

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра Информатики
Дисциплина «Программирование»

ОТЧЕТ
к лабораторной работе №9
на тему:
«МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ»
БГУИР 6-05-0612-02 113

Выполнил студент группы 453503
ХАЛАМОВ Николай Андреевич

(дата, подпись студента)

Проверил ассистент каф. Информатики
РОМАНЮК Максим Валерьевич

(дата, подпись преподавателя)

Минск 2025

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Задание 1. Вариант 4. 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. Предметная область: Кухонный процессор.

2 ВЫПОЛНЕНИЕ РАБОТЫ

Перед выполнением работы следует разработать диаграмму классов для наглядного выполнения поставленной задачи (см. рисунок 1).

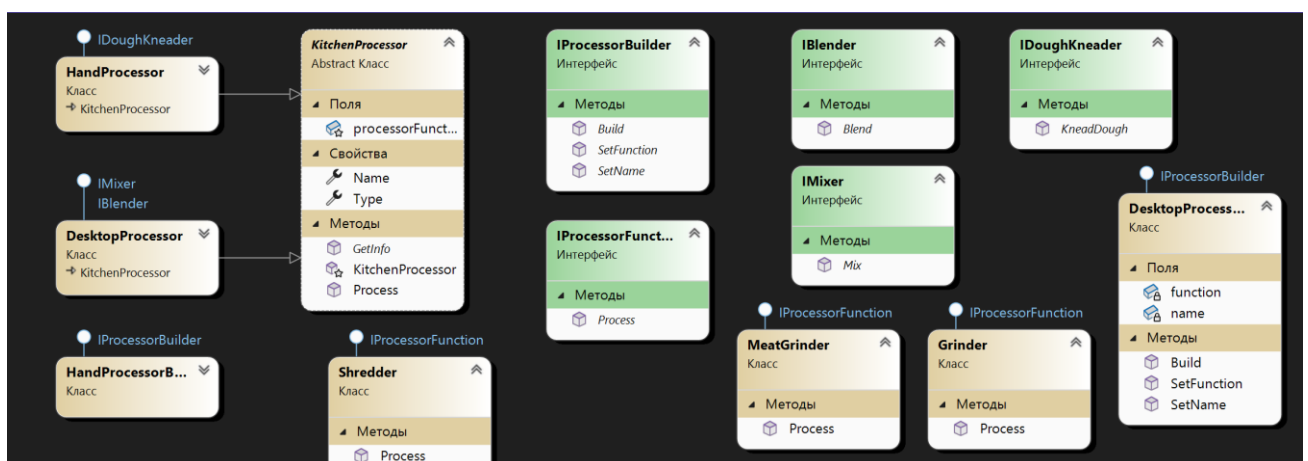


Рисунок 1 – Диаграмма классов

Для выполнения задания были созданы 4 папки с классами и интерфейсами для понятной и корректной структуры приложения. В папке

Abstract находятся основа и описание поведения модели. В абстрактном классе **KitchenProcessor** находятся общие свойства и функции для всех процессоров.

```
namespace KitchenProcessorApp.Abstract
{
    public abstract class KitchenProcessor
    {
        public string Name { get; set; }
        public string Type { get; set; }
        protected IProcessorFunction processorFunction;

        protected KitchenProcessor(string name, string type,
IProcessorFunction function)
        {
            Name = name;
            Type = type;
            processorFunction = function;
        }

        public abstract void GetInfo();

        public void Process() => processorFunction.Process(Name);
    }
}
```

Интерфейс IProcessorFunction описывает как процессор будет обрабатывать продукты.

```
public interface IProcessorFunction
{
    void Process(string processorName);
}
```

Интерфейсы IMixed, IBlender, IDoughKneader добавляют поведение к процессорам через интерфейсы.

```
public interface IBlender
{
    void Blend();
}
public interface IDoughKneader
{
    void KneadDough();
}
public interface IMixer
{
    void Mix();
}
```

Папка Functions содержит реализации поведения — что будет происходить, когда вызвать **Process()**.

```
public class Grinder : IProcessorFunction
{
    public void Process(string name) => Console.WriteLine($"{name}
измельчает продукты.");
}
public class MeatGrinder : IProcessorFunction
```

```

    {
        public void Process(string name) => Console.WriteLine($"{name}
превращает мясо в фарш.");
    }
    public class Shredder : IProcessorFunction
    {
        public void Process(string name) => Console.WriteLine($"{name}
шинкует продукты.");
    }

```

В папке Models находятся те кухонные процессоры, которые мы будем использовать. Оба наследуются от KitchenProcessor и реализуют GetInfo().

```

public class DesktopProcessor : KitchenProcessor, IMixer, IBlender
{
    public DesktopProcessor(string name, IProcessorFunction function)
        : base(name, "Настольный", function) { }

    public override void GetInfo() =>
        Console.WriteLine($"{Name} Тип: {Type}, Поддерживает миксер и
блендер");

    public void Mix() => Console.WriteLine($"{Name} взбивает тесто.");
    public void Blend() => Console.WriteLine($"{Name} делает смузи.");
}
public class HandProcessor : KitchenProcessor, IDoughKneader
{
    public HandProcessor(string name, IProcessorFunction function)
        : base(name, "Ручной", function) { }

    public override void GetInfo() =>
        Console.WriteLine($"{Name} Тип: {Type}, Поддерживает
тестомешалку");

    public void KneadDough() => Console.WriteLine($"{Name} замешивает
тесто.");
}

