МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім.Ігоря Сікорського»

Лабораторна робота №3

з дисципліни «ОРПЗ .NET»

на тему:

«Шаблони проектування.

Породжуючі шаблони»

Варіант №18

Виконала студентка групи ІС-03

Полюхович Іванна

Київ

2022

Шаблони проектування. Породжуючі шаблони**Мета роботи**: ознайомитися з основними шаблонами проектування, навчитися застосовувати їх при проектуванні і розробці ПЗ.

**Постановка задачі комп‘ютерного практикуму № 3**

При виконанні комп‘ютерного практикуму необхідно виконати наступні дії:

1) Вивчити породжуючі патерни. Знати загальну характеристику та призначення кожного з них, особливості реалізації кожного з породжуючих патернів та випадки їх застосування.

2) Реалізувати задачу згідно варіанту, запропонованого нижче. Розробити інтерфейси та класи з застосування одного або декількох патернів. Повністю реалізувати методи, пов‘язані з реалізацією обраного патерну.

3) Повністю описати архітектуру проекту (призначення методів та класів), особливості реалізації обраного патерну. Для кожного патерну необхідно вказати основні класи та їх призначення,

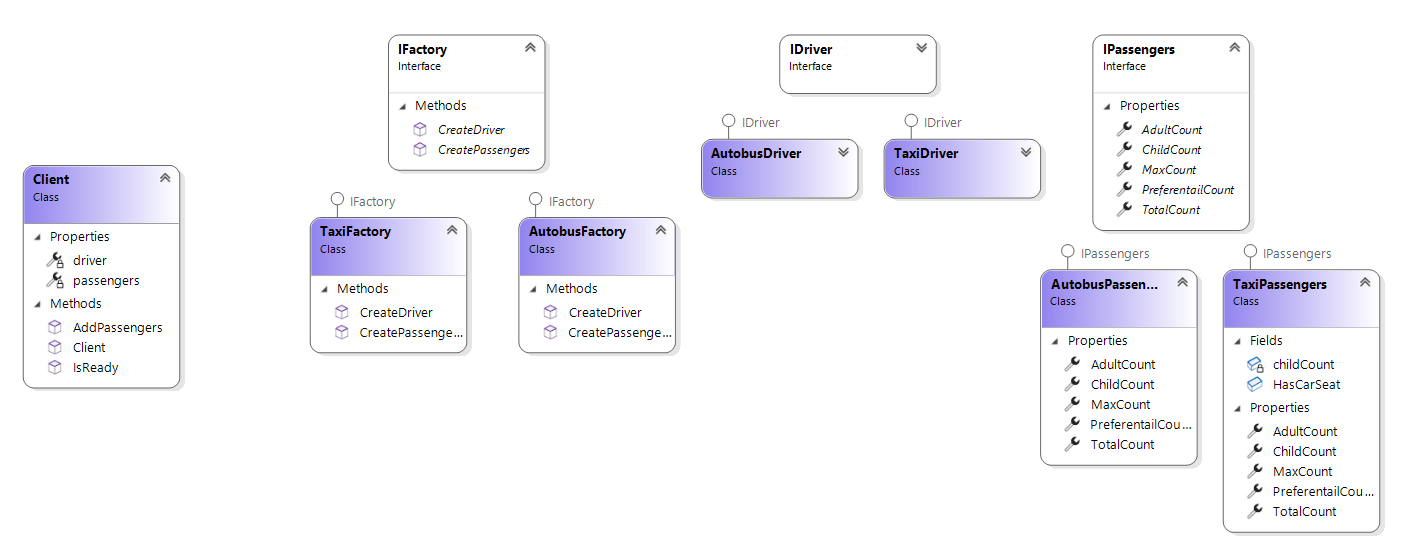
4) Навести UML-діаграму класів

**Варіанти індивідуальних завдань:**

18) Реалізувати задачу «Перевезення». Забезпечити контроль завантаження та готовності до відправлення автобусів і таксі. Водій таксі і автобуси мають права різної категорії. Без водія машина не поїде. Два водія в одну машину сісти не можуть. Без пасажирів машини не поїдуть. Є ліміт завантаження машин. Для автобуса 30 чол. Для таксі -4 чол. Є різниця між пасажирами автобуса і таксі. Для автобуса: три категорії пасажирів - дорослий, пільговий, дитина - різна вартість квитка. Для таксі: дорослий і дитина. Необхідно дитяче крісло.

Щоб реалізувати даний варіант, я обрала патерн для розробки Abstract Factory.

**UML-діаграму класів**

****

**Код програми**

/// <summary>

/// Transport client.

/// Here are the main components without which the vehicle will not go and

/// implementation of their interaction.

/// This does not use specific implementations, but basic interfaces,

/// which allows you to dynamically configure the components of the vehicle.

/// Configuration is performed by transfer to the factory designer.

/// It is important to understand that this class is in fact already a client of the abstract factory.

/// An abstract factory does not have to return a single object. It returns individual components,

/// which can both come together (as here),

/// and just interact with each other.

/// But an abstract factory guarantees that these components will

/// correctly compatible and properly interact with each other.

/// </summary>

public class Client

{

/// <summary>

/// Driver of the vehicle.

/// </summary>

IDriver driver { get; set; }

/// <summary>

/// Passengers of the vehicle.

/// </summary>

IPassengers passengers { get; set; }

/// <summary>

/// Create vehicle with driver and passengers.

/// </summary>

/// <param name="f"> Factory determinig what vehicle will be. </param>

public Client(IFactory f)

{

driver = f.CreateDriver();

passengers = f.CreatePassengers();

}

/// <summary>

/// Adding passengers.

/// </summary>

/// <param name="children"> Count of children`s seats. </param>

/// <param name="adult"> Count of adult`s seats. </param>

/// <param name="prefential"> Count of prefential`s seats. </param>

public void AddPassengers(int children, int adult, int prefential = 0)

{

int sum = children + adult + prefential;

if (sum <= passengers.MaxCount)

{

passengers.AdultCount = adult;

passengers.ChildCount = children;

passengers.PreferentailCount = prefential;

}

else

{

Console.WriteLine("Passengers cannot join to auto becuase count overrun maximal size! Add again.");

}

}

/// <summary>

/// Ready to move.

/// </summary>

/// <returns> Is ready to go?. </returns>

public bool IsReady()

{

bool isReady = false;

if (passengers.TotalCount != 0)

{

isReady = true;

}

return isReady;

}

}

/// <summary>

/// Abstract factory of creating products.

/// </summary>

public interface IFactory

{

IPassengers CreatePassengers();

IDriver CreateDriver();

}

/// <summary>

/// Factory of autobus.

/// Determining family of components.

/// Contains factory methods.

/// </summary>

class AutobusFactory : IFactory

{

public IDriver CreateDriver()

{

return new AutobusDriver();

}

public IPassengers CreatePassengers()

{

return new AutobusPassengers();

}

}

/// <summary>

/// Factory of taxi.

/// Determining family of components.

/// Contains factory methods.

/// </summary>

class TaxiFactory : IFactory

{

public IDriver CreateDriver()

{

return new TaxiDriver();

}

public IPassengers CreatePassengers()

{

return new TaxiPassengers();

}

}

/// <summary>

/// Vehicle driver.

/// </summary>

public interface IDriver

{

}

/// <summary>

/// Taxi driver.

/// </summary>

class TaxiDriver : IDriver

{

}

/// <summary>

/// Autobus driver.

/// </summary>

class AutobusDriver : IDriver

{

}

/// <summary>

/// Vehicle passengers.

/// </summary>

public interface IPassengers

{

int MaxCount { get; }

int TotalCount { get; set; }

int ChildCount { get; set; }

int AdultCount { get; set; }

int PreferentailCount { get; set; }

}

/// <summary>

/// Autobus passengers.

/// </summary>

class AutobusPassengers : IPassengers

{

/// <summary>

/// Maximal count of passengers.

/// </summary>

public int MaxCount => 30;

/// <summary>

/// Total count of passengers.

/// </summary>

public int TotalCount {

get

{

return ChildCount + AdultCount + PreferentailCount;

}

set

{

TotalCount = ChildCount + AdultCount + PreferentailCount;

}

}

/// <summary>

/// Count of children in autobus.

/// </summary>

public int ChildCount { get; set; }

/// <summary>

/// Count of adult in autobus.

/// </summary>

public int AdultCount { get; set; }

/// <summary>

/// Count of preferential in autobus.

/// </summary>

public int PreferentailCount { get; set; }

}

/// <summary>

/// Taxi passengers.

/// </summary>

class TaxiPassengers : IPassengers

{

/// <summary>

/// Maximal count of passengers.

/// </summary>

public int MaxCount => 4;

int childCount;

/// <summary>

/// Total count of passengers.

/// </summary>

public int TotalCount

{

get

{

return ChildCount + AdultCount;

}

set

{

TotalCount = ChildCount + AdultCount;

}

}

/// <summary>

/// Count of adult in taxi.

/// </summary>

public int AdultCount { get; set; }

/// <summary>

/// Count of children in taxi.

/// </summary>

public int ChildCount

{

get => childCount;

set {

HasCarSeat = true;

childCount = value;

}

}

/// <summary>

/// Count of preferential in taxi.

/// </summary>

public int PreferentailCount { get; set; } = 0;

/// <summary>

/// THE PRESENCE OF A CHILD SEAT.

/// </summary>

public bool HasCarSeat = false;

}

static void Main(string[] args)

{

IFactory factory = new TaxiFactory();

Client client = new Client(factory);

client.AddPassengers(1,3);

Console.WriteLine(client.IsReady());

factory = new AutobusFactory();

client = new Client(factory);

client.AddPassengers(14, 15, 2);

Console.WriteLine(client.IsReady());

}