

Лабораторная работа №1

Задание 1. Для создания нового проекта выполните следующие действия (см. Приложение 1):

1. Создайте новый проект: File ► New Project, либо Create Project в окне Start Page
2. В окне New project в левой части выберите Visual C# Projects, в правой – пункт Console Application
3. В поле Name введите имя проекта, в поле Location – место его сохранения на диске
4. Ознакомьтесь с основными окнами среды.
5. Рассмотрите каждую строку заготовки программы.
6. Наберите приведенный пример программы (Листинг 1). Вставьте свои значения соответствующих типов в пропущенных местах операторов.

Листинг 1

```
using System;
namespace matemat1
{
    class Program
    {
        public static void Main(string[] args)
        //после знака = для x,y,z вставьте Ваши данные
        {
            int x= ;
            double y= ;
            Console.WriteLine("x: " + x);
            double b = ;
            Console.WriteLine ("y: " + y);
        }
    }
}
```

7. Сохраните весь проект на диске: File ► Save All
 8. Для выполнения программы: Debug ► Start Without Debugging
- Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr1_1
{
    internal class Program
    {
        public static void Main(string[] args)
```

```

    {
        int x = 18;
        double y = 3;
        Console.WriteLine("x: " + x);
        double b = 0;
        Console.WriteLine("y: " + y);
    }
}

```

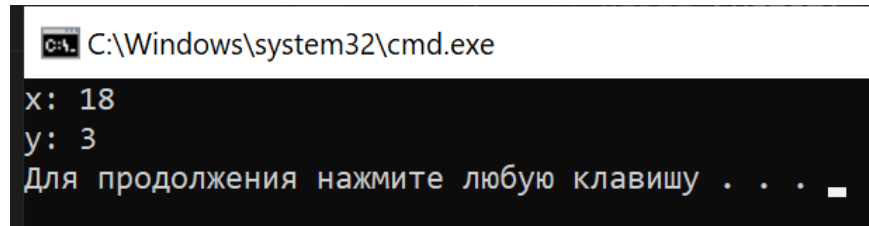


Рисунок 1 – Полученный результат

Задание 2. Создайте новое консольное приложение для решения задачи.

Введите вещественные числа x , y , z из области допустимых значений исходных данных. Для преобразования к числовой форме используйте класс `Convert` и метод `Parse`. Вычислите a , b . Результаты выведите на экран с использованием формата и шаблонов.

$$3. a = (1 + y)^2 \frac{x^2 + 4}{e^{-x} + x^2 + 4}; b = \frac{1}{x^4/2 + \sin^4 z + 1};$$

Рисунок 2 – Решаемые примеры

Код программы:

```

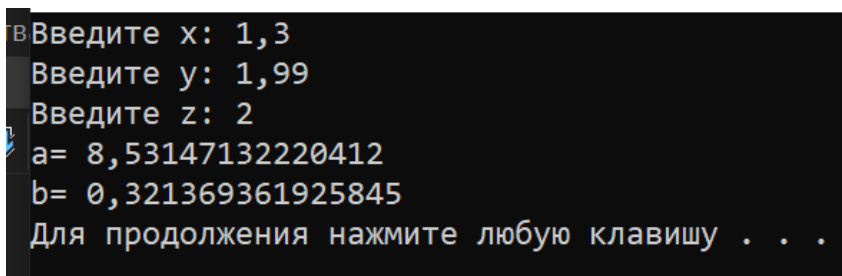
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr1_2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Введите x: ");
            double x = double.Parse(Console.ReadLine());

```

```

Console.Write("Введите y: ");
double y = double.Parse(Console.ReadLine());
Console.Write("Введите z: ");
double z = double.Parse(Console.ReadLine());
double a = Math.Pow((1 + y),2) * ((Math.Pow(x,2)+4)/(Math.Exp(-x)+
Math.Pow(x, 2) + 4));
Console.WriteLine("a= {0}", a);
double b= 1/(Math.Pow(x,4)/2+ Math.Pow(Math.Sin(z),4)+1);
Console.WriteLine("b= {0}", b);}}}

```



```

ВВведите x: 1,3
Введите y: 1,99
Введите z: 2
a= 8,53147132220412
b= 0,321369361925845
Для продолжения нажмите любую клавишу . . .

```

Рисунок 3 – Полученный результат

Контрольные вопросы:

1. Основные принципы технологии .NET.

– Платформенная независимость: Приложения .NET могут выполняться на различных операционных системах благодаря использованию среды исполнения CLR (Common Language Runtime).

– Безопасность типов: Все операции проверяются на этапе компиляции и выполнения, чтобы избежать ошибок, связанных с неправильной работой с памятью.

– Многоязыковая поддержка: Поддержка множества языков программирования через общий промежуточный язык CIL (Common Intermediate Language), который транслируется в машинный код на этапе выполнения.

– Унифицированная библиотека классов: .NET Framework включает обширную библиотеку классов, упрощающую разработку приложений.

2. Что представляет собой платформа Visual Studio.NET?

Visual Studio — это интегрированная среда разработки (IDE), предназначенная для создания приложений на платформе .NET. Она поддерживает различные языки программирования, такие как C#, [VB.NET](#), F#, C++ и другие. Встроенный редактор кода, инструменты отладки, система управления версиями и другие функции делают её мощным инструментом для разработчиков.

3. Как создать консольное приложение?

Чтобы создать консольное приложение в Visual Studio:

1 Откройте Visual Studio.

2 Выберите «Файл» → «Создать» → «Проект».

3 Найдите шаблон «Консольное приложение (.NET)» и выберите его.

4 Задайте имя проекта и нажмите «ОК».

4. Принципы объектно-ориентированного программирования.

– Инкапсуляция – скрывание реализации класса и предоставление интерфейсов для взаимодействия с ним;

– Наследование – возможность создавать новые классы на основе существующих, расширяя их функциональность;

– Полиморфизм – способность объектов разных классов реагировать на одинаковые вызовы методов по-разному.

5. Литералы. Как определяются типы литералов?

Литералы — это фиксированные значения, которые записываются непосредственно в исходном коде. Типы литералов в C# определяются автоматически на основании их формы записи:

– Целочисленные литералы: «int», «long» (например, «123»).

– Дробные литералы: «float», «double» (например, «3,14»).

– Символьные литералы: «char» (например, «A»).

– Строковые литералы: «string» (например, «Hello»)

6. Какие типы относятся к встроенным?

Встроенными типами являются базовые типы данных, предоставляемые языком C#:

– «bool» логический тип (true/false).

– «byte», «sbyte» 8-битные целые числа.

– «short», «ushort» 16-битные целые числа.

– «int», «uint» 32-битные целые числа.

– «long», «ulong» 64-битные целые числа.

– «float», «double» числовые типы с плавающей точкой.

– «decimal»: Высокоточный числовой тип.

– «char» символ Unicode.

– «object» базовый класс всех типов.

– «string» последовательность символов Unicode.

7. Чем отличаются типы-значения и ссылочные типы?

Типы значений хранятся непосредственно в стеке памяти и содержат сами данные. Примеры: int, struct.

Ссылочные типы хранят ссылку на область памяти, где находятся данные. Примеры: class, interface, delegate. Когда вы присваиваете один ссылочный тип другому, копируется ссылка, а не сам объект.

8. Какие типы числовых переменных имеются?

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 4 |

Числовые переменные делятся на два вида:

- Целочисленные: byte, sbyte, short, ushort, int, uint, long, ulong.
- Дробные: float, double, decimal.

9. Что такое объявление и инициализация?

Объявление – создание переменной с указанием её типа.

Инициализация – присвоение начального значения переменной.

10. Для чего используется упаковка и распаковка?

Упаковка (boxing) — процесс преобразования типа-значения в тип-объект (например, упаковать int в объект типа object).

Распаковка (unboxing) — преобразование объекта обратно в тип-значение.

11. Как в C# выполняется преобразование типа?

Преобразование может быть неявным (например, от меньшего к большему типу) или явным (например, (int)3.14). Также используется метод Convert.

12. Как осуществляется консольный ввод?

Используется класс Console. Для чтения строки используется метод Console.ReadLine(), для чтения символа — метод Console.Read().

13. Чем отличаются методы Read и ReadLine?

Метод ReadLine считывает данные и очищает буфер. Метод Read считывает из буфера только один символ, и в отличие от ReadLine, не очищает буфер, поэтому следующий после него ввод будет выполняться с того места, на котором закончился предыдущий, т.е. будет читаться код клавиши. Поэтому необходимо прочесть остаток строки методом ReadLine().

14. Как обеспечить вывод данных на экран?

Для вывода данных на экран используется метод Console.WriteLine() или Console.Write().

15. Для чего предназначен и как используется форматный вывод данных?

Форматный вывод позволяет контролировать представление данных (например, количество знаков после запятой). Используется с помощью метода Console.WriteLine() с форматированием.

16. Каковы основные правила использования стандартных функций?

Следует соблюдать синтаксис вызова функции, передавать необходимые параметры, учитывать область видимости переменных и обрабатывать возможные исключения.

17. Основные приемы работы в среде разработки Visual Studio.NET:

- Как создать консольное приложение?

Откройте Visual Studio. Выберите «Файл» → «Создать» → «Проект». Найдите шаблон «Консольное приложение (.NET)» и выберите его. Задайте имя проекта и нажмите «ОК».

– Как сохранить проект с заданным именем?

Выберите «Файл» → «Сохранить как...» и укажите имя.

– Как загрузить проект?

Выберите «Файл» → «Открыть» → «Проект/Решение» и выберите файл проекта.

– Как выполнить отладку программы?

Нажмите F5 или выберите «Отладка» → «Запустить отладку».

– Как откомпилировать и выполнить программу?

Нажмите Ctrl + F5 для компиляции и выполнения без отладки.

– Как просмотреть результаты выполнения программы?

Результаты выводятся в окне консоли или в окне вывода Visual Studio.

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 6 |

Лабораторная работа № 2

Объявление и определение методов

Задание 1. Создайте проект для решения задачи: На экран выводить исходные данные и результаты. В работе использовать только стандартные типы: числовые, символьный и булевский.

6. Напишите программу, которая выполняет конверсию из сантиметров в дюймы и футы и наоборот. При вводе величин укажите единицу измерения – 'i' для дюймов, 'с' для см., f – для футов.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lr2_1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Введите единицу измерения (i - дюймы, с - см, f - футы): ");
            char ed = char.Parse(Console.ReadLine());
            Console.WriteLine("Введите значение: ");
            double znach = double.Parse(Console.ReadLine());
            if (ed == 'с') // Сантиметры
            {
                double dum = znach / 2.54;
                double fut = dum / 12;
                Console.WriteLine("{0} см = {1} дюймов = {2} футов", znach, dum, fut);
            }
            else if (ed == 'i') // Дюймы
            {
                double cm = znach * 2.54;
                double fut = znach / 12;
                Console.WriteLine("{0} дюймов = {1} см = {2} футов", znach, cm, fut);
            }
            else if (ed == 'f') // Футы
            {
                double fut = znach * 12;
                double cm = fut / 2.54;
                Console.WriteLine("{0} футов = {1} дюймов = {2} см", znach, fut, cm);
            }
        }
    }
}
```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 7 |

```

double cm = znach * 30.48;
double dum = znach * 12;
Console.WriteLine("{0} футов = {1} см = {2} дюймов", znach, cm, dum);
}
else
{
    Console.WriteLine("Неизвестная единица измерения. Используйте 'i', 'c'
или 'f.'");
}
}}}

```

```

Введите единицу измерения (i - дюймы, c - см, f - футы):
c
Введите значение:
10
10 см = 3,93700787401575 дюймов = 0,328083989501312 футов
Для продолжения нажмите любую клавишу . . .

```

Рисунок 4 – Полученный результат

Задание 2. Напишите функции в виде методов. Напишите тестирующую программу с выдачей результатов на экран.

6. Найти первый элемент, больший K, последовательностей {x} и {y}, определяемых рекуррентными соотношениями:

$$x_i = x_{i-1} + 2y_{i-1}$$

$$y_i = y_{i-1} - 2x_{i-1}$$

$$x_1 = 1, y_1 = 0.$$

Листинг программы:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lr2_2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Введите значение K:");
            double k = double.Parse(Console.ReadLine());
            FindFirst(k);
        }
    }
}

```



```

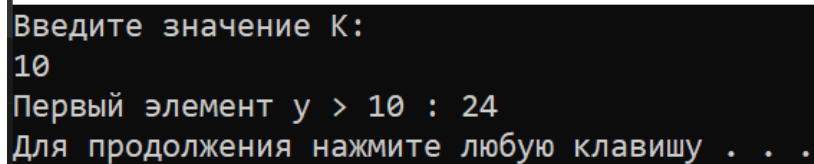
}

static void FindFirst(double k)
{
    double x = 1, y = 0;

    while (true)
    {
        if (x > k)
        {
            Console.WriteLine("Первый элемент x > {0}: {1}", k, x);
            break;
        }
        if (y > k)
        {
            Console.WriteLine("Первый элемент y > {0}: {1}", k, y);
            break;
        }

        double newX = x + 2 * y;
        double newY = y - 2 * x;
        x = newX;
        y = newY;
    }
}

```



```

Введите значение K:
10
Первый элемент y > 10 : 24
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5 – Полученный результат

Контрольные вопросы:

1. Выражением какого типа является условие в операторе if? Какие значения оно может принимать?

Условие в операторе if является логическим выражением, которое может принимать значения true (истина) или false (ложь). Если выражение истинно, выполняется блок кода внутри if; если ложно, блок кода пропускается.

2. Как работает оператор if, если отсутствует часть else <оператор2>?

Если часть else отсутствует, то просто не выполняется никакой код, если условие в if ложно. Это означает, что программа продолжит выполнение следующих инструкций после блока if.

if (условие)

```
{// Код выполняется, если условие истинно}
```

```
// Код продолжается здесь, независимо от результата условия
```

3. В каких случаях используется оператор switch?

Оператор switch используется для выбора одного из множества возможных вариантов выполнения кода в зависимости от значения выражения. Это удобно, когда нужно сравнить одно значение с несколькими константами.

4. Какого типа может быть <выражение> в операторе switch?

Выражение в операторе switch может быть:

- Целочисленным типом (int, byte, short, long).
- Символьным типом (char).
- Строковым типом (string).
- Перечислением (enum).

5. В каком случае выполняется последовательность инструкций default-ветви?

Инструкции в ветви default выполняются, если ни одно из значений в case не совпадает со значением выражения в switch. Ветвь default является необязательной, но полезной для обработки случаев, когда ни один из вариантов не подходит.

6. В чем отличие операторов while и do ... while

While проверяет условие перед выполнением тела цикла. Если условие ложно с самого начала, тело цикла не выполнится ни разу.

```
while (условие)
```

```
{// Код выполняется, пока условие истинно}
```

do...while выполняет тело цикла хотя бы один раз, а затем проверяет условие. Даже если условие ложно, тело цикла выполнится один раз.

```
do
```

```
{// Код выполняется хотя бы один раз}
```

```
while (условие);
```

7. Что представляет собой элемент <инициализация> в операторе for?

Элемент <инициализация> в операторе for используется для инициализации одной или нескольких переменных перед началом цикла. Обычно это счетчик цикла.

```
for (int i = 0; i < 5; i++)
```

```
{// Код цикла}
```

8. Какого типа может быть элемент <условие> в цикле for?.

Элемент <условие> в цикле for должен быть логическим выражением, которое возвращает значение true или false. Цикл продолжается до тех пор, пока это выражение истинно.

9. Назначение управляющей переменной цикла for?

Управляющая переменная цикла for (например, счетчик) используется для отслеживания количества итераций цикла и управления его выполнением. Она обычно инициализируется перед началом цикла и изменяется в каждой итерации.

10. Назначение управляющих операторов goto, break, continue, return.

Goto переходит к указанной метке в коде, что может привести к трудному для понимания коду и его следует использовать осторожно.

Break прерывает выполнение текущего цикла или блока switch и переходит к следующей инструкции после него.

Continue пропускает оставшиеся инструкции текущей итерации цикла и переходит к следующей итерации.

Return завершает выполнение метода и возвращает управление вызывающему коду. Может также возвращать значение, если метод не является void.

11. Как программируются циклические алгоритмы с явно заданным числом повторений цикла?

Циклические алгоритмы с заданным числом повторений обычно реализуются с помощью цикла for, где количество итераций задается в условии:

```
for (int i = 0; i < кол-во итераций; i++)  
{// Код для выполнения в каждой итерации}
```

12. Что представляют собой методы?

Методы – блоки кода, которые выполняют определенные действия и могут быть вызваны из других частей программы. Методы могут принимать параметры и возвращать значения.

13. Как объявляется метод?

Метод объявляется с указанием модификаторов доступа, возвращаемого типа, имени метода и параметров (если они есть):

```
public int Method(int a)  
{// Код метода  
    return a;}
```

14. Какова область действия параметров метода?

Область действия параметров метода ограничивается телом метода. Они доступны только внутри метода и недоступны вне его.

15. Как вызываются методы?

Методы вызываются по имени с указанием необходимых аргументов (если есть параметры):

```
int res = Method(1); // Вызов метода с аргументом 1
```

16. Общие (статические) методы класса.

Общие (статические) методы класса объявляются с использованием ключевого слова `static`. Они могут быть вызваны без создания экземпляра класса и могут работать только с статическими членами класса:

```
public class MyClass
{
    public static void StaticMethod()
    { // Код статического метода }
}
// Вызов статического метода
MyClass.StaticMethod();
```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| | | | | | | |
| Изм. | Лист | №докум. | Подпись | Дата | | 12 |

Лабораторная работа №3

Задание 1. Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы без параметров и с параметрами (имена некоторых полей должны совпадать с идентификаторами параметров), методы и свойства. Методы и свойства должны обеспечивать непротиворечивый и удобный интерфейс класса.

В программе должна выполняться проверка всех разработанных элементов класса, вывод состояния объекта.

6. Описать класс, представляющий полукруг с основанием, параллельным оси x . Предусмотреть методы для вычисления площади полукруга, длину обрамляющей линии, и проверки попадания данной точки внутрь полукруга.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr3_1
{
    class Circle
    {
        private double radius;
        public Circle(double radius)
        {
            this.radius = radius;
        }
        public double Plosh()
        {
            return (Math.PI * Math.Pow(radius, 2))/2;
        }

        public double Perimeter()
        {
            return Math.PI * radius + 2 * radius; // Полукруг + основание
        }

        public bool Point(double x, double y)
        {

```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 13 |

```

        if ((x * x + y * y <= radius * radius) && (y >= 0))
        {
            Console.WriteLine("Точка внутри полукруга.");
        }
        else {
            Console.WriteLine("Точка вне полукруга.");
        }
        return (x * x + y * y <= radius * radius) && (y >= 0);
    }
    public void Vivod()
    {
        Console.WriteLine("Полукруг: радиус = {0}, площадь = {1}, периметр = {2}", radius, Plosh(), Perimeter());
    }
}
internal class Program
{
    static void Main(string[] args)
    {
        Console.Write("Введите радиус окружности: ");
        double n = double.Parse(Console.ReadLine());
        Circle circle = new Circle(n);
        circle.Vivod();
        Console.WriteLine("Введите координаты точки (x, y):");
        string[] vvod = Console.ReadLine().Split(',');
        if (vvod.Length == 2 && double.TryParse(vvod[0], out double x) &&
double.TryParse(vvod[1], out double y))
        {
            Console.WriteLine(circle.Point(x, y));
        }
        else
        {
            Console.WriteLine("Неверный ввод.");
        }
    }
}

```

```

Введите радиус окружности: 10
Полукруг: радиус = 10, площадь = 157,07963267949, периметр = 51,4159265358979
Введите координаты точки (x, y):
4,3
Точка внутри полукруга.
True

```

Рисунок 6 – Полученный результат

Задание 2. Включите в проект Задания 1 обработку исключений.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lr3_2
{
    class Circle
    {
        private double radius;
        public Circle(double radius)
        {
            if (radius <= 0)
            {
                throw new ArgumentException("Радиус должен быть положительным
числом.");
            }
            this.radius = radius;
        }

        public double Plosh()
        {
            return (Math.PI * Math.Pow(radius, 2))/2;
        }

        public double Perimeter()
        {
            return Math.PI * radius + 2 * radius; // Полукруг + основание
        }

        public bool Point(double x, double y)
        {
            if ((x * x + y * y <= radius * radius) && (y >= 0))
            {
                Console.WriteLine("Точка внутри полукруга.");
            }
            else {
                Console.WriteLine("Точка вне полукруга.");
            }
            return (x * x + y * y <= radius * radius) && (y >= 0);
        }
    }
}
```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист 15 |
| Изм. | Лист | №докум. | Подпись | Дата | | |

```

public void Vivod()
{
    Console.WriteLine("Полукруг: радиус = {0}, площадь = {1}, периметр = {2}", radius, Plosh(), Perimeter());
}
}
internal class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.Write("Введите радиус окружности: ");
            double n = double.Parse(Console.ReadLine());
            Circle circle = new Circle(n);
            circle.Vivod();
            Console.WriteLine("Введите координаты точки (x, y):");
            string[] vvod = Console.ReadLine().Split(',');
            if (vvod.Length == 2 && double.TryParse(vvod[0], out double x) && double.TryParse(vvod[1], out double y))
            {
                Console.WriteLine(circle.Point(x, y));
            }
            else
            {
                Console.WriteLine("Неверный ввод.");
            }
        }
        catch (FormatException)
        {
            Console.WriteLine("Ошибка: Введено некорректное число. Пожалуйста, введите числовое значение.");
        }
        catch (ArgumentException ex)
        {
            Console.WriteLine("Ошибка: {0}", ex.Message);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Произошла непредвиденная ошибка: {0}", ex.Message);
        }
    }
}

```


}}}

Введите радиус окружности: -10
Ошибка: Радиус должен быть положительным числом.
Для продолжения нажмите любую клавишу . . .

Рисунок 7 – Полученный результат

Контрольные вопросы:

1. Как описываются классы в C#?

Классы в C# описываются с помощью ключевого слова `class`, за которым следует имя класса и тело класса, заключенное в фигурные скобки. Внутри класса можно определять поля, методы, свойства и другие члены.

```
public class MyClass  
{private int Field;  
    public void Method()  
    { // Код метода }}
```

2. Что относится к членам класса?

Членами класса являются:

- Поля: переменные, которые хранят данные.
- Методы: функции, которые выполняют действия.
- Свойства: специальные методы для доступа к полям (геттеры и сеттеры).
- Конструкторы: специальные методы, используемые для инициализации объектов.
- Индексы: позволяют обращаться к элементам класса как к массиву.
- События: используются для реализации паттерна "наблюдатель".

3. Что такое статические члены класса?

Статические члены класса (поля, методы, свойства) принадлежат самому классу, а не его экземплярам. Они могут быть вызваны без создания объекта класса и обычно используются для хранения данных или методов, общих для всех экземпляров.

```
public class MyClass  
{public static int StaticField;  
    public static void StaticMethod()  
    { // Код статического метода }}
```

4. Данные: поля и константы.

Поля – переменные, объявленные внутри класса, которые хранят состояние объекта. Поля могут изменяться в процессе работы программы.

Константы – значения, которые не могут изменяться после их инициализации. Объявляются с использованием ключевого слова `const`.

5. Спецификаторы полей и констант класса.

Спецификаторы доступа определяют уровень доступа к полям и константам. Основные спецификаторы:

- `public`: доступен из любого места.
- `private`: доступен только внутри самого класса.
- `protected`: доступен внутри класса и его производных.
- `internal`: доступен внутри текущей сборки.
- `protected internal`: доступен внутри текущей сборки и производным классам.

6. Как передаются параметры в методы?

Параметры могут передаваться в методы по значению (копируется значение) или по ссылке (ссылается на оригинальный объект). По умолчанию параметры передаются по значению.

```
public void Method(int value) // Передача по значению
```

```
{ }
```

```
public void Method(ref int value) // Передача по ссылке
```

```
{ }
```

7. Для чего предназначен параметр `params`?

Параметр `params` позволяет передавать переменное количество аргументов в метод. Он должен быть последним параметром в списке параметров метода и может принимать массив значений.

```
public void Method(params int[] num)
```

```
{
```

```
    foreach (var number in num)
```

```
    { // Обработка каждого числа }
```

```
// Вызов метода с разным количеством аргументов
```

```
Method(1, 2, 3); Method(4, 5);
```

8. Что представляет собой конструктор? Для чего он используется?

Конструктор — это специальный метод класса, который вызывается при создании нового экземпляра этого класса. Он используется для инициализации полей объекта и выполнения других необходимых действий при создании объекта.

```
public class MyClass
```

```
{ public int Field;
```

```
    // Конструктор
```

```
    public MyClass(int Val)
```

```
    { Field = Val; // Инициализация поля } }
```

9. Какие бывают конструкторы?

Существует несколько типов конструкторов:

| | | | | | | |
|------|------|----------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | № докум. | Подпись | Дата | | 18 |

По умолчанию: не принимает параметров и может быть автоматически создан компилятором.

```
public MyClass() { }
```

Параметризованный: принимает параметры для инициализации полей.

```
public MyClass(int Val) { Field = Val; }
```

Статический конструктор: используется для инициализации статических членов класса. Вызывается автоматически перед первым обращением к статическим членам.

```
static MyClass()
```

```
{// Инициализация статических полей}
```

10. Может ли класс не иметь конструктора?

Да, класс может не иметь явного конструктора. В этом случае компилятор автоматически создаст конструктор по умолчанию.

11. Для чего предназначена система сбора мусора?

Система сборки мусора (Garbage Collection) в C# отвечает за автоматическое управление памятью. Она освобождает память, занятую объектами, которые больше не используются или недоступны, что помогает предотвратить утечки памяти и улучшает производительность приложения.

Лабораторная работа №4

Задание 1. Создайте проект, в котором опишите класс для решения задачи Вашего варианта. Каждый разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы с параметрами и без параметров, методы, свойства, индексаторы.

Класс должен реализовывать следующие операции над массивами:

- задание произвольной размерности массива при создании объекта;
- доступ к элементу по индексам с контролем выхода за пределы массива;
- вывод на экран элемента массива по заданному индексу и всего массива.

При возникновении ошибок должны выбрасываться исключения.

В программе должна выполняться проверка всех разработанных элементов класса.

6. Описать класс, реализующий тип данных «матрица целых чисел». Класс должен реализовывать метод проверки, является ли матрицы верхней треугольной.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;
namespace Lp4
{
    public class IntMatrix
    {
        private int[,] matrix; // Скрытое поле для хранения матрицы
        public int Rows { get; private set; } // Открытое свойство для кол-ва строк
        public int Columns { get; private set; } // Открытое свойство для кол-ва
        столбцов
        // Конструктор с параметрами, проверка размера матрицы
        public IntMatrix(int rows, int columns)
        { if (rows <= 0 || columns <= 0)
            { throw new ArgumentException("Размеры матрицы должны быть
            положительными числами."); }
            Rows = rows;
            Columns = columns;
            matrix = new int[Rows, Columns];
        }
        // Индексатор для доступа к элементам матрицы
        public int this[int row, int column]
```

```

{
    get
    { if (row < 0 || row >= Rows || column < 0 || column >= Columns)
      { throw new IndexOutOfRangeException("Индекс за пределами
массива.");}
      return matrix[row, column];
    }
    set
    { if (row < 0 || row >= Rows || column < 0 || column >= Columns)
      { throw new IndexOutOfRangeException("Индекс за пределами
массива.");}
      matrix[row, column] = value;
    }
}
public void Full()
{
    Console.WriteLine("Введите элементы матрицы:");
    for (int i = 0; i < Rows; i++)
    {
        for (int j = 0; j < Columns; j++)
        {
            Console.Write("Элемент [{0}, {1}]: ", i, j);
            matrix[i, j] = int.Parse(Console.ReadLine());
        }
    }
}
public void Print()
{
    Console.WriteLine("Матрица:");
    for (int i = 0; i < Rows; i++)
    {
        for (int j = 0; j < Columns; j++)
        {
            Console.Write(matrix[i, j] + "\t ");
        }
        Console.WriteLine();
    }
}
public bool UpperTria()
{
    for (int i = 1; i < Rows; i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (matrix[i, j] != 0)
            {
                return false;
            }
        }
    }
}

```

```

    }
    }
    }
    return true;
}
}
internal class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.Write("Введите количество строк: ");
            int rows = int.Parse(Console.ReadLine());
            Console.Write("Введите количество столбцов: ");
            int columns = int.Parse(Console.ReadLine());

            IntMatrix matrix = new IntMatrix(rows, columns);
            matrix.Full();
            matrix.Print();
            if (matrix.UpperTria())
            {
                Console.WriteLine("Матрица является верхней треугольной.");
            }
            else {
                Console.WriteLine("Матрица не является верхней треугольной.");
            }
        }
        catch (FormatException)
        {
            Console.WriteLine("Ошибка: введено некорректное число. Пожалуйста, введите целое число.");
        }
        catch (ArgumentException ex)
        {
            Console.WriteLine("Ошибка: {0}", ex.Message);
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine("Ошибка: {0}", ex.Message);
        }
        catch (Exception ex)
        {

```

```

Console.WriteLine("Произошла непредвиденная ошибка: {0}",
ex.Message);
}}}}

```

```

Введите количество строк: 4
Введите количество столбцов: 4
Введите элементы матрицы:
Элемент [0, 0]: 1
Элемент [0, 1]: 1
Элемент [0, 2]: 1
Элемент [0, 3]: 1
Элемент [1, 0]: 0
Элемент [1, 1]: 1
Элемент [1, 2]: 1
Элемент [1, 3]: 1
Элемент [2, 0]: 0
Элемент [2, 1]: 0
Элемент [2, 2]: 1
Элемент [2, 3]: 1
Элемент [3, 0]: 0
Элемент [3, 1]: 0
Элемент [3, 2]: 0
Элемент [3, 3]: 1
Матрица:
1      1      1      1
0      1      1      1
0      0      1      1
0      0      0      1
Матрица является верхней треугольной.

```

Рисунок 8 – Полученный результат

Контрольные вопросы:

1. Что понимается под массивом?

Массив — это структура данных, которая позволяет хранить фиксированное количество элементов одного типа. Элементы массива располагаются в памяти последовательно, и к ним можно обращаться по индексу.

2. Каковы возможные способы описания массивов (одномерных и многомерных)?

Одномерный массив:

```
int[] Array = new int[5];
```

Можно инициализировать сразу:

```
int[] Array = { 1, 2, 3, 4, 5 };
```

Двумерный массив:

```
int[,] Array = new int[3, 4];
```

С инициализацией:

```
int[,] Array = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };
```

Многомерный массив (например, трехмерный):

```
int[, ,] Array = new int[2, 3, 4]; // Массив 2x3x4
```

3. В каких случаях целесообразно описывать двумерный массив с помощью одномерных?

Целесообразно использовать одномерные массивы для представления двумерной структуры, когда:

- Необходимо динамически изменять размеры массива.
- Требуется более простая логика доступа к элементам.
- Доступ к элементам можно организовать через вычисление индекса (например, $index = row * numberOfColumns + column$).

4. Какие типы допустимы для описания индексов массивов?

Индексы массивов должны быть целочисленными типами. Это может быть `int`, `long`, `short` или `byte`. Однако чаще всего используются `int`.

5. Какие типы могут использоваться в качестве базовых для описания массивов?

В качестве базовых типов для массивов могут использоваться:

- Примитивные типы (например, `int`, `float`, `char`, `bool`).
- Пользовательские классы и структуры.
- Делегаты.

6. Как осуществляется ввод и вывод массивов?

Ввод и вывод массивов можно осуществлять с помощью циклов.

// Ввод

```
for (int i = 0; i < array.Length; i++)  
{ array[i] = Convert.ToInt32(Console.ReadLine());}
```

// Вывод

```
foreach (var i in array)  
{ Console.WriteLine(item);}
```

7. Для чего предназначен цикл `foreach`?

Цикл `foreach` предназначен для упрощенного перебора элементов коллекций (включая массивы) без необходимости управлять индексами. Он автоматически обрабатывает итерацию по элементам.

8. Можно ли использовать цикл `foreach` для ввода элементов массива?

Нет, цикл `foreach` не предназначен для ввода элементов массива, так как он не предоставляет доступа к индексу. Для ввода элементов массива лучше использовать стандартный цикл `for`.

9. Как определяется базовый тип индексатора?

Базовый тип индексатора определяется как тип данных, который возвращается при обращении к элементу индексатора. Он задается в определении индексатора.

```
public class Collection
{
    private int[] array = new int[50];
    public int this[int index] // Индексатор
    {
        get { return array[index]; }
        set { array[index] = value; }
    }
}
```

10. Что записывается в качестве имени индексатора?

Индексатор не имеет имени в традиционном смысле; вместо этого используется ключевое слово `this` в определении индексатора.

11. Что содержит список параметров индексатора?

Список параметров индексатора содержит один или несколько параметров, которые используются для доступа к элементам коллекции. Обычно это один параметр (индекс), но можно определить индексаторы с несколькими параметрами.

```
public int this[int row, int column] // Индексатор с двумя параметрами
{
    get { return array[row, column]; }
    set { array[row, column] = value; }}
```

Лабораторная работа №5

Задание 1. Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Каждый разрабатываемый класс должен, содержать следующие элементы: скрытые и открытые поля, конструкторы (один из них должен передавать параметром массив), перегруженные операции.

В программе должна выполняться проверка всех разработанных элементов класса.

6. Описать класс для работы с n-мерным вектором. Класс должен реализовывать возможность: перегруженные операции отношений, выполняющие сравнение длин векторов;

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr5
{
    public class Vector
    {
        private double[] vector; // Скрытое поле для хранения вектора
        // Конструктор по умолчанию
        public Vector()
        {
            vector = new double[0];
        }
        // Конструктор с параметром массив
        public Vector(double[] elements)
        {
            if (elements == null || elements.Length == 0)
            {
                throw new ArgumentException("Вектор не может быть пустым.");
            }
            vector = new double[elements.Length];
            Array.Copy(elements, vector, elements.Length);
        }
        // получение длины вектора
        public int Length => vector.Length;
        public static void Srav(Vector v1, Vector v2)
```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 26 |

```

{
    if (v1.vector.Length > v2.vector.Length)
    {
        Console.WriteLine("Вектор 1 длиннее.");
    }
    else if (v1.vector.Length < v2.vector.Length)
    {
        Console.WriteLine("Вектор 2 длиннее.");
    }
    else
    {
        Console.WriteLine("Длины векторов равны.");
    }
}

public void Print()
{
    Console.WriteLine("Вектор: [" + string.Join(", ", vector) + "]");
}
}

internal class Program
{
    static void Main(string[] args)
    {
        try
        {
            Random rnd = new Random();
            Console.Write("Введите длину первого вектора: ");
            int length1 = int.Parse(Console.ReadLine());
            double[] e1 = new double[length1];

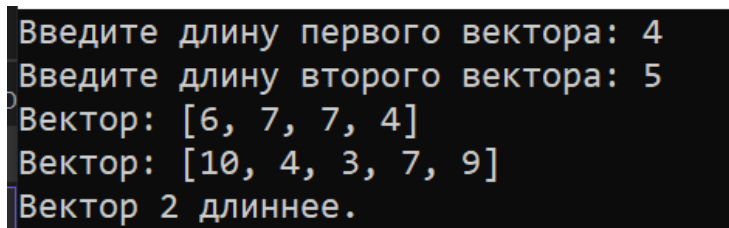
            for (int i = 0; i < length1; i++)
            {
                e1[i] = rnd.Next(1,11);
            }
            Vector vector1 = new Vector(e1);
            Console.Write("Введите длину второго вектора: ");
            int length2 = int.Parse(Console.ReadLine());
            double[] e2 = new double[length2];
            for (int i = 0; i < length2; i++)
            {
                e2[i] = rnd.Next(1, 11);
            }
            Vector vector2 = new Vector(e2);

```

```

vector1.Print();
vector2.Print();
Vector.Srav(vector1, vector2);
}
catch (FormatException)
{
    Console.WriteLine("Ошибка: введено некорректное число. Пожалуйста,
введите число.");
}
catch (Exception ex)
{
    Console.WriteLine("Произошла непредвиденная ошибка: {0}",
ex.Message);
}
}
}
}

```



```

Введите длину первого вектора: 4
Введите длину второго вектора: 5
Вектор: [6, 7, 7, 4]
Вектор: [10, 4, 3, 7, 9]
Вектор 2 длиннее.

```

Рисунок 9 – Полученный результат

Контрольные вопросы:

1. Что представляет собой перегрузка методов?

Перегрузка методов — это возможность создавать несколько методов с одинаковым именем, но с разными параметрами (различающимися по количеству, типу или порядку). Это позволяет вызывать один и тот же метод с различными аргументами.

```

public class Example
{
    public void Display(int value) { Console.WriteLine(value); }
    public void Display(string value) { Console.WriteLine(value); }
}

```

2. Что представляет собой перегрузка операций

Перегрузка операций позволяет определить, как стандартные операторы (например, +, -, *, /) будут работать с пользовательскими типами данных. Это делает код более читаемым и интуитивно понятным.

```

public class Complex
{
    public double Real { get; set; }
}

```

```
public double Imaginary { get; set; }
public static Complex operator +(Complex a, Complex b)
{ return new Complex { Real = a.Real + b.Real, Imaginary = a.Imaginary +
b.Imaginary };}
```

3. Формат описания операции класса.

Операция определяется с помощью ключевого слова `operator`, за которым следует символ операции и параметры, которые она принимает. Возвращаемый тип – тип, который будет возвращен операцией.

```
public static Return Type operator OperatorSymbol(Parameters)
{ }
```

4. Какие операции нельзя перегружать?

Некоторые операции нельзя перегружать:

- . (точка)
- :: (двойное двоеточие)
- ?: (тернарный оператор)
- new (оператор создания объекта)
- is (оператор проверки типа)
- as (оператор приведения типа)
- sizeof (оператор получения размера типа)

5. Что является результатом перегрузки унарных операций?

Результатом перегрузки унарных операций является новый объект, который создается в результате применения операции к одному операнду. Например, перегрузка оператора ++ может увеличить значение свойства объекта.

```
public static MyClass operator ++(MyClass obj)
{ obj.Value++;
return obj;}
```

6. Какие параметры могут быть у бинарных операций класса?

Бинарные операции принимают два параметра – оба могут быть экземплярами пользовательского класса или одним экземпляром пользовательского класса и другим типом данных.

```
public static MyClass operator +(MyClass a, MyClass b) { /* ... */ }
```

7. Как выполняется перегрузка операций отношения?

Перегрузка операций отношения (например, <, >, <=, >=) осуществляется через определение соответствующих операторов в классе. Эти операторы должны возвращать булево значение (true или false).

```
public static bool operator <(MyClass a, MyClass b)
{ return a.Value < b.Value;}
```

8. Чем являются строки в C#?

В C# строки представляют собой последовательности символов и являются объектами класса `String`. Строки являются неизменяемыми (`immutable`), что означает, что после создания они не могут быть изменены.

9. Какие операции определены для строк?

Для строк в C# определены различные операции:

- Конкатенация (+).
- Сравнение строк (`==`, `!=`).
- Доступ к символам по индексу.
- Методы для поиска, замены и извлечения подстрок.

10. Как создаются строки?

Строки создаются с помощью литералов строк или конструктора класса `String`.

```
string a1 = "Hello, World!";  
string a2 = new String(new char[] { 'H', 'e', 'l', 'l', 'o' });
```

11. Можно ли изменять значение строки?

Нет, строки в C# неизменяемы. Любые операции, которые изменяют строку, на самом деле создают новую строку.

```
string original = "ABCD";  
string modified = original.Replace("A", "F"); // original остается "ABCD"
```

Лабораторная работа №6

Наследование

Задание 1. Составить программу с одним родительским классом и потомком. Все поля должны быть закрытыми. Базовый класс должен содержать конструкторы с параметрами, методы доступа к закрытым полям, вывод полей и указанный в таблице метод. Производный класс содержит дополнения и изменения, организовать вывод новых полей потомка, при этом имена методов совпадают с именами методов базового класса. Составить тестирующую программу с выдачей результатов. Создать объекты базового и производного типов. В программе должна выполняться проверка всех разработанных элементов класса.

6. Базовый класс: Стол (поля: название, площадь S в см)

Метод: Стоимость $C = S^{2/3} + k$, где k – коэффициент.

Потомок: Письменный (поле – используемый материал, стоимость отделки)

Изменения в потомках: Найти стоимость отделки, определяемую как 10% от стоимости и полную стоимость.

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LR_6
{
    class Table
    {
        private string name;
        private double area;

        public Table(string name, double area)
        {
            this.name = name;
            this.area = area;
        }
        public double Cost(double k)
        {
            return Math.Pow(area, 2) / 3 + k;
        }
        public void Display()
```

```

    {
        Console.WriteLine("Название: {0}, Площадь: {1} см", name, area );
    }
}
class Writing : Table
{
    private string material; // Используемый материал
    private double finish; // Стоимость отделки
    public Writing(string name, double area, string material)
        : base(name, area)
    {
        this.material = material;
        this.finish = CalcF();
    }
    // Метод для вычисления стоимости отделки
    private double CalcF()
    {
        double cost = Cost(0);
        return cost * 0.1; // 10% от стоимости
    }
    public new void Display()
    {
        base.Display();
        Console.WriteLine("Материал: {0}, Стоимость отделки: {1}", material,
finish);
    }
    // Метод для получения полной стоимости
    public double GetCost(double k)
    {
        return Cost(k) + finish;
    }
}
internal class Program
{
    static void Main(string[] args)
    {
        Console.Write("Введите количество обычных столов: ");
        int a = int.Parse(Console.ReadLine());
        Table[] tables = new Table[a];
        for (int i = 0; i < a; i++)
        {
            Console.Write("Введите название стола {0}: ", i);
            string name = Console.ReadLine();
            Console.Write("Введите площадь стола {0} (в см): ", i);

```



```

double area = double.Parse(Console.ReadLine());
Console.Write("Введите коэффициент k для стола {0}: ", i);
double k = double.Parse(Console.ReadLine());
tables[i] = new Table(name, area);
tables[i].Display();
Console.Write("Стоимость стола: {0}", tables[i].Cost(k));
Console.WriteLine();
}
Console.Write("Введите количество письменных столов: ");
int b = int.Parse(Console.ReadLine());
Writing[] writing = new Writing[b];
for (int i = 0; i < b; i++)
{
    Console.Write("Введите название письменного стола {0}: ", i);
    string name = Console.ReadLine();
    Console.Write("Введите площадь письменного стола {0} (в см): ", i);
    double area = double.Parse(Console.ReadLine());
    Console.Write("Введите материал письменного стола {0}: ", i);
    string material = Console.ReadLine();
    Console.Write("Введите коэффициент k для письменного стола {0}:", i);
    double k = double.Parse(Console.ReadLine());
    writing[i] = new Writing(name, area, material);
    writing[i].Display();
    Console.Write("Полная стоимость: {0}", writing[i].GetCost(k));
    Console.WriteLine();
}
}
}
}

```

```

Введите количество обычных столов: 2
Введите название стола 0: прима
Введите площадь стола 0 (в см): 30
Введите коэффициент k для стола 0: 5
Название: прима, Площадь: 30 см
Стоимость стола: 305
Введите название стола 1: августин
Введите площадь стола 1 (в см): 50
Введите коэффициент k для стола 1: 9
Название: августин, Площадь: 50 см
Стоимость стола: 842,333333333333
Введите количество письменных столов: 2
Введите название письменного стола 0: обычный
Введите площадь письменного стола 0 (в см): 33
Введите материал письменного стола 0: темное дерево
Введите коэффициент k для письменного стола 0:22
Название: обычный, Площадь: 33 см
Материал: темное дерево, Стоимость отделки: 36,3
Полная стоимость: 421,3
Введите название письменного стола 1: аврора
Введите площадь письменного стола 1 (в см): 100
Введите материал письменного стола 1: хрусталь
Введите коэффициент k для письменного стола 1:30
Название: аврора, Площадь: 100 см
Материал: хрусталь, Стоимость отделки: 333,333333333333
Полная стоимость: 3696,66666666667

```

Рисунок 10 – Полученный результат

Контрольные вопросы:

1. В чем состоит принцип наследования?

Принцип наследования позволяет создавать новый класс на основе существующего класса (базового), унаследовав его члены (поля и методы). Это способствует повторному использованию кода и созданию иерархий классов.

2. Какие члены класса наследуются?

Наследуются публичные (public) и защищенные (protected) члены класса.

Статические члены не наследуются, но могут быть доступны через базовый класс.

3. Что представляет собой защищенный доступ?

Защищенный доступ (protected) позволяет членам класса быть доступными только внутри самого класса и в производных классах (наследниках). Это ограничивает доступ к членам класса из других классов.

4. Как происходит вызов конструкторов базового класса?

Конструкторы базового класса вызываются с помощью ключевого слова base в конструкторе производного класса:

```
public class BaseClass
```

```
{public BaseClass(int value) { }}
public class DerivedClass : BaseClass
{public DerivedClass(int value) : base(value) { }}
```

5. Что такое сокрытие имен при наследовании?

Соккрытие имен происходит, когда производный класс определяет член с тем же именем, что и член базового класса. Это может привести к путанице, так как доступ к членам будет зависеть от контекста.

```
public class BaseClass
{public void Display() { Console.WriteLine("Base Class Display"); }}
public class DerivedClass : BaseClass
{public new void Display() { Console.WriteLine("Derived Class Display"); } //
```

Соккрытие

```
}
```

6. Как получить доступ к сокрытому члену базового класса?

Чтобы получить доступ к сокрытому члену базового класса, можно использовать явное приведение типа к базовому классу:

```
DerivedClass derived = new DerivedClass();
((BaseClass)derived).Display();
```

Лабораторная работа №7

Полиморфизм. Виртуальные методы

Задание 1.

Составить программу с одним родительским классом и двумя потомками. Потомки должны содержать виртуальные функции. Создать виртуальную функцию выдачи результатов расчета методов на экран монитора с указанием названий и полей и их значений соответствующего объекта. Составить тестирующую программу с выдачей протокола на экран монитора.

При этом создать объекты базового и производных типов, используя полиморфный контейнер - массив ссылок базового класса на объекты базового и производных классов (количество объектов ≥ 5).

| Родительский класс | Потомки | Полиморфные методы |
|--------------------------|--|--|
| Одежда (поле - название) | Пальто (поле размер V) Костюм (поле рост H) | Расход ткани Пальто $V/6.5+0.5$ Костюм $2*H+0.3$ |

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr7_1
{
    public class Clothes
    {
        public string Name { get; set; }

        public Clothes(string name)
        { Name = name; }
        public virtual double CalcFab()
        { return 0; }
        public virtual void Display()
        {
            Console.WriteLine("Одежда: {0}, Расход ткани: {1}", Name, CalcFab());
        }
    }
    public class Coat : Clothes
    {
        public double Size { get; set; }
    }
}
```

```

public Coat(string name, double size) : base(name)
{ Size = size;}
public override double CalcFab()
{
    return Size / 6.5 + 0.5;
}
public override void Display()
{
    Console.WriteLine("Пальто: {0}, Размер: {1}, Расход ткани: {2}", Name, Size,
CalcFab());
}
}
public class Suit : Clothes
{
    public double Height {get; set;}
    public Suit(string name, double height) : base(name)
    { Height = height;}
    public override double CalcFab()
    {
        return 2 * Height + 0.3;
    }
    public override void Display()
    {
        Console.WriteLine("Костюм: {0}, Рост: {1}, Расход ткани: {2}", Name,
Height, CalcFab());
    }
}
internal class Program
{
    static void Main(string[] args)
    {
        Clothes[] Items = new Clothes[5];
        Items[0] = new Coat("Пальто зимнее", 42);
        Items[1] = new Suit("Костюм деловой", 180);
        Items[2] = new Coat("Пальто осеннее", 38);
        Items[3] = new Suit("Костюм вечерний", 175);
        Items[4] = new Coat("Пальто бежевое", 40);
        Items[3] = new Suit("Костюм женский", 160);
        Items[4] = new Coat("Пальто мужское", 44);
        foreach (var i in Items)
        {
            i.Display();
        }
    }
}
}

```

```

Пальто: Пальто зимнее, Размер: 42, Расход ткани: 6,96153846153846
Костюм: Костюм деловой, Рост: 180, Расход ткани: 360,3
Пальто: Пальто осеннее, Размер: 38, Расход ткани: 6,34615384615385
Костюм: Костюм вечерний, Рост: 175, Расход ткани: 350,3
Пальто: Пальто бежевое, Размер: 40, Расход ткани: 6,65384615384615
Костюм: Костюм женский, Рост: 160, Расход ткани: 320,3
Пальто: Пальто мужское, Размер: 44, Расход ткани: 7,26923076923077

```

Рисунок 11 – Полученный результат

Задание 2. Составить программу с абстрактным родительским классом и двумя объектами - потомками. Для этого модифицировать задание 1. Составить тестирующую программу с выдачей протокола на экран монитора. В ней нужно реализовать циклический вывод параметров объектов, используя полиморфный контейнер - массив объектов базового класса (количество объектов ≥ 5).

6. Организовать вычисление суммарного расхода ткани.

Листинг программы:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr7_2
{
    public abstract class Clothes
    {
        public string Name { get; set; }
        public Clothes(string name)
        { Name = name; }
        public abstract double CalcFab();
        public virtual void Display()
        {
            Console.WriteLine("Одежда: {0}, Расход ткани: {1}", Name, CalcFab());
        }
    }
    public class Coat : Clothes
    {
        public double Size { get; set; }
        public Coat(string name, double size) : base(name)
        { Size = size; }
        public override double CalcFab()
        { return Size / 6.5 + 0.5; }
        public override void Display()
        {

```

```

        Console.WriteLine("Пальто: {0}, Размер: {1}, Расход ткани: {2}", Name, Size,
CalcFab());
    } }
    public class Suit : Clothes
    {
        public double Height { get; set; }
        public Suit(string name, double height) : base(name)
        { Height = height; }
        public override double CalcFab()
        {
            return 2 * Height + 0.3;
        }
        public override void Display()
        {
            Console.WriteLine("Костюм: {0}, Рост: {1}, Расход ткани: {2}", Name,
Height, CalcFab());
        }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            Clothes[] Items = new Clothes[7];
            Items[0] = new Coat("Пальто зимнее", 42);
            Items[1] = new Suit("Костюм деловой", 180);
            Items[2] = new Coat("Пальто осеннее", 38);
            Items[3] = new Suit("Костюм вечерний", 175);
            Items[4] = new Coat("Пальто бежевое", 40);
            Items[5] = new Suit("Костюм женский", 160);
            Items[6] = new Coat("Пальто мужское", 44);
            double sum = 0;
            Console.WriteLine("Вывод информации о одежде:");
            foreach (var item in Items)
            {
                item.Display();
                sum += item.CalcFab();
            }
            Console.WriteLine("Суммарный расход ткани: {0}", sum);
        } } }

```

Вывод информации о одежде:

Пальто: Пальто зимнее, Размер: 42, Расход ткани: 6,96153846153846
Костюм: Костюм деловой, Рост: 180, Расход ткани: 360,3
Пальто: Пальто осеннее, Размер: 38, Расход ткани: 6,34615384615385
Костюм: Костюм вечерний, Рост: 175, Расход ткани: 350,3
Пальто: Пальто бежевое, Размер: 40, Расход ткани: 6,65384615384615
Костюм: Костюм женский, Рост: 160, Расход ткани: 320,3
Пальто: Пальто мужское, Размер: 44, Расход ткани: 7,26923076923077
Суммарный расход ткани: 1058,13076923077

Рисунок 12 – Полученный результат

Контрольные вопросы:

1. Что означает принцип полиморфизма?

Полиморфизм позволяет объектам разных классов обрабатывать данные через один и тот же интерфейс или базовый класс. Это означает, что можно использовать один и тот же метод для разных типов объектов.

2. Для чего используется позднее связывание?

Позднее связывание используется для динамического определения метода, который будет вызван во время выполнения программы. Это позволяет использовать интерфейсы и абстрактные классы и обеспечивает большую гибкость в программировании.

3. В каких случаях используются виртуальные методы?

Виртуальные методы используются в ситуациях, когда необходимо переопределить поведение метода в производном классе. Это позволяет создать более специфичное поведение для различных подклассов.

4. Какие условия необходимо соблюдать при переопределении виртуального метода?

При переопределении виртуального метода необходимо:

- Использовать ключевое слово `override`.
- Сохранять ту же сигнатуру метода (имя, возвращаемый тип и параметры).
- Убедиться, что базовый метод объявлен как виртуальный (`virtual`) или абстрактный (`abstract`).

5. Что представляют собой абстрактные классы? Для чего они предназначены?

Абстрактные классы – классы, которые не могут быть инстанцированы и предназначены для того, чтобы служить базой для других классов. Они могут содержать абстрактные методы (без реализации), которые должны быть переопределены в производных классах.

6. Могут ли в абстрактном классе быть неабстрактные методы?

Да, в абстрактном классе могут быть неабстрактные методы с реализацией. Это позволяет предоставлять общую функциональность для всех производных классов.

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| | | | | | | |
| Изм. | Лист | №докум. | Подпись | Дата | | 41 |

Лабораторная работа №8

Интерфейсы

Задание 1. Интерфейсы Ix, Iy, Iz, содержат объявления методов с одной и той же сигнатурой следующим образом

```
interface Ix
{
    void IxF0(параметр);
    void IxF1();
}
interface Iy
{
    void F0(параметр);
    void F1();
}
interface Iz
{
    void F0(параметр);
    void F1();
}
```

Эти интерфейсы наследуются в классе TestClass, содержащий член w типа параметр и реализуются так, как задано в варианте. В каждом методе задать вывод результата.

Рассмотреть случай

- неявной реализации интерфейсов
- явной реализации интерфейса Iz

В программе должна выполняться:

- неявная неоднозначная реализация методов интерфейсов Iy и Iz,
- вызов функций с явным приведением к типу интерфейса,
- вызов метода для объекта посредством интерфейсной ссылки.

| Параметр | IxF0, IxF1 возвращают | F0 F1 возвращают | |
|----------|--------------------------|-----------------------|------------------------|
| | | Неявная реализация | Явная реализация Iz |
| int | 7w-4 | w*3 | 6+w |

Листинг программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```

using System.Threading.Tasks;
namespace Lr8_1
{
    interface Ix
    { void IxF0(int parameter);
      void IxF1(); }
    interface Iy
    { void F0(int parameter);
      void F1(); }
    interface Iz
    { void F0(int parameter);
      void F1(); }
    class TestClass : Ix, Iy, Iz
    {
        private int w;
        public TestClass(int w)
        { this.w = w; }
        // Неявная реализация интерфейсов Iy и Ix
        public void F0(int parameter)
        { Console.WriteLine("Iy.F0 возвращает: {0}", w * 3); }
        public void F1()
        { Console.WriteLine("Iy.F1 вызван."); }
        public void IxF0(int parameter)
        {
            Console.WriteLine("IxF0 возвращает: {0}", 7 * w - 4);
        }
        public void IxF1()
        {
            Console.WriteLine("IxF1 вызван.");
        }
        // Явная реализация интерфейса Iz
        void Iz.F0(int parameter)
        {
            Console.WriteLine("Iz.F0 возвращает: {0}", 6 + w);
        }
        void Iz.F1()
        {
            Console.WriteLine("Iz.F1 вызван.");
        }
    }
    internal class Program
    {
        static void Main(string[] args)

```

```

{
    TestClass Obj = new TestClass(5);
    Console.WriteLine("Вызов методов с неявной реализацией");
    Obj.IxF0(10);
    Obj.IxF1();
    Obj.F0(10); // Вызов метода из интерфейса Iy (неявно)
    Obj.F1();   // Вызов метода из интерфейса Iy (неявно)
    Console.WriteLine("Явный вызов методов интерфейса Iz через приведение
типа");
    Iz izObj = Obj;
    izObj.F0(10);
    izObj.F1();
    Console.WriteLine("Для проверки вызова методов из интерфейса Iy явно");
    Iy iyObj = Obj;
    iyObj.F0(10);
    iyObj.F1();
}
}
}

```

```

Вызов методов с неявной реализацией
IxF0 возвращает: 31
IxF1 вызван.
Iy.F0 возвращает: 15
Iy.F1 вызван.
Явный вызов методов интерфейса Iz через приведение типа
Iz.F0 возвращает: 11
Iz.F1 вызван.
Для проверки вызова методов из интерфейса Iy явно
Iy.F0 возвращает: 15
Iy.F1 вызван.

```

Рисунок 13 – Полученный результат

Задание 2. Выполнить задания, используя для хранения экземпляров разработанных классов стандартные параметризованные коллекции. Во всех классах реализовать интерфейсы `Comparable` и `Comparer` перегрузить операции отношения для реализации сравнения объектов по указанному полю. Результат вывести на экран.

6. Составить инвентарную ведомость игрушек, включив следующие данные: название игрушки, ее стоимость (в руб.), возрастные границы детей, для которых предназначена игрушка. Вывести в новый список информацию о тех игрушках, которые предназначены для детей от N до M лет, отсортировав их по стоимости.

Листинг программы:

```

using System;
using System.Collections.Generic;

```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 44 |

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lr8_2
{
    public class Toy : IComparable<Toy>
    {
        public string Name { get; set; }
        public double Price { get; set; }
        public int AgeFrom { get; set; }
        public int AgeTo { get; set; }
        public Toy(string name, double price, int ageFrom, int ageTo)
        {
            Name = name;
            Price = price;
            AgeFrom = ageFrom;
            AgeTo = ageTo;
        }
        public int CompareTo(Toy other)
        {
            if (other == null) return 1;
            return Price.CompareTo(other.Price);
        }
        public override string ToString()
        {
            return ("${Name}, Цена: {Price} руб., Возраст: от {AgeFrom} до {AgeTo}
лет");
        }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            List<Toy> toys = new List<Toy>
            {
                new Toy("Мягкая игрушка", 500, 3, 7),
                new Toy("Конструктор", 1200, 5, 12),
                new Toy("Кукла", 800, 4, 10),
                new Toy("Автомобиль", 300, 2, 6),
                new Toy("Настольная игра", 1500, 8, 14),
                new Toy("Кубик Рубика", 100, 9, 99),
                new Toy("Набор Лего", 100000, 1, 99),
                new Toy("Мяч", 10, 3, 13),
            }
        }
    }
}

```

| | | | | | | |
|------|------|---------|---------|------|---------------------------|------|
| | | | | | ЛР 2-40 01 01.35.40.18.25 | Лист |
| Изм. | Лист | №докум. | Подпись | Дата | | 45 |

```

new Toy("Карты", 500, 9, 30),
new Toy("Шахматы", 1000, 4, 10)
};
foreach (var i in toys) {
    Console.WriteLine(i);
}
Console.Write("Введите начальный возраст: ");
int n = int.Parse(Console.ReadLine());
Console.Write("Введите до какого возраста: ");
int m = int.Parse(Console.ReadLine());
if ((n > m) || (n < 0) || (m < 0)) { Console.WriteLine("Неверный ввод!"); }
else
{
    var filteredToys = toys.FindAll(t => t.AgeFrom >= n && t.AgeTo <= m);
    filteredToys.Sort();

    Console.WriteLine("Сортировка найденных игрушек по стоимости:");
    foreach (var toy in filteredToys)
    {
        Console.WriteLine(toy);
    }
}
}
}
}
}

```

```

Мягкая игрушка, Цена: 500 руб., Возраст: от 3 до 7 лет
Конструктор, Цена: 1200 руб., Возраст: от 5 до 12 лет
Кукла, Цена: 800 руб., Возраст: от 4 до 10 лет
Автомобиль, Цена: 300 руб., Возраст: от 2 до 6 лет
Настольная игра, Цена: 1500 руб., Возраст: от 8 до 14 лет
Кубик Рубика, Цена: 100 руб., Возраст: от 9 до 99 лет
Набор Лего, Цена: 100000 руб., Возраст: от 1 до 99 лет
Мяч, Цена: 10 руб., Возраст: от 3 до 13 лет
Карты, Цена: 500 руб., Возраст: от 9 до 30 лет
Шахматы, Цена: 1000 руб., Возраст: от 4 до 10 лет
Введите начальный возраст: 3
Введите до какого возраста: 10
Сортировка найденных игрушек по стоимости:
Мягкая игрушка, Цена: 500 руб., Возраст: от 3 до 7 лет
Кукла, Цена: 800 руб., Возраст: от 4 до 10 лет
Шахматы, Цена: 1000 руб., Возраст: от 4 до 10 лет

```

Рисунок 14 – Полученный результат

Контрольные вопросы:

1. Как описывается интерфейс? Его назначение.

Интерфейс в C# описывается с помощью ключевого слова `interface`, за которым следует имя интерфейса. Интерфейсы предназначены для определения контрактов, которые классы должны реализовать. Они позволяют создавать гибкие и расширяемые системы, обеспечивая возможность работы с различными классами через общий интерфейс.

```
public interface Example
{ void MethodA();
  int MethodB(string input);}
```

2. Какие члены может содержать интерфейс?

Интерфейс может содержать:

- Метод (без реализации)
- Свойства (без реализации)
- Индексаторы
- События

Все члены интерфейса по умолчанию являются публичными и не могут иметь других спецификаторов доступа.

3. Какие спецификаторы допустимы у методов, реализующих интерфейс?

Методы, реализующие интерфейс, могут иметь только спецификатор доступа `public`. Если метод не объявлен как `public`, он не будет доступен через интерфейс.

```
public class Ex : Example
{
    public void MethodA() { /* реализация */ }
    public int MethodB(string input) { /* реализация */ return 0; }
}
```

4. В каких случаях используется явная реализация интерфейса?

Явная реализация интерфейса используется, когда необходимо избежать конфликтов имен между методами класса и методами интерфейса или когда нужно скрыть реализацию метода от общего интерфейса класса. Это позволяет вызывать метод только через экземпляр интерфейса.

```
public class Ex: Example
{
    void Example.MethodA() { /* реализация */ } // явная реализация
}
```

5. Как осуществляется наследование интерфейсов?

Интерфейсы могут наследоваться от других интерфейсов, используя двоеточие. Класс, реализующий наследуемый интерфейс, должен реализовать все его методы.

```
public interface Basa
{void BasaMethod();}
public interface Saga: Basa
{void SagaMethod();}
```

6. Можно ли явно реализованные методы объявлять виртуальными?

Нет, явно реализованные методы не могут быть объявлены как виртуальные. Это связано с тем, что явная реализация скрывает метод от общего интерфейса класса, и он не может быть переопределен.

7. Можно ли повторно реализовать интерфейс, указав его имя в списке предков класса наряду с классом-предком?

Да, можно. Если класс наследует от базового класса и реализует интерфейс, он может повторно реализовать тот же интерфейс, если это необходимо. Однако это не является обязательным — класс может просто унаследовать реализацию интерфейса от базового класса.

```
public class Basa
{public void MethodA() { /* реализация */ }}
public interface Example
{void MethodA();}
public class Saga: Basa, Example
{public new void MethodA() { /* новая реализация */ }}
```

8. Какие стандартные интерфейсы используются для работы с коллекциями?

Некоторые стандартные интерфейсы для работы с коллекциями в C# включают:

- IEnumerable<T> — для перебора коллекции.
- ICollection<T> — для определения методов для работы с коллекцией.
- IList<T> — для работы с индексированными коллекциями.
- IDictionary<TKey, TValue> — для работы с парами "ключ-значение".

9. Чем отличаются интерфейсы IComparable и IComparer?

IComparable используется для определения порядка сортировки объектов одного типа. Он содержит метод CompareTo, который сравнивает текущий объект с другим объектом того же типа.

```
public class MyClass : IComparable<MyClass>
{public int CompareTo(MyClass other) { /* реализация */ }}
```

IComparer используется для определения порядка сортировки объектов разных типов или для предоставления дополнительной логики сортировки. Он содержит метод Compare, который принимает два объекта и возвращает результат их сравнения.

```
public class MyClassComparer : IComparer<MyClass>
{public int Compare(MyClass x, MyClass y) { /* реализация */ }}
```