

Міністерство освіти і науки України  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики  
Кафедра програмного забезпечення комп'ютерних систем

**КУРСОВА РОБОТА**

з дисципліни «Об'єктно-орієнтоване програмування»

на тему

**Шаблони проектування в ООП. Магазин побутової техніки.**

Виконала студентка

II курсу групи КП-72

Городченко Анна Володимирівна

залікова книжка КП-7208

Керівник роботи

доц., к.т.н. Заболотня Т.М.

Оцінка

---

(дата, підпис)

## ЗМІСТ

<b>ВСТУП.....</b>	<b>3</b>
<b>1. ОПИС СТРУКТУРНО-ЛОГІЧНОЇ ОРГАНІЗАЦІЇ ПРОГРАМИ.....</b>	<b>5</b>
1.1. Модульна організація програми.....	5
1.2. Функціональні характеристики.....	7
1.3. Опис реалізованих класів.....	8
<b>2. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ</b>	
<b>    ШАБЛОНІВ ПРОЕКТУВАННЯ.....</b>	<b>22</b>
2.1. Обґрунтування вибору та опис шаблонів проектування для реалізації програмного забезпечення автомату.....	22
2.2. Діаграма класів.....	29
2.3. Результати роботи програми.....	30
<b>ВИСНОВКИ.....</b>	<b>35</b>
<b>СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....</b>	<b>36</b>

## ВСТУП

Дана курсова робота присвячена розробці програмного забезпечення торгівельного автомату за допомогою використання шаблонів проектування. Задача моделювання програмного забезпечення для торгівельного автомату є досить актуальною, адже магазин із продажу техніки для дому – звична річ для великого міста, для бездоганного процесу купівлі товару відвідувачем магазину, необхідно розробити певні рішення, які представлять основні потоки даних об'єктів, з якими взаємодіють покупці під час купівлі побутової техніки у відповідному магазині. Дана тематика вибрана для курсової роботи тому, що результати абстрагування об'єктів у даній спроектованій системі дозволяють застосувати всі вивчені методи та принципи об'єктно-орієнтованого програмування для створення програмного забезпечення, зокрема мати змогу правильно організувати код за допомогою шаблонів проектування.

**Об'єктом** курсової роботи є *магазин побутової техніки*.

**Метою** роботи є *розроблення програмного забезпечення для магазину побутової техніки* з використанням шаблонів проектування.

Для досягнення описаної мети необхідно виконати такі **пункти завдань**:

- Абстрагувати об'єкти предметної галузі;
- Розробити структурну організацію програмного забезпечення за допомогою використання основних принципів об'єктно-орієнтованого програмування та шаблонів проектування;
- Визначити та описати функціональні характеристики програми;
- Обґрунтувати вибір шаблонів проектування;
- Розробити графічний інтерфейс користувача;
- Виконати реалізацію програмного забезпечення відповідно до технічного завдання;
- Виконати тестування розробленої програми;

- Оформити документацію з курсової роботи.

Розроблене програмне забезпечення складається з таких логічних частин: інтерфейсу користувача, створення та відображення меню основних кроків із купівлі товару покупцем, модуля, що відповідає за додавання товарів у кошик та оплати цих товарів.

Використані шаблони проектування: **State, Flyweight, Factory, Prototype, Bridge, Decorator, Proxy**( у вигляді Protection Proxy).

Розроблене програмне забезпечення може бути використане у окремій частині бізнес-сфери, яка передбачає роботу із організацією бізнес-процесів у магазині.

Пояснювальна записка складається зі вступу, двох розділів, загальних висновків та списку використаних джерел (*три* найменування). Робота містить \_\_ рисунків. Загальний обсяг роботи – \_\_ друкованих сторінок.

## 1. ОПИС СТРУКТУРНО-ЛОГІЧНОЇ СХЕМИ ПРОГРАМИ

### 1.1. Модульна організація програми

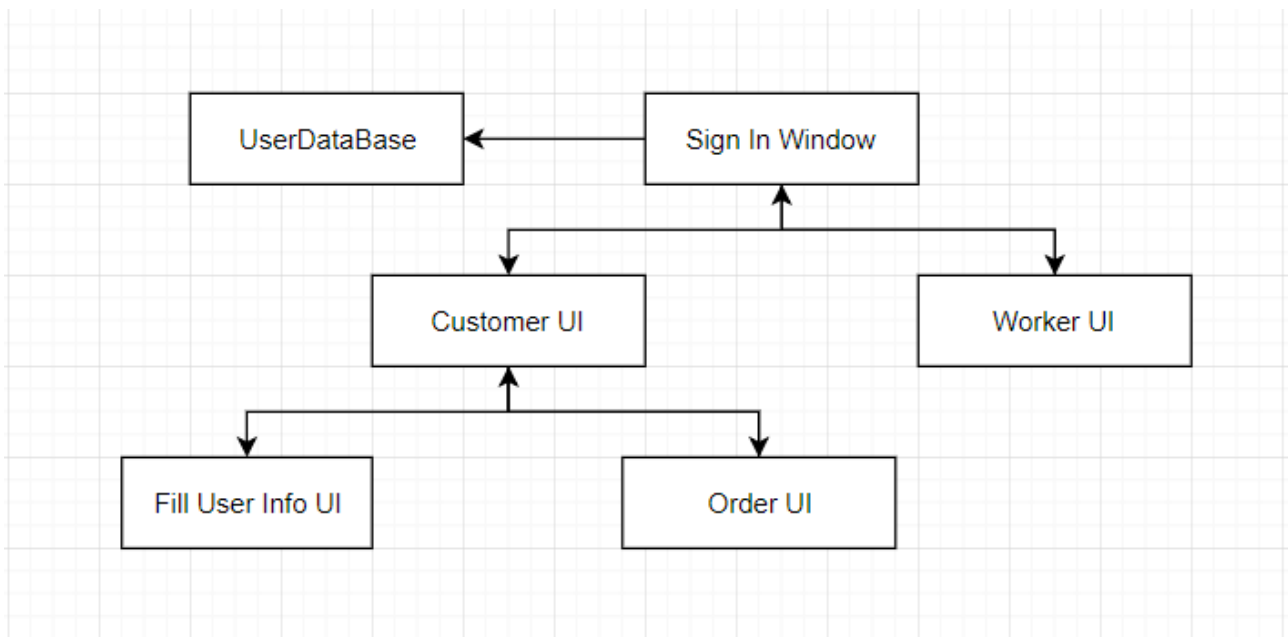


Рис. 1.1.1. Модульна організація програми

**SignInWindow** – модуль, який відповідає за вхід у систему за допомогою логіна та паролю, якщо користувач є у системі. Якщо користувача немає у систему, можна зайти у магазин безпосередньо без авторизації.

**CustomerUI** – модуль, який відповідає за функції роботи безпосередньо з покупцем та із самим кошиком товарів користувача.

**WorkerUI** – модуль, який призначений для працівників магазину, в якому вони встановлюють знижки на товари.

**OrderUI** – модуль, що відповідає за формування замовлення.

**FillUserInfoUI** – модуль, який відповідає за заповнення інформації про користувача для замовлення, якщо цей користувач попередньо не увійшов у систему.

**UserDataBase** — модуль, який являє собою сховище існуючих користувачів.

### 1.2. Функціональні характеристики

Робота з програмою починається з форми SignInWindow. Користувачу пропонується увійти у систему, ввівши пароль і логін, або пропустити етап входу у систему. В залежності від того, який логін введе користувач, обирається або інтерфейс покупця, або інтерфейс працівника. Якщо користувач входить без логіну, то автоматично відкривається інтерфейс покупця.

В інтерейсі покупця можна переглядати товари, що є в магазині, додавати їх в кошик, видаляти. Також можна перелогінитись( зайти в систему з іншого аккаунта). Також відображається поточний баланс користувача(якщо він увійшов в систему). У цьому меню можна перейти безпосередньо до оплати товарів, але якщо користувач не надав інформацію про себе( не увійшов у систему), користувачу пропонується ввести дані про свою особу( повне ім'я, вік та стать), у такому випадку баланс для всіх користувачів, які оплачують покупки таким способом, надається за замовчуванням 50 000 грошових одиниць.

В інтерейсі OrderUI , який відповідає за формування замовлення, є відображення кошику з товарами з їх кількістю. Є можливість повернутись до редагування кошику(додавання чи видалення товарів). Покупець бачить загальну ціну за товари та загальну ціну за все замовлення. Користувач може обрати тип оплати товару(картою чи готівкою) та вибрати доставку та тип доставки: літаком, потягом чи таксі. Можна обрати декілька типів доставки.

Після того, як покупець закінчить із формуванням замовлення, він нажимає на кнопку «Замовити», після якої висвічується стан замовлення(успішне та неуспішне в разі нестачі грошей).

Після успішного замовлення користувач повертається в меню магазину, де можна переглядати та додавати товари. (CustomerUI).

В інтерфейсі працівника( WorkerUI) працівник магазину можна встановити знижку на товари. Якщо увійшов працівник з посадою «Менеджер», то він може встановлювати знижку на будь який товар більшу, ніж половина вартості товару, але меншу, ніж повна вартість товару. Працівники з іншими посадами можуть встановити тільки таку знижку, яка менша, ніж половина вартості товару.

### 1.3. Опис реалізованих класів

1. **User** – абстрактний клас користувача, від якого наслідуються класи **Customer**, **Worker**. Має такі поля: тип користувача, повне ім'я, вік та стать.

#### User.java

```
package Users;

public abstract class User {
    protected UserType type;
    private String fullname;
    private String sex;
    private int age;

    public User(String fullname, String sex) {
        this.fullname = fullname;
        this.sex = sex;
    }
    public User(String fullname, String sex, int age) {
        this.fullname = fullname;
        this.sex = sex;
        this.age = age;
    }

    public String getFullname() {
        return fullname;
    }

    public String getSex() {
        return sex;
    }

    public int getAge() {
        return age;
    }
    public UserType getType() {
        return type;
    }
}
```

## 2. UserType -тип користувача.

### UserType.java

```
package Users;

public enum UserType {
    WORKER,
    CUSTOMER,
    USER;
}
```

## 3. Customer – клас для оголошення сутності покупця, роботи покупця із кошиком та роботи покупця по оплаті продуктів із кошика.

### Customer.java

```
package Users;

import Delivery.DeliveryManager;
import Delivery.DeliveryManager.ShippingType;
import Payment.*;
import Products.Product;

import java.util.HashMap;

public class Customer extends User {

    private CustomerPayment paymentType;

    private float money;
    private int allProductsCost;
    private int shippingCost;

    public HashMap<Product, Integer> basket = new HashMap<>();

    public Customer(String fullname, String sex, float money) {
        super(fullname, sex);
        this.paymentType = new CustomerPayment(new CashPayment());
        this.money = money;
        this.type = UserType.CUSTOMER;
    }

    public Customer(int age, String fullname, String sex) {
        super(fullname, sex, age);
        this.money = 50000;
        this.type = UserType.CUSTOMER;
        this.paymentType = new CustomerPayment(new CashPayment());
    }

    public float getMoney() {
        return this.money;
    }

    public int getAllProductsCost() {
        if (allProductsCost == 0) {
```



```

        basket.keySet().forEach((Product key) -> {
            this.allProductsCost += key.getCurrentPrice() *
basket.get(key);
        });
    }
    return this.allProductsCost;
}

public void setBasket(HashMap<Product, Integer> b) {
    this.basket = b;
    this.allProductsCost = 0;
}

public void setCustomerPayment(CustomerPayment p) {
    this.paymentType = p;
}

public int getTotalCost() {
    return this.allProductsCost + this.shippingCost;
}

public boolean payForProducts() {
    float toPay = paymentType.pay(getTotalCost());
    if (money >= toPay) {
        money -= toPay;
        System.out.println("Your balance decreased by " + (int) toPay);
        return true;
    } else {
        return false;
    }
}

public void addShippingCost(int cost) {
    this.shippingCost += cost;
}

public void removeShippingCost() {
    this.shippingCost = 0;
}

public boolean hasMoneyToPay(ShippingType type) {
    return money >= paymentType.pay(getAllProductsCost() +
DeliveryManager.getShippingCost(type));
}
}

```

**4. Worker-** клас працівника магазину. Відповідає за встановлення знижки на товари.

#### Worker.java

```
package Users;

import Products.Product;

public class Worker extends User {

    private String position;

    public Worker(String fullname, String sex, String position) {
        super(fullname, sex);
        this.position = position;
        this.type = UserType.WORKER;
    }

    public String getPosition() {
        return position;
    }

    public void setDiscount(Product p, int value) {
        p.setDiscount(value);
    }

}
```

**5. ICustomerPayForProduct** – інтерфейс , призначений для реалізації шаблону «Міст».

#### ICustomerPayForProduct.java

```
package Payment;

interface ICustomerPayForProduct {
    float payForProducts(int toPay);
}
```

**6. CustomerPayment** – клас для оголошення сутності, яка відображатиме спосіб оплати користувачем товару – карткою, чи готівкою. Цей клас є імплементацією паттерну «Міст».

### CustomerPayment.cs

```
package Payment;
// Implementor for a "bridge" design pattern.

public class CustomerPayment {

    ICustomerPayForProduct paymentType;

    public CustomerPayment(ICustomerPayForProduct paymentType) {
        this.paymentType = paymentType;
    }

    public void setPaymentType(ICustomerPayForProduct paymentType) {
        this.paymentType = paymentType;
    }

    public float pay(int toPay) {
        return this.paymentType.payForProducts(toPay);
    }

}
```

**6. CashPayment, CardPayment** – конкретні «виконавці» шаблону «Міст».

### CardPayment.java, CaPayment.java

```
package Payment;

// First concrete abstraction - paying with credit card.
public class CardPayment implements ICustomerPayForProduct {

    @Override
    // Get a 10% cashback as a card credit card owner.
    public float payForProducts(int toPay) {
        return (float) (toPay * 0.9);
    }

}
```

```
package Payment;

// Second concrete abstraction - paying with cash.
public class CashPayment implements ICustomerPayForProduct{

    @Override
    // Regular payment without cashback.
    public float payForProducts(int toPay) {
        return toPay;
    }

}
```

## 7. IDiscount – інтерфейс, для реалізації шаблону «Proxy».

### IDiscount.java

```
package Discount;

import Products.Product;
import Users.Worker;

interface IDiscount {
    void setDiscount(Product p, Worker w, int value);
}
```

8. **SimpleDiscountSystem** – клас, для реалізації шаблону «Proxy». Слугує для встановлення знижки на товар працівником магазину. Будь-яку знижку (але меншу, ніж вартість товару) може встановлювати будь який працівник.

### SimpleDiscountSystem.java

```
package Discount;

import Products.Product;
import Users.Worker;

public class SimpleDiscountSystem implements IDiscount {

    @Override
    public void setDiscount(Product product, Worker worker, int value) {
        if (value >= 0 && value <= product.getPrice()) {
            worker.setDiscount(product, value);
        }
    }
}
```

9. **ProtectedDiscountSystem** – клас, для реалізації шаблону «Proxy».

Відповідає за встановлення знижки в магазині. Перевіряє працівника магазину, щоб знижка не була більша ніж 50% від вартості товару. Тільки «Менеджер», може встановлювати знижку > 50%.

### ProtectedDiscountSystem.java

```
package Discount;

import Products.Product;
import Users.Worker;
import java.awt.Frame;
import javax.swing.JOptionPane;

public class ProtectedDiscountSystem implements IDiscount {

    private SimpleDiscountSystem basic = new SimpleDiscountSystem();

    @Override
    public void setDiscount(Product p, Worker w, int value) {
        // ONLY manager can set discount more than a half of price of product
    }
}
```

```

        if (value >= p.getPrice() / 2) {
            if (!"manager".equals(w.getPosition().toLowerCase())) {

                JOptionPane.showMessageDialog(
                    new Frame("Exceeding the value of the discount"),
                    "Sorry, you cant set discount value more than a half
of product price.Only manager can do it.",
                    "Exceeding the value of the discount",
                    JOptionPane.WARNING_MESSAGE);

            } else {
                basic.setDiscount(p, w, value);
            }
        } else {
            basic.setDiscount(p, w, value);
        }
    }
}

```

**10.UsersDataBase** – статичний клас для зберігання користувачів системи.

### ProtectedDiscountSystem.java

```

package DB;
import Users.*;
import java.util.HashMap;

public class UsersDataBase {
    static HashMap<String,User> users = new HashMap<>();
    static{
        users.put("goroanya", new Customer("Horodchenko Anna","female",
100000));
        users.put("traumgedanken", new Customer("Bulaievskiy Ihor","male",
20000));

        users.put("ariana", new Worker("Grande Ariana","female", "cashier"
));
        users.put("taylor", new Worker("Swift Taylor","female", "seller-
consultant" ));
        users.put("weekend", new Worker("Abel Tesfaye;", "male", "manager"
));
    }
    public User getUser(String login){
        return users.get(login);
    }
}

```

### 11. State -інтерфейс для реалізації шаблону «Стан».

#### State.java

```

package State;

import Forms.SignInWindow;

public interface State {
    void execute(SignInWindow MV);
}

```

**11.CustomerState** – конкретна реалізація в шаблоні «**State**». Призначений для відкриття форми CustomerUI.form для переходу в режим покупця.

#### CustomerState.java

```
package State;

import Forms.SignInWindow;

public class CustomerState implements State {

    @Override
    public void execute(SignInWindow SW) {
        SW.setVisible(false);
        SW.openCustomerUI();
    }
}
```

**12.WorkerState** - конкретна реалізація в шаблоні «**State**». Призначений для відкриття форми WorkerUI.form для переходу в режим працівника.

#### WorkerState.java

```
package State;

import Forms.SignInWindow;

public class WorkerState implements State {

    @Override
    public void execute(SignInWindow SW) {
        SW.setVisible(false);
        SW.openWorkerUI();
    }
}
```

**13. StateFactory** – клас, реалізація шаблону «**Flyweight**», створює об'єкти-пристосуванці та управляє ними. Коли клієнт звертається до пристосування, об'єкт **StateFactory** надає існуючий екземпляр або створює новий, якщо готового ще немає. Пристосуваннями є екземпляри шаблону «**State**» : **CustomerState** і **WorkerState**.

#### StateFactory.java

```
package State;

import Users.UserType;
import java.util.HashMap;

public class StateFactory {
    static HashMap<UserType, State> states = new HashMap<>();
    static {
        states.put(UserType.CUSTOMER, null);
        states.put(UserType.WORKER, null);
    }
}
```

```

    }
    public State getState(UserType type){
        if(states.get(type) == null ){
            if(type == UserType.CUSTOMER){
                states.put(type, new CustomerState());
            }
            else{
                states.put(type, new WorkerState());
            }
        }
        return states.get(type);
    }
}

```

**13.Product** – клас, який є об'єктом товару в магазині. Наслідується класами **TVSet, Fridge, WashingMachine**. Та реалізує інтерфейс **IPdoduct**.

### Product.java

```

package Products;

public abstract class Product implements IProduct {

    protected int price;
    protected int discount;
    protected String manufacturer;
    protected String name;
    protected Color color;

    public int getPrice() {
        return this.price;
    }

    @Override
    public int getCurrentPrice() {
        return this.price - this.discount;
    }

    public void setDiscount(int discount) {
        this.discount = discount;
    }

    public int getDiscount() {
        return this.discount;
    }

    public Product(Color color, String name, int price, String manufacturer)
    {
        this.color = color;
        this.name = name;
        this.price = price;
        this.manufacturer = manufacturer;
    }

    public String getName() {
        return this.name;
    }

    public boolean equals(Object object) {
        if (this == object) {
            return true;
        }
    }
}

```

```

        if (object == null || getClass() != object.getClass()) {
            return false;
        }
        Product product = (Product) object;
        return price == product.price
            && manufacturer.equals(product.manufacturer)
            && name.equals(product.name)
            && color.equals(product.color);
    }

    public int hashCode() {
        return java.util.Objects.hash(this.getClass(), price, manufacturer,
name, color);
    }
}

```

## IProduct.java

```

package Products;

public interface IProduct {
    int getCurrentPrice();
}

```

## TVSet.java, Fridge.java, WashingMachine.java

```

package Products;

public class TVSet extends Product implements Copyable {

    private int diagonal;

    public TVSet(Color color, String name, int price, String manufacturer,
int diagonal) {
        super(color, name, price, manufacturer);
        this.diagonal = diagonal;
        this.discount = 0;
    }

    public TVSet(Color color, String name, int price, String manufacturer,
int diagonal, int discount) {
        super(color, name, price, manufacturer);
        this.diagonal = diagonal;
        this.discount = discount;
    }

    @Override
    public int getCurrentPrice() {
        return diagonal >= 100 ? (this.price - discount + 500) : (this.price
- discount);
    }

    @Override
    public Product copy() {
        return new TVSet(this.color, this.name, this.price,
this.manufacturer, this.diagonal, this.discount);
    }

    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;
    }
}

```



```

        if (!super.equals(object)) return false;
        TVSet tvSet = (TVSet) object;
        return diagonal == tvSet.diagonal &&
            discount == tvSet.discount;
    }

    public int hashCode() {
        return java.util.Objects.hash(super.hashCode(), diagonal, discount);
    }
}

```

```

package Products;

public class Fridge extends Product implements IProduct, Copyable {

    private boolean hasFreezer;

    public Fridge(Color color, String name, int price, String manufacturer,
boolean hasFreezer) {
        super(color, name, price, manufacturer);
        this.hasFreezer = hasFreezer;
        this.discount = 0;
    }

    private Fridge(Color color, String name, int price, String manufacturer,
boolean hasFreezer, int discount) {
        super(color, name, price, manufacturer);
        this.hasFreezer = hasFreezer;
        this.discount = discount;
    }

    @Override
    public int getCurrentPrice() {
        return hasFreezer ? (this.price - discount + 1000) : (this.price -
discount);
    }

    @Override
    public Product copy() {
        return new Fridge(this.color, this.name, this.price,
this.manufacturer, this.hasFreezer, this.discount);
    }

    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;
        if (!super.equals(object)) return false;
        Fridge fridge = (Fridge) object;
        return hasFreezer == fridge.hasFreezer &&
            discount == fridge.discount;
    }

    public int hashCode() {
        return java.util.Objects.hash(super.hashCode(), hasFreezer,
discount);
    }
}

```

```

package Products;

public class WashingMachine extends Product implements IProduct, Copyable {

    // no discount
    private int capacity;
}

```

```

    public WashingMachine(Color color, String name, int price, String
manufacturer, int capacity) {
        super(color, name, price, manufacturer);
        this.capacity = capacity;
    }

    @Override
    public int getCurrentPrice() {
        return this.price;
    }

    @Override
    public Product copy() {
        return new WashingMachine(this.color, this.name, this.price,
this.manufacturer, this.capacity);
    }

    public boolean equals(Object object) {
        if (this == object) return true;
        if (object == null || getClass() != object.getClass()) return false;
        if (!super.equals(object)) return false;
        WashingMachine that = (WashingMachine) object;
        return capacity == that.capacity;
    }

    public int hashCode() {
        return java.util.Objects.hash(super.hashCode(), capacity);
    }
}

```

## ProductType.java

```

package Products;

public enum ProductType {
    SAMSUNG_TV,
    SAMSUNG_FRIDGE,
    SATURN_TV,
    SATURN_FRIDGE,
    LG_TV,
    LG_FRIDGE;
}

```

## 14.IProductFactory – інтерфейс, призначений для реалізації шаблону «Factory method».

### IProductFactory.java

```

package Products.ProductFactory;

import Products.Fridge;
import Products.TVSet;
import Products.WashingMachine;
import Users.Worker;

interface IProductFactory {
    WashingMachine getWashingMachine();
    Fridge getFridge();
}

```

```

TVSet getTVSet();
int setTVDiscount(int value, Worker w);
int setFridgeDiscount(int value, Worker w);
}

```

**15.LGProductsFactory, SamsungProductsFactory, SaturnProductsFactory** – реалізації шаблону «**Factory method**», а також використовує шаблон «**Prototype**». Призначені для видачі товару різної категорії(TVSet, Fridge, WashingMachine). При видачі створюють копії товару, а не вертають реальний об'єкт.

