Балтийский государственный технический университет
«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра И5 «Информационные системы и программная инженерия»

Лабораторная работа №1
по дисциплине «Программирование на языке высокого уровня»
по теме «Разработка иерархии классов с использованием интерфейсов, абстрактных классов и других механизмов наследования»

Выполнил:
Студент Альков В. С.
Группа И407Б
Вариант 2

Преподаватель:
Кимсанбаев К. А.

Санкт-Петербург
2021 г.

Задача:

Работа состоит из последовательного выполнения трёх заданий: создание класса, иерархии классов и интерфейса. Все задания выполняются последовательно в одном файле с исходным текстом в рамках одного проекта. В отчёте к работе должны быть отдельно отражены результаты выполнения каждого задания.

вариативная часть

Пельмешки! Класс для первой части – хинкали. Варианты свойств: размер, вес, приготовлен (bool изначально false), количество мяса, наличие кинзы, количество складок. Варианты методов: приготовить, съесть, добавить соус,узнать соотношение мяса и теста и наличие кинзы, получить рецепт идеального хинкали (статический)

Вспомогательные функции и классы

Функция чтения параметров для конструирования объектов.

Входные данные: кол-во параметров. Конструкторов 3, принимающих 0, 1, 2 параметра.
Выходные данные: лист прочитанных данных.

```
public static List<Object> readParam(int count)
{
    var list = new List<Object>();
    if (count == 0)
        return list;
    var success = false;
    double res = 0d;
    Console.WriteLine("Имя: ");
    list.Add(Console.ReadLine());
    if (count == 1)
        return list;
    while (!success)
    {
        Console.Clear();
        Console.WriteLine("Кол-во теста: ");
        success = double.TryParse(Console.ReadLine(), out res);
    }
    list.Add(res);
    success = false;
    while (!success)
    {
        Console.Clear();
        Console.WriteLine("Кол-во начинки: ");
        success = double.TryParse(Console.ReadLine(), out res);
    }
    list.Add(res);
    return list;
}
```

Функция чтения целого числа из диапазона.

Входные данные: начало диапазона, конец диапазона, выводимое сообщение
Выходные данные: прочитанное целое число

```
public static int readInt(int p1, int p2, string message = "")
{
```

```
    int input = 0;
    bool success = false;
    while (!success && input < p1 || input > p2)
    {
        Console.Clear();
        Console.Write(message);
        success = int.TryParse(Console.ReadLine(), out input);
    }
    return input;
}
```

Функция, конструирующая объект.

Входные данные: параметр типа T, массив параметров для объекта.
Выходные данные: объект.

```
static public T? Constructor<T>(params Object[] args) where T : class =>
(T?)Activator.CreateInstance(typeof(T), args);
```

Функция выбора объекта из списка.

Входные данные: параметр типа T, список объектов.
Выходные данные: объект.

```
static T? chooseObject<T>(List<T> list) where T : class
{
    if (list.Count < 1)
    {
        Console.Write("Объектов нет");
        return null;
    }
    int input = readInt(1, list.Count, $"Всего объктов: {list.Count}\nВведите номер
объекта: ");
    return list[input - 1];
}
```

Функция, тестирующая методы объекта.

Входные данные: параметр типа T, объект.
Выходные данные: нет.

```
static void methods<T>(T? obj) where T : Khinkali // Dumpling во 2 части, IEatable в 3
части
{
    if (obj == null)
    {
        Console.ReadLine();
        return;
    }
    string? input;
    do
    {
        Console.Clear();
        Console.WriteLine("1. Вес\n2. Соотношение начинки к тесту\n3. Съесть\n4.
Приготовить\n5. Добавить соус\n6. Метод ToString\n7. Имя класса\n8. Назад");
        input = Console.ReadLine();
        switch (input)
        {
            case "1":
                obj.weight();
                break;
            case "2":
```

```
                    obj.ratioFillingToDough();
                    break;
                case "3":
                    obj.eat();
                    break;
                case "4":
                    obj.cook();
                    break;
                case "5":
                    obj.addSouce();
                    break;
                case "6":
                    Console.Write(obj.ToString());
                    break;
                case "7":
                    obj.printClassName();
                    break;
                case "8": break;
                default: Console.Write("Неправильный ввод"); break;
            }
            Console.ReadLine();
        } while (input != "8");
    }
```

Класс пункт меню.

```
public class Item
{
    public List<Item> items = new List<Item>(); //подпункты
    public Item? parent = null; //родитель
    public bool back; //нужно ли возвращаться после выполнения
    public Item? backItem = null; //по умолчанию отсутствует возвращаемый пункт
    public string text; //текст пункта
    public Action? process; //выполняемое действие
    public Item(string t, Action? proc = null, bool b = false, Item? backitem = null)
    {
        backItem = backitem;
        back = b;
        process = proc;
        text = t;
    }
    public Item add(Item newItem)
    {
        newItem.parent = this; //установка родителя
        if (newItem.backItem == null && newItem.back) //если возвращаться надо, но
возвращаемый пункт не задан, то это родитель
            newItem.backItem = newItem.parent;
        items.Add(newItem);
        return newItem;
    }
    public void exec()
    {
        if (process != null)
            process();
    }
}
```

Класс меню.

```
public class Menu : Item
{
    public Item? current; //текущий пункт
```

```
        public Menu(string t, Action? proc = null, bool b = false, Item? backitem = null) :
    base(t, proc, b, backitem) { }
        public void run()
        {
            current = this; //изначально текущий пункт сам объект меню
            int input;
            string print;

            while (current != null)
            {
                Console.Clear();
                current.exec(); //выполнение действия текущего пункта
                if (current == null) //проверка выхода
                    break;
                if (current.back && current.backItem != null) //возвращение, если оно есть
                {
                    current = current.backItem;
                    continue;
                }
                print = "";
                current.items.ForEach(obj => print += obj.text + "\n");
                input = readInt(1, current.items.Count, print);
                current = current.items[input - 1];
            }
        }
        public void exit() => current = null;
    }
```

Часть 1

Класс Хинкали.

```
public class Khinkali
{
    int countFolds;
    const string DefaultName = "Хинкали";
    static string idealRecipe = $"Тесто: 10\nНачинка: 50\nКорица: Да\nСоус: Нет\nКол-во складок:
18";
    string Name { get; set; }
    bool Cooked { get; set; }
    bool Cilantro { get; set; }
    bool Eaten { get; set; }
    bool Souce { get; set; }
    double Dough { get; set; }
    double Filling { get; set; }
    public Khinkali() : this(DefaultName) { }
    public Khinkali(string name) : this(name, 50d, 250d) { }
    public Khinkali(string name, double dough, double filling)
    {
        Name = name;
        Dough = dough;
        Filling = filling;
        Cooked = Cilantro = Eaten = Souce = false;
        countFolds = new Random().Next(20);
        Cilantro = true;
    }
    public void ratioFillingToDough() => Console.WriteLine(100.0 * Math.Abs(Dough - Filling) /
Math.Max(Dough, Filling));
    public void weight() => Console.WriteLine(Dough + Filling);
    public void eat()
    {
        if (!Cooked)
        {
            Console.WriteLine($"{Name} еще не приготовлен");
```

```
            return;
        }
        Console.WriteLine($"{Name} {(Eaten ? "уже был съеден!" : "съеден!")}");
        Eaten = true;
    }
    public virtual void cook()
    {
        Console.WriteLine($"{Name} {(Cooked ? "уже был приготовлен!" : "приготовлен!")}");
        Cooked = true;
    }
    public virtual void addSouce()
    {
        Console.WriteLine($"В {Name} {(Souce ? "уже добавлен соус!" : "добавлен соус!")}");
        Souce = true;
    }
    public void printClassName() => Console.WriteLine(base.GetType().Name);
    public override string ToString() =>
$"Класс:{base.GetType().Name}\nИмя:{Name}\nТесто:{Dough}\nНачинка:{Filling}\nПриготовлен:{(Cooked ?
"Да" : "Нет")}\nКорица:{(Cilantro ? "Да" : "Нет")}\nКол-во складок:{countFolds}\nСоус:{(Souce ? "Да"
: "Нет")}\nСъеден:{(Eaten ? "Да" : "Нет")}";
    public static void printIdealRecipe() => Console.WriteLine(idealRecipe);
}
```

Главная функция.

```
static void Main(string[] args)
{
    Menu main = new Menu("");
    var part1_list = new List<Khinkali>();
    main.add(new Item("1. Создать объект", delegate {
        int input = readInt(1, 3, "1. Конструктор без параметров\n2. Конструктор с параметром\n3.
Конструктор с параметрами\n");
        var obj = Constructor<Khinkali>(readParam(input - 1).ToArray());
        if (obj != null) part1_list.Add(obj);
    }, true));
    main.add(new Item("2. Методы", delegate {
        methods(chooseObject(part1_list));
    }, true));
    main.add(new Item("3. Статический метод", delegate {
        Khinkali.printIdealRecipe();
        Console.ReadLine();
    }, true));
    main.add(new Item("4. Свойства", delegate {
        Console.Write(chooseObject(part1_list));
        Console.ReadLine();
    }, true));
    main.add(new Item("5. Выйти", delegate { main.exit(); }));
    main.run();
}
```

