Лабораторная работа №9
Множественное наследование

Цель работы:

получить навыки проектирования приложения, состоящих из нескольких взаимосвязанных классов и интерфейсов

Время выполнения – 8 академических часов

Задание к работе

1. Описать семейство классов, имеющих общий функционал (), при этом в каждом классе присутствует дополнительно свой функционал. Набор дополнительных функций в разных классах может быть произвольным.
2. Дополнительный функционал описать в виде набора интерфейсов
3. Одна из **общих** функций должна быть реализована по-своему в каждом классе.
4. Одна из **общих** функций должна быть реализована в других классах (например, изменение скорости, использование оружия, доставка груза). При этом должно быть несколько вариантов реализации (несколько классов), например, персонажам игры доступны разные инструменты – каждый инструмент может использоваться разными персонажами. Конкретный вариант реализации выбирается **при создании объекта** (применить шаблон проектирования **«Мост» («Bridge»**)).
5. Для создания объектов использовать шаблон проектирования **«Абстрактная фабрика» (Abstract factory)** или **«Построитель» (Builder)**
6. В классе Program создать коллекцию разных объектов (см. п.1). Затем для каждого элемента коллекции вызвать **все методы, доступные** для данного объекта.

Примечание: НЕ НУЖНО разрабатывать реальную реализацию функций. Достаточно вывести в консоль сообщение о том, какое действие выполняется, и кем.

Примечание: меню для взаимодействия с пользователем реализовывать не нужно. Для демонстрации работы приложения достаточно инициализацию исходных данных и проверку функционала приложения реализовать в коде

Индивидуальные задания

1. Предметная область: Телевизор.

- Пример общих свойств: наименование, тип экрана (например: плазма, OLED …)

- Общие функции: GetInfo (в каждом классе реализуется по-своему), Show (варианты описаны в других классах, примеры: эфир, кабель, HDMI …)
- Примеры дополнительных функций: Интернет-браузер, Распознавание движений, Голосовой помощник …

2. Предметная область: Строительная бригада.

- Пример общих свойств: наименование, тип (организация, ИП, частное лицо …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), BuildWalls (варианты описаны в других классах, примеры: из кирпича, из пеноблоков, монолитные …)
- Примеры дополнительных функций: поставка стройматериалов, разработка проекта, консультация заказчика …

3. Предметная область: Риелтор.

- Пример общих свойств: имя, тип недвижимости
- Общие функции: GetInfo (в каждом классе реализуется по-своему), DecribeObject (варианты описаны в других классах, примеры: по E-mail, по телефону, через Viber …)
- Примеры дополнительных функций: выезд на объект, подготовка документов, проверка банкнот …

4. Предметная область: Кухонный процессор.

- Пример общих свойств: название, тип(настольный, ручной, встроенный …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), Process (варианты описаны в других классах, примеры: шинкование, измельчение, мясорубка …)
- Примеры дополнительных функций: миксер, блендер, тестомешалка …

5. Предметная область: Фирма доставки товаров.

- Пример общих свойств: название, тип(городской, междугородний, международный …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), Deliver (варианты описаны в других классах, примеры: автомобилем, DHL, AutoLight Express …)
- Примеры дополнительных функций: сопровождение груза, страхование, отслеживание этапов доставки …

6. Предметная область: Пиццерия.

- Пример общих свойств: название, размер (маленькая, средняя, большая …)

- Общие функции: GetInfo (в каждом классе реализуется по-своему), Bake (варианты описаны в других классах, примеры: на толстой основе, на тонкой основе, закрытая)
- Примеры дополнительных функций: (добавки) острая, с голубым сыром, на углях…

7. Предметная область: Медперсонал.

- Пример общих свойств: имя, профиль (хирург, терапевт, медсестра …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), Diagnose (варианты описаны в других классах, примеры: осмотр, МРТ, рентген…)
- Примеры дополнительных функций: сделать укол, выписать больничный, сделать перевязку …

8. Предметная область: Гостиница.

- Пример общих свойств: название, тип (гостиница, хостел, агроусадьба …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), Pay (варианты описаны в других классах, примеры: наличные, карта, банковский перевод …)
- Примеры дополнительных функций: трансфер от аэропорта, минибар, бассейн, интернет …

9. Предметная область: Автомобиль.