### LGProductsFactory.java

```

package Products.ProductFactory;

import Discount.ProtectedDiscountSystem;
import Products.*;
import static Products.Color.*;
import Users.Worker;

public class LGProductsFactory implements IProductFactory {

    private ProtectedDiscountSystem discountSystem = new
ProtectedDiscountSystem();

    private WashingMachine wash;
    private TVSet tv;
    private Fridge fridge;

    public LGProductsFactory() {
        this.wash = new WashingMachine(RED, "LG WW6WIH2109SDUA", 4000, "lg",
30);
        this.tv = new TVSet(BLACK, "LG KGYVIK57C", 5000, "lg", 90);
        this.fridge = new Fridge(WHITE, "LG MLKcRT6467", 20000, "lg", false);
    }

    @Override
    public WashingMachine getWashingMachine() {
        return (WashingMachine) wash.copy();
    }

    @Override
    public Fridge getFridge() {
        return (Fridge) fridge.copy();
    }

    @Override
    public TVSet getTVSet() {
        return (TVSet) tv.copy();
    }

    @Override
    public int setTVDiscount(int value, Worker w) {
        this.discountSystem.setDiscount(tv, w, value);
        return this.tv.getDiscount();
    }

    @Override
    public int setFridgeDiscount(int value, Worker w) {
        this.discountSystem.setDiscount(fridge, w, value);
        return this.fridge.getDiscount();
    }
}

```

### SamsungProductsFactory.java

```
package Products.ProductFactory;

import Discount.ProtectedDiscountSystem;
import Products.*;
import static Products.Color.GREEN;
import Users.Worker;

public class SamsungProductsFactory implements IProductFactory {

    private ProtectedDiscountSystem discountSystem = new
ProtectedDiscountSystem();

    private WashingMachine wash;
    private TVSet tv;
    private Fridge fridge;

    public SamsungProductsFactory() {
        this.wash = new WashingMachine(GREEN, "Samsung VKCUTF46VJ", 5000,
"samsung", 5);
        this.tv = new TVSet(GREEN, "Samsung 177VJCuDYCDX", 15000, "samsung",
120);
        this.fridge = new Fridge(GREEN, "Samsung LNLIDGB42343", 20000,
"samsung", true);
    }

    @Override
    public WashingMachine getWashingMachine() {
        return (WashingMachine) wash.copy();
    }

    @Override
    public Fridge getFridge() {
        return (Fridge) fridge.copy();
    }

    @Override
    public TVSet getTVSet() {
        return (TVSet) tv.copy();
    }

    @Override
    public int setTVDiscount(int value, Worker w) {
        this.discountSystem.setDiscount(tv, w, value);
        return this.tv.getDiscount();
    }

    @Override
    public int setFridgeDiscount(int value, Worker w) {
        this.discountSystem.setDiscount(fridge, w, value);
        return this.fridge.getDiscount();
    }
}
```

### SaturnProductsFactory.java

```
package Products.ProductFactory;

import Discount.ProtectedDiscountSystem;
import Products.*;
import static Products.Color.*;
import Users.Worker;

public class SaturnProductsFactory implements IProductFactory {

    private ProtectedDiscountSystem discountSystem = new
ProtectedDiscountSystem();
```

```

private WashingMachine wash;
private TVSet tv;
private Fridge fridge;

public SaturnProductsFactory() {
    this.wash = new WashingMachine(WHITE, "SATURN ST-WK7602", 23000,
"saturn", 10);
    this.tv = new TVSet(WHITE, "SATURN ST-BBIYF68", 30000, "saturn", 150);
    this.fridge = new Fridge(BLACK, "SATURN ST-WKGIP", 10000, "saturn",
false);
}

@Override
public WashingMachine getWashingMachine() {
    return (WashingMachine) wash.copy();
}

@Override
public Fridge getFridge() {
    return (Fridge) fridge.copy();
}

@Override
public TVSet getTVSet() {
    return (TVSet) tv.copy();
}

@Override
public int setTVDiscount(int value, Worker w) {
    this.discountSystem.setDiscount(tv, w, value);
    return this.tv.getDiscount();
}

@Override
public int setFridgeDiscount(int value, Worker w) {
    this.discountSystem.setDiscount(fridge, w, value);
    return this.fridge.getDiscount();
}
}

```

**16.Copyable** – інтерфейс для реалізації шаблону «**Prototype**». Реалізовується такими класами : **TVSet, Fridge, WashingMachine**.

#### Copyable.java

```

package Products;

public interface Copyable {
    Product copy();
}

```

**17.DeliveryService** – абстрактний клас для основи шаблону «**Decorator**».

Визначає основні методи для модулю, який працює із регулюванням способів доставки придбаного покупцем товару.

#### DeliveryService.java

```

package Delivery;

import Users.Customer;
public abstract class DeliveryService {

```

```
protected void deliverProducts(Customer customer){
}
}
```

**18. BasicDeliveryService** – клас, призначений для звичайної доставки замовлення (клієнт сам забирає своє замовлення). Реалізує шаблон «**Decorator**».

#### BasicDeliveryService.java

```
package Delivery;

import Users.Customer;
import java.awt.Frame;
import javax.swing.JOptionPane;

public class BasicDeliveryService extends DeliveryService {

    @Override
    public void deliverProducts(Customer customer) {
        boolean hasEnoughMoney = customer.payForProducts();

        if (!hasEnoughMoney) {
            JOptionPane.showMessageDialog(
                new Frame("Failed order"),
                "You don't have enough money to get this order",
                "Unsuccessful order",
                JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(
                new Frame("Order"),
                "Your order is ready.",
                "Order is ready",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```

**19. AirDeliveryService, TrainDeliveryService, TaxiDeliveryService** – конкретні декоратори у шаблоні «**Decorator**». Визначають спосіб доставки: літаком, потягом, таксі.

#### AirDeliveryService.java

```
package Delivery;

import Users.Customer;
import java.awt.Frame;
import javax.swing.JOptionPane;

public class AirDeliveryService extends Decorator {

    private final int shippingCost = 1000;

    @Override
    public void deliverProducts(Customer customer) {
        customer.addShippingCost(shippingCost);
        super.deliverProducts(customer);
        JOptionPane.showMessageDialog(
            new Frame("Order"),
            "Your is arriving by air, please wait. Thanks for choosing"
        );
    }
}
```

```

our shop.\nBest wishes.",
        "Order is ready",
        JOptionPane.INFORMATION_MESSAGE);
    }
}

```

### TrainDeliveryService.java

```

package Delivery;

import Users.Customer;
import java.awt.Frame;
import javax.swing.JOptionPane;

public class TrainDeliveryService extends Decorator {

    private final int shippingCost = 500;

    @Override
    public void deliverProducts(Customer customer) {
        customer.addShippingCost(shippingCost);
        super.deliverProducts(customer);
        JOptionPane.showMessageDialog(
            new Frame("Order"),
            "Your is arriving by train,please wait. Thanks for
choosing our shop.\nBest wishes.",
            "Order is ready",
            JOptionPane.INFORMATION_MESSAGE);
    }
}

```

### TaxiDeliveryService.java

```

package Delivery;

import Users.Customer;
import java.awt.Frame;
import javax.swing.JOptionPane;

public class TaxiDeliveryService extends Decorator {

    private final int shippingCost = 300;

    @Override
    public void deliverProducts(Customer customer) {
        customer.addShippingCost(shippingCost);
        super.deliverProducts(customer);
        JOptionPane.showMessageDialog(
            new Frame("Order"),
            "Your is arriving by taxi,please wait. Thanks for choosing
our shop.\nBest wishes.",
            "Order is ready",
            JOptionPane.INFORMATION_MESSAGE);
    }
}

```

## 20. DeliveryManager – клас, який використовується для доставки замовлення.

### DeliveryManager.java

```
package Delivery;

import Users.Customer;

public class DeliveryManager {

    public static final int trainShippingCost = 1500;
    public static final int airShippingCost = 2000;
    public static final int taxiShippingCost = 1000;

    public enum ShippingType {
        TRAIN,
        TAXI,
        AIR,
        AIR_TAXI,
        AIR_TRAIN,
        TRAIN_TAXI,
        AIR_TRAIN_TAXI,
        NONE;
    }

    private DeliveryService deliveryService;

    private Decorator airDecorator;
    private Decorator taxiDecorator;
    private Decorator trainDecorator;

    public DeliveryManager() {
        deliveryService = new BasicDeliveryService();

        airDecorator = new AirDeliveryService();
        taxiDecorator = new TaxiDeliveryService();
        trainDecorator = new TrainDeliveryService();
    }

    public void ship(Customer customer, ShippingType type) {
        switch (type) {
            case TRAIN: {
                train_Shipping(customer);
                break;
            }
            case TAXI: {
                taxi_Shipping(customer);
                break;
            }
            case AIR: {
                air_Shipping(customer);
                break;
            }
            case AIR_TAXI: {
                air_taxi_Shipping(customer);
                break;
            }
            case AIR_TRAIN: {
                air_train_Shipping(customer);
                break;
            }
        }
    }
}
```



```

        case TRAIN_TAXI: {
            train_taxi_Shipping(customer);
            break;
        }
        case AIR_TRAIN_TAXI: {
            air_train_taxi_Shipping(customer);
            break;
        }
        case NONE: {
            deliveryService.deliverProducts(customer);
            break;
        }
        default: {
            throw new IllegalArgumentException("Invalid type of
shipping!");
        }
    }

}

public static int getShippingCost(ShippingType type) {
    switch (type) {
        case TRAIN: {
            return trainShippingCost;
        }
        case TAXI: {
            return taxiShippingCost;
        }
        case AIR: {
            return airShippingCost;
        }
        case AIR_TAXI: {
            return taxiShippingCost + airShippingCost;
        }
        case AIR_TRAIN: {
            return airShippingCost + trainShippingCost;
        }
        case TRAIN_TAXI: {
            return trainShippingCost + taxiShippingCost;
        }
        case AIR_TRAIN_TAXI: {
            return airShippingCost + trainShippingCost + taxiShippingCost;
        }
        case NONE: {
            return 0;
        }
        default: {
            throw new IllegalArgumentException("Invalid type of
shipping!");
        }
    }

}

private void taxi_Shipping(Customer customer) {
    taxiDecorator.addShippingRoute(deliveryService);
    taxiDecorator.deliverProducts(customer);
}

private void air_Shipping(Customer customer) {
    airDecorator.addShippingRoute(deliveryService);
    airDecorator.deliverProducts(customer);
}

private void train_Shipping(Customer customer) {
    trainDecorator.addShippingRoute(deliveryService);

```

```

        trainDecorator.deliverProducts(customer);
    }

    private void train_taxi_Shipping(Customer customer) {
        trainDecorator.addShippingRoute(deliveryService);
        taxiDecorator.addShippingRoute(trainDecorator);
        taxiDecorator.deliverProducts(customer);
    }

    private void air_taxi_Shipping(Customer customer) {
        airDecorator.addShippingRoute(deliveryService);
        taxiDecorator.addShippingRoute(airDecorator);
        taxiDecorator.deliverProducts(customer);
    }

    private void air_train_Shipping(Customer customer) {
        airDecorator.addShippingRoute(deliveryService);
        trainDecorator.addShippingRoute(airDecorator);
        trainDecorator.deliverProducts(customer);
    }

    private void air_train_taxi_Shipping(Customer customer) {
        airDecorator.addShippingRoute(deliveryService);
        trainDecorator.addShippingRoute(airDecorator);
        taxiDecorator.addShippingRoute(trainDecorator);
        taxiDecorator.deliverProducts(customer);
    }
}

```

**21.SignInWindow** – клас, який реалізує інтерфейс входу в систему. Має багато логіки.

### SignInWindow.java

```

package Forms;

import DB.UsersDataBase;
import Products.ProductFactory.LGProductsFactory;
import Products.ProductFactory.SamsungProductsFactory;
import Products.ProductFactory.SaturnProductsFactory;
import State.*;
import Users.*;
import java.awt.Frame;
import javax.swing.JOptionPane;

public class SignInWindow extends javax.swing.JFrame {

    private final UsersDataBase db = new UsersDataBase();
    private StateFactory factory = new StateFactory();

    private UserType userType;
    private User user;

    private WorkerUI workerUI;
    private CustomerUI customerUI;

    public final SamsungProductsFactory samsungFactory = new
    SamsungProductsFactory();
    public final LGProductsFactory lgFactory = new LGProductsFactory();
    public final SaturnProductsFactory saturnFactory = new
    SaturnProductsFactory();

    public SignInWindow() {

```

```

        initComponents();
        this.setVisible(true);
        this.user = null;
    }

    public void openCustomerUI() {
        customerUI = new CustomerUI(this);
        customerUI.setVisible(true);
    }

    public void openWorkerUI() {
        workerUI = new WorkerUI(this);
        workerUI.setVisible(true);
    }

    public void showNextForm() {
        if (user != null) {
            JOptionPane.showMessageDialog(
                new Frame("Successful login to the system"),
                "Welcome, " + user.getFullname(),
                "Login to the system",
                JOptionPane.INFORMATION_MESSAGE);
        }

        // state + fllweight
        factory.getState(userType).execute(this);
    }

    public User getUser() {
        return this.user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    private void SignInBtnActionPerformed(java.awt.event.ActionEvent evt) {

        String login = Login.getText();

        setUser(db.getUser(login));

        if (user == null) {
            JOptionPane.showMessageDialog(
                new Frame("Invalid login"),
                "There is no such user, try again",
                "Login to the system",
                JOptionPane.WARNING_MESSAGE);
        } else {
            this.userType = user.getType();
            showNextForm();
        }
    }

    private void MissLoginBtnActionPerformed(java.awt.event.ActionEvent evt)
this.user = null;
        this.userType = UserType.CUSTOMER;
        showNextForm();
    }

```

## 21. CustomerUI – клас, який працює з кошиком покупця, має багато логіки.

### CustomerUI.java

```
package Forms;

import Products.Product;
import Products.ProductFactory.*;
import Users.*;
import java.awt.Frame;
import static java.lang.Integer.parseInt;
import java.util.HashMap;
import javax.swing.JOptionPane;

public class CustomerUI extends javax.swing.JFrame {

    private SignInWindow parentWindow;
    private OrderUI orderWindow;
    private UserFillInfoUI infoWindow;
    public HashMap<Product, Integer> curBasket = new HashMap<>();

    public CustomerUI(SignInWindow sw) {

        this.parentWindow = sw;
        initComponents();

        updateUserInfo();
        initDiscounts();
    }

    private void initDiscounts() {
        LGTVDiscount.setText(parentWindow.lgFactory.getTVSet().getDiscount()
        == 0 ? "" : "Discount : " + parentWindow.lgFactory.getTVSet().getDiscount());

        SamsungTVDiscount.setText(parentWindow.samsungFactory.getTVSet().getDiscount()
        == 0 ? "" : "Discount : " +
        parentWindow.samsungFactory.getTVSet().getDiscount());

        SaturnTVDiscount.setText(parentWindow.saturnFactory.getTVSet().getDiscount()
        == 0 ? "" : "Discount : " +
        parentWindow.saturnFactory.getTVSet().getDiscount());

        LGFridgeDiscount.setText(parentWindow.lgFactory.getFridge().getDiscount() == 0
        ? "" : "Discount : " + parentWindow.lgFactory.getFridge().getDiscount());

        SamsungFridgeDiscount.setText(parentWindow.samsungFactory.getFridge().getDisco
        unt() == 0 ? "" : "Discount : " +
        parentWindow.samsungFactory.getFridge().getDiscount());

        SaturnFridgeDiscount.setText(parentWindow.saturnFactory.getFridge().getDiscoun
        t() == 0 ? "" : "Discount : " +
        parentWindow.saturnFactory.getFridge().getDiscount());
    }

    public Customer getUser() {
        return (Customer) parentWindow.getUser();
    }

    public void setCustomer(Customer c) {
        parentWindow.setUser(c);
    }

    void clearBasket() {
        curBasket.clear();
        this.Basket.removeAll();
    }
}
```

```

        SamsungFridgeCounter.setText("0");
        LGFridgeCounter.setText("0");
        SaturnFridgeCounter.setText("0");
        SamsungTVCounter.setText("0");
        LGTVCounter.setText("0");
        SaturnTVCounter.setText("0");
        SamsungWashCounter.setText("0");
        LGWashCounter.setText("0");
        SaturnWashCounter.setText("0");
    }

    private void ReloginBtnActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_ReloginBtnActionPerformed
        this.dispose();
        parentWindow.setVisible(true);

    } //GEN-LAST:event_ReloginBtnActionPerformed

    public void updateUserInfo() {
        if (parentWindow.getUser() != null) {
            welcomeLbl.setText("Welcome, " +
parentWindow.getUser().getFullname());
            BalanceLbl.setText("Your balance : " + (int) ((Customer)
parentWindow.getUser()).getMoney());
        } else {
            welcomeLbl.setText("Welcome!");
            BalanceLbl.setText("");
        }
    }

    private void BuyBtnActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_BuyBtnActionPerformed
        if (parentWindow.getUser() == null) {
            infoWindow = new UserFillInfoUI(this);
            JOptionPane.showMessageDialog(
                new Frame("No information about user"),
                "Please, fill the form to make a purchase",
                "Fill information",
                JOptionPane.INFORMATION_MESSAGE);
            infoWindow.setVisible(true);
        } else {
            this.setVisible(false);
            ((Customer) parentWindow.getUser()).setBasket(curBasket);
            orderWindow = new OrderUI(this);
        }

    }

    } //GEN-LAST:event_BuyBtnActionPerformed

    private void SamsungTVPlusBtnActionPerformed(java.awt.event.ActionEvent evt)
    { //GEN-FIRST:event_SamsungTVPlusBtnActionPerformed
        SamsungTVCounter.setText("" + (parseInt(SamsungTVCounter.getText()) + 1));
        Basket.add("TV: " + SamsungTVLbl.getText());
        curBasket.put(parentWindow.samsungFactory.getTVSet(),
parseInt(SamsungTVCounter.getText()));
    } //GEN-LAST:event_SamsungTVPlusBtnActionPerformed

```

**22.WorkerUI** – клас, який призначений для роботи працівника магазину. Містить багато логіки.

## WorkerUI.java

```
package Forms;

import Products.ProductType;

public class WorkerUI extends javax.swing.JFrame {

    private SignInWindow parent;

    WorkerUI(SignInWindow sw) {

        this.parent = sw;

        initComponents();
        initDiscounts();

        welcomeLbl.setText("Welcome, " + parent.getUser().getFullname());

        addEventListeners();

    }

    private void initDiscounts() {
        LGTVDiscount.setText(" " + parent.lgFactory.getTVSet().getDiscount());
        SamsungTVDiscount.setText(" " +
parent.samsungFactory.getTVSet().getDiscount());
        SaturnTVDiscount.setText(" " +
parent.saturnFactory.getTVSet().getDiscount());
        LGFridgeDiscount.setText(" " +
parent.lgFactory.getFridge().getDiscount());
        SamsungFridgeDiscount.setText(" " +
parent.samsungFactory.getFridge().getDiscount());
        SaturnFridgeDiscount.setText(" " +
parent.saturnFactory.getFridge().getDiscount());

    }

    private void addEventListeners() {
        LGTVDiscount.addActionListener(new ActionListener(LGTVDiscount,
parent, ProductType.LG_TV));
        SamsungTVDiscount.addActionListener(new
MyActionListener(this.SamsungTVDiscount, parent, ProductType.SAMSUNG_TV));
        SaturnTVDiscount.addActionListener(new
MyActionListener(SaturnTVDiscount, parent, ProductType.SATURN_TV));
        LGFridgeDiscount.addActionListener(new
MyActionListener(LGFridgeDiscount, parent, ProductType.LG_FRIDGE));
        SamsungFridgeDiscount.addActionListener(new
MyActionListener(SamsungFridgeDiscount, parent, ProductType.SAMSUNG_FRIDGE));
        SaturnFridgeDiscount.addActionListener(new
MyActionListener(SaturnFridgeDiscount, parent, ProductType.SATURN_FRIDGE));
    }

    private void ReloginBtnActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_ReloginBtnActionPerformed
        this.dispose();
        parent.setVisible(true);
    }
}
```

**23. UserFillInfo** – клас, який пропонує користувачі ввести свої дані для здійснення замовлення.

**UserFillInfo.java**

```

package Forms;

import Users.Customer;
import java.awt.Frame;
import static java.lang.Integer.parseInt;
import javax.swing.JOptionPane;

public class UserFillInfoUI extends javax.swing.JFrame {

    private Customer customer;
    private CustomerUI customerUI;

    public UserFillInfoUI(CustomerUI ui) {
        initComponents();
        customerUI = ui;
        this.customer = null;
    }
    private void CancelBtnActionPerformed(java.awt.event.ActionEvent evt)
    { //GEN-FIRST:event_CancelBtnActionPerformed
        this.dispose();
    } //GEN-LAST:event_CancelBtnActionPerformed

    private void OkBtnActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_OkBtnActionPerformed
        try {
            if (!FullnameField.getText().isEmpty() &&
            !AgeField.getText().isEmpty() && parseInt(AgeField.getText()) > 16) {
                customerUI.setCustomer(new Customer(parseInt(AgeField.getText()),
                FullnameField.getText(), MaleRadio.isSelected() ? "male" : "female"));
                customerUI.updateUserInfo();
                this.dispose();
            } else {
                JOptionPane.showMessageDialog(
                    new Frame("ALL fields must be filled"),
                    "Please, fill all fields (age > 16)",
                    "Fill fields",
                    JOptionPane.WARNING_MESSAGE);
            }
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(
                new Frame("Bad value"),
                "Please, fill correct age (age > 16 )",
                "Fill correct age",
                JOptionPane.INFORMATION_MESSAGE);
        }
    } //GEN-LAST:event_OkBtnActionPerformed

```

**24. OrderUI** – клас, який містить багато логіки. Працює безпосередньо із замовлення покупця, дає можливість обрати доставку, тип оплати.

**OrderUI.java**

```

package Forms;

import Delivery.DeliveryManager;
import Delivery.DeliveryManager.ShippingType;
import static Delivery.DeliveryManager.ShippingType.*;
import Payment.*;

```

```

import Products.Product;
import java.awt.Frame;
import java.util.HashMap;
import javax.swing.JOptionPane;

public class OrderUI extends javax.swing.JFrame {

    private CustomerUI parent;
    private DeliveryManager delivery = new DeliveryManager();
    private ShippingType type;

    public OrderUI(CustomerUI parent) {
        initComponents();

        this.parent = parent;

        fillComponents(this.parent.curBasket);

        DeliveryPanel.setVisible(false);
        this.setVisible(true);
    }

    private void fillComponents(HashMap<Product, Integer> map) {
        map.keySet().forEach((Product key) -> {
            if (map.get(key) != 0) {
                Basket.add(key.getName() + " (x" + map.get(key) + ")");
            }
        });
        this.ProductsCostLbl.setText("" + parent.getUser().getAllProductsCost());
        this.BalanceLbl.setText("Your balance : " + (int)
parent.getUser().getMoney());
        this.TotalCostLbl.setText(ProductsCostLbl.getText());
    }

    private void GoBackBtnActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_GoBackBtnActionPerformed
        this.parent.setVisible(true);
        this.dispose();
    } //GEN-LAST:event_GoBackBtnActionPerformed

    private void MakeOrderBtnActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_MakeOrderBtnActionPerformed
        if (Basket.getItemCount() == 0) {
            JOptionPane.showMessageDialog(
                new Frame("Empty basket"),
                "The basket is empty, add some products",
                "Empty basket",
                JOptionPane.ERROR_MESSAGE);
        } else {
            if (!parent.getUser().hasMoneyToPay(chooseShippingType())) {
                JOptionPane.showMessageDialog(
                    new Frame("Failed order"),
                    "You don't have enough money to get this order",
                    "Unsuccessful order",
                    JOptionPane.INFORMATION_MESSAGE);
            } else {
                delivery.ship(parent.getUser(), chooseShippingType());
                parent.setVisible(true);
                parent.updateUserInfo();
                parent.clearBasket();
                this.dispose();
            }
        }
    }
}

```



```

} //GEN-LAST:event_MakeOrderBtnActionPerformed

private ShippingType chooseShippingType() {
    if (AirCheckBox.isSelected() && TrainCheckBox.isSelected() &&
        TaxiCheckBox.isSelected()) {
        return AIR_TRAIN_TAXI;
    } else if (TrainCheckBox.isSelected() && TaxiCheckBox.isSelected()) {
        return TRAIN_TAXI;
    } else if (AirCheckBox.isSelected() && TaxiCheckBox.isSelected()) {
        return AIR_TAXI;
    } else if (AirCheckBox.isSelected() && TrainCheckBox.isSelected()) {
        return AIR_TRAIN;
    } else if (AirCheckBox.isSelected()) {
        return AIR;
    } else if (TrainCheckBox.isSelected()) {
        return TRAIN;
    } else if (TaxiCheckBox.isSelected()) {
        return TAXI;
    } else {
        return NONE;
    }
}

private void getDeliveryCheckBoxActionPerformed(java.awt.event.ActionEvent
    evt) { //GEN-FIRST:event_getDeliveryCheckBoxActionPerformed
    DeliveryPanel.setVisible(getDeliveryCheckBox.isSelected());
    if (!getDeliveryCheckBox.isSelected()) {
        TaxiCheckBox.setSelected(false);
        TrainCheckBox.setSelected(false);
        AirCheckBox.setSelected(false);
    }
} //GEN-LAST:event_getDeliveryCheckBoxActionPerformed

private void RefreshCostActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_RefreshCostActionPerformed
    this.TotalCostLbl.setText("" + (parent.getUser().getAllProductsCost() +
        delivery.getShippingCost(chooseShippingType())));
} //GEN-LAST:event_RefreshCostActionPerformed

private void CardPaymentRadioActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_CardPaymentRadioActionPerformed
    parent.getUser().setCustomerPayment(new CustomerPayment(new
        CardPayment()));
} //GEN-LAST:event_CardPaymentRadioActionPerformed

private void CashPaymentRadioActionPerformed(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_CashPaymentRadioActionPerformed
    parent.getUser().setCustomerPayment(new CustomerPayment(new
        CashPayment()));
} //GEN-LAST:event_CashPaymentRadioActionPerformed
}

```

**25. Program** – клас для запуску мейну – тобто, створення вікна входу в систему(**SignInWindow**).

### Program.java

```

import Forms.SignInWindow;

public class Program {
    public static void main(String[] args) {
        SignInWindow form = new SignInWindow();
    }
}

```





### 3. Flyweight

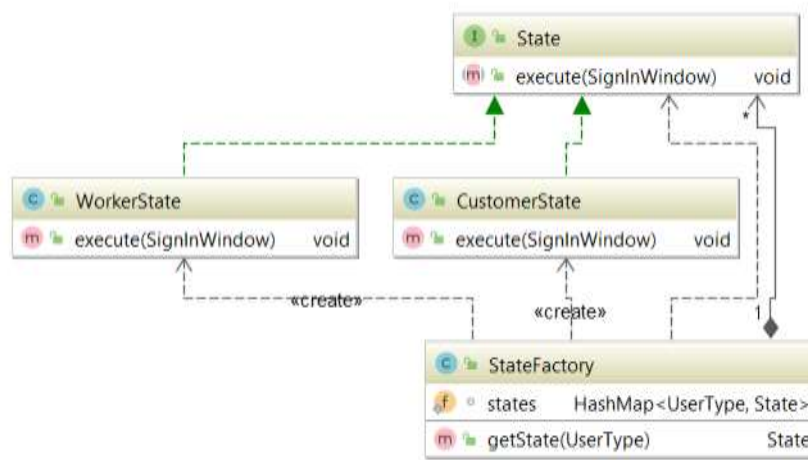


Рис. 2.1.5. UML-діаграма шаблону “Flyweight”

**Структурний шаблон.** Структурує об'єкти таким чином, що з них створюється лише обмежений набір екземплярів замість великої множини об'єктів. Полегшує повторне використання багатьох малих об'єктів, роблячи використання великої кількості об'єктів більш ефективною.

**Структура.** **State** визначає інтерфейс, за допомогою якого пристосуванці можуть отримувати зовнішній стан. **WorkerState** та **CustomerState** - конкретні пристосуванці, реалізують інтерфейс **State**. **StateFactory** - створює об'єкти-пристосуванці та управляє ними. Коли **SignInWindow** звертається до пристосування, об'єкт **StateFactory** надає існуючий екземпляр або створює новий, якщо готового ще немає.

**Обґрунтування використання даного шаблону.** Оскільки у системі додано можливість перезайти в систему (relogin), то було б недоцільно створювати кожного разу стан, через який відкривається нове вікно (інтерфейс покупця або працівника). Тому було вирішено додати **StateFactory**, яка на кожен запит «relogin» вертає потрібний стан.

## 4. Factory Method

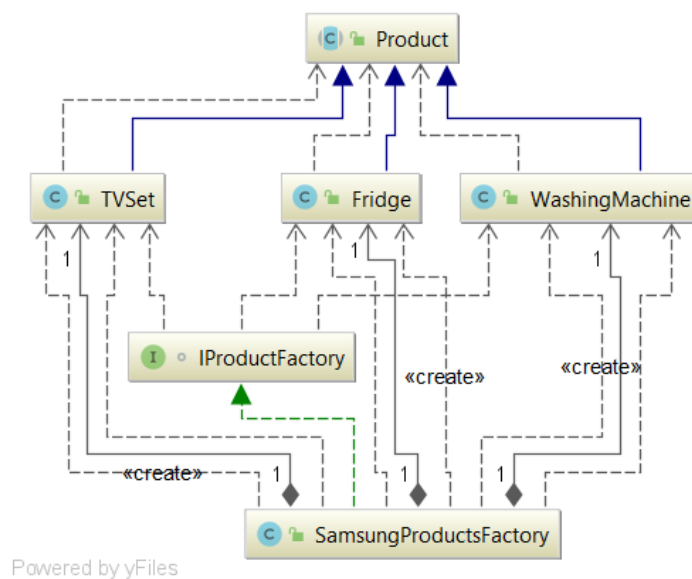


Рис. 2.1.3. UML-діаграма шаблону “*Factory Method*”

*Породжуючий шаблон.* Визначає інтерфейс для створення об'єктів, але дозволяє підкласам змінювати тип створюваних об'єктів.

*Структура.* **IProductFactory** – головний інтерфейс даного шаблону. Має в собі методи повернення товарів різного типу (**TVSet**, **Fridge**, **Washing Machine**). Цей інтерфейс реалізують 3 інші класи, але вище наведений тільки один клас **SamsungProductsFactory**. Ці три класи і представляють весь спектр наявних у програмі продуктів.

*Обґрунтування використання даного шаблону.* За допомогою даного шаблону можна організувати гнучку реалізацію різновидів певного класу, і в даній програмі така реалізація потрібна була для створення різних товарів.

## 5. Proxy

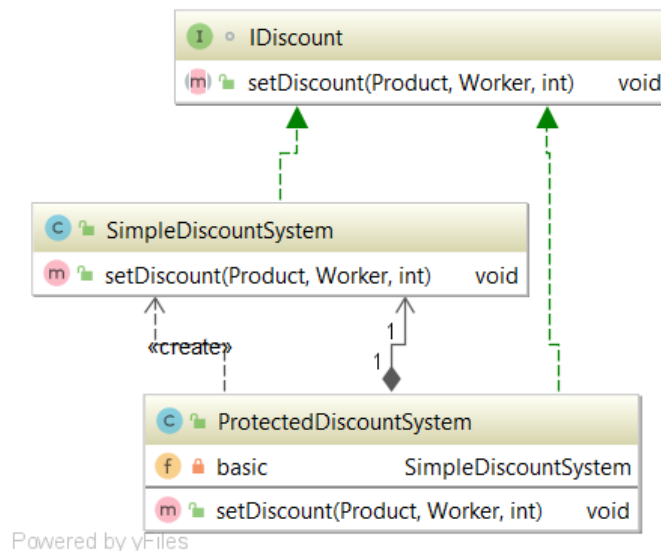


Рис. 2.1.4. UML-діаграма шаблону “**Proxy**”

*Структурний шаблон.* Обертає корисний об'єкт або сервіс спеціальним об'єктом-замінником, який «прикидається» оригіналом і перехоплює всі виклики до нього, а потім, після деякої обробки, направляє їх до обгорнутого об'єкту.

*Структура.* **ProtectedDiscountSystem** вміщає у собі об'єкт класу **SimpleDiscountSystem**. Усі методи, невизначені для контролювання у цьому шаблоні, просто переадресуються об'єкту основного класу. Інші методи (а саме метод *setDiscount*), мають у середині додаткову перевірку перед виконанням основної логіки, і вона буде виконана тільки тоді, коли **ProtectedDiscountSystem** проведе необхідні перевірки.

*Обґрунтування використання даного шаблону.* Для забезпечення того, щоб працівники (окрім менеджера) не могли поставити знижку на товар більшу, ніж 50%, був використаний шаблон «Проксу»(у вигляді *protection proxy*).

## 6. Bridge

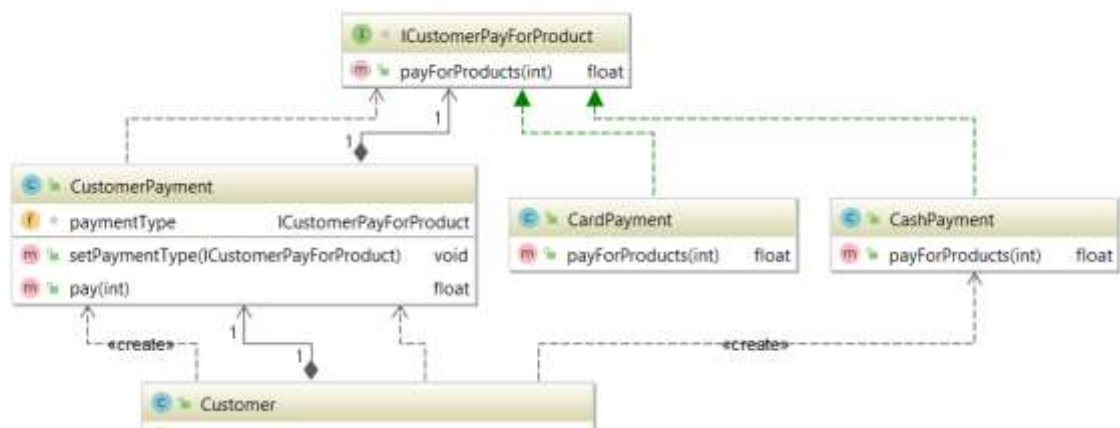


Рис. 2.1.5. UML-діаграма шаблону “**Bridge**”

**Структурний шаблон.** Це структурний шаблон проектування, що дозволяє розділяти абстракцію і реалізацію таким чином, щоб вони могли змінюватися незалежно. Цей шаблон використовує інкапсуляцію, агрегування та успадкування для того, щоб розділити відповідальність між класами.

**Структура.** **ICustomerPayment** - інтерфейс, що дозволяє платити за товар. **CashPayment** – клас, що повертає повну вартість покупки, **CardPayment**- повертає 10% кешбеку за користування картою. **CashPayment**, **CardPayment** – конкретні виконавці шаблону.

**Обґрунтування використання даного шаблону.** Оскільки конкретну реалізацію необхідно вибирати під час виконання програми, тобто вибирати тип оплати, було використано шаблон «**Bridge**».

## 7. Prototype

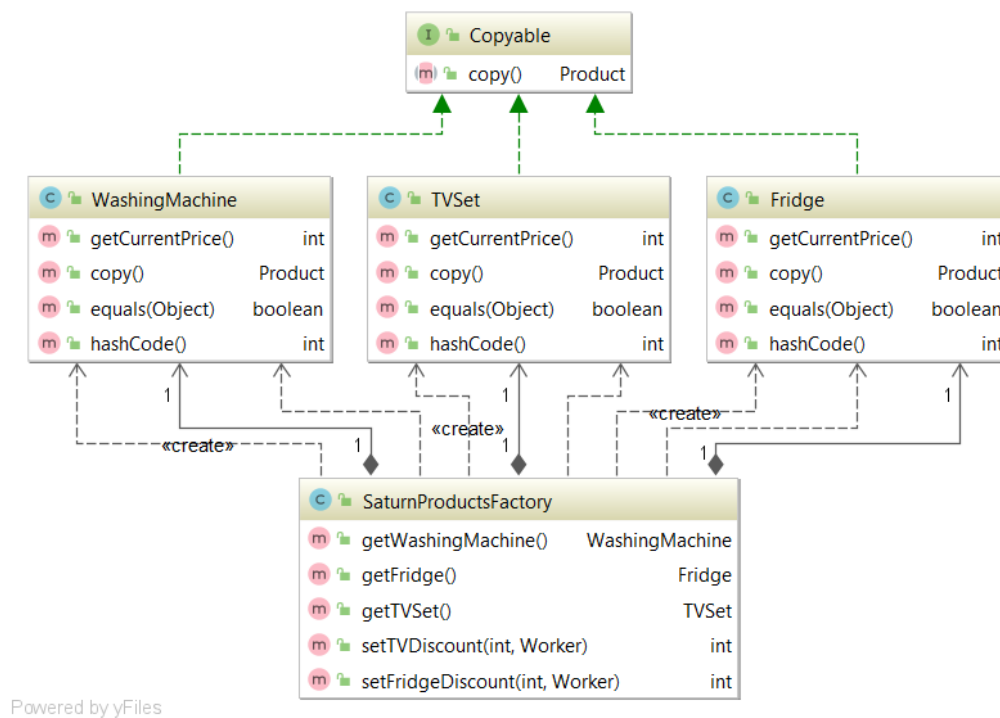


Рис. 2.1.6. UML-діаграма шаблону “**Prototype**”

*Породжуючий шаблон.* Задає види створюваних об’єктів за допомогою екземпляра-прототипу і створює нові об’єкти шляхом копіювання цього прототипу.

*Структура.* **Copyable** – головний інтерфейс даного шаблону. **TVSet**, **Fridge**, **WashingMachine** — класи, які реалізують інтерфейс **Copyable**. **SamsungProductsFactory**, **LGProductsFactory**, **SaturnProductsFactory** – класи, які повертають копії об’єктів **TVSet**, **Fridge**, **WashingMachine**.

*Обґрунтування використання даного шаблону.* Оскільки класи, які породжуються, визначаються під час виконання програми.



## 1.4. Діаграма класів

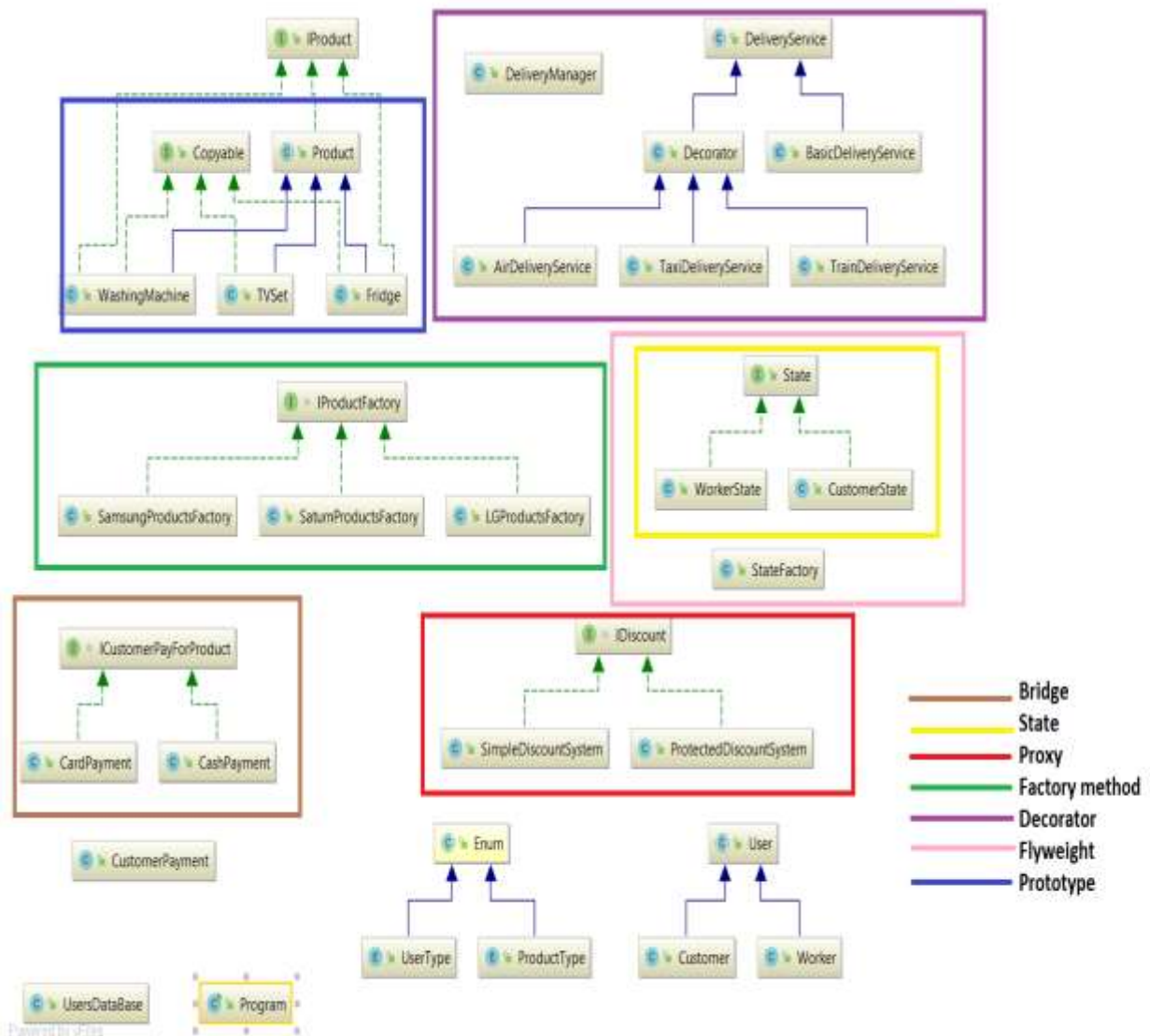



Рис. 2.2.1. Діаграма класів програми

## 1.5. Результати роботи програми

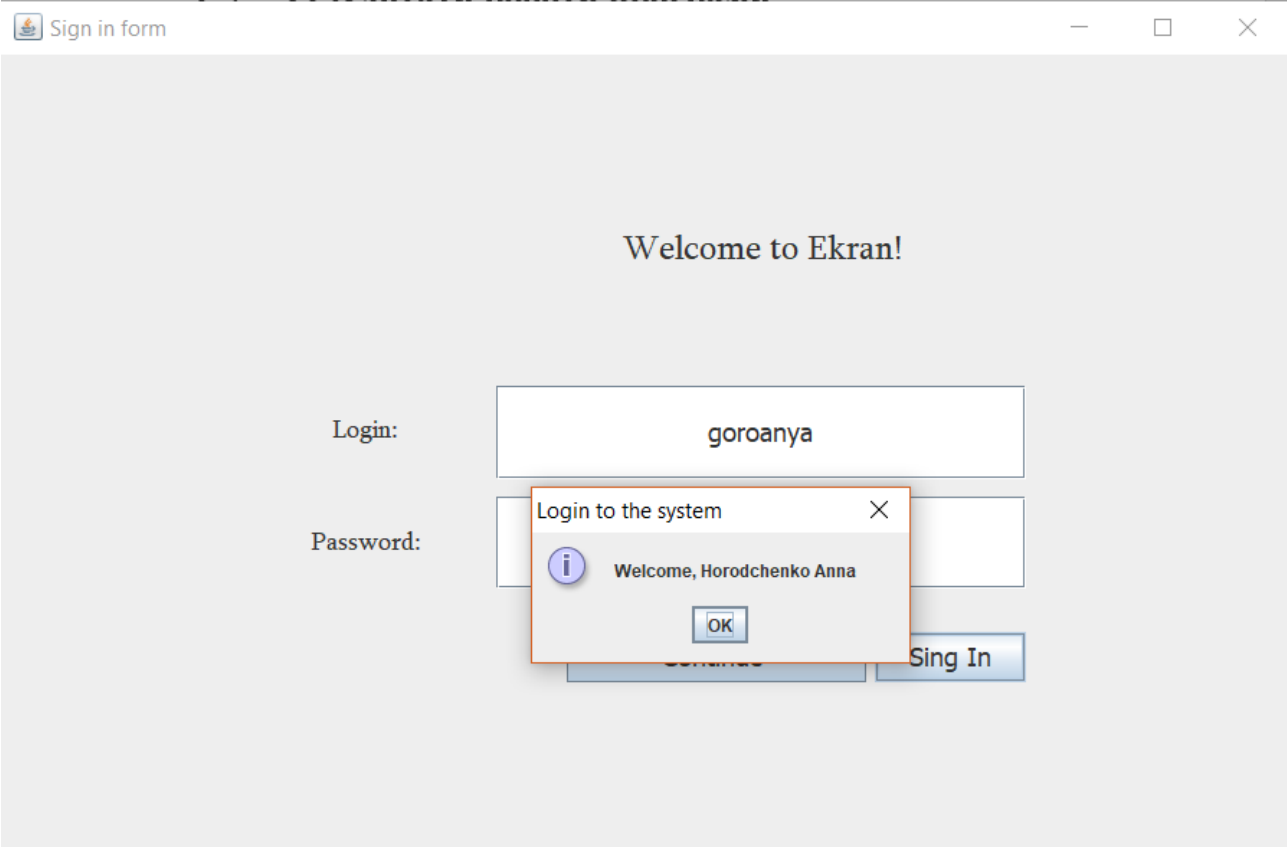
Початок роботи з магазином техніки з входу в систему:



The screenshot shows a window titled "Sign in form" with a light gray background. At the top center, it says "Welcome to Ekran!". Below this, there are two input fields: "Login:" and "Password:". The "Login:" field is empty, and the "Password:" field is also empty. At the bottom, there are two buttons: "Continue" and "Sing In" (note the typo).

Рис. 2.3.1. Вхід в систему

Успішно увійшли в магазин, як покупець, ввівши логін:



The screenshot shows the same "Sign in form" window, but now the "Login:" field contains the text "goroanya". A small dialog box titled "Login to the system" is overlaid on the form. The dialog box has a close button (X) in the top right corner. Inside the dialog, there is an information icon (i) and the text "Welcome, Horodchenko Anna". At the bottom of the dialog is an "OK" button. The "Password:" field is still empty, and the "Continue" and "Sing In" buttons are still visible.

Рис. 2.3.2. Успішний вхід

Додаємо товари в кошик, натискуючи кнопки + та – біля вибраного товару. На екрані бачимо відображення кількості грошей у клієнта та його ім'я.

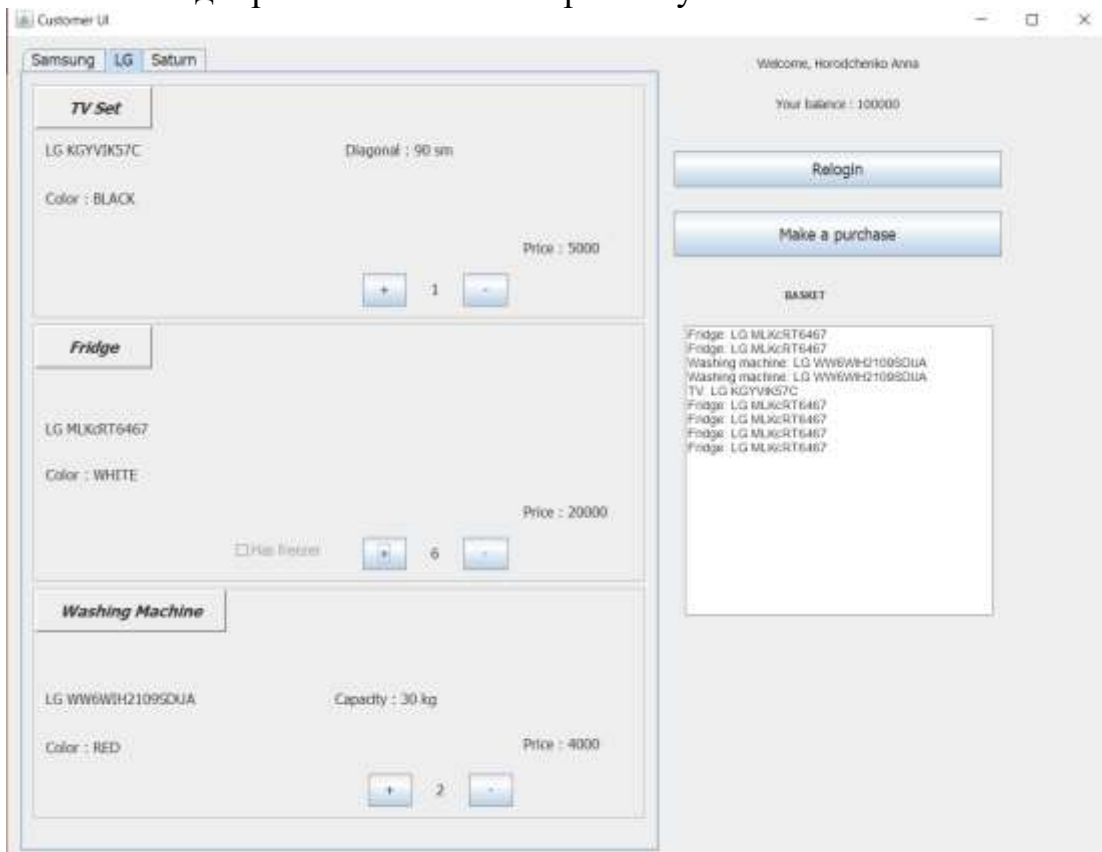


Рис. 2.3.3. Додавання в кошик

Після нажаття на кнопку «Зробити замовлення», переходимо в інше вікно, у вікно замовлення. Тут відображається кошик з кількістю товарів, ціну за товари, кнопки обрання типу оплати та можливості взяти доставку.

