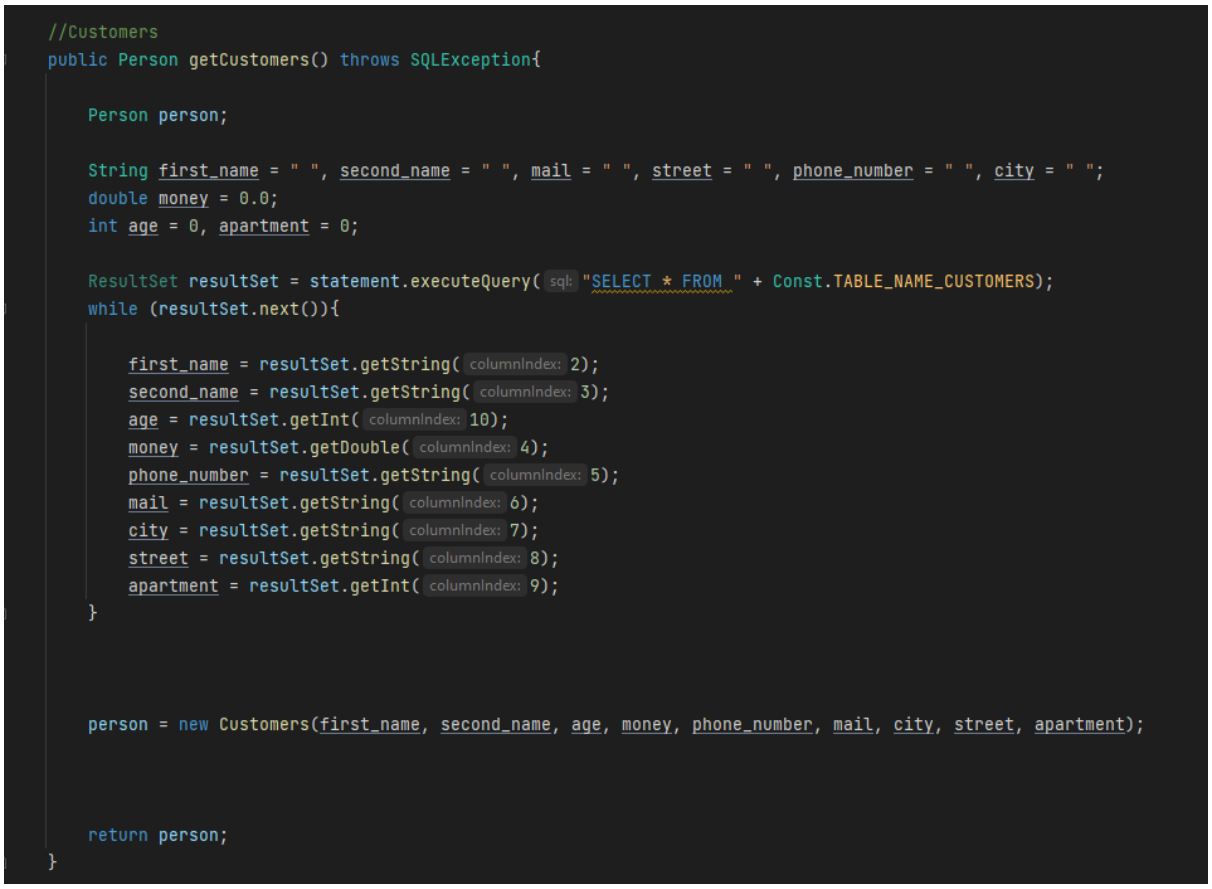
1. **Implementation core functions:**
   First, I connected a SQLite database and wrote the basic functions for connecting.

   1. In the DBHandler class, I used the Singletone design pattern. In order that it would not be possible to create an instance of this object, because it is unlikely with this pattern, I will be able to create a new instance of this object, and I will not have problems with the connection.

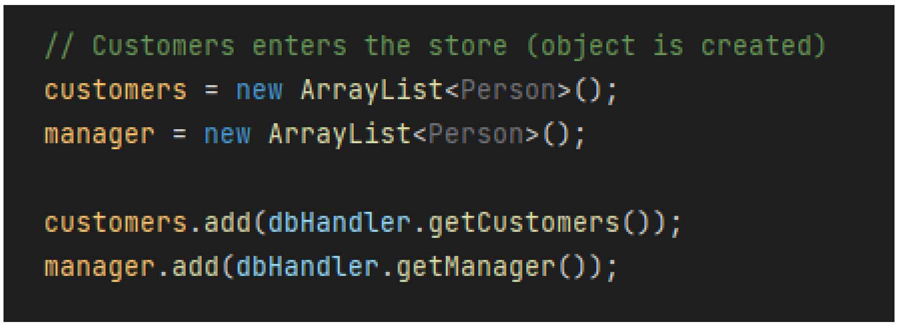


```java
public class DBHandler {

    private static DBHandler instance;
    Connection connection;
    Statement statement;
    ResultSet resultSet;


    private DBHandler() {

        try {

            connection = DriverManager.getConnection(Configs.getUrl());
            statement = connection.createStatement();
            System.out.println("Connected to SQL.");

        }catch(SQLException throwables){
            throwables.printStackTrace();
            System.err.println("Connection error.");
        }

    }

    public static DBHandler getInstance() throws SQLException {
        if (instance == null)
            instance = new DBHandler();
        return instance;
    }
```

2. Then, in the DBHandler class, I wrote the function “public Person getCustomers ()”, which fill the object with data, return this object, and then add this object to the ArrayList <Person> customers.

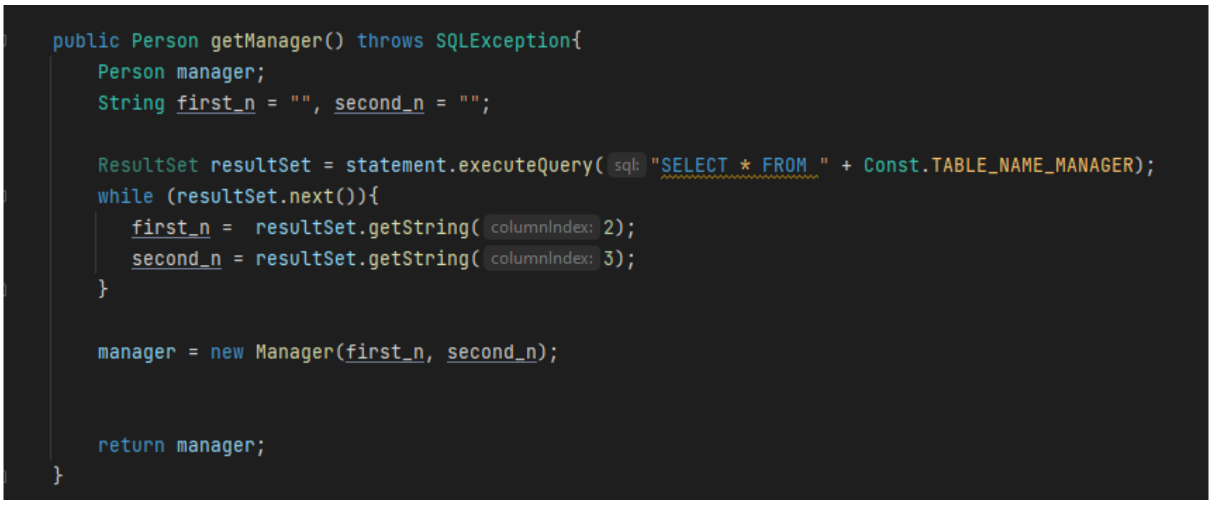


```java
//Customers
public Person getCustomers() throws SQLException{

    Person person;

    String first_name = " ", second_name = " ", mail = " ", street = " ", phone_number = " ", city = " ";
    double money = 0.0;
    int age = 0, apartment = 0;

    ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM " + Const.TABLE_NAME_CUSTOMERS);
    while (resultSet.next()){

        first_name = resultSet.getString( columnIndex: 2);
        second_name = resultSet.getString( columnIndex: 3);
        age = resultSet.getInt( columnIndex: 10);
        money = resultSet.getDouble( columnIndex: 4);
        phone_number = resultSet.getString( columnIndex: 5);
        mail = resultSet.getString( columnIndex: 6);
        city = resultSet.getString( columnIndex: 7);
        street = resultSet.getString( columnIndex: 8);
        apartment = resultSet.getInt( columnIndex: 9);
    }



    person = new Customers(first_name, second_name, age, money, phone_number, mail, city, street, apartment);



    return person;
}
```