- Пример общих свойств: модель, тип (легковой, грузовой …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), ChangeGear (варианты описаны в других классах и зависит от коробки передач, примеры: механика, автомат, вариатор…)
- Примеры дополнительных функций: передвижение, перевозка груза, перевозка пассажиров …

10 **Предметная область: Принтер.**

- Пример общих свойств: модель, формат печати (A4, A3, 44” …)
- Общие функции: GetInfo (в каждом классе реализуется по-своему), Print (варианты описаны в других классах и зависит от технологии печати, примеры: лазерная, струйная, сублимационная…)
- Примеры дополнительных функций: печать на листах, печать с рулона, сортировка, web-интерфейс …

11 **Предметная область: Путешествия**

- Пример общих свойств: маршрут, вид (индивидуальный, групповой, семейный)

- Общие функции: GetInfo (в каждом классе реализуется по-своему), Travel (варианты описаны в других классах и зависит от способа путешествия, примеры: пешком, на велосипеде, на автобусе…)
- Примеры дополнительных функций: посещение достопримечательностей, мангал, прыжок с тарзанки …

Пример выполнения работы

Предметная область: Пошаговая стратегия.

Семейство объектов – персонажи игры: Peasant и Necromancer

Общие свойства: Name (имя) и Gender (пол)

Пол описан как перечисление:

```
enum Gender
{
    Male, Female
}
```

Дополнительные свойства:

Возможность перемещения. Описано в интерфейсе IMovable. Реализуется в классе Peasant.

```
internal interface IMovable
    {
        void Walk();
        void Run();
    }
```

Выполнение работы. Описано в интерфейсе IWorker. Реализуется в классе Peasant.

```
internal interface IWorker
    {
        void DoWork();
    }
```

Наложение заклятья. Описано в интерфейсе IWizard. Реализуется в классе Necromancer.

```
internal interface IWizard
    {
        void Spell();
    }
```

Пример реализации метода DoWork:

```
public void DoWork()
        {
```

```
            Say("I work on my garden");
        }
```

Общий метод GetInfo().

В классе Peasant реализуется так:

```
public override void GetInfo()
        {
            Say($"I am peasant, my gender is {Gender}");
        }
```

В классе Necromancer реализуется так:

```
public override void GetInfo()
        {
            Say($"I am necromancer, my gender is {Gender}");
        }
```

Метод Say:

```
protected void Say(string message)
    {
        Console.WriteLine($"{Name} says: {message}");
    }
```

Общий метод UseWeapon – использовать оружие

Предлагаемые виды оружия: Longbow, Sword и Poleaxe

Оружие реализует интерфейс IWeapon:

```
internal interface IWeapon
{
    void Attack();
}
```