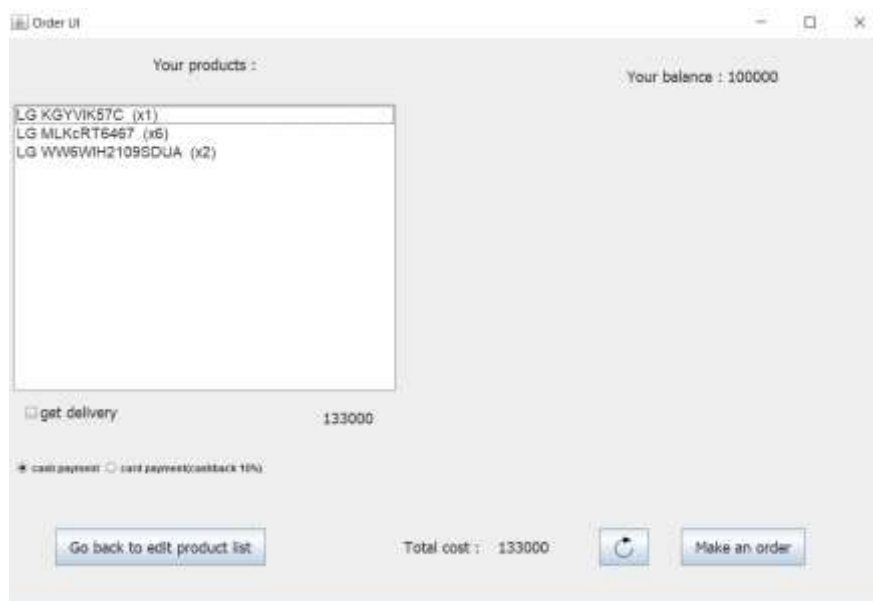
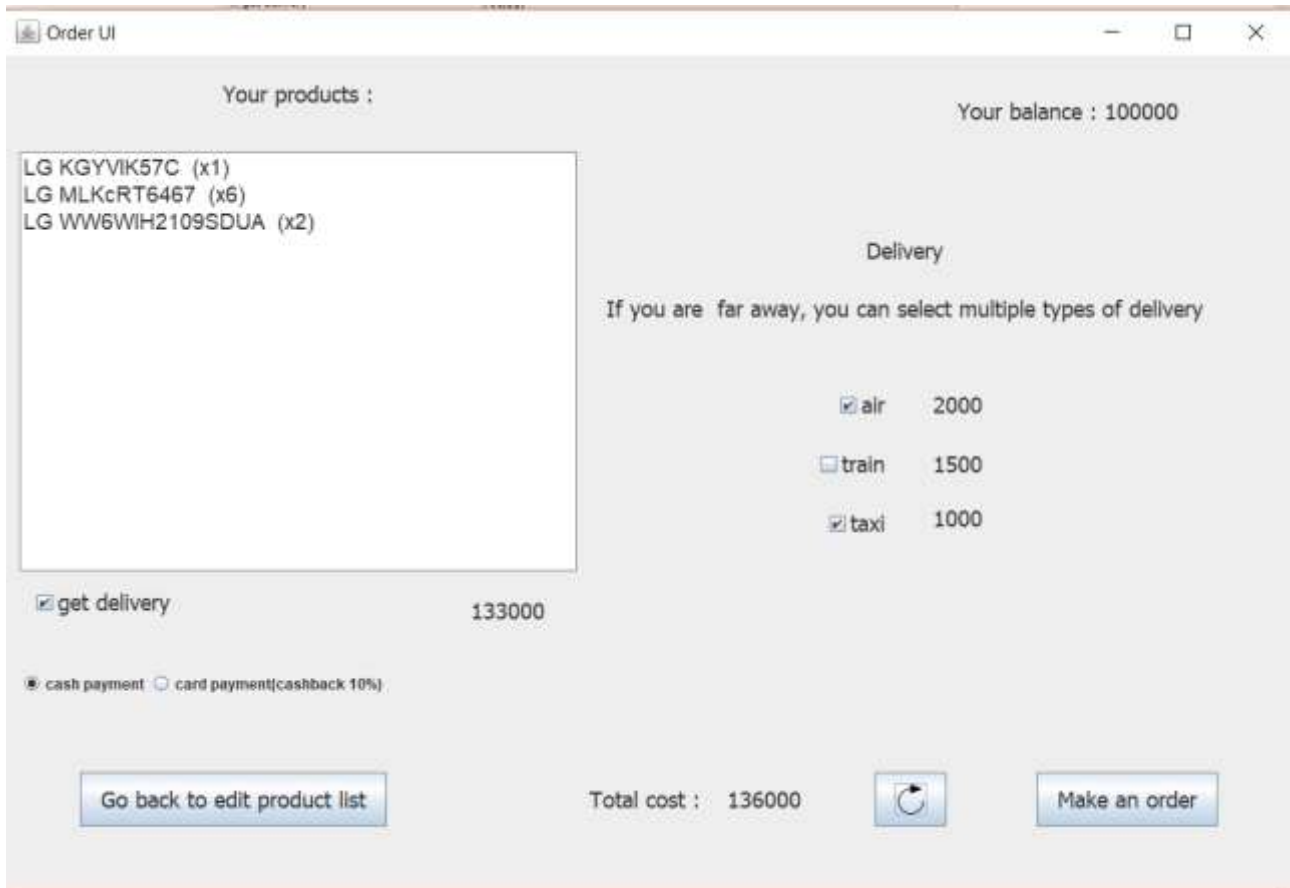


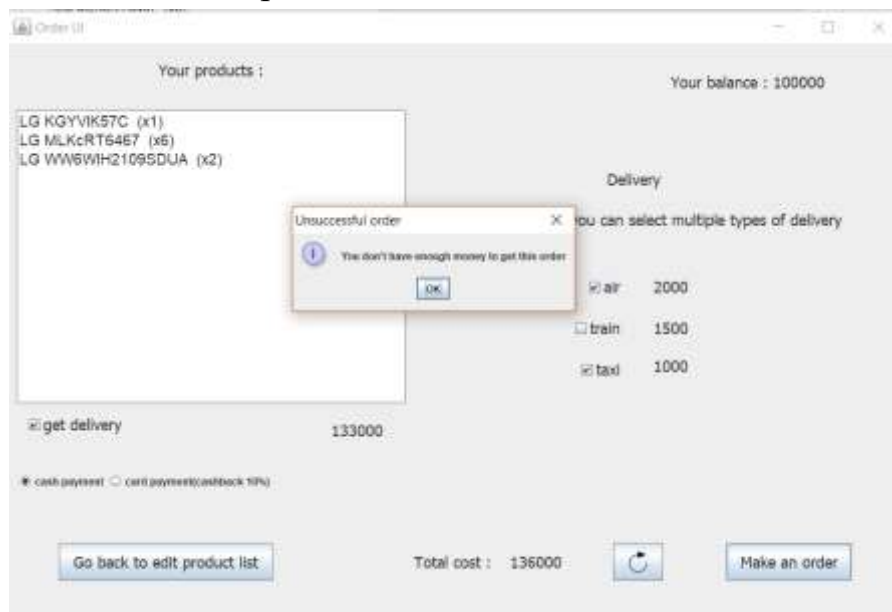
Рис. 2.3.4. Вікно замовлення

Обираємо доставку : літаком та таксі. Оновлюємо загальну вартість за замовлення. Бачимо, що загальна вартість збільшилась, оскільки ми замовили доставку.



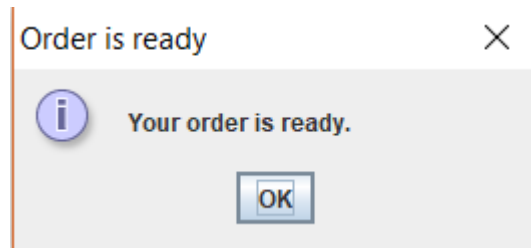
*Рис. 2.3.5.Вікно замовлення з вибраною доставкою.*

Замовляємо. Бачимо, що у нас не вистачає коштів, повертаємось і редагуємо кошик, щоб нам вистачило грошей.

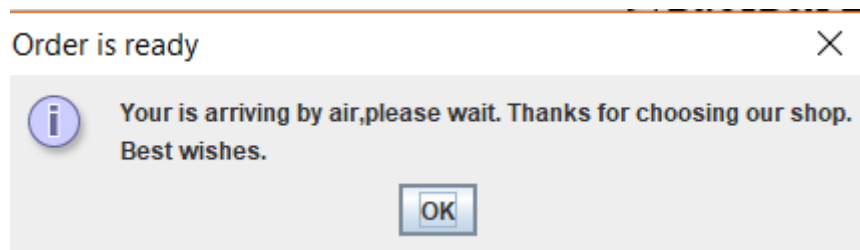


*Рис. 2.3.6.Вікно замовлення . Недостатньо коштів.*

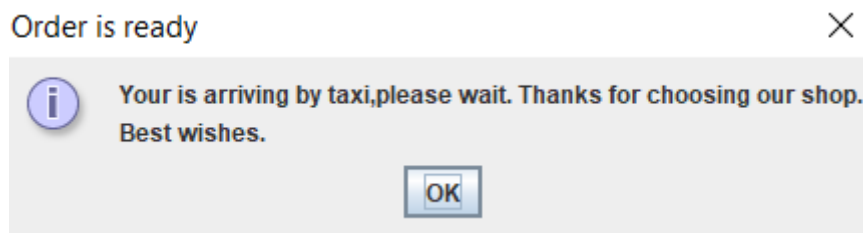
Коштів достатньо, замовлення обробляється і відбувається доставка:



*Рис. 2.3.7.Замовлення готове.*



*Рис. 2.3.8.Замовлення доставляється літаком.*



*Рис. 2.3.9. Замовлення доставляється таксі.*

Перезаходимо в магазин в акаут працівника(посада НЕ менеджер). Ставимо знижку на товар більшу, ніж 50%. Бачимо помилку:

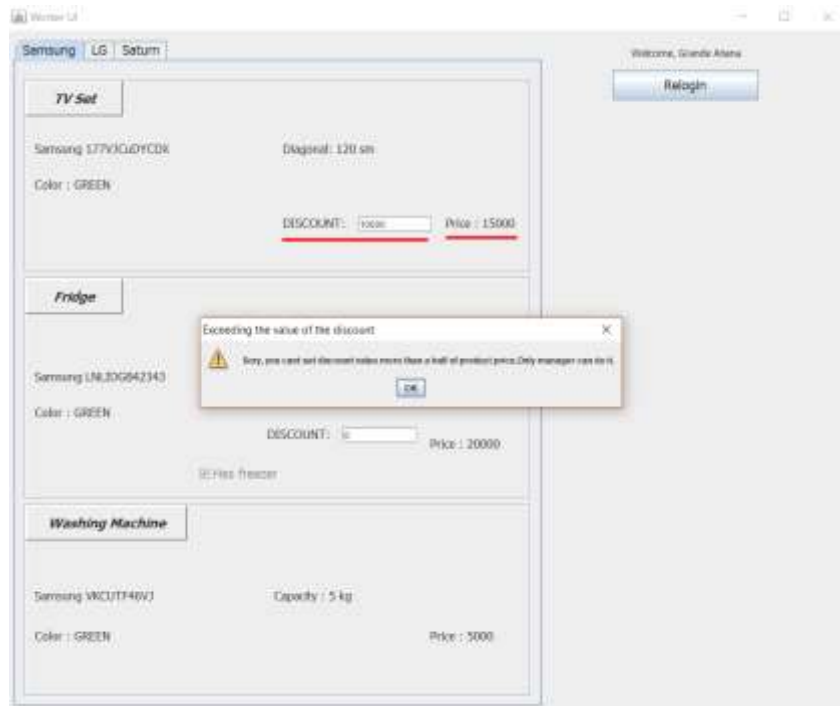


Рис. 2.3.10.Помилка встановлення знижки працівником.

Тепер встановлюємо знижку 5000 ( 30% від ціни товару). Перезаходимо в систему без авторизації(нажавши кнопку Continue).Бачимо , що встановилась знижка на товар .

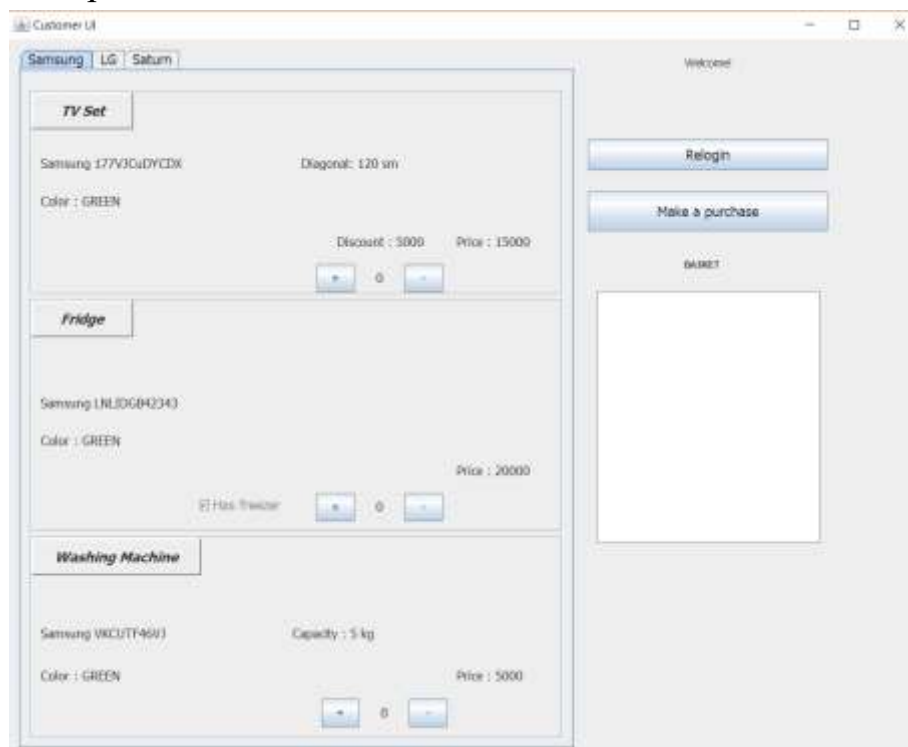
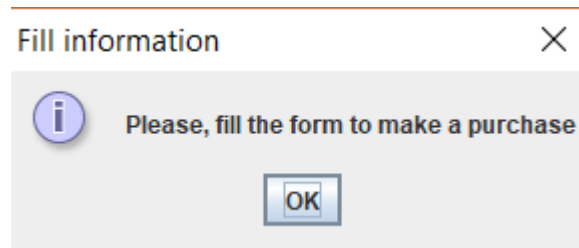


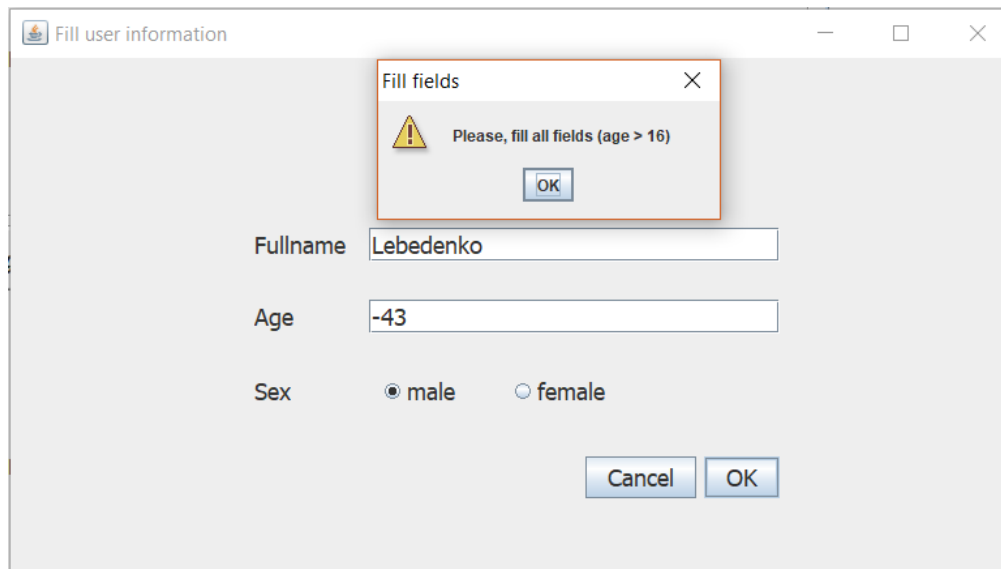
Рис. 2.3.11.Вікно товарів після встановлення знижки.

Спробуємо зробити замовлення, бачимо, що потрібно надати дані про себе, щоб зробити замовлення.



*Рис. 2.3.12. Вікно товарів після встановлення знижки.*

Бачимо вікно вводу даних. Через некоректний ввід не можна зробити замовлення.



*Рис. 2.3.13. Вікно вводу даних про користувача та помилка некоректних даних.*

## ВИСНОВКИ

Метою даної курсової роботи було розроблення програмного забезпечення магазину для продажу техніки з використанням шаблонів проектування. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Об'єктно-орієнтоване програмування» студентами II курсу кафедри ПЗКС НТУУ «КПІ».

Для досягнення поставленої мети у повному обсязі виконано завдання, визначені у аркуші завдання на курсову роботу; розроблено графічні матеріали; реалізовано всі вимоги до програмного продукту, наведені у технічному завданні; створено відповідну документацію. Розроблене програмне забезпечення дозволяє користувачу купляти товари, домовлятись про доставку.

Програму створено на основі використання шаблонів проектування. Зокрема, до структури програмного забезпечення входить реалізація 7 шаблонів, які належать до різних груп.

Для розроблення програмного забезпечення була використана мова програмування Java та IDE NetBeans.



## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Язык шаблонов. Города. Здания. Строительство. / Кристофер Вольфганг Александер. – 1977. – 1096 с.
2. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. – 1994. – 395 с.
3. Руководство Microsoft по проектированию архитектуры приложений. / С. Сомасегар, Скотт Гатри, Дэвид Хилл. – 2009. – 529 с.