Furkan Baytak

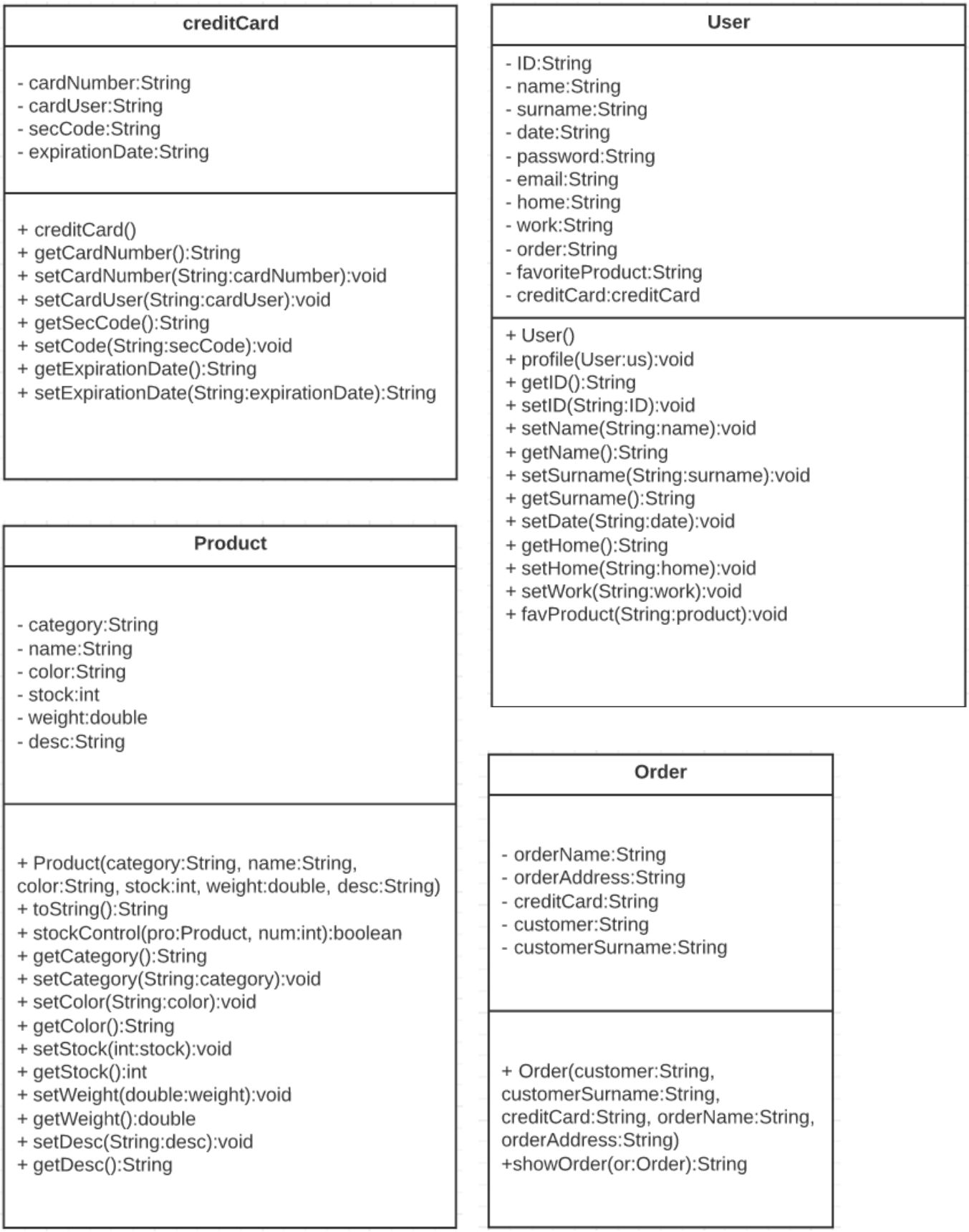
210316033

Computer Eng. Night Class

E-Commerce Application Project

Contents:

UML Diagrams:



Classes:

In this application, there is five Classes; creditCart, User, Product, Order, and Test.

Constructors:

```
public class Order {
    2 usages
    private String orderName;
    2 usages
    private String orderAddress;
    2 usages
    private String creditCard;
    2 usages
    private String customer;
    2 usages
    private String customerSurname;

    2 usages
    public Order(String customer,String customerSurname,
            String orderName,  String creditCard, String orderAddress
            ){
    this.orderName = orderName;
    this.customer = customer;
    this.creditCard = creditCard;
    this.orderAddress = orderAddress;
    this.customerSurname = customerSurname;
    }
```

In general, all usages of Constructors are the same. They take information from place where called “Data Field” (first 5 lines in this code.) and puts them in objects and then creates a new object with these informations.

For example; “this.orderName = orderName” code, takes orderName from Data Field and makes it equal with the orderName which defined in Constructor.

In data field all information is private because we don’t want to let anyone change them without using getter-setter methods because this can lead to complexity.

All Constructors have the same codes, implementations, and same reason to be.