```

В папке Builder реализуется паттерн Строитель. Интерфейс IprocessorBuilder задает правила сборки процессора и устанавливает единые правила для всех строителей (Desktop, Hand), а в классах реально задаются все параметры нового процессора.

```

public interface IProcessorBuilder
{
    void SetName(string name);
    void SetFunction(IProcessorFunction function);
    KitchenProcessor Build();
}
public class HandProcessorBuilder : IProcessorBuilder
{
    private string name;
    private IProcessorFunction function;

    public void SetName(string name) => this.name = name;
    public void SetFunction(IProcessorFunction function) => this.function
= function;
    public KitchenProcessor Build() => new HandProcessor(name, function);
}
public class DesktopProcessorBuilder : IProcessorBuilder

```

```

{
    private string name;
    private IProcessorFunction function;

    public void SetName(string name) => this.name = name;
    public void SetFunction(IProcessorFunction function) =>
this.function = function;
    public KitchenProcessor Build() => new DesktopProcessor(name,
function);
}

```

Класс Program создаёт объекты через билдеры, кладёт их в List<KitchenProcessor>, проходит по списку, вызывает у каждого: GetInfo(), Process(), Mix() / Blend() / KneadDough() — если реализованы. Демонстрирует, что всё работает правильно.

```

class Program
{
    static void Main()
    {
        var processors = new List<KitchenProcessor>();

        var builder1 = new DesktopProcessorBuilder();
        builder1.SetName("Процессор A");
        builder1.SetFunction(new Shredder());
        processors.Add(builder1.Build());

        var builder2 = new HandProcessorBuilder();
        builder2.SetName("Процессор B");
        builder2.SetFunction(new MeatGrinder());
        processors.Add(builder2.Build());

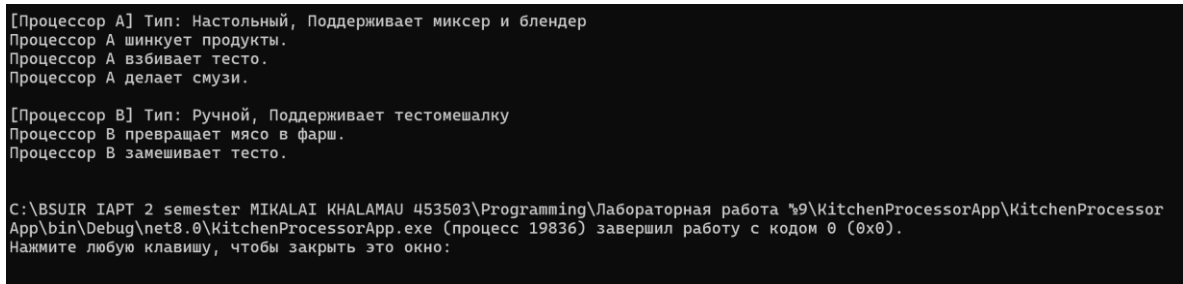
        foreach (var proc in processors)
        {
            proc.GetInfo();
            proc.Process();

            if (proc is IMixer mixer) mixer.Mix();
            if (proc is IBlender blender) blender.Blend();
            if (proc is IDoughKneader kneader) kneader.KneadDough();

            Console.WriteLine();
        }
    }
}

```

Результат работы программы продемонстрирован ниже (см. рисунок 2).



```

[Процессор A] Тип: Настольный, Поддерживает миксер и блендер
Процессор A шинкует продукты.
Процессор A взбивает тесто.
Процессор A делает смузи.

[Процессор B] Тип: Ручной, Поддерживает тестомешалку
Процессор B превращает мясо в фарш.
Процессор B замешивает тесто.

C:\BSUIR IAPT 2 semester MIKALAI KHALAMAU 453503\Programming\Лабораторная работа №9\KitchenProcessorApp\KitchenProcessorApp\bin\Debug\net8.0\KitchenProcessorApp.exe (процесс 19836) завершил работу с кодом 0 (0x0).
Нажмите любую клавишу, чтобы закрыть это окно:

```

Рисунок 2 – Результат работы программы

ВЫВОД

В ходе выполнения лабораторной работы была достигнута цель по разработке приложения с использованием объектно-ориентированных принципов и шаблонов проектирования. Было реализовано семейство классов кухонных процессоров, обладающих общим функционалом (KitchenProcessor) и различным дополнительным поведением, заданным через интерфейсы (IMixer, IBlender, IDoughKneader). Метод GetInfo() реализован по-разному в каждом классе процессора, что позволяет отличать их поведение. Метод Process() делегируется внешним классам (Shredder, Grinder, MeatGrinder) с использованием паттерна "Мост" (Bridge) — это позволяет изменять поведение независимо от типа устройства. Для создания объектов использован паттерн "Строитель" (Builder), реализованный в классах DesktopProcessorBuilder и HandProcessorBuilder. Это позволило удобно и поэтапно создавать объекты с нужной конфигурацией. В классе Program создана коллекция из различных объектов, и для каждого вызваны доступные методы, включая основные (GetInfo, Process) и дополнительные (Mix, Blend, KneadDough). Таким образом, работа продемонстрировала понимание и корректное применение принципов множественного поведения, делегирования и шаблонов проектирования в объектно-ориентированной архитектуре.