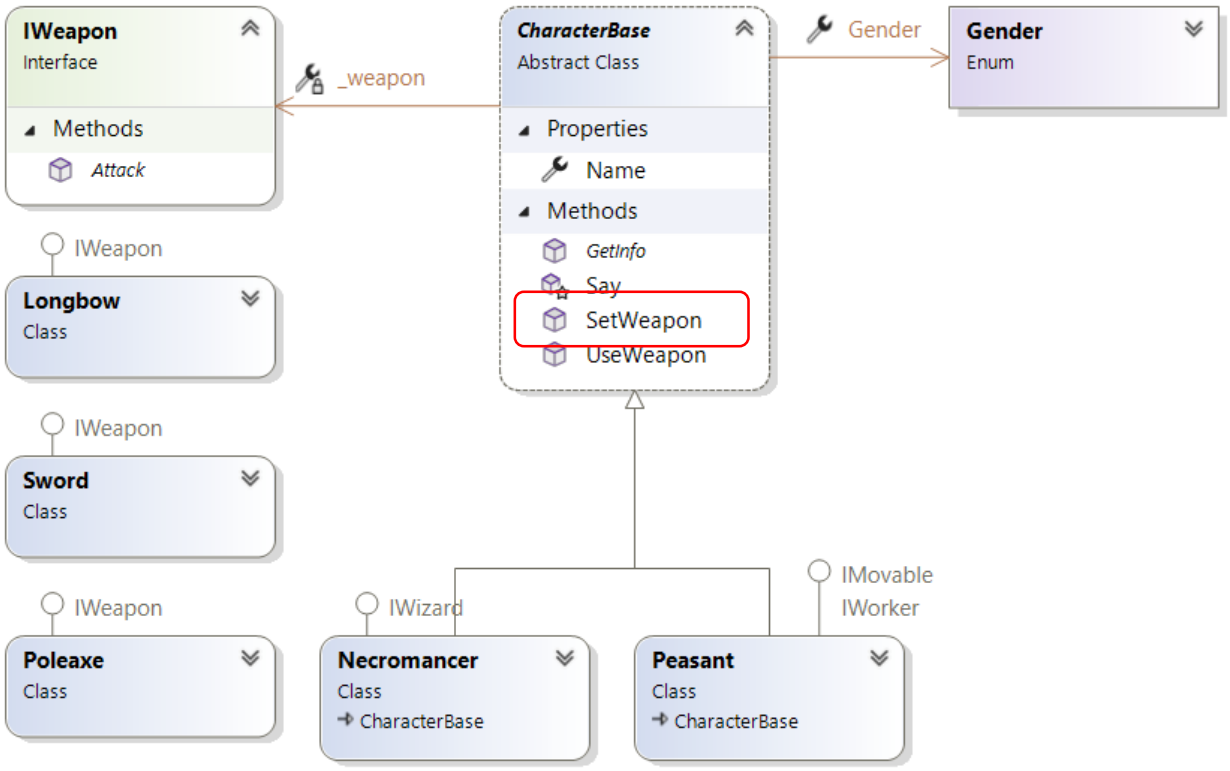
Пример реализации в классе Longbow:

```
public void Attack()
        {
            Console.WriteLine("An arrow shot from a bow");
        }
```

Шаблон проектирования «Мост»

В качестве абстракции (моста) выделим абстрактный класс CharacterBase, описывающий общие члены классов персонажей, и взаимодействующий с оружием через интерфейс IWeapon.

Диаграмма классов:



> Теперь с помощью метода SetWeapon можно любому персонажу назначить любое оружие.

Пример создания объекта Peasant:

```
var peasant = new Peasant {Name=”Bob”, Gender=Gender.Male};
peasant.SetWeapon(new Sword());
```

Создание объектов с помощью шаблона Строитель

Пример абстрактного класса строителя:

```
internal abstract class AbstractBuilder
    {
        protected string name = String.Empty;
        protected Gender gender;
        protected IWeapon? weapon;
        public AbstractBuilder SetName(string name)
        { this.name = name; return this; }
        public AbstractBuilder SetGender(Gender gender)
        {  this.gender = gender; return this;}
        public AbstractBuilder SetWeapon(IWeapon weapon)
        { this.weapon = weapon; return this; }

        public abstract CharacterBase Build();
    }
```

Метод Build() будет реализован в наследниках PeasantBuilder и NecromancerBuilder, например:

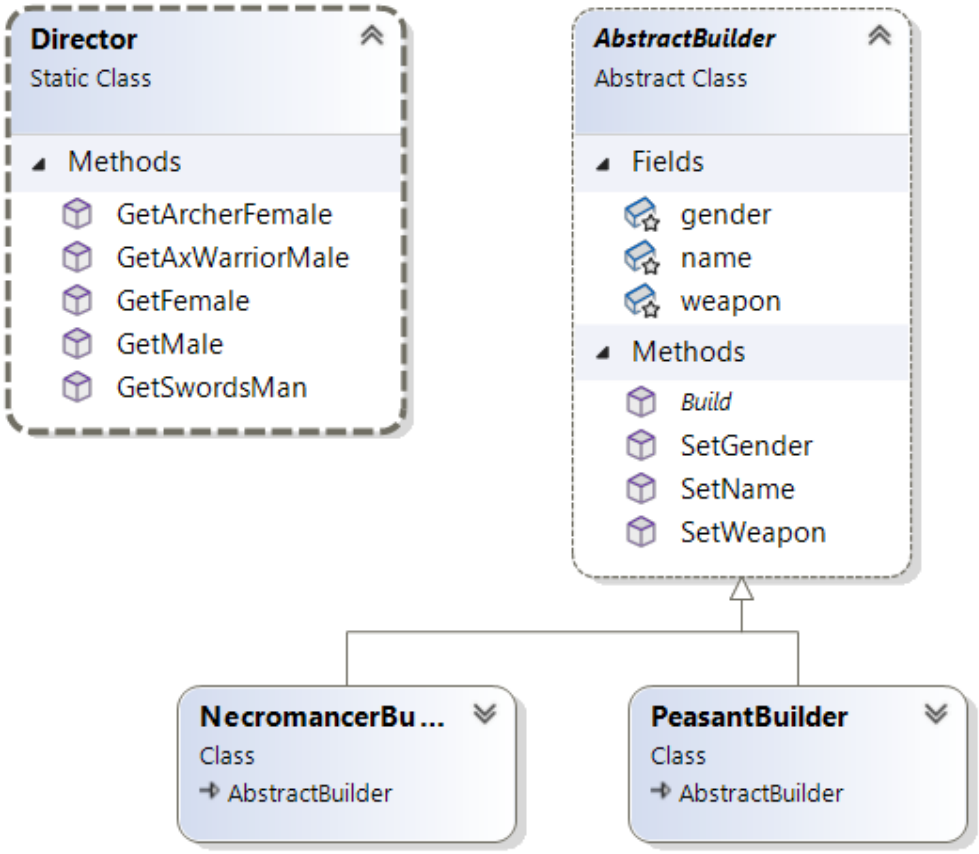
```
public override CharacterBase Build()
        {
            var peasant = new Peasant {Name=name, Gender=gender};
            if(weapon!=null)
                peasant.SetWeapon(weapon);
            return peasant;
        }
```

Класс Director с помощью построителей будет создавать конкретные объекты классов, например:

```
internal static class Director
    {
        /// <summary>
        /// Создать невооруженного мужчину
        /// </summary>
        /// <param name="name">Имя персонажа</param>
        /// <param name="builder">объект построителя</param>
        /// <returns></returns>
        public static CharacterBase GetMale(string name, AbstractBuilder builder)
        {        }

        /// <summary>
        /// Создать невооруженную женщину
        /// </summary>
        /// <param name="name">Имя персонажа</param>
        /// <param name="builder">объект построителя</param>
        /// <returns></returns>
        public static CharacterBase GetFemale(string name, AbstractBuilder
builder)
        {        }

        /// <summary>
        /// Создать женщину-лучника
        /// </summary>
        /// <param name="name">Имя персонажа</param>
        /// <param name="builder">объект построителя</param>
        /// <returns></returns>
        public static CharacterBase GetArcherFemale(string name, AbstractBuilder
builder)
        {
            return builder.SetName(name)
                        .SetGender(Gender.Female)
                        .SetWeapon(new Longbow())
                        .Build();
        }
```

```
        /// <summary>
        /// Создать мечника
        /// </summary>
        /// <param name="name">Имя персонажа</param>
        /// <param name="builder">объект построителя</param>
        /// <returns></returns>
        public static CharacterBase GetSwordsMan(string name, AbstractBuilder
builder)
        {       }

        /// <summary>
        /// Создать мужчину, вооруженного топором
        /// </summary>
        /// <param name="name">Имя персонажа</param>
        /// <param name="builder">объект построителя</param>
        /// <returns></returns>
        public static CharacterBase GetAxWarriorMale(string name, AbstractBuilder
builder)
        {       }
    }
```

Диаграмма классов:



Пример создания объектов в классе Program:

```
var characters = new List<CharacterBase>();

var peasantBuilder = new PeasantBuilder();
```

```
var necromancerBuilder = new NecromancerBuilder();

characters.AddRange(new CharacterBase[]
{
    Director.GetFemale("Mary", peasantBuilder),
    Director.GetSwordsMan("Merlin", necromancerBuilder),
    Director.GetAxWarriorMale("Cliff", peasantBuilder),
    Director.GetArcherFemale("Katana", peasantBuilder)
});
```

> Далее в цикле нужно обойти коллекцию **characters** и вызвать **Все** доступные методы

Пример результата работы программы: