ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КПІ імені Ігоря Сікорського”

Кафедра програмного забезпечення комп’ютерних систем

**ЛАБОРАТОРНА РОБОТА №4**

з дисципліни

Програмування

C# .Net. Реалізація основних

принципів ООП мовою C#

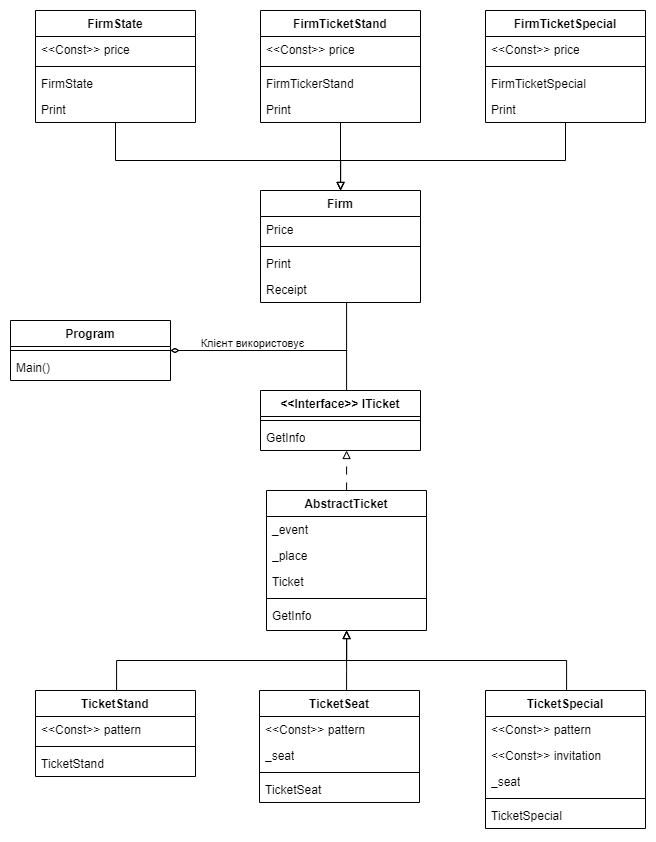
Студента 2 курсу ФПМ

гр. КП-23 Верьовочкіна М. С.

Київ – 2024

**Варіант №6**

1. За допомогою шаблону забезпечити друк квитків на концерти різними фірмами. Білети можуть бути просто вхідними (без місця), звичайними (з місцем) та вкладеними у спеціальне запрошення. Білети на театральні вистави друкують на державних підприємствах, а білети на гастролі та приватних театрів – в комерційних фірмах. Поліграфічні фірми відрізняються кількістю кольорів, які використовуються при друці білетів, та відповідно ціною на послуги. В залежності від типу білетів перенаправляти їх для друку на ту чи іншу фірму.
2. Реалізувати віртуальний конструктор корпусних меблів та забезпечити можливість за бажанням клієнта обирати матеріали каркасу (натуральна деревина, ДСП), матеріали фасадів (ДСП, МДФ фарбований, МДФ плівковий, натуральна деревина), фурнітуру (пластитикову, металеву, дерев’яну).
   1. Задача на масовий («промисловий») випуск продукції різних видів залежно від умов. Оскільки в задачі наведений тільки 1 тип продукту (квитки), то не вийде сформувати промислові лінійки з багатьох типів, отже немає необхідності у використанні шаблону «абстрактна фабрика», натомість для даної задачі достатньо реалізації «фабричного методу». Преставимо UML-діаграму структури класів:



Текст програми (головна функція визначає один з варіантів виконання програми в якості прикладу):

class Program

{

static void Main()

{

Firm firm = new FirmState();

foreach (ITicket ticket in firm.Print("a", "b", 3))

Console.WriteLine("------------------------------\n" + ticket.GetInfo());

Console.WriteLine("====================================================================");

firm = new FirmTicketStand();

foreach (ITicket ticket in firm.Print("a", "b", 5))

Console.WriteLine("------------------------------\n" + ticket.GetInfo());

Console.WriteLine("====================================================================");

firm = new FirmTicketSpecial();

foreach (ITicket ticket in firm.Print("a", "b", 2))

Console.WriteLine("------------------------------\n" + ticket.GetInfo());

}

}

public interface ITicket

{

string GetInfo();

}

public abstract class AbstractTicket : ITicket

{

protected string \_event;

protected string \_place;

protected string Ticket;

public string GetInfo()

{

return Ticket;

}

}

public class TicketStand : AbstractTicket

{

private const string pattern = "This ticket is the basis for admission to the concert\nEvent: {0}\nPlace: {1}";

public TicketStand(string e, string p)

{

\_event = e;

\_place = p;

Ticket = String.Format(pattern, \_event, \_place);

}

}

public class TicketSeat : AbstractTicket

{

private const string pattern = "This ticket is the basis for admission to the concert\nEvent: {0}\nPlace: {1}\nSeat: {2}";

private int \_seat;

public TicketSeat(string e, string p, int s)

{

\_event = e;

\_place = p;

\_seat = s;

Ticket = String.Format(pattern, \_event, \_place, \_seat);

}

}

public class TicketSpecial : AbstractTicket

{

private const string pattern = "This ticket is the basis for admission to the concert\nEvent: {0}\nPlace: {1}\nSeat: {2}";

private const string invitation = "Dear Guests,\n\nJoin us for an exclusive private concert. \n" +

"Experience an unforgettable evening of music and entertainment with us.";

private int \_seat;

public TicketSpecial(string e, string p, int s)

{

\_event = e;

\_place = p;

\_seat = s;

Ticket = invitation + "\n\n" + String.Format(pattern, \_event, \_place, \_seat);

}

}

public abstract class Firm

{

protected double Price;

public abstract List<ITicket> Print(string e, string place, int count);

protected void Receipt(int count) => Console.WriteLine("Your price for {0} tickets is {1}", count, Price \* count);

}

public class FirmState : Firm

{

private const double price = 250.00;

public FirmState()

{

Price = price;

}

public override List<ITicket> Print(string event\_, string place, int count)

{

List<ITicket> tickets = new List<ITicket>();

for (int i = 1; i <= count; i++)

tickets.Add(new TicketSeat(event\_, place, i));

Receipt(count);

return tickets;

}

}

public class FirmTicketStand : Firm

{

private const double price = 200.00;

public FirmTicketStand()

{

Price = price;

}

public override List<ITicket> Print(string event\_, string place, int count)

{

List<ITicket> tickets = new List<ITicket>();

for (int i = 1; i <= count; i++)

tickets.Add(new TicketStand(event\_, place));

Receipt(count);

return tickets;

}

}

public class FirmTicketSpecial : Firm

{

private const double price = 420.00;

public FirmTicketSpecial()

{

Price = price;

}

public override List<ITicket> Print(string event\_, string place, int count)

{

List<ITicket> tickets = new List<ITicket>();

for (int i = 1; i <= count; i++)

tickets.Add(new TicketSpecial(event\_, place, i));

Receipt(count);

return tickets;

}

}

Приклад виконання розробленого коду:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

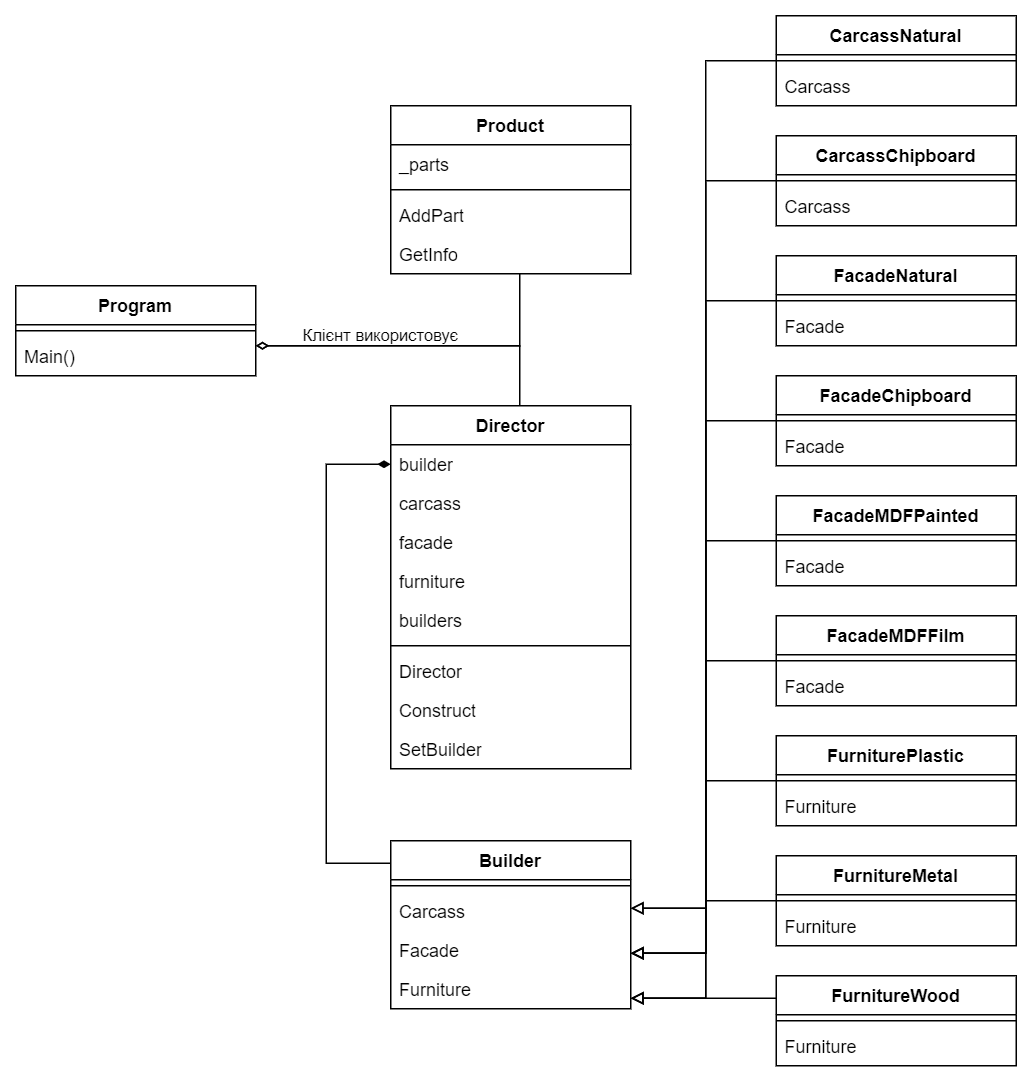
Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. За умовою потрібно зібрати цілісний продукт з окремих компонентів різного виду. Для цього доцільно використовувати шаблон «будівельник», причому клієнт матиме доступ тільки до директора та власне продукту, а роботу з обранням правильного будівельника вже буде виконувати директор.

Преставимо UML-діаграму структури класів:



Текст програми:

class Program

{

static void Main()

{

Console.Write("Enter the number of products: ");

int n = int.Parse(Console.ReadLine());

List<Product> products = new List<Product>();

Director director = new Director();

for (int i = 0; i < n; i++)

products.Add(director.Construct());

foreach(Product product in products)

product.GetInfo();

}

}

public abstract class Builder

{

public virtual void Carcass(Product product) { }

public virtual void Facade(Product product) { }

public virtual void Furniture(Product product) { }

}

public class CarcassNatural : Builder

{

public override void Carcass(Product product) => product.AddPart("natural wood carcass");

}

public class CarcassChipboard : Builder

{

public override void Carcass(Product product) => product.AddPart("chipboard carcass");

}

public class FacadeNatural : Builder

{

public override void Facade(Product product) => product.AddPart("natural wood facade");

}

public class FacadeChipboard : Builder

{

public override void Facade(Product product) => product.AddPart("chipboard facade");

}

public class FacadeMFDPainted : Builder

{

public override void Facade(Product product) => product.AddPart("painted MDF facade");

}

public class FacadeMDFFilm : Builder

{

public override void Facade(Product product) => product.AddPart("film MDF facade");

}

public class FurniturePlastic : Builder

{

public override void Furniture(Product product) => product.AddPart("plastic furniture");

}

public class FurnitureMetal : Builder

{

public override void Furniture(Product product) => product.AddPart("metal furniture");

}

public class FurnitureWood : Builder

{

public override void Furniture(Product product) => product.AddPart("wood furniture");

}

public class Product

{

private List<string> \_parts = new List<string>();

public void AddPart(string part) => \_parts.Add(part);

public void GetInfo()

{

Console.WriteLine("Product parts:");

foreach (string part in \_parts)

Console.WriteLine(" - " + part);

}

}

public class Director

{

Builder? builder = null;

static Dictionary<string, Builder> carcass = new Dictionary<string, Builder>();

static Dictionary<string, Builder> facade = new Dictionary<string, Builder>();

static Dictionary<string, Builder> furniture = new Dictionary<string, Builder>();

static Dictionary<Dictionary<string, Builder>, string> builders = new Dictionary<Dictionary<string, Builder>, string>();

static Director()

{

carcass.Add("Natural wood", new CarcassNatural());

carcass.Add("Chipboard", new CarcassChipboard());

facade.Add("Natural wood", new FacadeNatural());

facade.Add("Chipboard", new FacadeChipboard());

facade.Add("MFD painted", new FacadeMFDPainted());

facade.Add("MDF film", new FacadeMDFFilm());

furniture.Add("Plastic", new FurniturePlastic());

furniture.Add("Metal", new FurnitureMetal());

furniture.Add("Wood", new FurnitureWood());

builders.Add(carcass, "carcass");

builders.Add(facade, "facade");

builders.Add(furniture, "furniture");

}

public Product Construct()

{

Product product = new Product();

SetBuilder(carcass);

builder.Carcass(product);

SetBuilder(facade);

builder.Facade(product);

SetBuilder(furniture);

builder.Furniture(product);

return product;

}

private void SetBuilder(Dictionary<string, Builder> builders)

{

int select\_builder = 0;

Console.WriteLine("Choose the material of {0}:", Director.builders[builders]);

for (int i = 0; i < builders.Count; i++)

Console.WriteLine("{0}) {1}", i + 1, builders.Keys.ElementAt(i));

do

{

try { select\_builder = int.Parse(Console.ReadLine()); }

catch (FormatException) { Console.WriteLine("Wrong format..."); }

} while(select\_builder < 1 || select\_builder > builders.Count);

builder = builders[builders.Keys.ElementAt(select\_builder - 1)];

}

}

Приклад виконання розробленого коду:

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Як бачимо, програма коректно «побудувала» меблі відповідно до запиту користувача та вивела інформацію про них на екран.

**Висновки:** в ході виконання лабораторної роботи було опановано роботу з такими особливостями мови C#, як твірні шаблони, вивчено відмінності між ними та способи їх доцільного підбору. Були продемонстровані різні види UML-діаграм для обраних шаблонів. Безпосередньо в роботі були використані шаблони «фабричний метод» та «будівельник».