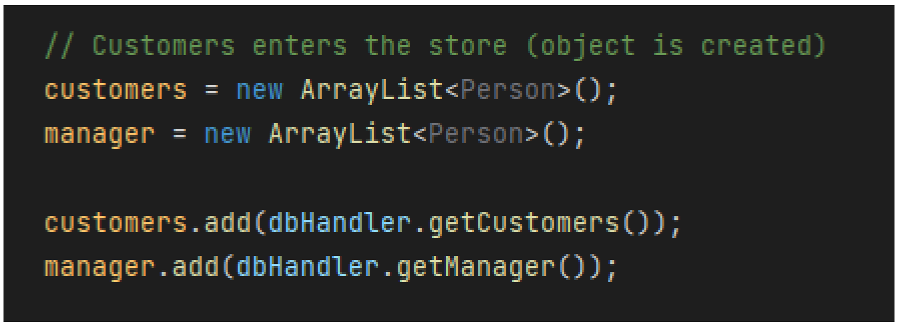


```java
// Customers enters the store (object is created)
customers = new ArrayList<Person>();
manager = new ArrayList<Person>();

customers.add(dbHandler.getCustomers());
manager.add(dbHandler.getManager());
```

3. I also designed the “public Person getManager ()” function:

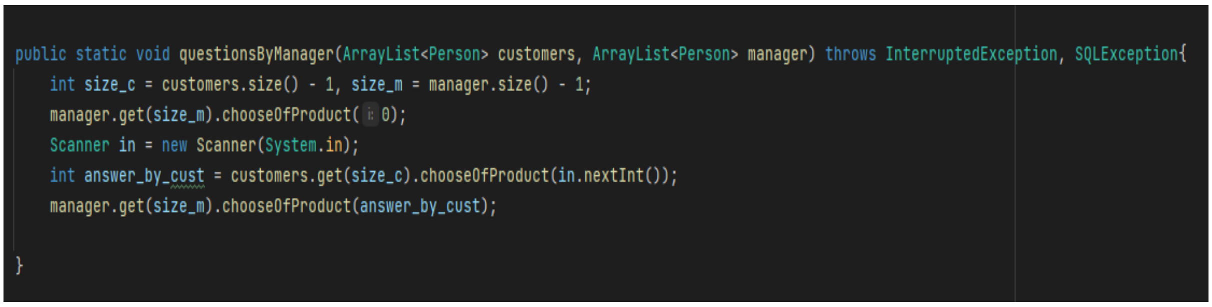


```
public Person getManager() throws SQLException{
    Person manager;
    String first_n = "", second_n = "";

    ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM " + Const.TABLE_NAME_MANAGER);
    while (resultSet.next()){
        first_n =  resultSet.getString( columnIndex: 2);
        second_n = resultSet.getString( columnIndex: 3);
    }

    manager = new Manager(first_n, second_n);


    return manager;
}
```



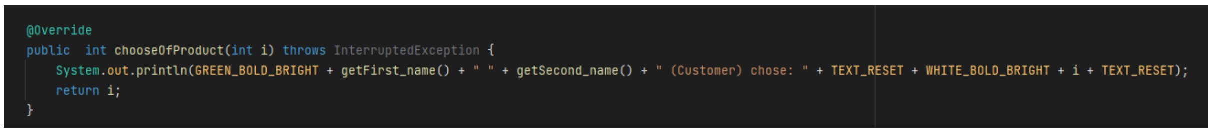
```
// Customers enters the store (object is created)
customers = new ArrayList<Person>();
manager = new ArrayList<Person>();

customers.add(dbHandler.getCustomers());
manager.add(dbHandler.getManager());
```

4. One of the main functions is “questionsByManager”, this is one of the most important functions because it starts the interaction between the two objects “Customer” and “Manager”. The manager asks what the Buyer wants.



```
public static void questionsByManager(ArrayList<Person> customers, ArrayList<Person> manager) throws InterruptedException, SQLException{
    int size_c = customers.size() - 1, size_m = manager.size() - 1;
    manager.get(size_m).chooseOfProduct( i: 0);
    Scanner in = new Scanner(System.in);
    int answer_by_cust = customers.get(size_c).chooseOfProduct(in.nextInt());
    manager.get(size_m).chooseOfProduct(answer_by_cust);

}
```

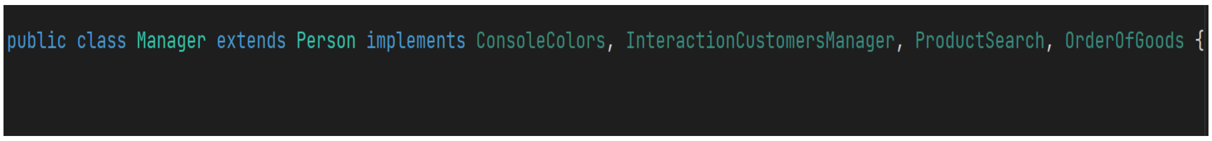
5. The “chooseOfProduct” function - it is responsible for the choice that the Buyer will make, after which the Manager object receives the answer.



```java
@Override
public  int chooseOfProduct(int i) throws InterruptedException {
    System.out.println(GREEN_BOLD_BRIGHT + getFirst_name() + " " + getSecond_name() + " (Customer) chose: " + TEXT_RESET + WHITE_BOLD_BRIGHT + i + TEXT_RESET);
    return i;
}
```

2. Inheritance and Multiple Inheritance:

The first best example for Multiple Inheritance would be the Class Manager, since it inherits from Person itself, and implements many of the interfaces it uses.



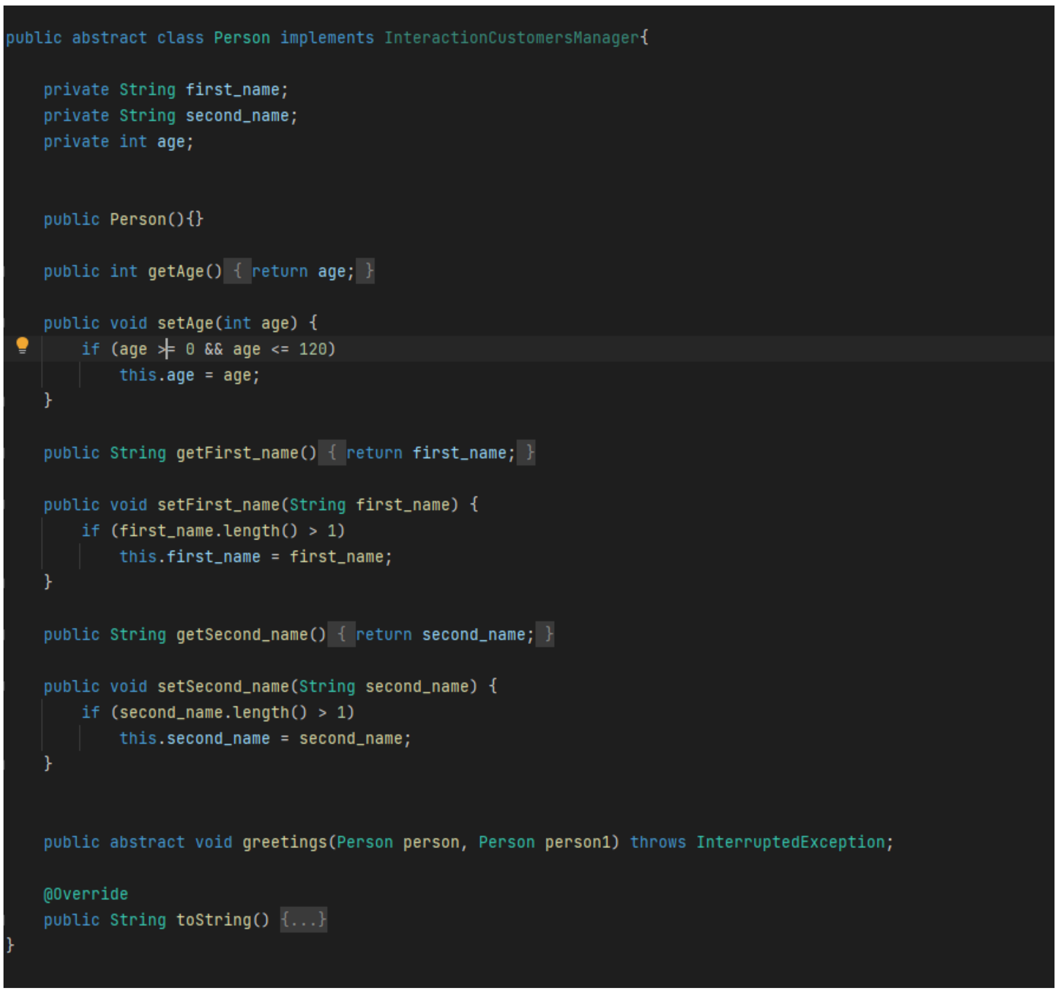
```java
public class Manager extends Person implements ConsoleColors, InteractionCustomersManager, ProductSearch, OrderOfGoods {
```

1. **Inheritance**
   I have a large number of inheritances that I use in a program, but the most important example of inheritance is class Person and class Product, which are abstract, from which a large number of classes inherit.

   Person is a superclass for **Customer, Manager, Cashier classes.**

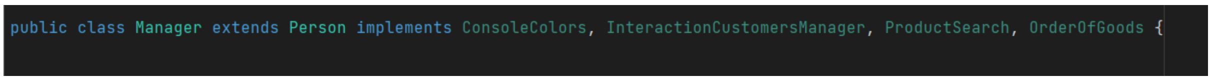
   Product is a superclass for **Technic, Furniture, Lighting classes.**

Person:



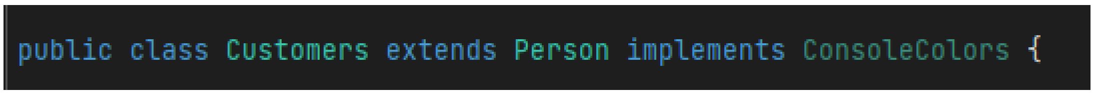
```java
public abstract class Person implements InteractionCustomersManager{

    private String first_name;
    private String second_name;
    private int age;


    public Person(){}

    public int getAge() { return age; }

    public void setAge(int age) {
        if (age >= 0 && age <= 120)
            this.age = age;
    }

    public String getFirst_name() { return first_name; }

    public void setFirst_name(String first_name) {
        if (first_name.length() > 1)
            this.first_name = first_name;
    }

    public String getSecond_name() { return second_name; }

    public void setSecond_name(String second_name) {
        if (second_name.length() > 1)
            this.second_name = second_name;
    }


    public abstract void greetings(Person person, Person person1) throws InterruptedException;

    @Override
    public String toString() {...}
}
```

Manager:



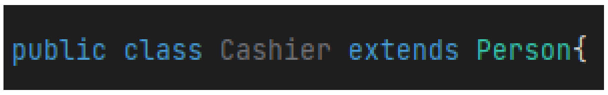
```java
public class Manager extends Person implements ConsoleColors, InteractionCustomersManager, ProductSearch, OrderOfGoods {
```

Customers:



```java
public class Customers extends Person implements ConsoleColors {
```

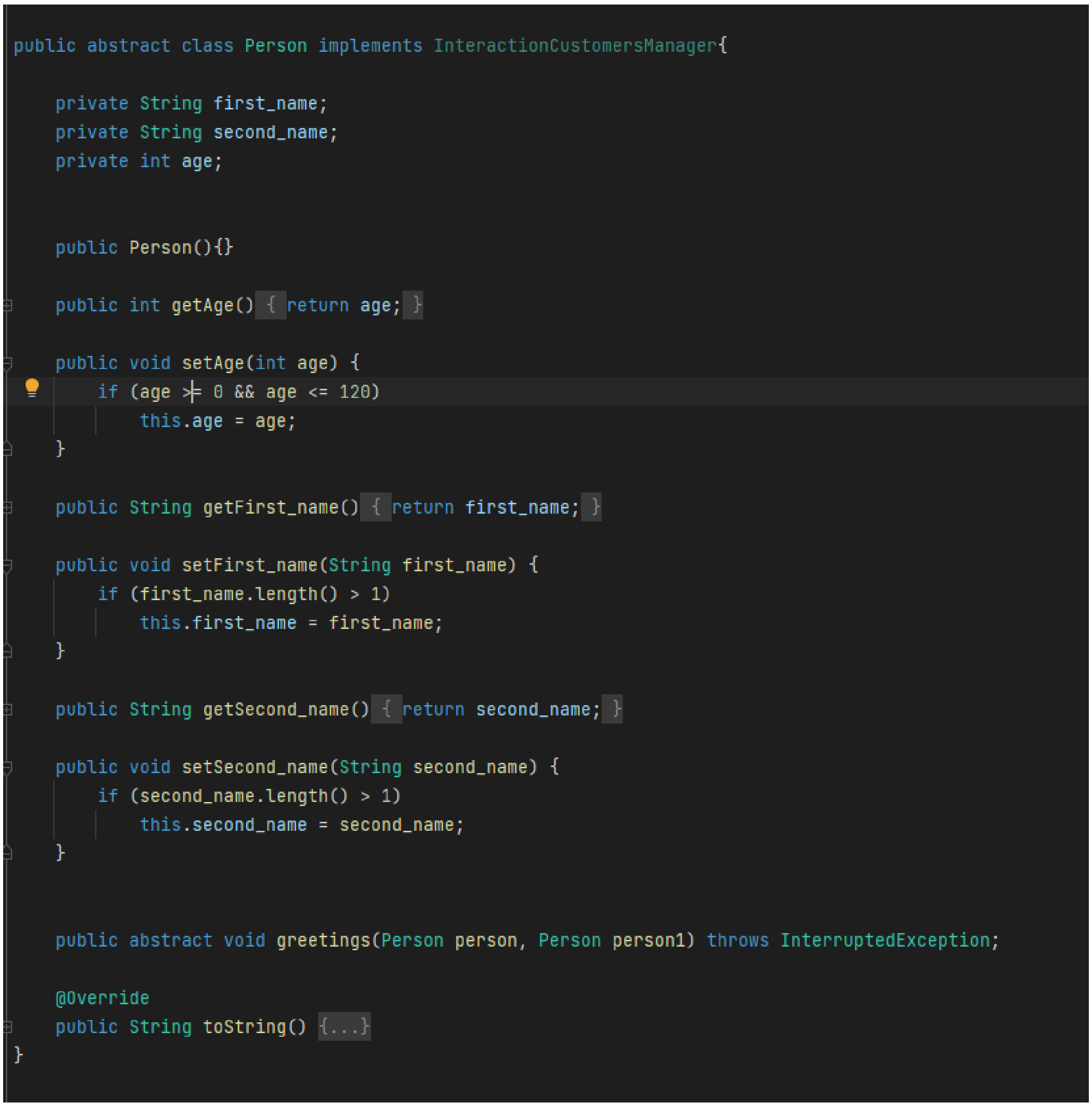
Cashier:



```java
public class Cashier extends Person{
```

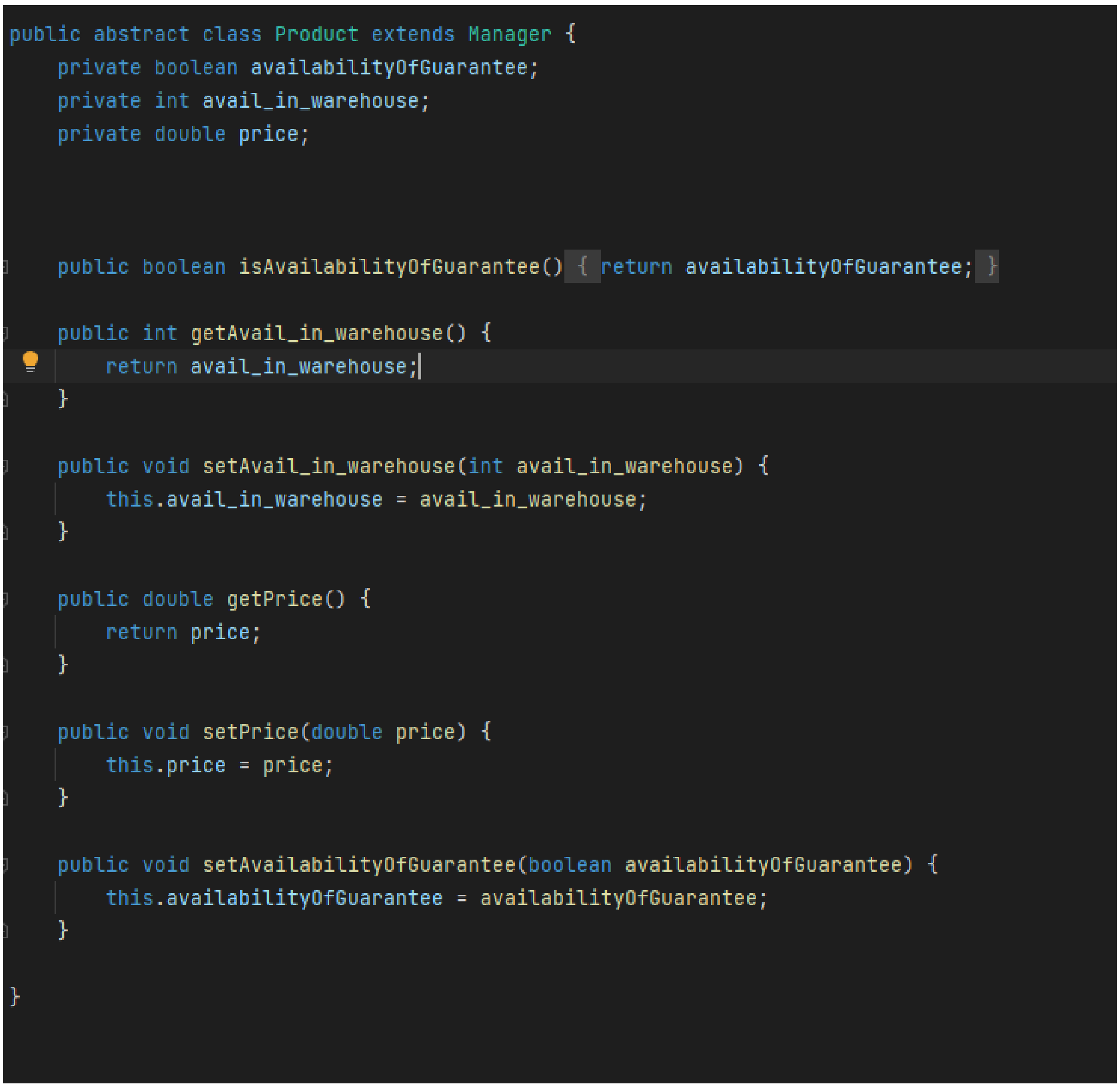
3. Encapsulation and access modifiers:

a)



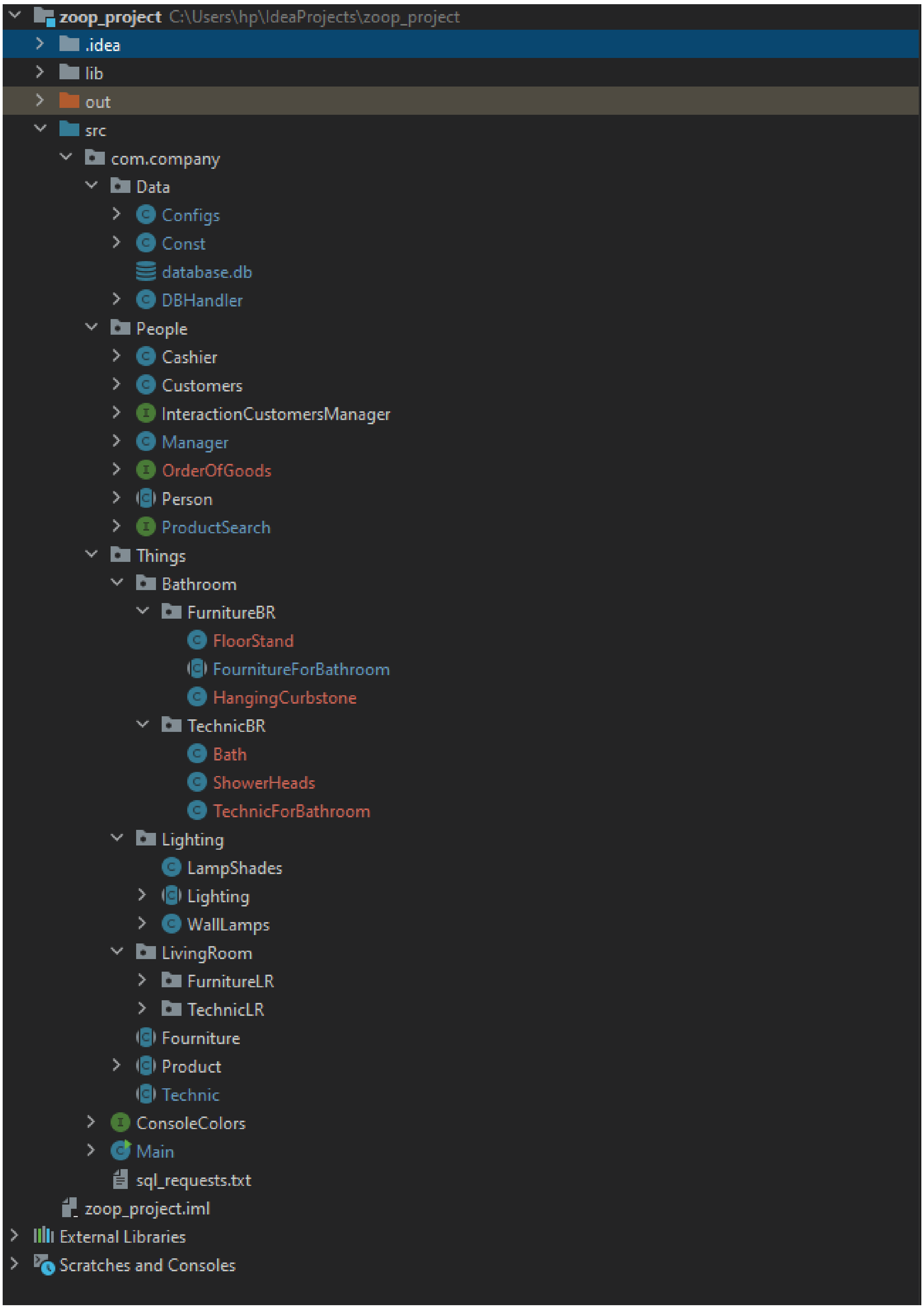
```java
public abstract class Person implements InteractionCustomersManager{

    private String first_name;
    private String second_name;
    private int age;


    public Person(){}

    public int getAge() { return age; }

    public void setAge(int age) {
        if (age >= 0 && age <= 120)
            this.age = age;
    }

    public String getFirst_name() { return first_name; }

    public void setFirst_name(String first_name) {
        if (first_name.length() > 1)
            this.first_name = first_name;
    }

    public String getSecond_name() { return second_name; }

    public void setSecond_name(String second_name) {
        if (second_name.length() > 1)
            this.second_name = second_name;
    }


    public abstract void greetings(Person person, Person person1) throws InterruptedException;

    @Override
    public String toString() {...}
}
```

b)



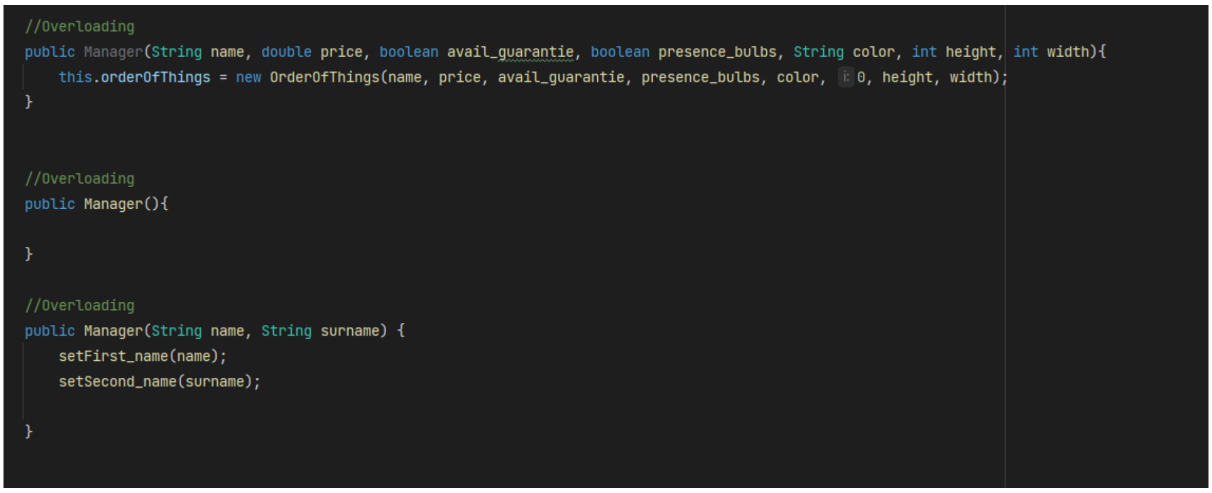
```java
public abstract class Product extends Manager {
    private boolean availabilityOfGuarantee;
    private int avail_in_warehouse;
    private double price;



    public boolean isAvailabilityOfGuarantee() { return availabilityOfGuarantee; }

    public int getAvail_in_warehouse() {
        return avail_in_warehouse;
    }

    public void setAvail_in_warehouse(int avail_in_warehouse) {
        this.avail_in_warehouse = avail_in_warehouse;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public void setAvailabilityOfGuarantee(boolean availabilityOfGuarantee) {
        this.availabilityOfGuarantee = availabilityOfGuarantee;
    }

}
```

4. Code organized into package:



```
zoop_project C:\Users\hp\IdeaProjects\zoop_project
  .idea
  lib
  out
  src
    com.company
      Data
        Configs
        Const
        database.db
        DBHandler
      People
        Cashier
        Customers
        InteractionCustomersManager
        Manager
        OrderOfGoods
        Person
        ProductSearch
      Things
        Bathroom
          FurnitureBR
            FloorStand
            FournitureForBathroom
            HangingCurbstone
          TechnicBR
            Bath
            ShowerHeads
            TechnicForBathroom
        Lighting
          LampShades
          Lighting
          WallLamps
        LivingRoom
          FurnitureLR
          TechnicLR
        Fourniture
        Product
        Technic
      ConsoleColors
      Main
      sql_requests.txt
  zoop_project.iml
External Libraries
Scratches and Consoles
```

5. Overloading:

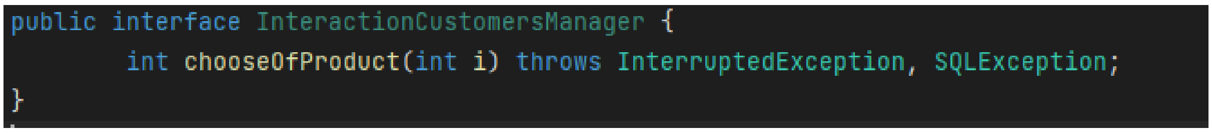


```java
//Overloading
public Manager(String name, double price, boolean avail_guarantie, boolean presence_bulbs, String color, int height, int width){
    this.orderOfThings = new OrderOfThings(name, price, avail_guarantie, presence_bulbs, color, i: 0, height, width);
}


//Overloading
public Manager(){

}

//Overloading
public Manager(String name, String surname) {
    setFirst_name(name);
    setSecond_name(surname);

}
```

6. Overriding:

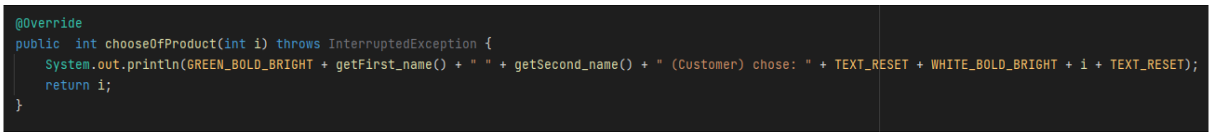
1.An example of using the **Overriding** method is **“chooseOfProduct”**, this method is written in the **“InteractionCustomersManager”** interface that the **Manage**r and **Customers** classes implement.

