Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра Информатики

Дисциплина «Программирование»

**ОТЧЕТ**

к лабораторной работе №8

на тему:

**«Множественное наследование»**

БГУИР 6-05-0612-02 34

|  |
| --- |
| Выполнил студент группы 353505  Хорошко Кирилл Николаевич |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 1 Индивидуальное задание

Задание 1. Вариант 2. Предметная область: Описать семейство классов, имеющих общий функционал (), при этом в каждом классе присутствует дополнительно свой функционал. Набор дополнительных функций в разных классах может быть произвольным. Дополнительный функционал описать в виде набора интерфейсов. Одна из общих функций должна быть реализована по-своему в каждом классе. 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»)). Для создания объектов использовать шаблон проектирования «Абстрактная фабрика» (Abstract factory) или «Построитель» (Builder). В классе Program создать коллекцию разных объектов. Затем для каждого элемента коллекции вызвать все методы, доступные для данного объекта.

# 2 Выполнение работы

Весь проект разбит на отдельные файлы для удобства работы, каждый класс имеет свой файл. Для начала был реализован класс Type, который содержал в себе перечисление типов строительных фирм, это могли быть организация, частное лицо, ип.

internal class Type

{

public enum type

{

Organization,

IP,

Person\_Company

}

}

Далее был выбран шаблон проектирования Builder, а также использовался шаблон проектирования Bridge. По шаблону Builder создавался класс по конкретному примеру, а с помощью шаблона Bridge был выделен абстрактный класс, от которого наследовались еще 2 класса.

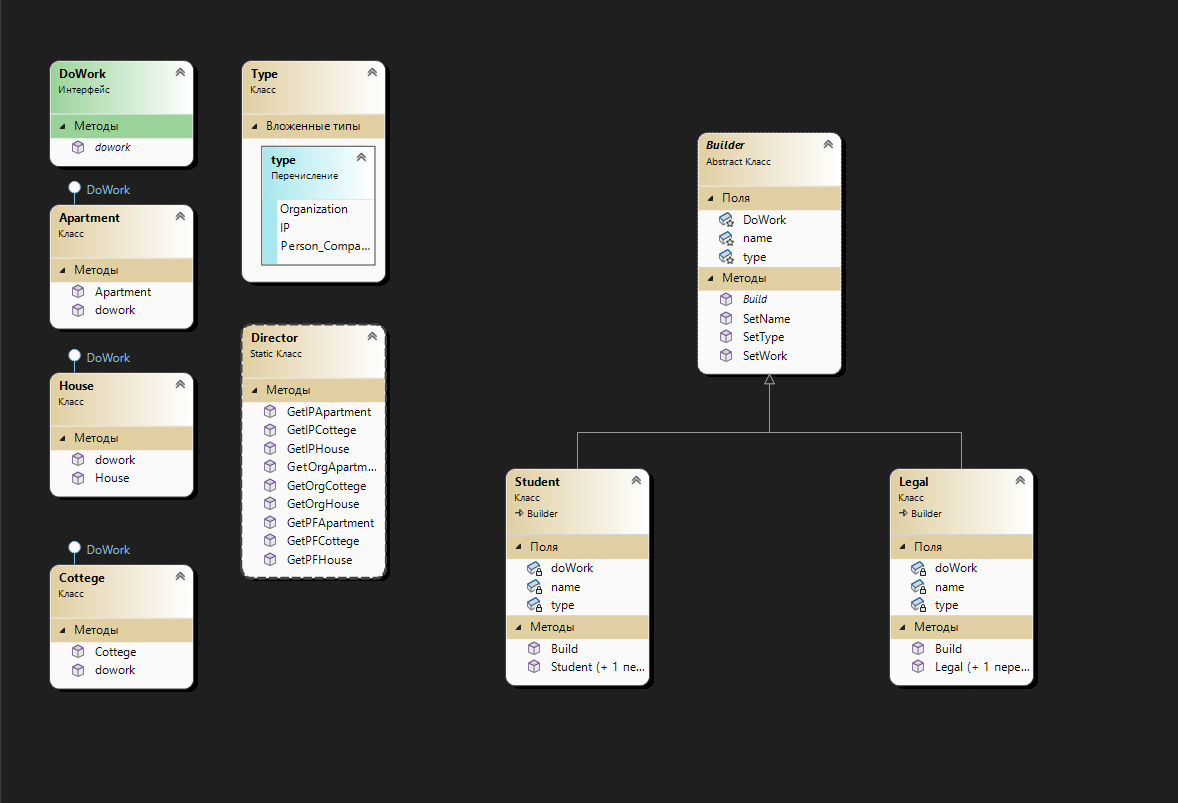


Рисунок 1 – Диаграмма классов

Класс Builder

internal abstract class Builder

{

protected string name = String.Empty;

protected Type.type type;

protected DoWork? DoWork;

public Builder SetName(string name)

{

this.name = name;

return this;

}

public Builder SetType( Type.type type)

{

this.type = type;

return this;

}

public Builder SetWork(DoWork DoWork)

{

this.DoWork = DoWork;

Console.WriteLine("Мы {0} типа {1}", name, type);

DoWork.dowork();

return this;

}

public abstract Builder Build();

}

Интерфейс DoWork

internal interface DoWork

{

void dowork();

}

Класс Director, с помощью статических методов которого создаются объекты классов.

static class Director

{

public static Builder GetOrgCottege(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.Organization)

.SetWork(new Cottege())

.Build();

}

public static Builder GetOrgHouse(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.Organization)

.SetWork(new House())

.Build();

}

public static Builder GetOrgApartment(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.Organization)

.SetWork(new Apartment())

.Build();

}

public static Builder GetIPCottege(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.IP)

.SetWork(new Cottege())

.Build();

}

public static Builder GetIPHouse(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.IP)

.SetWork(new House())

.Build();

}

public static Builder GetIPApartment(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.IP)

.SetWork(new Apartment())

.Build();

}

public static Builder GetPFCottege(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.Person\_Company)

.SetWork(new Cottege())

.Build();

}

public static Builder GetPFHouse(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.Person\_Company)

.SetWork(new House())

.Build();

}

public static Builder GetPFApartment(string name, Builder builder)

{

return builder.SetName(name)

.SetType(Type.type.Person\_Company)

.SetWork(new Apartment())

.Build();

}

}

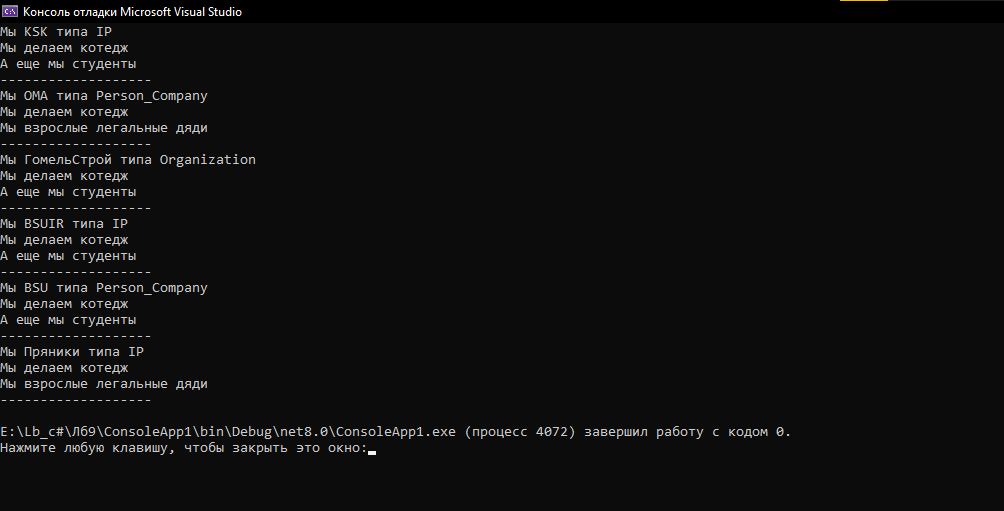


Рисунок 2 – Результат работы программы

# Вывод

В данной работе была поставлена цель разработать приложение, состоящее из нескольких взаимосвязанных классов и интерфейсов, и получить навыки проектирования такого приложения. В рамках выполнения работы были выполнены следующие задания: Описано семейство классов, имеющих общий функционал. Каждый класс в семействе имеет свой дополнительный функционал, что позволяет различать их. Набор дополнительных функций в разных классах может быть произвольным. Дополнительный функционал был описан в виде набора интерфейсов. Интерфейсы определяют контракты, которые классы должны реализовывать, чтобы предоставить свой дополнительный функционал. Одна из общих функций была реализована по-своему в каждом классе. Это позволяет каждому классу иметь уникальный функционал, несмотря на общую базу. Одна из общих функций была реализована в других классах с использованием разных вариантов реализации. Для этого был применен шаблон проектирования "Мост" (Bridge), который позволяет разделить абстракцию и реализацию, и выбрать конкретную реализацию при создании объекта. Для создания объектов был использован шаблон проектирования "Абстрактная фабрика" (Abstract Factory). Эти шаблоны позволяют создавать объекты с определенными свойствами и настройками, что обеспечивает гибкость и удобство при создании экземпляров классов. В классе Program была создана коллекция разных объектов из семейства классов. Затем для каждого элемента коллекции вызывались все методы, доступные для данного объекта. Это позволяет проверить работу всех функций и методов, реализованных в классах и интерфейсах.