Результат работы программы

```
1. Создать объект
2. Методы
3. Статический метод
4. Свойства
5. Выйти
```

1. Создать объект

```
1. Конструктор без параметров
2. Конструктор с параметром
3. Конструктор с параметрами
```

```
1. Конструктор без параметров
2. Конструктор с параметром
3. Конструктор с параметрами
3
Имя:
123
```

```
Кол-во теста:
56
```

```
Кол-во начинки:
78
```

2. Методы

```
Всего объктов: 1
Введите номер объекта: 1
```

```
1. Вес
2. Соотношение начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Имя класса
8. Назад
```

```
1. Вес
2. Соотношение начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Имя класса
8. Назад
1
134
```

```
1. Вес
2. Соотношение начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Имя класса
8. Назад
3
123 еще не приготовлен
```

```
1. Вес
2. Соотношение начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Имя класса
8. Назад
4
123 приготовлен!
```

```
1. Вес
2. Соотношение начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Имя класса
8. Назад
3
123 съеден!
```

```
1. Вес
2. Соотношение начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Имя класса
8. Назад
7
Khinkali
```

3. Статический метод

```
Тесто: 10
Начинка: 50
Корица: Да
Соус: Нет
Кол-во складок: 18
```

4. Свойства

```
Всего объктов: 1
Введите номер объекта: 1
Класс:Khinkali
Имя:123
Тесто:56
Начинка:78
Приготовлен:Да
Корица:Да
Кол-во складок:4
Соус:Нет
Съеден:Да
```

Выводы

В данной части задания был спроектирован класс Хинкали со свойствами, методами, в том числе статическим.

Часть 2

Классы

```
public abstract class Dumpling
{
    private const string DefaultName = "Дамплинг";
    protected string Name { get; set; }
    protected bool Cooked { get; set; }
    protected bool Cilantro { get; set; }
    protected bool Eaten { get; set; }
    protected bool Souce { get; set; }
    protected double Dough { get; set; }
    protected double Filling { get; set; }
    public Dumpling() : this(DefaultName) { }
    public Dumpling(string name) : this(name, 50d, 300d) { }
    public Dumpling(string name, double dough, double filling)
    {
        Name = name;
        Dough = dough;
        Filling = filling;
        Cooked = Cilantro = Eaten = Souce = false;
    }
    public void ratioFillingToDough() => Console.WriteLine(100.0 * Math.Abs(Dough - Filling) /
Math.Max(Dough, Filling));
    public void weight() => Console.WriteLine(Dough + Filling);
    public void eat()
    {
        if (!Cooked)
        {
            Console.WriteLine($"{Name} еще не приготовлен");
            return;
        }
        Console.WriteLine($"{Name} {(Eaten ? "уже был съеден!" : "съеден!")}");
        Eaten = true;
    }
    public virtual void cook()
    {
        Console.WriteLine($"{Name} {(Cooked ? "уже был приготовлен!" : "приготовлен!")}");
        Cooked = true;
    }
    public virtual void addSouce()
    {
```

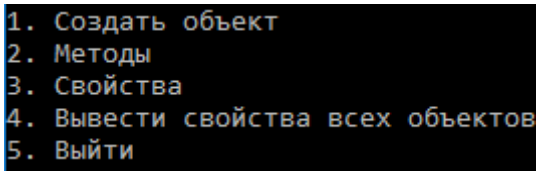
```
        Console.WriteLine($"В {Name} {(Souce ? "уже добавлен соус!" : "добавлен соус!")}");
        Souce = true;
    }
    public override string ToString() =>
$"Класс:{base.GetType().Name}\nИмя:{Name}\nТесто:{Dough}\nНачинка:{Filling}\nПриготовлен:{(Cooked ?
"Да" : "Нет")}\nКорица:{(Cilantro ? "Да" : "Нет")}\nСоус:{(Souce ? "Да" : "Нет")}\nСъеден:{(Eaten ?
"Да" : "Нет")}";
    public void printClassName() => Console.WriteLine(base.GetType().Name);
}
public class Khinkali : Dumpling
{
    private int countFolds;
    private const string DefaultName = "Хинкали";
    public static string idealRecipe = $"Тесто: 10\nНачинка: 50\nКорица: Да\nСоус: Нет\nКол-во
складок: 18";
    public Khinkali() : this(DefaultName) { }
    public Khinkali(string name) : this(name, 50d, 250d) { }
    public Khinkali(string name, double dough, double filling) : base(name, dough, filling)
    {
        countFolds = new Random().Next(20);
        Cilantro = true;
    }
    public override string ToString() =>
$"Класс:{base.GetType().Name}\nИмя:{Name}\nТесто:{Dough}\nНачинка:{Filling}\nПриготовлен:{(Cooked ?
"Да" : "Нет")}\nКорица:{(Cilantro ? "Да" : "Нет")}\nКол-во складок:{countFolds}\nСоус:{(Souce ? "Да"
: "Нет")}\nСъеден:{(Eaten ? "

    public static void printIdealRecipe() => Console.WriteLine(idealRecipe);

}
public class Pelmen : Dumpling
{
    private const string DefaultName = "Пельмени";
    public Pelmen() : this(DefaultName) { }
    public Pelmen(string name) : this(name, 10d, 60d) { }
    public Pelmen(string name, double dough, double filling) : base(name, dough, filling) => Souce =
true;
    public override void addSouce()
    {
        Console.Write("Происходит добавление майонеза...");
        base.addSouce();
    }
    public override void cook()
    {
        if (!Cooked)
        {
            Console.WriteLine("Начинаю процесс приготовления пелеьменей... Тесто замешано...
Раскатано...\nСборка начата... Лепка окончена...\nПроцесс варки... Завершен...\nПельмени на
тарелке!\n");
            Cooked = true;
        }
        else
            base.cook();
    }
}
public class Varenik : Dumpling
{
    private const string DefaultName = "Вареник";
    public Varenik() : this(DefaultName) { }
    public Varenik(string name) : this(name, 5d, 70d) { }
    public Varenik(string name, double dough, double filling) : base(name, dough, filling) => Souce
= Cilantro = true;
    public override void cook()
    {
```

```
        if (!Cooked)
        {
            Console.WriteLine("Начинаю процесс готовки вареников.. Подготовка теста... Подготовка
картошки для начинки...\nСборка... Процесс варки... Завершен...\nВареники готовы!");
            Cooked = true;
        }
        else
            base.cook();
    }
}
public sealed class Wonton : Dumpling
{
    private const string DefaultName = "Вонтон";
    public Wonton() : this(DefaultName) { }
    public Wonton(string name) : this(name, 25d, 150d) { }
    public Wonton(string name, double dough, double filling) : base(name, dough, filling) { }
}
```

Главная функция

```
public delegate T CounstructorHolder<T>(params Object[] args); //делегат для упрощения создания
объектов

static void Main(string[] args)
{
    CounstructorHolder<Dumpling?>[] counstructors = { Constructor<Khinkali>, Constructor<Pelmen>,
Constructor<Varenik>, Constructor<Wonton> };

    Menu main = new Menu("");

    var part2_list = new List<Dumpling>();
    main.add(new Item("1. Создать объект", delegate {
        int input1 = readInt(1, 4, "1. Хинкали\n2. Пельмени\n3. Вареники\n4. Вонтон\n");
        int input2 = readInt(1, 3, "1. Конструктор без параметров\n2. Конструктор с параметром\n3.
Конструктор с параметрами\n");
        var obj = counstructors[input1 - 1](readParam(input2 - 1).ToArray());
        if (obj != null) part2_list.Add(obj);
    }, true));
    main.add(new Item("2. Методы", delegate {
        methods(chooseObject(part2_list));
    }, true));
    main.add(new Item("3. Свойства", delegate {
        Console.Write(chooseObject(part2_list));
        Console.ReadLine();
    }, true));
    main.add(new Item("4. Вывести свойства всех объектов", delegate {
        if (part2_list.Count < 1)
            Console.WriteLine("Объектов нет");
        else
            part2_list.ForEach(obj => Console.WriteLine(obj + "\n"));
        Console.ReadLine();
    }, true));
    main.add(new Item("5. Выйти", delegate { main.exit(); }));

    main.run();
}
```

Результат работы программы



Создам по объекту каждого типа и выведу свойства всех объектов.

1. Создать объект

```
1. Хинкали
2. Пельмени
3. Вареники
4. Вонтон
```

4. Вывести свойства всех объектов

```
Класс:Khinkali
Имя:Хинкали
Тесто:50
Начинка:250
Приготовлен:Нет
Корица:Да
Кол-во складок:9
Соус:Нет
Съеден:Нет

Класс:Pelmen
Имя:Пельмени
Тесто:10
Начинка:60
Приготовлен:Нет
Корица:Нет
Соус:Да
Съеден:Нет

Класс:Varenik
Имя:Вареник
Тесто:5
Начинка:70
Приготовлен:Нет
Корица:Да
Соус:Да
Съеден:Нет

Класс:Wonton
Имя:Вонтон
Тесто:25
Начинка:150
Приготовлен:Нет
Корица:Нет
Соус:Нет
Съеден:Нет
```

Выводы

В этой части задания была разработана иерархия классов, было выполнено переопределение методов по необходимости. Благодаря тому что классы потомки единственного базового класса, их можно хранить в рамках базового вместе, а также к ним можно обращаться через базовый класс. Это упрощает написание кода, так как нет необходимости писать функции под каждый класс отдельно.

Часть 3

Итоговый вариант программы

```
using System;
using System.Collections.Generic;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {

            #nullable enable
            ConstructorHolder<Dumpling?>[] constructors = { Constructor<Khinkali>,
Constructor<Pelmen>, Constructor<Varenik>, Constructor<Wonton> };
            ConstructorHolder<IEatable?>[] constructors2 = { Constructor<Khinkali>,
Constructor<Pelmen>, Constructor<Varenik>, Constructor<Wonton>, Constructor<Pizza> };
            #nullable disable

            Menu main = new Menu("");
            var part1 = main.add(new Item("1. Часть 1"));
            var part2 = main.add(new Item("2. Часть 2"));
            var part3 = main.add(new Item("3. Часть 3"));
            var exit = main.add(new Item("4. Выйти", delegate { main.exit(); }));

            var part1_list = new List<Khinkali>();
            part1.add(new Item("1. Создать объект",  delegate {
```

```
                int input = readInt(1, 3, "1. Конструктор без параметров\n2. Конструктор с
параметром\n3. Конструктор с параметрами\n");
                var obj = Constructor<Khinkali>(readParam(input - 1).ToArray());
                if (obj != null) part1_list.Add(obj);
            }, true));
            part1.add(new Item("2. Методы", delegate {
                methods(chooseObject(part1_list));
            }, true));
            part1.add(new Item("3. Статический метод", delegate {
                Khinkali.printIdealRecipe();
                Console.ReadLine();
            }, true));
            part1.add(new Item("4. Свойства", delegate {
                Console.Write(chooseObject(part1_list));
                Console.ReadLine(); }, true));
            part1.add(new Item("5. Назад", null, true, main));

            var part2_list = new List<Dumpling>();
            part2.add(new Item("1. Создать объект", delegate {
                int input1 = readInt(1, 4, "1. Хинкали\n2. Пельмени\n3. Вареники\n4. Вонтон\n");
                int input2 = readInt(1, 3, "1. Конструктор без параметров\n2. Конструктор с
параметром\n3. Конструктор с параметрами\n");
                var obj = counstructors[input1-1](readParam(input2 - 1).ToArray());
                if (obj != null) part2_list.Add(obj);
            }, true));
            part2.add(new Item("2. Методы", delegate {
                methods(chooseObject(part2_list));
            }, true));
            part2.add(new Item("3. Свойства", delegate {
                Console.Write(chooseObject(part2_list));
                Console.ReadLine();
            }, true));
            part2.add(new Item("4. Вывести свойства всех объектов", delegate {
                if (part2_list.Count < 1)
                    Console.WriteLine("Объектов нет");
                else
                    part2_list.ForEach(obj => Console.WriteLine(obj + "\n"));
                Console.ReadLine();
            }, true));
            part2.add(new Item("5. Назад", null, true, main));

            var part3_list = new List<IEatable>();
            part3.add(new Item("1. Создать объект", delegate {
                int input1 = readInt(1, 5, "1. Хинкали\n2. Пельмени\n3. Вареники\n4. Вонтон\n5.
Пицца\n");
                int input2 = readInt(1, 3, "1. Конструктор без параметров\n2. Конструктор с
параметром\n3. Конструктор с параметрами\n");
                var obj = counstructors2[input1 - 1](readParam(input2 - 1).ToArray());
                if (obj != null) part3_list.Add(obj);
            }, true));
            part3.add(new Item("2. Методы", delegate {
                methods(chooseObject(part3_list));
            }, true));
            part3.add(new Item("3. Свойства", delegate {
                Console.Write(chooseObject(part3_list));
                Console.ReadLine();
            }, true));
            part3.add(new Item("4. Вывести свойства всех объектов", delegate {
                    if (part3_list.Count < 1)
                        Console.WriteLine("Объектов нет");
                    else
                        part3_list.ForEach(obj => Console.WriteLine(obj + "\n"));
                Console.ReadLine();
            }, true));
```

```
            part3.add(new Item("5. Общая функция", delegate {
                cookAndEat(chooseObject(part3_list));
                Console.ReadLine();
            }, true));
            part3.add(new Item("6. Назад", null, true, main));

            main.run();

        }

        interface IEatable
        {
            public void ratioFillingToDough();
            void weight();
            void eat();
            void cook();
            void addSouce();
            string ToString();
            void printClassName();
        }
        abstract class Dumpling : IEatable
        {
            private const string DefaultName = "Дамплинг";
            protected string Name { get; set; }
            protected bool Cooked { get; set; }
            protected bool Cilantro { get; set; }
            protected bool Eaten { get; set; }
            protected bool Souce { get; set; }
            protected double Dough { get; set; }
            protected double Filling { get; set; }
            public Dumpling() : this(DefaultName) { }
            public Dumpling(string name) : this(name, 50d, 300d) { }
            public Dumpling(string name, double dough, double filling)
            {
                Name = name;
                Dough = dough;
                Filling = filling;
                Cooked = Cilantro = Eaten = Souce = false;
            }
            public void ratioFillingToDough() => Console.WriteLine(100.0 * Math.Abs(Dough - Filling)
/ Math.Max(Dough, Filling));
            public void weight() => Console.WriteLine(Dough + Filling);
            public void eat()
            {
                if (!Cooked)
                {
                    Console.WriteLine($"{Name} еще не приготовлен");
                    return;
                }
                Console.WriteLine($"{Name} {(Eaten ? "уже был съеден!" : "съеден!")}");
                Eaten = true;
            }
            public virtual void cook()
            {
                Console.WriteLine($"{Name} {(Cooked ? "уже был приготовлен!" : "приготовлен!")}");
                Cooked = true;
            }
            public virtual void addSouce()
            {
                Console.WriteLine($"В {Name} {(Souce ? "уже добавлен соус!" : "добавлен соус!")}");
                Souce = true;
            }
            public override string ToString() =>
$"Класс:{base.GetType().Name}\nИмя:{Name}\nТесто:{Dough}\nНачинка:{Filling}\nПриготовлен:{(Cooked ?
```

```
"Да" : "Нет")}\nКорица:{(Cilantro ? "Да" : "Нет")}\nСоус:{(Souce ? "Да" : "Нет")}\nСъеден:{(Eaten ?
"Да" : "Нет")}";
            public void printClassName() => Console.WriteLine(base.GetType().Name);
        }
        class Khinkali : Dumpling
        {
            private int countFolds;
            private const string DefaultName = "Хинкали";
            public static string idealRecipe = $"Тесто: 10\nНачинка: 50\nКорица: Да\nСоус: Нет\nКол-
во складок: 18";
            public Khinkali() : this(DefaultName) { }
            public Khinkali(string name) : this(name, 50d, 250d) { }
            public Khinkali(string name, double dough, double filling) : base(name, dough, filling)
            {
                countFolds = new Random().Next(20);
                Cilantro = true;
            }
            public override string ToString() =>
$"Класс:{base.GetType().Name}\nИмя:{Name}\nТесто:{Dough}\nНачинка:{Filling}\nПриготовлен:{(Cooked ?
"Да" : "Нет")}\nКорица:{(Cilantro ? "Да" : "Нет")}\nКол-во складок:{countFolds}\nСоус:{(Souce ? "Да"
: "Нет")}\nСъеден:{(Eaten ? "Да" : "Нет")}";

            public static void printIdealRecipe() => Console.WriteLine(idealRecipe);

        }
        class Pelmen : Dumpling
        {
            private const string DefaultName = "Пельмени";
            public Pelmen() : this(DefaultName) { }
            public Pelmen(string name) : this(name, 10d, 60d) { }
            public Pelmen(string name, double dough, double filling) : base(name, dough, filling) =>
Souce = true;
            public override void addSouce()
            {
                Console.Write("Происходит добавление майонеза...");
                base.addSouce();
            }
            public override void cook()
            {
                if (!Cooked)
                {
                    Console.WriteLine("Начинаю процесс приготовления пелеьменей... Тесто замешано...
Раскатано...\nСборка начата... Лепка окончена...\nПроцесс варки... Завершен...\nПельмени на
тарелке!\n");
                    Cooked = true;
                }
                else
                    base.cook();
            }
        }
        class Varenik : Dumpling
        {
            private const string DefaultName = "Вареник";
            public Varenik() : this(DefaultName) { }
            public Varenik(string name) : this(name, 5d, 70d) { }
            public Varenik(string name, double dough, double filling) : base(name, dough, filling)
=> Souce = Cilantro = true;
            public override void cook()
            {
                if (!Cooked)
                {
                    Console.WriteLine("Начинаю процесс готовки вареников.. Подготовка теста...
Подготовка картошки для начинки...\nСборка... Процесс варки... Завершен...\nВареники готовы!");
                    Cooked = true;
```

```
            }
            else
                base.cook();
        }
    }

    sealed class Wonton : Dumpling
    {
        private const string DefaultName = "Вонтон";
        public Wonton() : this(DefaultName) { }
        public Wonton(string name) : this(name, 25d, 150d) { }
        public Wonton(string name, double dough, double filling) : base(name, dough, filling) {
}
    }
    class Pizza : IEatable
    {
        private int countParts = 6;
        private const string DefaultName = "Пицца";
        protected string Name { get; set; }
        protected bool Cooked { get; set; }
        protected bool Eaten { get; set; }
        protected bool Souce { get; set; }
        protected double Dough { get; set; }
        protected double Filling { get; set; }
        public Pizza() : this(DefaultName) { }
        public Pizza(string name) : this(name, 50d, 300d) { }
        public Pizza(string name, double dough, double filling)
        {
            Name = name;
            Dough = dough;
            Filling = filling;
            Cooked = Eaten = Souce = false;
        }
        public void ratioFillingToDough() => Console.WriteLine(100.0 * Math.Abs(Dough - Filling)
/ Math.Max(Dough, Filling));
        public void weight() => Console.WriteLine(Dough + Filling);
        public void eat()
        {
            if (!Cooked)
            {
                Console.WriteLine($"{Name} еще не приготовлен!");
                return;
            }
            if (!Eaten)
            {
                countParts--;
                if (countParts < 1)
                    Eaten = true;
                Console.WriteLine($"Съеден кусок {Name}! Осталось {countParts} кусков!!");
            }
            else
                Console.WriteLine($"Все куски пиццы {Name} были съедены!!");
        }
        public void cook()
        {
            Console.WriteLine($"{Name} {(Eaten ? "уже был приготовлен!" : "приготовлен!")}");
            Cooked = true;
        }
        public void addSouce()
        {
            Console.WriteLine($"В {Name} {(Eaten ? "уже добавлен соус!" : "добавлен соус!" )}");
            Souce = true;
        }
```

```
            public override string ToString() =>
$"Класс:{base.GetType().Name}\nИмя:{Name}\nТесто:{Dough}\nНачинка:{Filling}\nПриготовлен:{(Cooked ?
"Да" : "Нет")}\nСоус:{(Souce ? "Да" : "Нет")}\nСъедена:{(Eaten ? "Да" : "Нет")}";
            public void printClassName() => Console.WriteLine(base.GetType().Name);
        }
        public static int readInt(int p1, int p2, string message = "")
        {
            int input = 0;
            bool success = false;
            while (!success && input < p1 || input > p2)
            {
                Console.Clear();
                Console.Write(message);
                success = int.TryParse(Console.ReadLine(), out input);
            }
            return input;
        }
        static void cookAndEat(IEatable obj)
        {
            if (obj != null)
            {
                obj.cook();
                obj.eat();
            }
        }
        public delegate T CounstructorHolder<T>(params Object[] args);
        #nullable enable
        static public T? Constructor<T>(params Object[] args) where T : class =>
(T?)Activator.CreateInstance(typeof(T), args);
        static T? chooseObject<T>(List<T> list) where T : class
        {
            if (list.Count < 1)
            {
                Console.Write("Объектов нет");
                return null;
            }
            int input = readInt(1, list.Count, $"Всего объктов: {list.Count}\nВведите номер объекта:
");
            return list[input - 1];
        }
        static void methods<T>(T? obj) where T: class, IEatable
        {
            if (obj == null)
            {
                Console.ReadLine();
                return;
            }
            string input;
            do
            {
                Console.Clear();
                Console.WriteLine("1. Вес\n2. Соотношение начинки к тесту\n3. Съесть\n4.
Приготовить\n5. Добавить соус\n6. Метод ToString\n7. Имя класса\n8. Назад");
                input = Console.ReadLine();
                switch (input)
                {
                    case "1":
                        obj.weight();
                        break;
                    case "2":
                        obj.ratioFillingToDough();
                        break;
                    case "3":
                        obj.eat();
```

```
                break;
            case "4":
                obj.cook();
                break;
            case "5":
                obj.addSouce();
                break;
            case "6":
                Console.Write(obj.ToString());
                break;
            case "7":
                obj.printClassName();
                break;
            case "8": break;
            default: Console.Write("Неправильный ввод"); break;
        }
        Console.ReadLine();
    } while (input != "8");
}
static public List<Object> readParam(int count)
{
    var list = new List<Object>();
    if (count == 0)
        return list;
    var success = false;
    double res = 0d;
    Console.WriteLine("Имя: ");
    list.Add(Console.ReadLine());
    if (count == 1)
        return list;
    while (!success)
    {
        Console.Clear();
        Console.WriteLine("Кол-во теста: ");
        success = double.TryParse(Console.ReadLine(), out res);
    }
    list.Add(res);
    success = false;
    while (!success)
    {
        Console.Clear();
        Console.WriteLine("Кол-во начинки: ");
        success = double.TryParse(Console.ReadLine(), out res);
    }
    list.Add(res);
    return list;
}
class Item
{
    public List<Item> items = new List<Item>();
    public Item? parent = null;
    public bool back;
    public Item? backItem = null;
    public string text;
    public Action? process;
    public Item(string t, Action? proc = null, bool b = false, Item? backitem = null)
    {
        backItem = backitem;
        back = b;
        process = proc;
        text = t;
    }
    public Item add(Item newItem)
    {
```

```
                newItem.parent = this;
                if (newItem.backItem == null && newItem.back)
                    newItem.backItem = newItem.parent;
                items.Add(newItem);
                return newItem;
            }
            public void exec()
            {
                if (process != null)
                    process();
            }

        }
        class Menu : Item
        {
            public Item? current;
            public Menu(string t, Action? proc = null, bool b = false, Item? backitem = null) :
base(t, proc, b, backitem) { }
            public void run()
            {
                current = this;
                int input;
                string print;

                while (current != null)
                {
                    Console.Clear();
                    current.exec();
                    if (current == null)
                        break;
                    if (current.back && current.backItem != null)
                    {
                        current = current.backItem;
                        continue;
                    }
                    print = "";
                    current.items.ForEach(obj => print += obj.text + "\n");
                    input = readInt(1, current.items.Count, print);
                    current = current.items[input - 1];
                }
            }
            public void exit() => current = null;
        }
        #nullable disable
    }
}
```

Результат работы программы

```
1. Часть 1
2. Часть 2
3. Часть 3
4. Выйти
```

3. Часть 3

```
1. Создать объект
2. Методы
3. Свойства
4. Вывести свойства всех объектов
5. Общая функция
6. Назад
```

Создам по объекту каждого типа, выведу свойства всех объектов, вызову общую функцию

1. Создать объект

```
1. Хинкали
2. Пельмени
3. Вареники
4. Вонтон
5. Пицца
```

4. Вывести свойства всех объектов

```
Класс:Khinkali
Имя:Хинкали
Тесто:50
Начинка:250
Приготовлен:Нет
Корица:Да
Кол-во складок:1
Соус:Нет
Съеден:Нет
```

```
Класс:Pelmen
Имя:Пельмени
Тесто:10
Начинка:60
Приготовлен:Нет
Корица:Нет
Соус:Да
Съеден:Нет
```

```
Класс:Varenik
Имя:Вареник
Тесто:5
Начинка:70
Приготовлен:Нет
Корица:Да
Соус:Да
Съеден:Нет
```

```
Класс:Wonton
Имя:Вонтон
Тесто:25
Начинка:150
Приготовлен:Нет
Корица:Нет
Соус:Нет
Съеден:Нет
```

```
Класс:Pizza
Имя:Пицца
Тесто:50
Начинка:300
Приготовлен:Нет
Соус:Нет
Съедена:Нет
```

5. Общая функция

```
Всего объктов: 5
Введите номер объекта: 5
Пицца приготовлен!
Съеден кусок Пицца! Осталось 5 кусков!!
```

```
Всего объктов: 5
Введите номер объекта: 3
Начинаю процесс готовки вареников.. Подготовка теста... Подготовка картошки для начинки...
Сборка... Процесс варки... Завершен...
Вареники готовы!
Вареник съеден!
```

```
Всего объктов: 5
Введите номер объекта: 2
Начинаю процесс приготовления пелеьменей... Тесто замешано... Раскатано...
Сборка начата... Лепка окончена...
Процесс варки... Завершен...
Пельмени на тарелке!

Пельмени съеден!
```

Выводы

В этой части был реализован интерфейс. Интерфейс позволяет связать объекты, которые не являются наследником определенного базового класса. Были реализованы такие возможности как:

1. Вывод веса
2. Вывод соотношения начинки к тесту
3. Съесть
4. Приготовить
5. Добавить соус
6. Метод ToString
7. Вывод имени класса