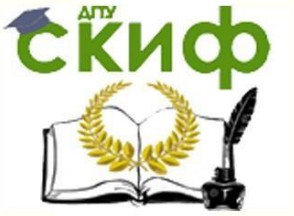


ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

Кафедра «Информационные технологии»

Сборник задач

по дисциплине

«Объектно-ориентированное программирование»

Авторы
Васильев П.В.,
Подколзина Л.А.

Ростов-на-Дону, 2023

Аннотация

Сборник задач по дисциплине «Объектно-ориентированное программирование» предназначен для студентов очной формы обучения направлений: 01.03.04 - Прикладная математика, 44.03.04 - Профессиональное обучение (по отраслям).

Авторы

Старший преподаватель кафедры «Информационные технологии»,
Васильев П.В.

Программист кафедры «Информационные технологии»,
Подколзина Л.А.

Оглавление

ЛАБОРАТОРНАЯ РАБОТА №1: ИНКАПСУЛЯЦИЯ

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе, и скрыть детали реализации от пользователя.

В информатике **инкапсуляцией** (лат. en capsula) называется упаковка данных и/или функций в единый компонент.

В объектно-ориентированных языках инкапсуляция, как правило, реализуется посредством механизма классов.

В общем случае, в разных языках программирования термин «инкапсуляция» относится к одной из или обеим одновременно следующим нотациям:

- языковая конструкция, позволяющая связать данные с методами, предназначенными для обработки этих данных;
- механизм языка, позволяющий ограничить доступ одних компонентов программы к другим.

Задание 1

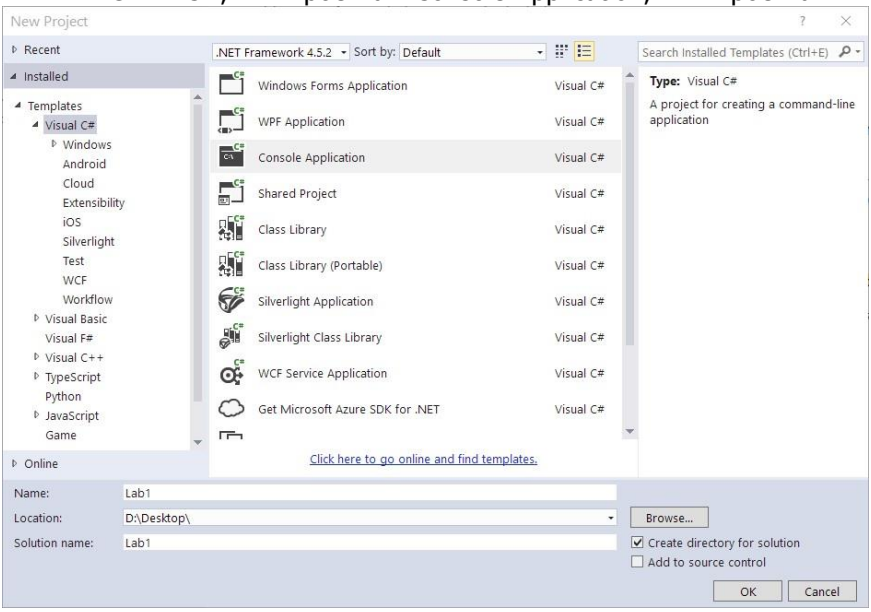
Создайте консольное приложение C#, в котором описать класс комплексных чисел и реализовать операции над ними. