Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему

**Шаблони проектування в ООП. Графічний редкатор**

Виконав студент

ІІ курсу групи КП-73

Булаєвський Ігор Олегович

залікова книжка КП-5104

Керівник роботи

доц., к.т.н. Заболотня Т.М.

Оцінка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, підпис)

Київ-2019

**ЗМІСТ**

**ВСТУП**.........................................................................................................................3

1. **ОПИС СТРУКТУРНО-ЛОГІЧНОЇ ОРГАНІЗАЦІЇ ПРОГРАМИ**.........5
   1. Модульна організація програми............................................................5
   2. Функціональні характеристики.............................................................7
   3. Опис реалізованих класів.......................................................................8
2. **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ**................................................................22
   1. Обґрунтування вибору та опис шаблонів проектування для реалізації програмного забезпечення автомату..................................22
   2. Діаграма класів......................................................................................29
   3. Результати роботи програми................................................................30

**ВИСНОВКИ**..............................................................................................................35

**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**......................................................36

**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, і для організації поставки товару на полиці магазину та 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 домашньої 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 із продажу техніки для дому*.

**Метою** роботи є *розроблення програмного забезпечення* *для магазину із продажу техніки для дому* 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.

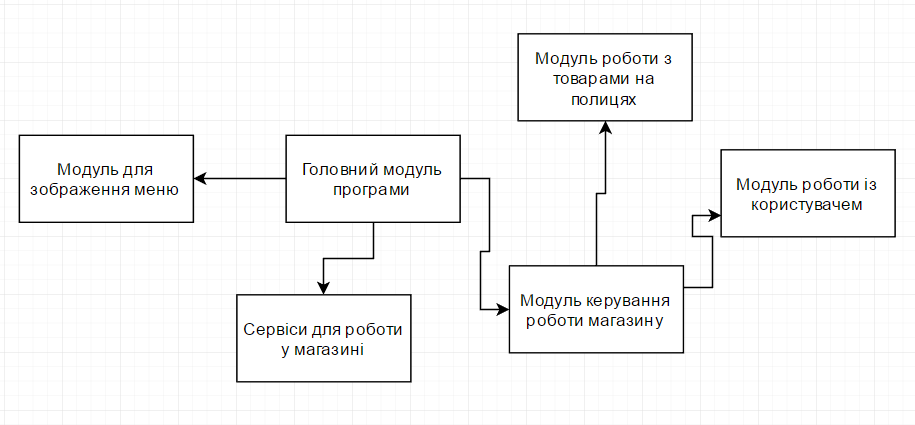
Розроблене програмне забезпечення складається з таких логічних частин: інтерфейсу користувача, створення та відображення меню основних кроків із купівлі товару покупцем, модуля, що містить інформацію про увесь магазин, касу для продажу товарів із взаємодією із покупцем та поточного покупця у магазині.

Використані шаблони проектування: **Abstract Factory, Strategy, Chain of responsibility, Proxy (**у вигляді Protection Proxy**), Bridge,** **Facade, Decorator, Singleton.**

Розроблене програмне забезпечення може бути використане у окремій частині бізнес-сфері, 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. **Модульна організація програми**



*Рис. 1.1.1. Модульна організація програми*

**Головний модуль системи** – модуль Shop (магазин), який містить у собі основні сутності: товари, касу, сервіси. Встановлює взаємодію з модулями покупця, полиць, каси та сервісів.

**Модуль керування роботи магазину для покупця** – модуль консольного терміналу (ConsoleUI), який містить основну логіку роботи магазину, кас, сервісів.

**Модуль роботи з товарами на полицях** – модуль, який відповідає за наповнення полиць магазину товарів правильними товарами.

**Модуль роботи з користувачем та кошиком користувача** – який відповідає за функції роботи безпосередньо з покупцем та із самим кошиком товарів користувача.

**Модуль відображення меню –** модуль, який містить інформацію про поточне меню магазину із точки зору покупця, а також займається відображенням цього меню (робота з інтерфейсом).

**Сервіси для роботи в магазині** – модуль, який містить у собі багато логіки для обробки замовлення покупця: обробка сервісу доставки, цін зі знижкою, цін при укладані особливого договору магазином із компанією-постачальником.

* 1. **Функціональні характеристики**

Робота з програмою розпочинається з *Main Menu* (головного меню). Користувачу пропонується вибрати певні товари до свого кошика. На початку користувач має 100000 ум.од., що є достатнім для купівлі товарів у магазині.  
На головному меню зображений поточний час для регулювання роботи робочих змін.

З головного меню користувач може потрапити до *Cash Register Menu –* меню каси. У цьому меню користувач може побачити які товари йому вдалось купити, а на які не вистачило грошей. Також на цьому меню користувач бачить, скільки у нього грошей було з самого початку, баланс накопичення для вираховування дисконтної системи та ставку поточної дисконтної картки. При купівлі товарів, поповнивши баланс накопичення до певної суми, покупцю видаються повідомлення про нові дисконтні картки, що збільшує економію покупця при наступних покупках.  
На кінець користувачу пропонується вибрати спосіб доставки купленої техніки: різні комбінації таких способів: на літаку, поїздом, пішим кур’єром.  
Користувач може відмовитись від доставки.

Із меню *Cash Register Menu* користувач програми попадає на останнє меню – меню стану після покупки. У цьому меню відображається стан користувача на момент після покупки та також відображається у якій послідовності були відправлені товари, замовлені користувачем.

* 1. **Опис реалізованих класів**

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

|  |
| --- |
| **Customer.cs** |
| class Customer  {  private CustomerPayment paymentType;  public List<Product> cart;  public String Name { get; set; }  public String Address { get; set; }  public uint Money { get; set; }  public int discountCardValue;  public uint boughtInTotal;  public Customer(String name, uint money)  {  this.paymentType = new CustomerPayment(new CashPayment());  this.cart = new List<Product>();  this.Name = name;  this.Money = money;  this.boughtInTotal = 0;  }  public void SetUpCustomer(String address, int discCardValue = 0)  {  this.Address = address;  this.discountCardValue = discCardValue;  if (discountCardValue != 0)  {  this.paymentType = new CustomerPayment(new CardPayment());  }  else if (discountCardValue == 0 && this.Money > 0)  {  this.paymentType = new CustomerPayment(new CashPayment());  }  }  public void AddToCart(Product product)  {  this.cart.Add(product);  Console.WriteLine("A {0} added to your cart.", product.name);  }  public void PayForProduct(Product productToBuy, uint discountValue)  {  uint toPayForProduct = this.paymentType.PayForProduct(productToBuy.price - discountValue);    if (toPayForProduct <= this.Money)  {  this.Money -= toPayForProduct;  this.boughtInTotal += toPayForProduct;  this.cart.Remove(productToBuy);  Console.WriteLine("Bought a {0} for {1}.", productToBuy.name, toPayForProduct);  }  else  {  Console.WriteLine("Sorry, not enough for {0}", productToBuy.name);  }  }  } |

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

Разом ізкласом **CustomerPayment**  у файлі CustomerPayment.cs знаходяться файли, які використовуються при побудові паттерну «міст»: **ICustomerPayForProduct, CardPayment, CashPayment.**

|  |
| --- |
| **CustomerPayment.cs** |
| // Implementor for a "bridge" design pattern.  class CustomerPayment  {  ICustomerPayForProduct paymentType;  public CustomerPayment(ICustomerPayForProduct paymentType)  {  this.paymentType = paymentType;  }  public void SetPaymentType(ICustomerPayForProduct paymentType)  {  if (paymentType == null)  return;  this.paymentType = paymentType;  }  public uint PayForProduct(uint toPay)  {  return this.paymentType.PayForProduct(toPay);  }  }  // Abstraction.  interface ICustomerPayForProduct  {  // return how much payed.  uint PayForProduct(uint toPay);  }  // First concrete abstraction - paying with credit card.  class CardPayment : ICustomerPayForProduct  {  // Get a 10% cashback as a card credit card owner.  public uint PayForProduct(uint toPay)  {  return (uint)(toPay \* 0.9);  }  }  // Second concrete abstraction - paying with cash.  class CashPayment : ICustomerPayForProduct  {  // Regular payment without cashback.  public uint PayForProduct(uint toPay)  {  return toPay;  }  } |

**3. DeliveryService –** абстрактний клас для основи паттерну «Декоратор». Визначає основні методи для модулю, який працює із регулюванням способів доставки придбаного покупцем товару.

Разом із класом **DeliveryService** у файлі DeliveryService.cs знаходяться клас **BasicDeliveryService,** який теж приймає участь у формуванні паттерну «Декоратор».

|  |
| --- |
| **DeliveryService.cs** |
| abstract class DeliveryService  {  public abstract void DeliverProduct(Customer customer, Product product);  public abstract void ShowDeliveryInfo();  }  // "Default" delivery implementation.  class BasicDeliveryService : DeliveryService  {  public override void DeliverProduct(Customer customer, Product product)  {  Console.WriteLine("Trying to send {0} to {1}.", product.name, customer.Name);  this.ShowDeliveryInfo();  }  public override void ShowDeliveryInfo()  {  }  } |

**4. Decorator** із DeliveryServiceDecorators.cs – предтавляє абрстрактний класс декоратору – основного для паттерну «декоратор». Також у цьому файлі існують інші «конкретні декоратори» - класи для відображення способу доставки: пішим кур’єром, потягом, чи літаком – **ByFootDelivery, ByTrainDevliery, ByAirDelivery**.

|  |
| --- |
| **DeliveryServiceDecorators.cs** |
| // Main abstract decorator class.  abstract class Decorator : DeliveryService  {  protected DeliveryService deliveryService;  public void AddShippingRoute(DeliveryService service)  {  this.deliveryService = service;  }  public override void DeliverProduct(Customer customer, Product product)  {  if (deliveryService != null)  {  deliveryService.DeliverProduct(customer, product);  }  }  public override void ShowDeliveryInfo()  {  if (deliveryService != null)  {  deliveryService.ShowDeliveryInfo();  }  }  }  // Example of concrete decorator  class ByAirDelivery : Decorator  {  public override void DeliverProduct(Customer customer, Product product)  {  Console.WriteLine("{0} air delivery order was sent to {1} by address {2}.", product.name, customer.Name, customer.Address);  base.DeliverProduct(customer, product);  }  public override void ShowDeliveryInfo()  {  Console.WriteLine("Shipping by air");  base.ShowDeliveryInfo();  }  }  class ByTrainDelivery : Decorator  {  public override void DeliverProduct(Customer customer, Product product)  {  Console.WriteLine("{0} train delivery order was sent to {1} by address {2}.", product.name, customer.Name, customer.Address);  base.DeliverProduct(customer, product);  }  public override void ShowDeliveryInfo()  {  Console.WriteLine("Shipping by train");  base.ShowDeliveryInfo();  }  }  class ByFootDelivery : Decorator  {  public override void DeliverProduct(Customer customer, Product product)  {  Console.WriteLine("{0} by foot delivery order was sent to {1} by address {2}.", product.name, customer.Name, customer.Address);  base.DeliverProduct(customer, product);  }  public override void ShowDeliveryInfo()  {  Console.WriteLine("Shipping by foot");  base.ShowDeliveryInfo();  } } |

**5. ShippingManager –** клас-менеджер для роботи із вхідними данними користувача із клавіатури з приводу формування власної послідовності доставки придбаного товару. Містить у собі використання паттерну «декоратор» - методи \*\*Shipping.  
Також у файлі ShippingManager.cs представлені допоміжні константи для вибору режиму доставки у класі **ShippingTypes.**

|  |
| --- |
| **ShippingManager.cs** |
| class ShippingManager  {  private DeliveryService \_deliveryService;  private Customer \_customerTmpObj;  private Product \_productTmpObj;  private Decorator footShipping;  private Decorator trainShipping;  private Decorator airShipping;  public ShippingManager()  {  \_deliveryService = new BasicDeliveryService();  footShipping = new ByFootDelivery();  trainShipping = new ByTrainDelivery();  airShipping = new ByAirDelivery();  }  public void ShipProduct(Product product, Customer customer, String shippingOption)  {  \_productTmpObj = product;  \_customerTmpObj = customer;  switch (shippingOption)  {  case ShippingTypes.ByFoot:  ByFootShipping();  break;  case ShippingTypes.ByTrain:  TrainShipping();  break;  case ShippingTypes.ByAir:  AirShipping();  break;  case ShippingTypes.ByAirAndTrain:  AirAndTrainShipping();  break;  case ShippingTypes.ByAirAndFoot:  AirAndByFootShipping();  break;  case ShippingTypes.ByAirAndTrainAndFoot:  AirAndByTrainAndByFootShipping();  break;  default:  Console.WriteLine("No shipping chosen.");  break;  }  }  private void AirShipping()  {  airShipping.AddShippingRoute(\_deliveryService);  ShowDeliveryTracking(airShipping, \_productTmpObj, \_customerTmpObj);  }  private void TrainShipping()  {  trainShipping.AddShippingRoute(\_deliveryService);  ShowDeliveryTracking(trainShipping, \_productTmpObj, \_customerTmpObj);  }  private void ByFootShipping()  {  footShipping.AddShippingRoute(\_deliveryService);  ShowDeliveryTracking(footShipping, \_productTmpObj, \_customerTmpObj);  }  private void AirAndTrainShipping()  {  trainShipping.AddShippingRoute(\_deliveryService);  airShipping.AddShippingRoute(trainShipping);  ShowDeliveryTracking(airShipping, \_productTmpObj, \_customerTmpObj);  }  private void AirAndByFootShipping()  {  footShipping.AddShippingRoute(\_deliveryService);  airShipping.AddShippingRoute(footShipping);  ShowDeliveryTracking(airShipping, \_productTmpObj, \_customerTmpObj);  }  private void AirAndByTrainAndByFootShipping()  {  footShipping.AddShippingRoute(\_deliveryService);  trainShipping.AddShippingRoute(footShipping);  airShipping.AddShippingRoute(trainShipping);  ShowDeliveryTracking(airShipping, \_productTmpObj, \_customerTmpObj);  }  private void ShowDeliveryTracking(Decorator lastDecoratorInChain, Product product, Customer customer)  {  lastDecoratorInChain.DeliverProduct(customer, product);  Console.WriteLine("Tracking delivery of {0} to {1}:", product.name, customer.Name);  lastDecoratorInChain.ShowDeliveryInfo();  }  } |

**6. PriceFormerStrategy** – клас для вибору стратегій із вираховування податків від компаній Samsung та Tochiba. У файлі також містяться основні складові паттерну «Стратегія» - інтерфейс для «конкретних» стратегій **IPriceFormerStrategy,** «конкретні» стратегії **SamsungProductPriceStrategy** i **TochibaProductPriceStrategy.**

|  |
| --- |
| **PriceFormerStrategy.cs** |
| public enum UkrainianCustomTaxes  {  Prepayed = 0,  Internal = 150,  FarEast = 250  }  public enum UkrainianGeneralTaxes  {  Prepayed = 0,  Basic = 250,  Complicated = 285  }  public interface IPriceFormerStrategy  {  uint GenerateProductPrice(uint basePrice);  }  public class SamsungProductPriceStrategy : IPriceFormerStrategy  {  public uint GenerateProductPrice(uint basePrice)  {  uint customsExpenses = (uint)UkrainianCustomTaxes.Prepayed;  uint taxesExpenses = (uint)UkrainianGeneralTaxes.Prepayed;  return basePrice + taxesExpenses + customsExpenses;  }  }  public class TochibaProductPriceStrategy : IPriceFormerStrategy  {  private const double taxesCoeficient = 0.7;  public uint GenerateProductPrice(uint basePrice)  {  uint customsExpenses = (uint)UkrainianCustomTaxes.FarEast;  uint taxesExpenses = (uint)UkrainianGeneralTaxes.Complicated;  double specialTax = (customsExpenses + taxesExpenses) \* taxesCoeficient;  return basePrice + taxesExpenses + customsExpenses + (uint)specialTax;  }  } |

**7. ProductPriceFormer** – клас-клієнт для паттерну «стратегія». Містить методи для запам’ятовування стратегії, необхідної для виконання, та основний керуючий метод поточної стратегії у даному класі. Є основним у паттерні «стратегія».

|  |
| --- |
| **ProductPriceFormer.cs** |
| class ProductPriceFormer  {  private IPriceFormerStrategy \_manufacturerPriceStrategy;  public void SetProductPriceStrategy(IPriceFormerStrategy manufacturerStrategy)  {  if (manufacturerStrategy == null)  return;  \_manufacturerPriceStrategy = manufacturerStrategy;  }  public uint GetProductPrice(uint productBasePrice)  {  return \_manufacturerPriceStrategy.GenerateProductPrice(productBasePrice);  }  } |

**8. Product –** основний абстрактний клас у проекті, відображає сутність, яку виробляють фабрики Samsung і Tochiba, сутності, які лежать на полицях у магазині, сутності, які купляє покупець магазину.  
Визначає через наслідування основні чотири товари, які продаються в магазині: холодильники **Fridge**, пральні машини **WashingMachine**, машини для миття посуду **DishWasher** та телевізори **TVSet**.  
Є одним із основних у паттерні «Абстрактна фабрика»

|  |
| --- |
| **Product.cs** |
| public enum ProductColor  {  Red = 1,  Green,  Blue,  Black,  White  }  public abstract class Product  {  public String name;  public String category;  public String manufacturer;  public ProductColor color;  public uint price;  public override string ToString()  {  return String.Format("{0}; {1}UAH", name, price);  }  }  public abstract class WashingMachine : Product  {  public int rpmNumber;  }  public abstract class Fridge : Product  {  public bool hasFreezer;  }  public abstract class DishWasher : Product  {  public int capacityInKg;  }  public abstract class TVSet : Product  {  public double screenDiagonal;  } |

**9. ProductManufacturer** – відображає паттерн «абстратктна фабрика» для створення товарів від виробників Samsung і Tochiba. Є абстрактним класом.

|  |
| --- |
| **ProductFactory.cs** |
| abstract class ProductManufacturer  {  public List<ProductColor> factoryProductsColors;  public String name;  protected ProductPriceFormer priceFormer = new ProductPriceFormer();  protected Random rand = new Random();  public abstract WashingMachine CreateWashingMachine(String name);  public abstract Fridge CreateFridge(String name);  public abstract DishWasher CreateDishWasher(String name);  public abstract TVSet CreateTVSet(String name);  } |

**10. SamsungProductManufacturer –** відображає виробника товарів марки Samsung, більш дорогих, але які мають менші податки при врахуванні остаточної ціни на полиці магазину. Є конкретною реалізацією абстрактної фабрики.

|  |
| --- |
| **ProductFactory.cs** |
| class SamsungProductManufacturer : ProductManufacturer  {  public SamsungProductManufacturer()  {  this.factoryProductsColors = new List<ProductColor>()  {  ProductColor.Green,  ProductColor.Red,  ProductColor.Blue  };  this.name = "Samsung";  this.priceFormer.SetProductPriceStrategy(new SamsungProductPriceStrategy());  }  public override WashingMachine CreateWashingMachine(String name)  {  return new SamsungWashingMachine(  this.priceFormer.GetProductPrice((uint)rand.Next(1000, 2000)),  this.factoryProductsColors[rand.Next(3)],  name,  rand.Next(200, 500)  );  }  public override Fridge CreateFridge(String name)  {  return new SamsungFridge(  this.priceFormer.GetProductPrice((uint)rand.Next(2000, 3000)),  this.factoryProductsColors[rand.Next(3)],  name,  true  );  }  public override DishWasher CreateDishWasher(String name)  {  return new SamsungDishWasher(  this.priceFormer.GetProductPrice((uint)rand.Next(1500, 2500)),  this.factoryProductsColors[rand.Next(3)],  name,  rand.Next(10, 80)  );  }  public override TVSet CreateTVSet(String name)  {  return new SamsungTVSet(  this.priceFormer.GetProductPrice((uint)rand.Next(10000, 20000)),  this.factoryProductsColors[rand.Next(3)],  name,  rand.NextDouble() \* 10  );  }  } |

**11. TochibaProductManufacturer.** Відображає фабрику, яка створює товари марки Tochiba – дешеві, але які стають значно дорожчими при ввезенні до країни через податки. Є конкретною реалізацією паттерну «Абстрактна фабрика»

|  |
| --- |
| **ProductFactory.cs** |
| class TochibaProductManufacturer : ProductManufacturer  {  public TochibaProductManufacturer()  {  this.factoryProductsColors = new List<ProductColor>()  {  ProductColor.Black,  ProductColor.White  };  this.name = "Tochiba";  this.priceFormer.SetProductPriceStrategy(new TochibaProductPriceStrategy());  }  public override WashingMachine CreateWashingMachine(String name)  {  return new TochibaWashingMachine(  this.priceFormer.GetProductPrice((uint)rand.Next(1800, 2800)),  this.factoryProductsColors[rand.Next(2)],  name,  rand.Next(200, 500)  );  }  public override Fridge CreateFridge(String name)  {  return new TochibaFridge(  this.priceFormer.GetProductPrice((uint)rand.Next(1200, 1750)),  this.factoryProductsColors[rand.Next(2)],  name,  false  );  }  public override DishWasher CreateDishWasher(String name)  {  return new TochibaDishWasher(  this.priceFormer.GetProductPrice((uint)rand.Next(600, 1500)),  this.factoryProductsColors[rand.Next(2)],  name,  rand.Next(10, 80)  );  }  public override TVSet CreateTVSet(String name)  {  return new TochibaTVSet(  this.priceFormer.GetProductPrice((uint)rand.Next(9000, 14000)),  this.factoryProductsColors[rand.Next(2)],  name,  rand.NextDouble() \* 10  );  }  } |

**12. SamsungProducts** – відображає файл для класів для вироблення товарів марки Samsung – **SamsungWashingMachine, SamsungFridge, SamsungDishWasher, SamsungTVSet**. Використовується у паттерні «абстрактная фабрика».

|  |
| --- |
| **SamsungProducts.cs** |
| class SamsungWashingMachine : WashingMachine  {  public SamsungWashingMachine(uint price, ProductColor color, String name, int rpmNum)  {  this.name = name;  this.category = "Washing machine";  this.color = color;  this.price = price;  this.rpmNumber = rpmNum;  this.manufacturer = "Samsung";  }  }  class SamsungFridge : Fridge  {  public SamsungFridge(uint price, ProductColor color, String name, bool hasFreezer)  {  this.name = name;  this.category = "Fridge";  this.color = color;  this.price = price;  this.hasFreezer = hasFreezer;  this.manufacturer = "Samsung";  }  }  class SamsungDishWasher : DishWasher  {  public SamsungDishWasher(uint price, ProductColor color, String name, int capacityInKg)  {  this.name = name;  this.category = "Dish washer";  this.color = color;  this.price = price;  this.capacityInKg = capacityInKg;  this.manufacturer = "Samsung";  }  }  class SamsungTVSet : TVSet  {  public SamsungTVSet(uint price, ProductColor color, String name, double diagonal)  {  this.name = name;  this.category = "TV set";  this.color = color;  this.price = price;  this.screenDiagonal = diagonal;  this.manufacturer = "Samsung";  }  } |

**13. TochibaProducts** – відображає файл для класів для вироблення товарів марки Tochiba – **TochibaWashingMachine, SamsungFridge, SamsungDishWasher, SamsungTVSet**.Використовується у паттерні «абстрактная фабрика».

|  |
| --- |
| **TochibaProducts.cs** |
| class TochibaWashingMachine : WashingMachine  {  public TochibaWashingMachine(uint price, ProductColor color, String name, int rpmNum)  {  this.name = name;  this.category = "Washing machine";  this.color = color;  this.price = price;  this.rpmNumber = rpmNum;  this.manufacturer = "Tochiba";  }  }  class TochibaFridge : Fridge  {  public TochibaFridge(uint price, ProductColor color, String name, bool hasFreezer)  {  this.name = name;  this.category = "Fridge";  this.color = color;  this.price = price;  this.hasFreezer = hasFreezer;  this.manufacturer = "Tochiba";  }  }  class TochibaDishWasher : DishWasher  {  public TochibaDishWasher(uint price, ProductColor color, String name, int capacityInKg)  {  this.name = name;  this.category = "Dish washer";  this.color = color;  this.price = price;  this.capacityInKg = capacityInKg;  this.manufacturer = "Tochiba";  }  }  class TochibaTVSet : TVSet  {  public TochibaTVSet(uint price, ProductColor color, String name, double diagonal)  {  this.name = name;  this.category = "TV set";  this.color = color;  this.price = price;  this.screenDiagonal = diagonal;  this.manufacturer = "Tochiba";  }  } |

**14. CashRegisterSeller** – основний клас у системі магазину. Відображає сутність каси із деякими утилітами – вираховування знижки, вироблення документації для продукту, розробка плану доставки придбаного покупцем товару. Продавці на цій касі працюють позмінно у чотири зміни. Містить методи роботи із клієнтом.

|  |
| --- |
| **CashRegisterSeller.cs** |
| class CashRegisterSeller  {  private Shift \_morningShift; // from 06:00-13:00  private Shift \_dayShift; // from 13:00-20:00  private Shift \_firstNightShift; // from 20:00-00:00  private Shift \_secondNightShift; // from 00:00-06:00  private DiscountCardCheckerManager \_discountCardChecker;  private ProductDocumentationFormer \_productDocsFormer;  private ShippingManager \_shippingManager;  public CashRegisterSeller()  {  \_morningShift = new SellerShift(6, 13, "Natasha");  \_dayShift = new SellerShift(13, 20, "Nikita");  \_firstNightShift = new SellerShift(20, 24, "Karina");  \_secondNightShift = new SellerShift(0, 6, "Wadim");  this.SetSchedule();  \_discountCardChecker = new DiscountCardCheckerManager();  \_productDocsFormer = new ProductDocumentationFormer();  \_shippingManager = new ShippingManager();  }  public void WorkWithClientAtTime(int time, Customer customer)  {  if (time < 0 || time > 24)  {  Console.WriteLine("Wrong time argument");  return;  }  // try to give call a certain sellers shift.  // starts from \_secondNight shift, because this is a start of the day.  Console.WriteLine(\_secondNightShift.Work(time));  // get money from a customer for each product in his cart.  Console.WriteLine("");  List<Product> customCartCopy = customer.cart.ToList();  foreach (Product itemInCart in customCartCopy)  {  Console.WriteLine("{0} is on the cash register. Price: {1}", itemInCart.name, itemInCart.price);  // Try to get a discount.  uint discountValue = (\_discountCardChecker.MakeDiscount(customer, itemInCart));  // Pay for an item from cart.  customer.PayForProduct(itemInCart, discountValue);  // Form documents package.  \_productDocsFormer.FormDocumentationPackage(itemInCart);  Console.WriteLine("");  }  // Select a shipping for all items.  Console.WriteLine("---------------------------------------------------------");  #region Shipping\_User\_Input  Console.WriteLine("Would you like to order a shipping?");  Console.WriteLine(String.Format(  "[f] - by foot\t[t] - by train\t[a] - by air\n\r[at] - by air and train\n[af] - by air and foot\n[atf] - by air, train and by foot.\n[anything else] - decline.")  );  char[] category = Console.ReadLine().ToLower().ToCharArray();  String categoryInString = new string(category);  Console.WriteLine(categoryInString);  if (categoryInString != "f" &&  categoryInString != "t" &&  categoryInString != "a" &&  categoryInString != "at" &&  categoryInString != "af" &&  categoryInString != "atf"  )  {  categoryInString = "none";  }  ConsoleUI.RestartScreen\_2();  #endregion  if (categoryInString != "none")  {  foreach (Product itemInCart in customCartCopy)  {  \_shippingManager.ShipProduct(itemInCart, customer, categoryInString);  Console.WriteLine("");  }  }  else  {  Console.WriteLine("No shipping was chosen.");  }  }  private void SetSchedule()  {  \_morningShift.NextShift = \_dayShift;  \_dayShift.NextShift = \_firstNightShift;  \_firstNightShift.NextShift = \_secondNightShift;  \_secondNightShift.NextShift = \_morningShift;  }  } |

**15. DiscountCardChecker** – клас для об’єктів, які вираховують знижку для товару, враховуючи дисконтну картку від відвідувачів магазину. Містить у собі паттерн «Proxy» для вираховування того, чи може користувач взагалі використовувати знижки – тобто чи є у нього картка, чи ні.  
Містить у собі усі складові паттерну «Proxy» - конкретний клас Proxy **DiscountCardCheckerManager;** абстрактний клас Proxy AbstractDiscountChecker та найпростіший клас, роботу якого заміняємо – **DiscountCardChecker.**

|  |
| --- |
| **DiscountCardChecker.cs** |
| // interface for a discount checker.  abstract class AbstractDiscountChecker  {  public abstract uint MakeDiscount(Customer customer, Product product);  public abstract void GiveDiscountCard(Customer customer);  }  // A real discount checker.  class DiscountCardChecker : AbstractDiscountChecker  {  public override uint MakeDiscount(Customer customer, Product product)  {  return (uint)(product.price \* customer.discountCardValue / 100);  }  public override void GiveDiscountCard(Customer customer)  {  customer.discountCardValue = (int)DiscountCardValues.FivePercentDiscount;  }  }  // A proxy class.  class DiscountCardCheckerManager : AbstractDiscountChecker  {  DiscountCardChecker cardChecker = new DiscountCardChecker();  public override uint MakeDiscount(Customer customer, Product product)  {  // If a customer does not have a discount card yet.  if (customer.discountCardValue == (int)DiscountCardValues.NoDiscountCard)  {  Console.WriteLine("You just earned a new 5% discount card. Congrats!");  // Give a new discount card to the customer.  this.GiveDiscountCard(customer);  // Get a discount with new discount card value.  return cardChecker.MakeDiscount(customer, product);  }  else // if a customer already has a discount card.  {  // try to give a discount card.  this.GiveDiscountCard(customer);  return cardChecker.MakeDiscount(customer, product);  }  }  public override void GiveDiscountCard(Customer customer)  {  switch (customer.discountCardValue)  {  case (int)DiscountCardValues.NoDiscountCard:  customer.discountCardValue = (int)DiscountCardValues.FivePercentDiscount;  break;  case (int)DiscountCardValues.FivePercentDiscount:  if (customer.boughtInTotal >= 5000)  {  customer.discountCardValue = (int)DiscountCardValues.TenPercentDiscount;  Console.WriteLine("Congrats, have a 10% discount card!");  }  break;  case (int)DiscountCardValues.TenPercentDiscount:  if (customer.boughtInTotal >= 10000)  {  customer.discountCardValue = (int)DiscountCardValues.FifteenPercentDiscount;  Console.WriteLine("Congrats, have a 15% discount card!");  }  break;  }  }  } |

**16. ProductDocumentationFormer** – клас для формування пакету документів при купівлі товара покупцем. Є основним у паттерні «фасад».  
Також поруч із ним лежать конкретні представники фасаду – класи **MainProductDocumentation, WarrantyCard, SaleContract.**

|  |
| --- |
| **ProductDocumentationFormer.cs** |
| // "Facade" design pattern.  class ProductDocumentationFormer  {  MainProductDocumentation mainDoc;  WarrantyCard warrantyCard;  SaleContract saleContract;  public ProductDocumentationFormer()  {  this.mainDoc = new MainProductDocumentation();  this.warrantyCard = new WarrantyCard();  this.saleContract = new SaleContract();  }  public List<String> FormDocumentationPackage(Product product)  {  Console.WriteLine("Docs for {0} are formed.", product.name);  return new List<string>() { mainDoc.ToString(), warrantyCard.ToString(), saleContract.ToString() };  }  }  // Abstract class for facade subclasses.  abstract class ProductDocument  {  public string Name;  public override string ToString()  {  return String.Format("{0} document", Name);  }  }  // Example of facade subclass.  class MainProductDocumentation : ProductDocument  {  public MainProductDocumentation()  {  this.Name = "Main product documentation.";  }  }  class WarrantyCard : ProductDocument  {  public WarrantyCard()  {  this.Name = "Warranty card";  }  }  class SaleContract : ProductDocument  {  public SaleContract()  {  this.Name = "Sale contract";  }  } |

**17. Shift** – абстрактний клас, який відображає сутність зміни продавців на касі. Є основним у паттерні «Ланцюжок обов’язків». Від нього наслідуються конкретні обробники певних станів – тут, це змін продавців – класи **SellerShift,** із методами, які дозволяють утворити певний зацикленний «ланцюжок».

|  |
| --- |
| **Shift.cs** |
| // Abstract handler  abstract class Shift  {  protected int \_startWrkTime;  protected int \_endWrkTime;  public String sellerName;  // Successor  public Shift NextShift { get; set; }  public abstract String Work(int currentTime);  }  // Concrete handler.  class SellerShift : Shift  {  public SellerShift(int startWrkTime, int endWrkTime, String sellerName)  {  \_startWrkTime = startWrkTime;  \_endWrkTime = endWrkTime;  this.sellerName = sellerName;  if (startWrkTime > endWrkTime)  {  \_startWrkTime = \_endWrkTime = endWrkTime;  }  }  public override String Work(int currentTime)  {  if (currentTime >= \_startWrkTime && currentTime <= \_endWrkTime)  {  return String.Format("Hello, I'm a seller {0}, what do you have in your cart?", this.sellerName);  }  else if (NextShift != null)  {  return NextShift.Work(currentTime);  }  return "No one works right now. This is an error"; // todo: debug?  }  public void SetNextShift(Shift nextShift)  {  if (nextShift == null)  return;  NextShift = nextShift;  }  } |

**18. Shop** – основна сутність програми. Відображає взаємодію основної вітрини магазину, представляє сам магазин, містить у собі усі додаткові сервіси, користувачів, продавців, касу та інші важливі компоненти програми.  
Використовується, в основному, у класах із **ConsoleUI.**

|  |
| --- |
| **Shop.cs** |
| class Shop  {  public Showcase mainShowCase;  private CashRegisterSeller \_cashRegisterSeller;  public Shop()  {  mainShowCase = Showcase.Instance();  \_cashRegisterSeller = new CashRegisterSeller();  }  public void WorkAtCashRegister(int currentTime, Customer customer)  {  \_cashRegisterSeller.WorkWithClientAtTime(currentTime, customer);  }  public void PrintShowcaseItems()  {  int ctr = 0;  Console.WriteLine("[W]ashing machines:");  foreach (Product item in mainShowCase.washingMachineStand)  {  Console.WriteLine(String.Format("\t[{0}] {1};", ctr++.ToString(), item.ToString()));  }  ctr = 0;  Console.WriteLine("[F]ridges:");  foreach (Product item in mainShowCase.fridgeStand)  {  Console.WriteLine(String.Format("\t[{0}] {1};", ctr++.ToString(), item.ToString()));  }  ctr = 0;  Console.WriteLine("[D]ish washers:");  foreach (Product item in mainShowCase.dishWasherStand)  {  Console.WriteLine(String.Format("\t[{0}] {1};", ctr++.ToString(), item.ToString()));  }  ctr = 0;  Console.WriteLine("[T]V sets:");  foreach (Product item in mainShowCase.tvSetStand)  {  Console.WriteLine(String.Format("\t[{0}] {1};", ctr++.ToString(), item.ToString()));  }  }  } |

**19. Showcase** – відображає сутність вітрини у магазині, на якій лежать товари для продажу та з яких покупець може взяти і купити товар.  
Є паттерном-«одинаком» у своїй реалізації. Має методи заповнення полиць товарами від виробників Samsung та Tochiba.

|  |
| --- |
| **Showcase.cs** |
| public class Showcase  {  // Showcase as Singleton instance.  static Showcase uniqueInstance;  // Showcase Data.  public List<WashingMachine> washingMachineStand;  public List<Fridge> fridgeStand;  public List<DishWasher> dishWasherStand;  public List<TVSet> tvSetStand;  // Public method to get instance of the showcase.  public static Showcase Instance()  {  if (uniqueInstance == null)  uniqueInstance = new Showcase();  return uniqueInstance;  }  // Private constructor for singleton pattern.  private Showcase()  {  this.washingMachineStand = new List<WashingMachine>();  this.fridgeStand = new List<Fridge>();  this.dishWasherStand = new List<DishWasher>();  this.tvSetStand = new List<TVSet>();    this.FillStandWithSamsungProducts();  this.FillStandWithTochibaProducts();  }  private void FillStand(ProductManufacturer manufacturer)  {  washingMachineStand.Add(manufacturer.CreateWashingMachine(manufacturer.name + " Washing Machine"));  fridgeStand.Add(manufacturer.CreateFridge(manufacturer.name + " Fridge"));  dishWasherStand.Add(manufacturer.CreateDishWasher(manufacturer.name + " Dish Washer"));  tvSetStand.Add(manufacturer.CreateTVSet(manufacturer.name + " TV set"));  }  private void FillStandWithSamsungProducts()  {  ProductManufacturer samsung = new SamsungProductManufacturer();  this.FillStand(samsung);  }  private void FillStandWithTochibaProducts()  {  ProductManufacturer tochiba = new TochibaProductManufacturer();  this.FillStand(tochiba);  }  } |

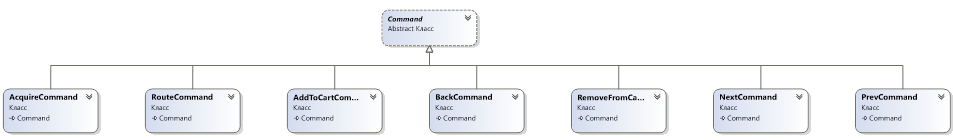
**20. ConsoleUI** – клас для взаємодії із користувачем та для показання злагодженної роботи усіх інших модулів програми.

|  |
| --- |
| **ConsoleUI.cs** |
| static class ConsoleUI  {  private static Shop \_shopInst;  private static Customer \_custInst;  private static int \_currentTime;    public static void SetMainInstances(Shop shopInst, Customer custInst)  {  \_shopInst = shopInst;  \_custInst = custInst;  }  public static void MainLoop()  {  PrintMainShowcase();  // Add items into your cart.  while (true)  {  Product productToAddToCart = null;  // Get a category.  Console.WriteLine("Choose category.");  char category = Console.ReadLine().ToLower().ToCharArray()[0];  while (category != 'w' && category != 'f' && category != 'd' && category != 't')  {  if (category == '>')  {  break;  }  RestartScreen();  Console.WriteLine("Choose category. Please, choose one from above.");  category = Console.ReadLine().ToLower().ToCharArray()[0];  }  if (category == '>')  {  break;  }  RestartScreen();  // Choose a product.  Console.WriteLine("Choose a product.");  char productId = Console.ReadLine().ToCharArray()[0];  while (productId != '0' && productId != '1')  {  if (productId == '>')  {  break;  }  RestartScreen();  Console.WriteLine("Choose a product. Please, choose one from above.");  category = Console.ReadLine().ToCharArray()[0];  }  if (productId == '>')  {  break;  }  RestartScreen();  // Choose an item to your cart.  switch (category)  {  case 'w':  switch (productId)  {  case '0':  productToAddToCart = \_shopInst.mainShowCase.washingMachineStand[0];  break;  case '1':  productToAddToCart = \_shopInst.mainShowCase.washingMachineStand[1];  break;  }  break;  case 'f':  switch (productId)  {  case '0':  productToAddToCart = \_shopInst.mainShowCase.fridgeStand[0];  break;  case '1':  productToAddToCart = \_shopInst.mainShowCase.fridgeStand[1];  break;  }  break;  case 'd':  switch (productId)  {  case '0':  productToAddToCart = \_shopInst.mainShowCase.dishWasherStand[0];  break;  case '1':  productToAddToCart = \_shopInst.mainShowCase.dishWasherStand[1];  break;  }  break;  case 't':  switch (productId)  {  case '0':  productToAddToCart = \_shopInst.mainShowCase.tvSetStand[0];  break;  case '1':  productToAddToCart = \_shopInst.mainShowCase.tvSetStand[1];  break;  }  break;  }  // Add an item.  \_custInst.AddToCart(productToAddToCart);  // Restart a screen at the end.  RestartScreen();  }  RestartScreen\_2();  // Buy items.  \_shopInst.WorkAtCashRegister(\_currentTime, \_custInst);  // End of the buying routine.  Console.WriteLine("End of the buying cycle.");  }  #region Console\_UI\_1  public static void PrintMainShowcase()  {  // todo: current time for shop.WorkAtCashRegister  Console.WriteLine("----------------------------------------------------------");  Console.WriteLine("Items in shop:");  \_shopInst.PrintShowcaseItems();  Console.WriteLine("----------------------------------------------------------");  Console.WriteLine("Current time: {0}", InUITime());  Console.WriteLine("----------------------------------------------------------");  Console.WriteLine("Press '>' and 'enter' to pay for products in the cart");  Console.WriteLine("----------------------------------------------------------");  }  public static void RestartScreen()  {  Console.Clear();  PrintMainShowcase();  }  private static int InUITime()  {  return (\_currentTime++ % 24) + 1;  }  #endregion  #region Console\_UI\_2  public static void PrintCustomerCart()  {  Console.WriteLine("----------------------------------------------------------");  Console.WriteLine("{0}. {1}UAH. Total bought: {2}UAH. Discount: {3}%",  \_custInst.Name, \_custInst.Money, \_custInst.boughtInTotal, \_custInst.discountCardValue);  Console.WriteLine("----------------------------------------------------------");  Console.WriteLine("{0} cart:", \_custInst.Name);  foreach (Product item in \_custInst.cart)  {  Console.WriteLine(item);  }  Console.WriteLine("----------------------------------------------------------");  Console.WriteLine("Current time: {0}", InUITime());  Console.WriteLine("----------------------------------------------------------");  }  public static void RestartScreen\_2()  {  Console.Clear();  PrintCustomerCart();  }  #endregion  } |

**21. Program** – клас для запуску мейну – тобто, ініціалізації сутностей магазину та одного користувача, а потім – пускання користувача до придбання товарів.

|  |
| --- |
| **Program.cs** |
| class Program  {  static void Main(string[] args)  {  Shop shop = new Shop();  Customer customer = new Customer("Maxim", 100000);  customer.SetUpCustomer("KPI"); // lives in KPI, no discount value.  ConsoleUI.SetMainInstances(shop, customer);  ConsoleUI.MainLoop();  Console.ReadKey();  }  } |

1. **ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗА ДОПОМОГОЮ ШАБЛОНІВ ПРОЕКТУВАННЯ**
   1. **Обґрунтування вибору та опис шаблонів проектування для реалізації програмного забезпечення автомату**
2. **Command**



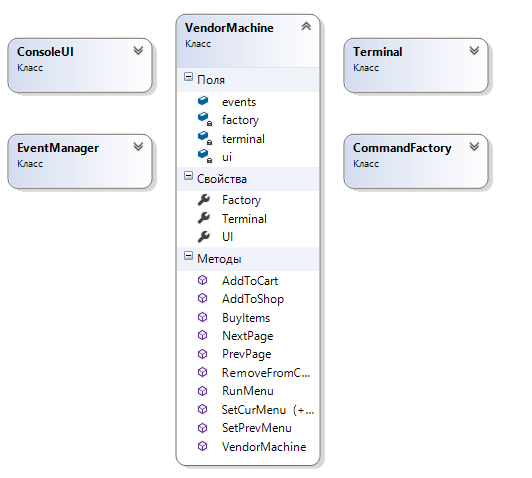
*Рис. 2.1.1. UML-діаграма шаблону* ***Command***

*Поведінковий шаблон*. Перетворює операцію в об'єкт. Це дозволяє передавати операції як аргументи при викликах методів, ставити операції в чергу, логувати їх, а також підтримувати можливість скасування операцій.

*Структура*.Реалізований для класу **MenuOption**. Об’єкти цього класу мають у середині об’єкт команди, і цей об’єкт слугує логічною частиною опції меню. Інша частина опції – інтерфейс – строка, яка виводиться користувачу і з якою і пов’язана логіка цієї команди.

*Обґрунтування використання даного шаблону*. При написанні коду завжди слід прагнути розділення інтерфейсу та логіки. За допомогою формування такого класу і можна досягти інкапсуляції логіки в одному об’єкті.

1. **Façade**



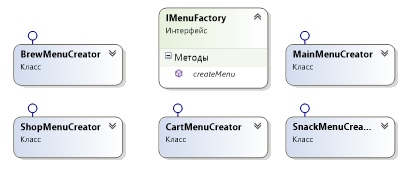
*Рис. 2.1.2. UML-діаграма шаблону “****Façade****”*

*Структурний шаблон*. Надає єдиний простий інтерфейс до безлічі інших інтерфейсів в деякій складній підсистемі.

*Структура*.Об’єкт, який включає в себе інші об’єкти: представників класів **Terminal**, **ConsoleUI**, **CommandFactory**, **EventManager**. **Terminal –** логіка роботи автомату, **ConsoleUI –** користувацький інтерфейс, **CommandFactory** – фабрика з виробництва **Command** (команд), **EventManager** – сутність з управління подіями у торгівельному автоматі.

*Обґрунтування використання даного шаблону*. Реалізований для того, щоб мати один об’єкт, через який можна безпечно та зручно взаємодіяти зі всією системою.

1. **Factory Method**



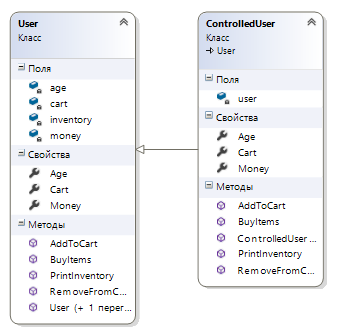
*Рис. 2.1.3. UML-діаграма шаблону “****Factory Method****”*

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

*Структура*. **IMenuFactory** – головний інтерфейс даного шаблону. Має у середині один-єдиний метод: створення меню. Цей інтерфейс реалізовують п’ять інших класів, представлених на скріншоті зверху. Ці п’ять класів і представляють весь спектр наявних у програмі меню.

*Обґрунтування використання даного шаблону*. За допомогою даного шаблона можна організувати гнучку реалізацію різновидів певного класу, і в даній програмі така реалізація потребувалася для створення меню.

1. **Proxy**



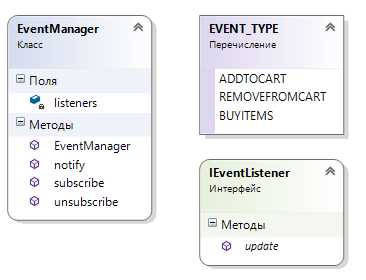
*Рис. 2.1.4. UML-діаграма шаблону “****Proxy****”*

*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*. **ControlledUser** вміщає у собі об’єкт класу **User**. Усі методи, невизначені для контролювання у цьому шаблоні, просто переадресуються об’єкту основного класу. Інші методи (а саме метод *AddToCart*), мають у середині додаткову перевірку перед виповненням основної логіки, і вона буде виповнена тільки тоді, коли **ControlledUser** проведе необхідні перевірки.

*Обґрунтування використання даного шаблону*.У торгівельному автоматі серед асортименту існують напої з маркуванням мінімального віку, необхідного для придбання такого продукту. Щоб мати змогу проконтролювати вік користувача, слід реалізувати структуру, описану у даному шаблоні.

1. **Observer**



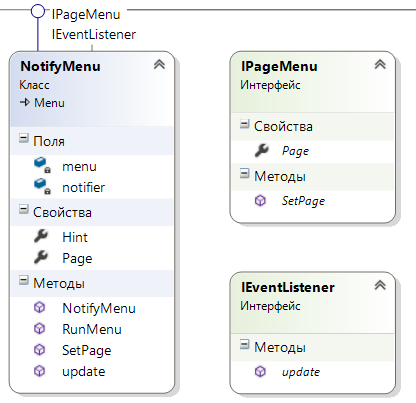
*Рис. 2.1.5. UML-діаграма шаблону “****Observer****”*

*Поведінковий шаблон*. Створює механізм підписки, за допомогою якого одні об'єкти можуть підписуватися на оновлення, що відбуваються в інших об'єктах.

*Структура*. **EventManager** – основний компонент системи, який має у середині структуру даних: підписників на деякі події, описані у перерахуванні **EVENT\_TYPE**. Також, у середині реалізовані стандартні методи, визначены даним шаблоном: підписку, відписку та оповіщення підписників. **IEventListener** – інтерфейс, реалізація якого необхідна у підписників, щоб мати змогу реагувати на відбуття певних подій.

*Обґрунтування використання даного шаблону*.При виконанні логіки, зв’язаної з певними опціями меню, відбуваються події, про які слід сповістити користувача системи. Даний шаблон дозволяє реалізувати таку можливість сповіщень.

1. **Decorator**



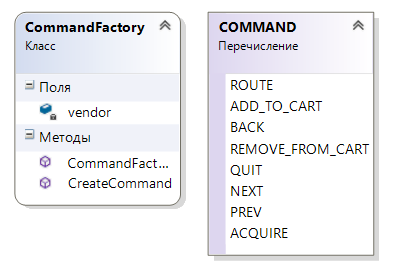
*Рис. 2.1.6. UML-діаграма шаблону “****Decorator****”*

*Структурний шаблон*. Динамічно додає об'єктам нову функціональність, обертаючи їх в корисні “обгортки”.

*Структура*. **NotifyMenu** – клас-обгортка для класу **Menu**, який водночас є представником підписника на події і декоратором відповідного класу. Об’єкт даного класу містить у собі об’єкт класу **Menu** (в тому числі – потенційно можливого меню, яке реалізовує інтерфейс **IPageMenu**) і переадресовує усі базові операції, які виконуються з меню, до представника, для якого він слугує “обгорткою”. Додатково, цей клас розширює функціонал меню і дозволяє меню виступати у якості підписника на події.

*Обґрунтування використання даного шаблону*.Після створення системи класів, яка представляє меню, з’явилася потреба мати можливість розширити функціонал певних меню, які структурно відрізнялися одне від одного, але мали один і той же абстрактний клас як фундамент. За допомогою “Декоратора” таке розширення функціоналу вдалося досягти, не змінюючи існуючого коду.

1. **Simple Factory**



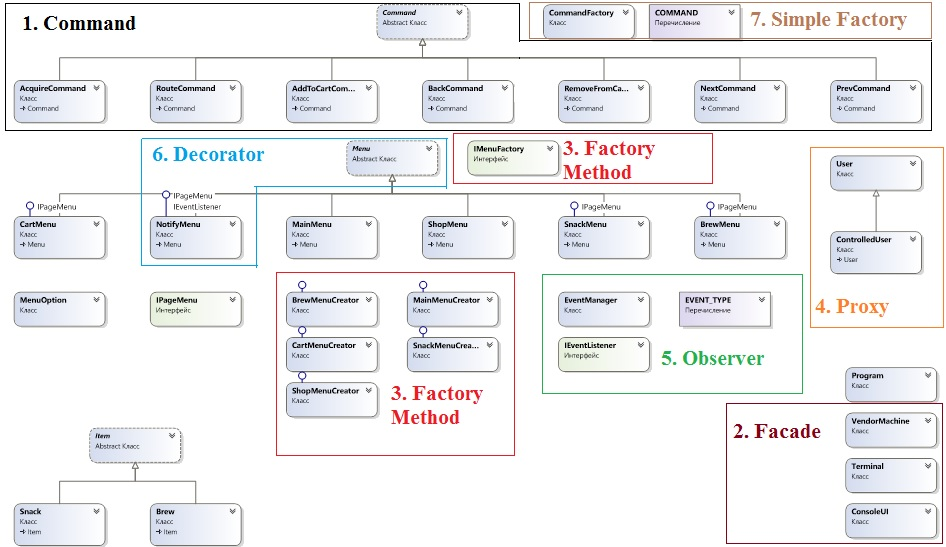
*Рис. 2.1.7. UML-діаграма шаблону “****Simple Factory****”*

*Породжуючий шаблон*. Клас, в якому є один метод з великим умовним оператором, що вибирають створюваний продукт. Цей метод викликають з якимось параметром, за яким визначається який з продуктів потрібно створити. У простій фабрики, зазвичай, немає підкласів.

*Структура*. Клас **CommandFactory** має у собі метод *CreateCommand*, одним з параметрів якого є параметр типу перерахування *COMMAND*. Передаючи в цей метод відповідне значення, запропоноване у даному перерахуванні, метод *CreateCommand* створює і повертає команду відповідного типу.

*Обґрунтування використання даного шаблону*. Після створення системи класів, яка представляє різновид наявних у програмі команд, треба було зручним чином організувати створення цих команд за допомогою якоїсь сутності. Так як кожна з команд має абстрактний клас **Command** у якості базового класу (фундаменту), то було прийняте рішення про використання структури шаблону “Simple Factory” за основу у проектуванні цієї частини системи.

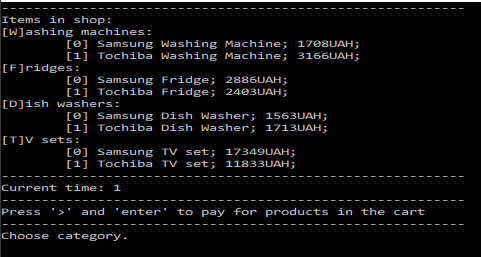
* 1. **Діаграма класів**



*Рис. 2.2.1. Діаграма класів програми*

* 1. **Результати роботи програми**

Початок роботи з магазином техніки для дому починається з головного меню автомату:



*Рис. 2.3.1. Головне меню*

Пропонується вибрати категорію (W, F, D, T – перші букви категорій), а потім – вибрати конкретний товар – від виробника Samsung, чи Tochiba.



*Рис 2.3.2. Вибір категорії по букві*



*Рис 2.3.3. Вибір продукта по номеру – 0, чи 1*

Після вибору необхідних товарів необхідно натиснути на ‘>’ та ‘enter’. Таким чином, ми перейдемо до меню оплати товару на касі та вибору режиму доставки.



*Рис 2.3.4. Перехід до наступної секції – оплати товару на касі*

Одразу вгорі бачимо статус покупця до покупок



*Рис 2.3.4. Статус покупця до покупок*

Під ним бачимо поточний кошик покупця.



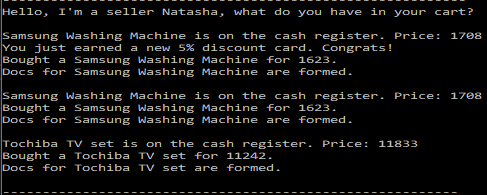
*Рис 2.3.5. Кошик покупця*

У наступній секції зображений поточний «віртуальний час» для визначення поточної зміни продавця. Так як ми стоїмо біля каси, то нас вітає продавець поточної зміни «Natasha»



*Рис 2.3.6. Привітання біля каси та поточний час*

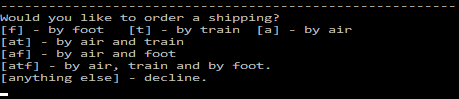
У наступній секції бачимо повідомлення про успішність, чи невдачу покупки. У нашому випадку усі транзакції пройшли успішно. Видно, що чим більше купляють покупці – тим більшими становляться їхні знижки.



*Рис 2.3.7. Статус придбання товарів покупцями*

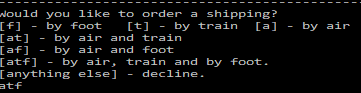
Пропонується на вибір різні варіанти доставки товару – через пішого кур’єра, через потяг, літак, або різні комбінації цих трьох варіантів.

Якщо покупець не бажає замовити доставку, то може натиснути будь-яку клавішу та ‘enter’.



*Рис 2.3.8. Секція вибору режиму доставки*

Я вибрав режим спочатку літаком, потім – потягом, а потім – пішим кур’єром, ввівши строку «atf» та натиснувши ‘enter’.



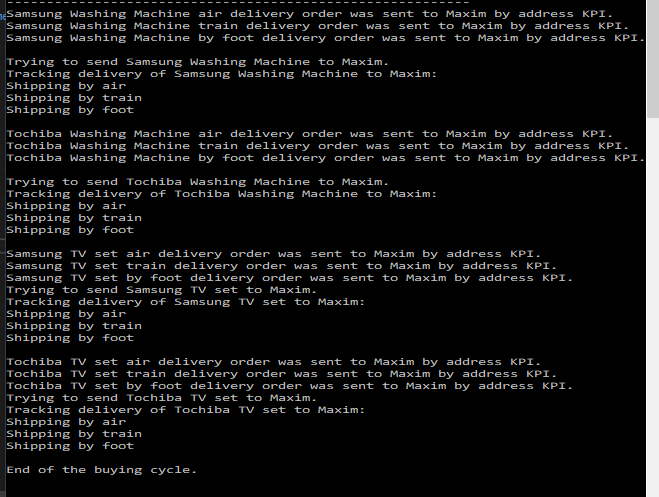
*Рис.2.3.9. Вибір усіх режимів доставки одночасно*

У наступному вікні бачимо статус покупця після покупки. Грошей стало менше, але накопичена вартість збереглась, а дисконтна картка тепер стала 10-відсотковою, а не 0%, як було до покупки.



*Рис 2.3.10. Стан покупця після покупки*

У наступній секції видно, як саме відправлялись товари до моєї адреси (тут: «КПІ»). Показаний шлях спочатку «на паперах», а потім – насправді, у режимі live-tracking. Це – завершення процесу покупки та кінець роботи програми.



*Рис 2.3.11. Кінець роботи програми.*

**ВИСНОВКИ**

Метою даної курсової роботи було розроблення програмного забезпечення магазину для продажу техніки для дому з використанням шаблонів проектування. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» студентами ІІ 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, отримувати документи на придбаний товар, а магазину, в свою чергу, заповнювати полиці товарами.

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

Для розроблення програмного забезпечення була використана мова програмування C#.

**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, Девид Хилл. – 2009. – 529 с.