InteractionCustomersManager:



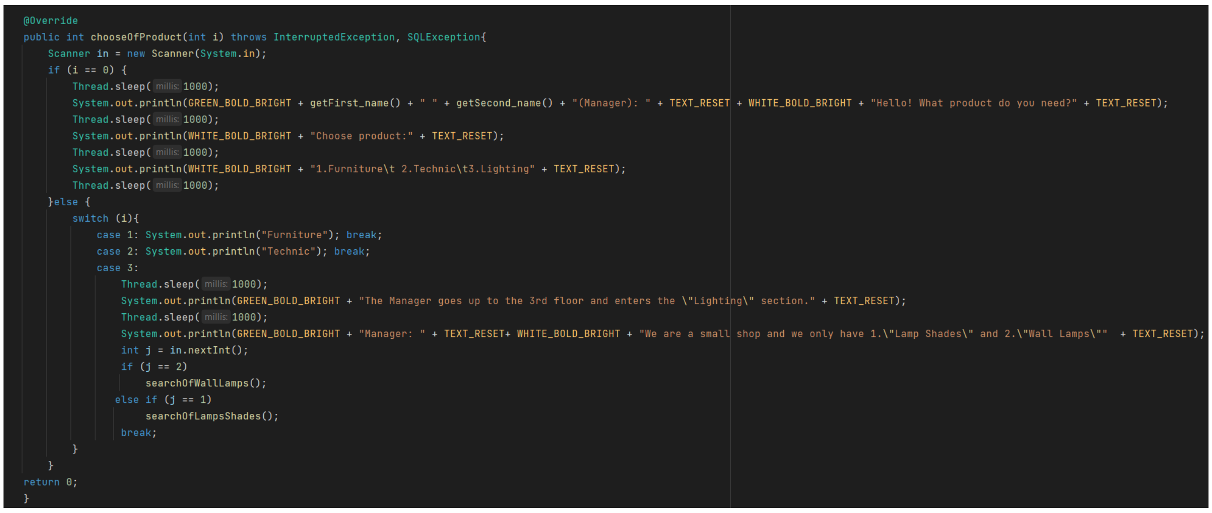
```java
public interface InteractionCustomersManager {
        int chooseOfProduct(int i) throws InterruptedException, SQLException;
}
```

Customers:



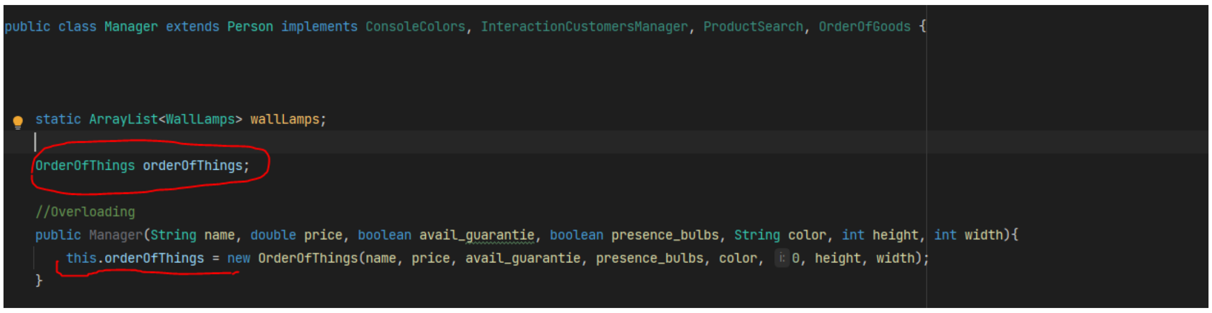
```java
@Override
public  int chooseOfProduct(int i) throws InterruptedException {
    System.out.println(GREEN_BOLD_BRIGHT + getFirst_name() + " " + getSecond_name() + " (Customer) chose: " + TEXT_RESET + WHITE_BOLD_BRIGHT + i + TEXT_RESET);
    return i;
}
```

Manager:



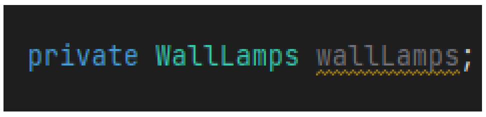
```java
@Override
public int chooseOfProduct(int i) throws InterruptedException, SQLException{
    Scanner in = new Scanner(System.in);
    if (i == 0) {
        Thread.sleep( millis: 1000);
        System.out.println(GREEN_BOLD_BRIGHT + getFirst_name() + " " + getSecond_name() + "(Manager): " + TEXT_RESET + WHITE_BOLD_BRIGHT + "Hello! What product do you need?" + TEXT_RESET);
        Thread.sleep( millis: 1000);
        System.out.println(WHITE_BOLD_BRIGHT + "Choose product:" + TEXT_RESET);
        Thread.sleep( millis: 1000);
        System.out.println(WHITE_BOLD_BRIGHT + "1.Furniture\t 2.Technic\t3.Lighting" + TEXT_RESET);
        Thread.sleep( millis: 1000);
    }else {
        switch (i){
            case 1: System.out.println("Furniture"); break;
            case 2: System.out.println("Technic"); break;
            case 3:
                Thread.sleep( millis: 1000);
                System.out.println(GREEN_BOLD_BRIGHT + "The Manager goes up to the 3rd floor and enters the \"Lighting\" section." + TEXT_RESET);
                Thread.sleep( millis: 1000);
                System.out.println(GREEN_BOLD_BRIGHT + "Manager: " + TEXT_RESET+ WHITE_BOLD_BRIGHT + "We are a small shop and we only have 1.\"Lamp Shades\" and 2.\"Wall Lamps\""  + TEXT_RESET);
                int j = in.nextInt();
                if (j == 2)
                    searchOfWallLamps();
               else if (j == 1)
                    searchOfLampsShades();
                break;
        }
    }
return 0;
}
```

7. Composition:

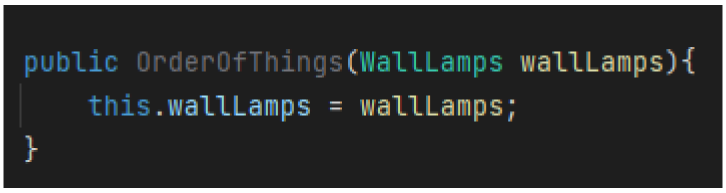


```java
public class Manager extends Person implements ConsoleColors, InteractionCustomersManager, ProductSearch, OrderOfGoods {


    static ArrayList<WallLamps> wallLamps;

    OrderOfThings orderOfThings;

    //Overloading
    public Manager(String name, double price, boolean avail_guarantie, boolean presence_bulbs, String color, int height, int width){
        this.orderOfThings = new OrderOfThings(name, price, avail_guarantie, presence_bulbs, color, 0, height, width);
    }
```

8. Association:

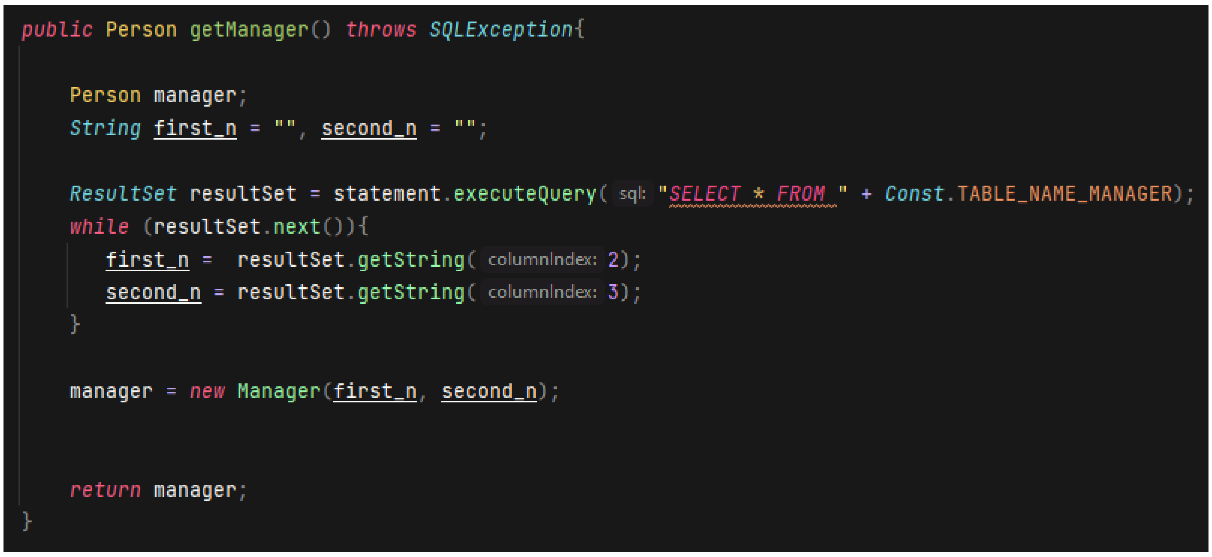


```java
private WallLamps wallLamps;
```

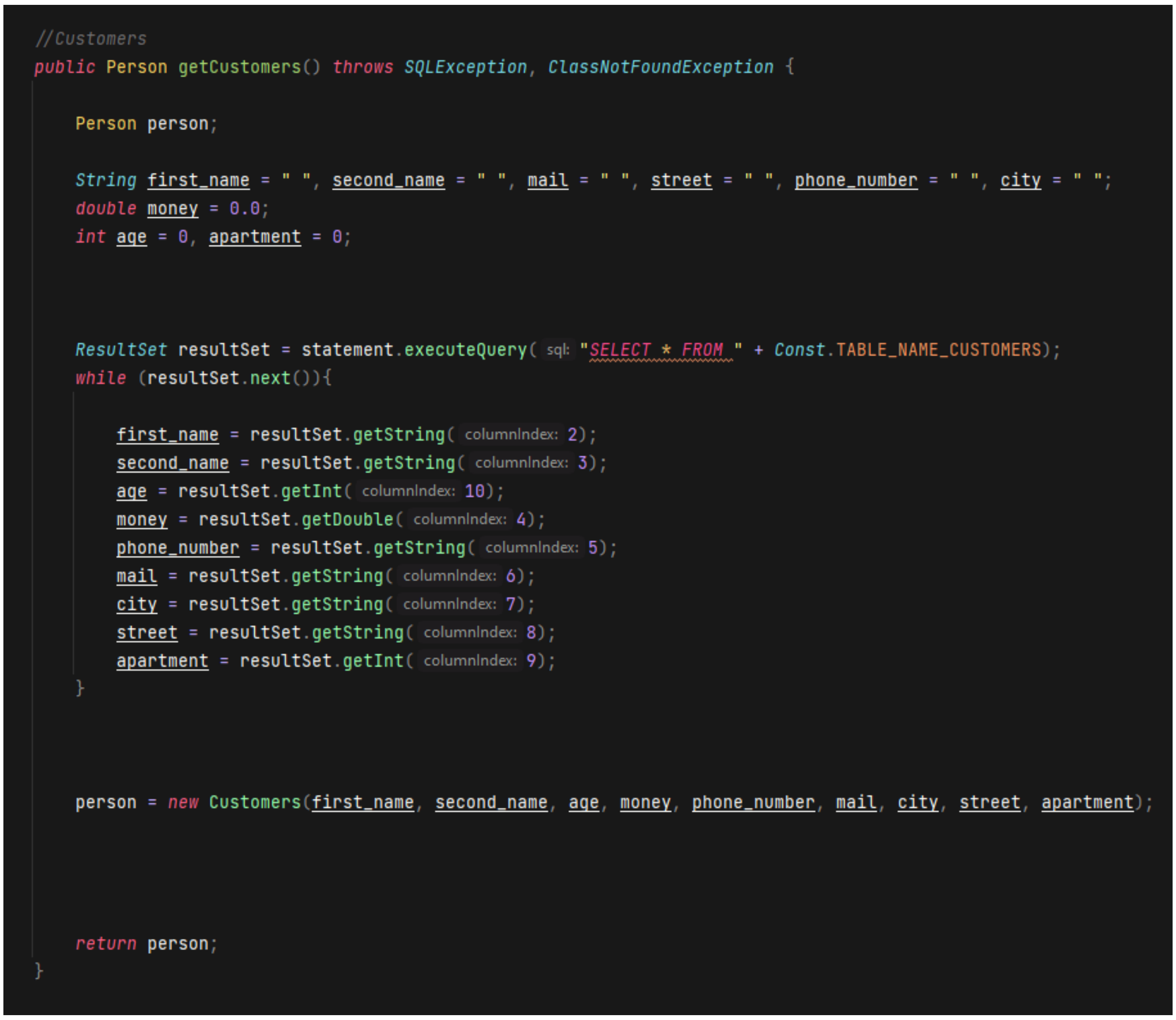


```java
public OrderOfThings(WallLamps wallLamps){
    this.wallLamps = wallLamps;
}
```

9. Polymorphism

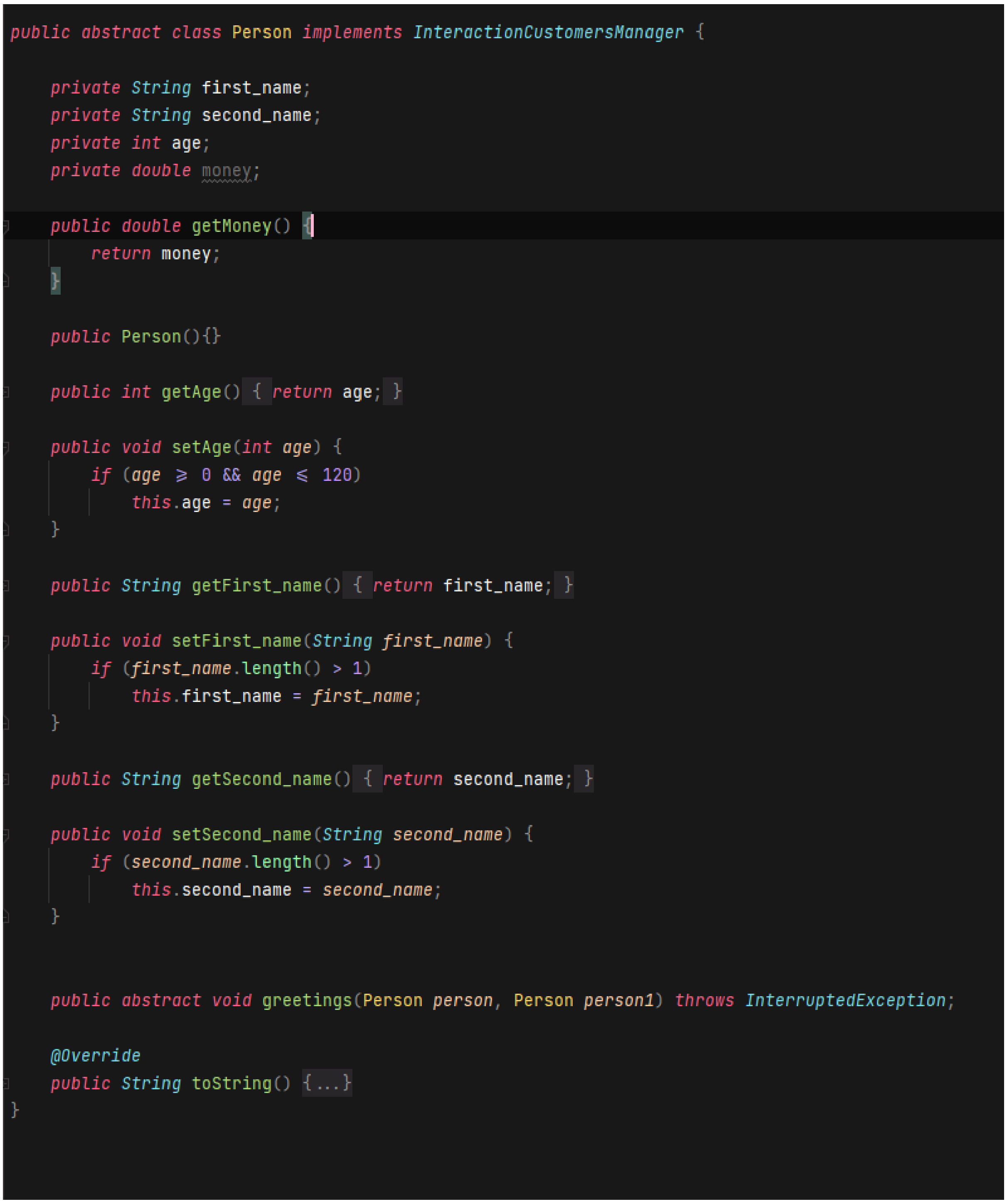


```
public Person getManager() throws SQLException{

    Person manager;
    String first_n = "", second_n = "";

    ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM " + Const.TABLE_NAME_MANAGER);
    while (resultSet.next()){
       first_n =  resultSet.getString( columnIndex: 2);
       second_n = resultSet.getString( columnIndex: 3);
    }

    manager = new Manager(first_n, second_n);


    return manager;
}
```



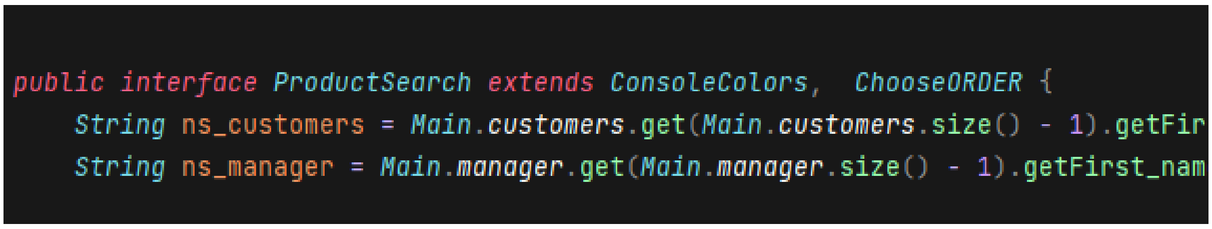
```
//Customers
public Person getCustomers() throws SQLException, ClassNotFoundException {

    Person person;

    String first_name = " ", second_name = " ", mail = " ", street = " ", phone_number = " ", city = " ";
    double money = 0.0;
    int age = 0, apartment = 0;



    ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM " + Const.TABLE_NAME_CUSTOMERS);
    while (resultSet.next()){

        first_name = resultSet.getString( columnIndex: 2);
        second_name = resultSet.getString( columnIndex: 3);
        age = resultSet.getInt( columnIndex: 10);
        money = resultSet.getDouble( columnIndex: 4);
        phone_number = resultSet.getString( columnIndex: 5);
        mail = resultSet.getString( columnIndex: 6);
        city = resultSet.getString( columnIndex: 7);
        street = resultSet.getString( columnIndex: 8);
        apartment = resultSet.getInt( columnIndex: 9);
    }



    person = new Customers(first_name, second_name, age, money, phone_number, mail, city, street, apartment);




    return person;
}
```

10. Abstract class



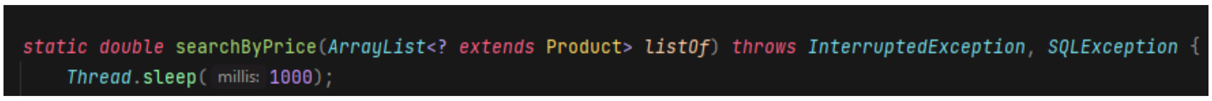
```java
public abstract class Person implements InteractionCustomersManager {

    private String first_name;
    private String second_name;
    private int age;
    private double money;

    public double getMoney() {
        return money;
    }

    public Person(){}

    public int getAge() { return age; }

    public void setAge(int age) {
        if (age ≥ 0 && age ≤ 120)
            this.age = age;
    }

    public String getFirst_name() { return first_name; }

    public void setFirst_name(String first_name) {
        if (first_name.length() > 1)
            this.first_name = first_name;
    }

    public String getSecond_name() { return second_name; }

    public void setSecond_name(String second_name) {
        if (second_name.length() > 1)
            this.second_name = second_name;
    }


    public abstract void greetings(Person person, Person person1) throws InterruptedException;

    @Override
    public String toString() {...}
}
```

11. Interface

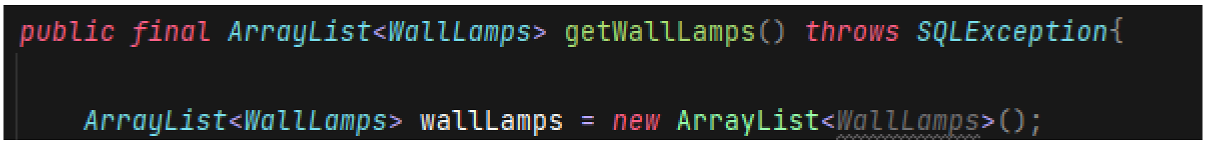


```
public interface ProductSearch extends ConsoleColors,  ChooseORDER {
    String ns_customers = Main.customers.get(Main.customers.size() - 1).getFir
    String ns_manager = Main.manager.get(Main.manager.size() - 1).getFirst_nam
```

12. Static and final method

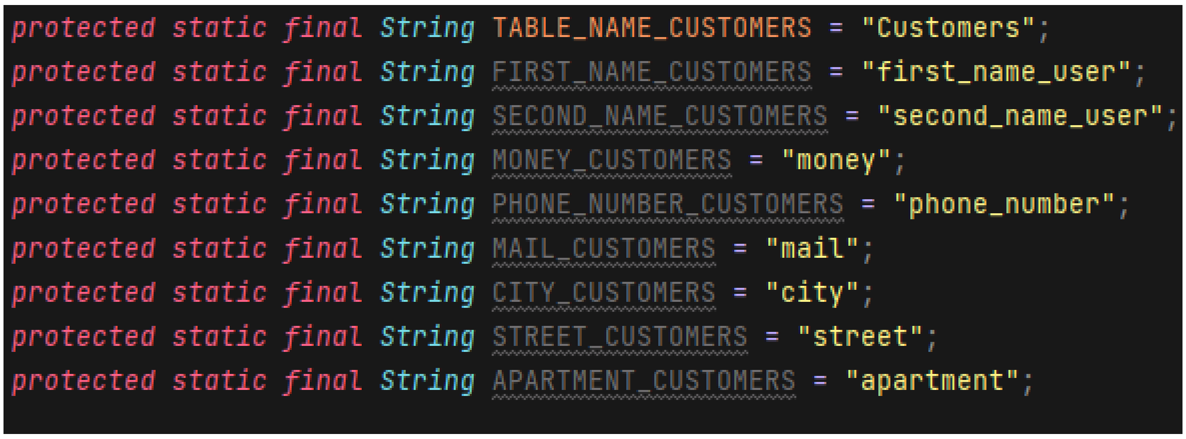


```
static double searchByPrice(ArrayList<? extends Product> listOf) throws InterruptedException, SQLException {
    Thread.sleep( millis: 1000);
```



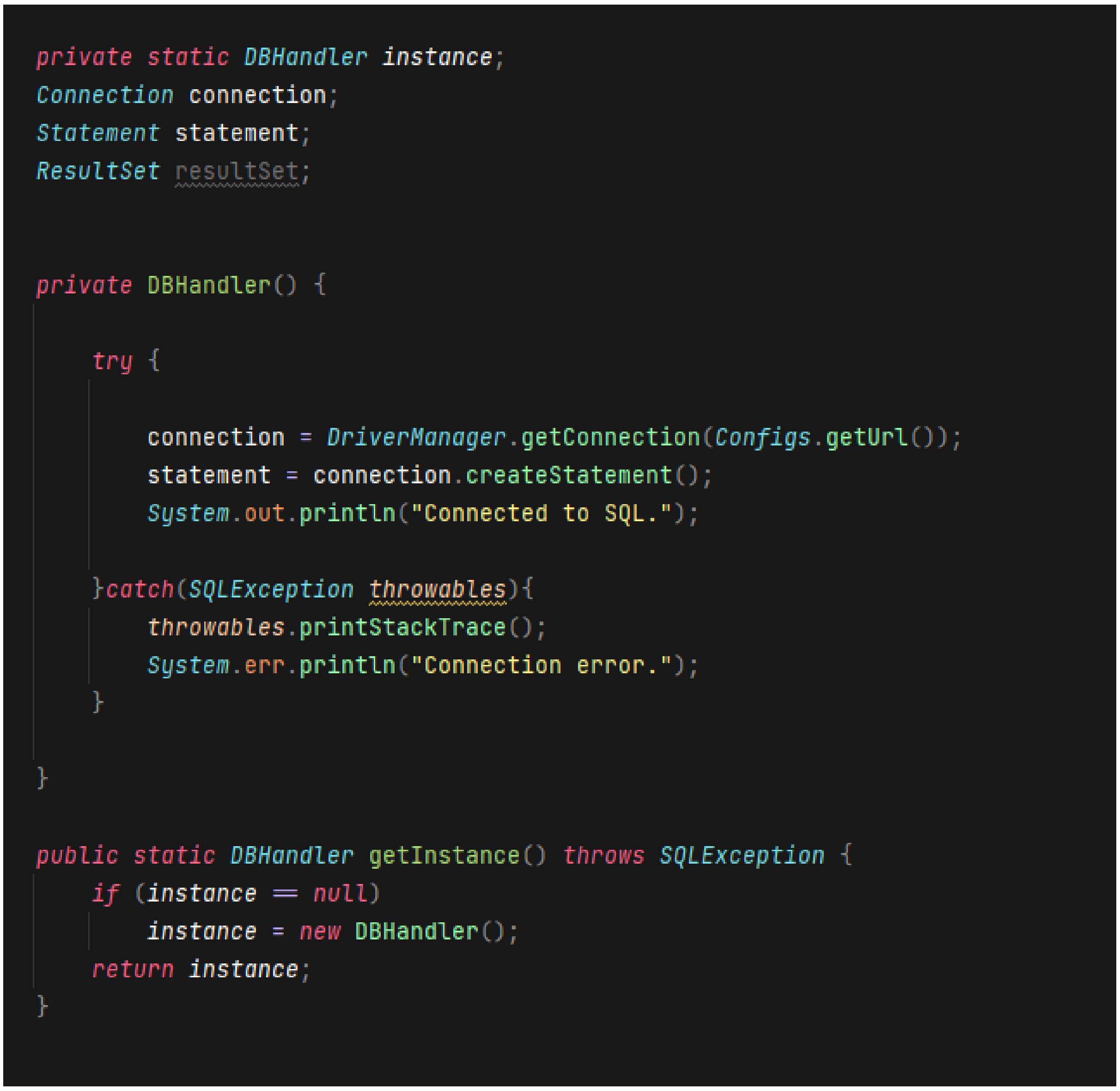
```
public final ArrayList<WallLamps> getWallLamps() throws SQLException{

    ArrayList<WallLamps> wallLamps = new ArrayList<WallLamps>();
```

13. Static and final attribution



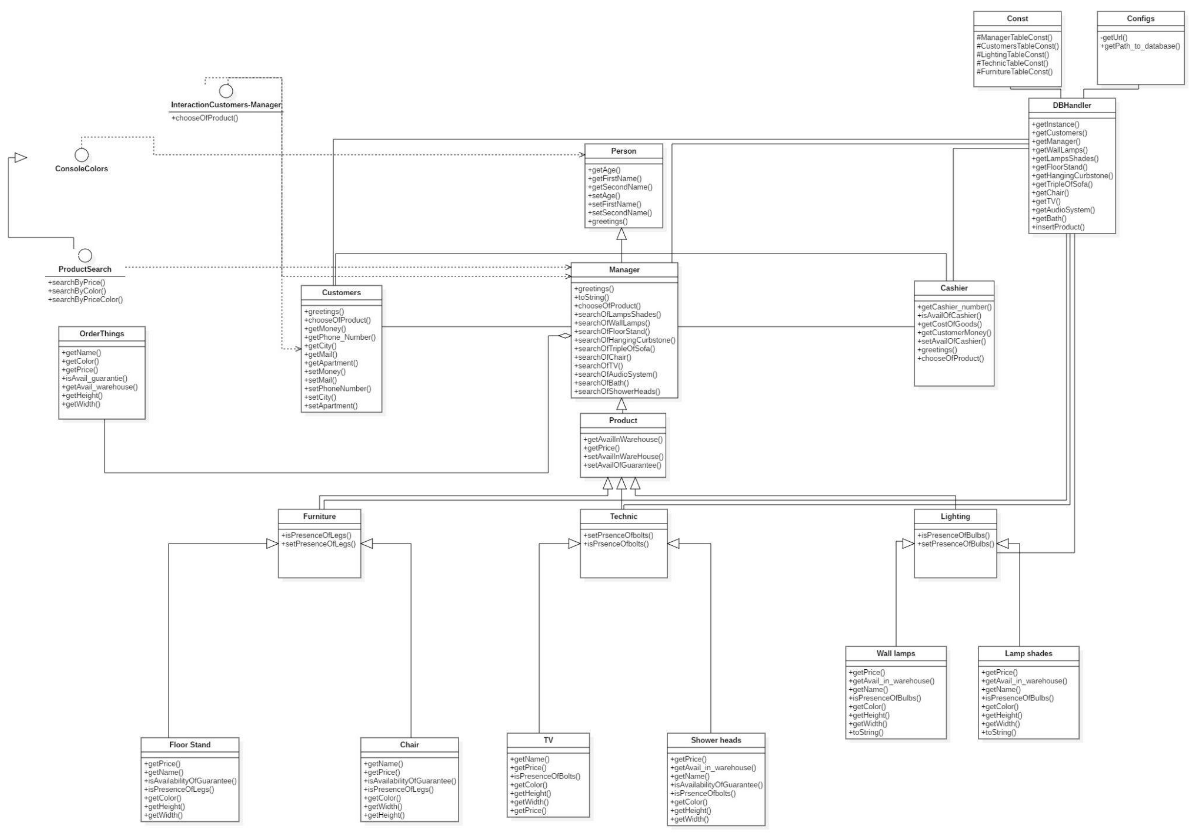
```
protected static final String TABLE_NAME_CUSTOMERS = "Customers";
protected static final String FIRST_NAME_CUSTOMERS = "first_name_user";
protected static final String SECOND_NAME_CUSTOMERS = "second_name_user";
protected static final String MONEY_CUSTOMERS = "money";
protected static final String PHONE_NUMBER_CUSTOMERS = "phone_number";
protected static final String MAIL_CUSTOMERS = "mail";
protected static final String CITY_CUSTOMERS = "city";
protected static final String STREET_CUSTOMERS = "street";
protected static final String APARTMENT_CUSTOMERS = "apartment";
```

14. Singleton



```java
private static DBHandler instance;
Connection connection;
Statement statement;
ResultSet resultSet;


private DBHandler() {

    try {

        connection = DriverManager.getConnection(Configs.getUrl());
        statement = connection.createStatement();
        System.out.println("Connected to SQL.");

    }catch(SQLException throwables){
        throwables.printStackTrace();
        System.err.println("Connection error.");
    }

}

public static DBHandler getInstance() throws SQLException {
    if (instance == null)
        instance = new DBHandler();
    return instance;
}
```

15. UML.


Const
#ManagerTableConst()
#CustomersTableConst()
#LightingTableConst()
#TechnicTableConst()
#FurnitureTableConst()
Configs
-getUrl()
+getPath_to_database()
DBHandler
+getInstance()
+getCustomers()
+getManager()
+getWallLamps()
+getLampsShades()
+getFloorStand()
+getHangingCurbstone()
+getTripleOfSofa()
+getChair()
+getTV()
+getAudioSystem()
+getBath()
+insertProduct()
InteractionCustomers-Manager
+chooseOfProduct()
ConsoleColors
ProductSearch
+searchByPrice()
+searchByColor()
+searchByPriceColor()
OrderThings
+getName()
+getColor()
+getPrice()
+isAvail_guarantie()
+getAvail_warehouse()
+getHeight()
+getWidth()
Person
+getAge()
+getFirstName()
+getSecondName()
+setAge()
+setFirstName()
+setSecondName()
+greetings()
Customers
+greetings()
+chooseOfProduct()
+getMoney()
+getPhone_Number()
+getCity()
+getMail()
+getApartment()
+setMoney()
+setMail()
+setPhoneNumber()
+setCity()
+setApartment()
Manager
+greetings()
+toString()
+chooseOfProduct()
+searchOfLampsShades()
+searchOfWallLamps()
+searchOfFloorStand()
+searchOfHangingCurbstone()
+searchOfTripleOfSofa()
+searchOfChair()
+searchOfTV()
+searchOfAudioSystem()
+searchOfBath()
+searchOfShowerHeads()
Cashier
+getCashier_number()
+isAvailOfCashier()
+getCostOfGoods()
+getCustomerMoney()
+setAvailOfCashier()
+greetings()
+chooseOfProduct()
Product
+getAvailInWarehouse()
+getPrice()
+setAvailInWareHouse()
+setAvailOfGuarantee()
Furniture
+isPresenceOfLegs()
+setPresenceOfLegs()
Technic
+setPrsenceOfbolts()
+isPrsenceOfbolts()
Lighting
+isPresenceOfBulbs()
+setPresenceOfBulbs()
Wall lamps
+getPrice()
+getAvail_in_warehouse()
+getName()
+isPresenceOfBulbs()
+getColor()
+getHeight()
+getWidth()
+toString()
Lamp shades
+getPrice()
+getAvail_in_warehouse()
+getName()
+isPresenceOfBulbs()
+getColor()
+getHeight()
+getWidth()
+toString()
Floor Stand
+getPrice()
+getName()
+isAvailabilityOfGuarantee()
+isPresenceOfLegs()
+getColor()
+getHeight()
+getWidth()
Chair
+getName()
+getPrice()
+isAvailabilityOfGuarantee()
+isPresenceOfLegs()
+getColor()
+getWidth()
+getHeight()
TV
+getName()
+getPrice()
+isPresenceOfBolts()
+getColor()
+getHeight()
+getWidth()
+getPrice()
Shower heads
+getPrice()
+getAvail_in_warehouse()
+getName()
+isAvailabilityOfGuarantee()
+isPrsenceOfbolts()
+getColor()
+getHeight()
+getWidth()