Getter – Setter Methods:

```
public String getCardNumber(){
    return cardNumber;
}
1 usage
public void setCardNumber(String cardNumber){
    this.cardNumber = cardNumber;
}
public String getCardUser(){
    return cardUser;
}
1 usage
public void setCardUser(String cardUser){
    this.cardUser = cardUser;
}
public String getSecCode(){
    return secCode;
}
1 usage
public void setCode(String secCode){
    this.secCode = secCode;
}
public String getExpirationDate(){
    return expirationDate;
}
1 usage
public void setExpirationDate(String expirationDate){
    this.expirationDate = expirationDate;
}
```

Getter and Setter methods (also known as Accessors and Mutators), are necessary for our creditCard class because when we're defining informations, we used private keyword. That makes us impossible to change these informations unless we don't have getter and setter methods.

Getter and setter methods lets us change these informations(setter methods) or get this informations in another class(getter methods).

creditCard Class:

```
public creditCard(){
Scanner sc = new Scanner(System.in);
System.out.println("Please write your credit card number");
setCardNumber(sc.nextLine());
System.out.println("Please write your card user");
setCardUser(sc.nextLine());
System.out.println("Please write your security code");
setCode(sc.nextLine());
System.out.println("Please write your expirationDate");
setExpirationDate(sc.nextLine());
}
```

creditCard class only has a Constructor and with this constructor we're able to create new Credit Card for our customer with asking them to write their credit card informations.

This Class let us create a Credit Card object with informations in it and then we can use this object in other classes to buy products or when we want to see our profile informations.

User Class:

```
public User(){
    Scanner sc = new Scanner(System.in);
    System.out.println("Please write your ID");
    setID(sc.nextLine());
    System.out.println("Please write your name and surname (name surname)(if you"
            + " have 2 names please write one)");
    setName(sc.nextLine());
    setSurname(sc.nextLine());
    System.out.println("Please write your birthday date. (mm.dd.yy)");
    setDate(sc.nextLine());
    System.out.println("Please write your password.(at least 3 characters)");
    boolean flag = false;
    while(!flag){
    String pass = sc.nextLine();
    if (pass.length()>=3){this.password = pass;flag = true;}else{
        System.out.println("at least 3 characters");}}
        System.out.println("Please write your email");
        setEmail(sc.nextLine());
        System.out.println("Please write your home address");
        setHome(sc.nextLine());
        System.out.println("Please write your work address (you can leave it blank "
            + "if you want)");
        setWork(sc.nextLine());
        System.out.printf("Profile created%n");
}
```

In User Class's constructor, we let the user enter their ID, Name and Surname, Birthday Date, Password, and Addresses. With this informations we created a profile for our User. In Test Class, we can see our profile informations later with using profile method.

User Class Methods:

```
public static void profile(User us){
Scanner sc = new Scanner(System.in);
StringBuilder password = new StringBuilder();
for(int i=0; i<us.password.length(); i++){
    password.append("*");
        }
System.out.printf("ID: %s %nName Surname: %s %s %nBirthday date: %s%nPassword: "
        + "%s%nE-mail: %s%nHome Address: %s%nWork Address: %s%n", us.ID, us.name,
        us.surname, us.date, password,us.email,us.home,us.work);
```

Profile Class allows us to print User's profile informations in test class.

Product Class:

Product class has a constructor, getter-setter methods, an overrided toString method and a stockControl method.

Product Class Methods:

```
    @Override
public String toString() {
        return String.format("Category: %s%nName: %s%nColor: %s%nWeight: "
                + "%.2f%nDescription: %s%nStock: %d%n",
                 this.getCategory(), this.getName(), this.getColor(),
                 this.getWeight(), this.getDesc(), this.getStock());
}
2 usages
public static boolean stockControl(Product pro, int num){
if (pro.stock >= num){
    pro.stock = pro.stock - num;
    return true;
}
else{
    System.out.println("Not enough products in stock");
    return false;
}
}
```

Overrided toString method lets us print Product Class's informations in string format and stockControl method checks for is there enough product to buy and if there is not it prints "Not enough products in stock" and if there is enough stock it decreases the amount of stock with the amount you bought.

Order Class:

Order class has a constructor and showOrder Method.

Order Class Methods:

```
public static String showOrder(Order or){
return String.format("Customer: %s %s%nProsecute: %s%nCredit Card: "
        + "%s%nAddress: %s%n", or.customer,or.customerSurname,or.orderName,
        or.creditCard,or.orderAddress);
}
```

showOrder Method, allows us to print our order's informations.

Test Class:

In Test Class, we can test our program.

```
public class Test {
    public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    boolean exit = false;
    User us = new User();
    creditCard CD = new creditCard();
    ArrayList<Product> favorites = new ArrayList<>();
    ArrayList<String> orders = new ArrayList<>();
    ArrayList<Product> electronics = new ArrayList<>();
    ArrayList<Product> fashions = new ArrayList<>();
    ArrayList<Product> cosmetics = new ArrayList<>();
    ArrayList<Product> stationary = new ArrayList<>();
    ArrayList<String> colors = new ArrayList<>();
    Random random = new Random();
    colors.add("Black");
    colors.add("White");
    colors.add("Red");
    colors.add("Blue");
    colors.add("Green");
    colors.add("Grey");
```

First, for creating products we made some ArrayLists to hold these products in it. For making our job easy we're creating products with random features so we can have more products.

```
int randomIndex = random.nextInt(colors.size());
electronics.add(new Product( category: "Electronic",  name: "Phone", colors.get(randomIndex),
        (int) (Math.random() * 101), Math.random(),  desc: "Android"));

randomIndex = random.nextInt(colors.size());

electronics.add(new Product( category: "Electronic",  name: "Tablet", colors.get(randomIndex),
        (int) (Math.random() * 101), Math.random(),  desc: "İos"));

randomIndex = random.nextInt(colors.size());
electronics.add(new Product( category: "Electronic",  name: "Laptop", colors.get(randomIndex),
        (int) (Math.random() * 101), Math.random(),  desc: "FreeDos"));

randomIndex = random.nextInt(colors.size());
fashions.add(new Product( category: "Fashion",  name: "Skirt", colors.get(randomIndex), (int)
        (Math.random() * 101), Math.random(),  desc: "Long slit skirt"));

randomIndex = random.nextInt(colors.size());
fashions.add(new Product( category: "Fashion",  name: "T-shirt", colors.get(randomIndex), (int
        ) (Math.random() * 101), Math.random(),  desc: "Anime printed "
                + "oversize t-shirt"));

randomIndex = random.nextInt(colors.size());
fashions.add(new Product( category: "Fashion",  name: "Jacket", colors.get(randomIndex), (int)
        (Math.random() * 101), Math.random(),  desc: "Vintage with pockets"));

randomIndex = random.nextInt(colors.size());
cosmetics.add(new Product( category: "Cosmetic",  name: "Lipstick", colors.get(randomIndex), (
        int) (Math.random() * 101), Math.random(),  desc: "Mat"));
```

With using these lines of codes, we're creating random objects for all our main products. Then we can start to test our Classes.

Using Switch-Case in this situation makes our job easier. We have a menu to pick operations from it.

Case-1: Show Profile

With using Profile method from User Class, we can print our User's profile information.

```
int button = sc.nextInt();
sc.nextLine();
switch(button){
    case 1:
        User.profile(us);
        break;
```

Case-2: Buy product

We can select which kind of product to buy first, then we have another switch-case int Case-2 switch-case. This one allows us to select what we want to this selected kind of product; we can buy or add favorite. If we want to buy it we can select how many product we want to buy and there is stockControl Method from Product Class to check is there enough stock to buy.

```
case 2:
    System.out.printf(" 1-Electronic%n 2-Fashion%n 3-Cosmetic%n"
            + " 4-Stationary%n 0-Return menu%n");
    int select = sc.nextInt();
    sc.nextLine();
    switch(select){
        case 1:
            showList(electronics);
            System.out.println("Select the product");
            int order = sc.nextInt();
            Product pro = electronics.get(order);
            System.out.println("press 1 to buy, press 2 to add to favorite");
            int n = sc.nextInt();
            if(n==1){
                sc.nextLine();
                System.out.println("How many");
                int num = sc.nextInt();
                sc.nextLine();
                boolean flag = Product.stockControl(pro, num);
                if (flag){
                Order or = new Order(us.getName(),us.getSurname(),pro.getName(),
                        CD.getCardNumber(),us.getHome());
                orders.add(Order.showOrder(or));
                }
                break;
            }
```

(This screenshot only shows for Case-1 in Case-2, you can check the other codes from test Class.)

Case-3: See Orders

If we bought any products before, we could see our orders if we press 3 in our menu.

```
case 3:
    for (String order : orders) {
        System.out.println(order);
    }
    break;
```

Case-4: Favorites

In Case-2, we could buy or add the product to our favorites. If we add any product to our favorite, we can see them with pressing 4 in our menu.

```
case 4:
    for (Product favorite : favorites) {
        System.out.println(favorite);
    }
    break;
```

Case-0: Exit

Exits the program.

```
case 0:
    exit = true;
    break;
```

Test Class Methods:

```
private static boolean ifForCase(Scanner sc, User us, creditCard CD, ArrayList<String> orders, Product pro, int n) {
    if(n==1){
        sc.nextLine();
        System.out.println("How many");
        int num = sc.nextInt();
        sc.nextLine();
        boolean flag = Product.stockControl(pro, num);
        if (flag){
        Order or = new Order(us.getName(),us.getSurname(),pro.getName(),
                CD.getCardNumber(),us.getHome());
        orders.add(Order.showOrder(or));
        }
        return true;
    }
    return false;
}
```

ifForCase method is used for large repeating lines of codes in switch-case in Case-2. With using this method, we get clearer lines of codes with not repeating lines.

```
private static void showList(ArrayList<Product> MyList) {
    for (int i = 0; i < MyList.size(); i++) {
        System.out.println(i + " -> " + MyList.get(i));
}
```

showList method is also used in switch-case in Case-2. We created our products randomly before and this method allows us to write these random products in array lists as a list of products. If we want to tell it more easily, we can say it shows us list of products.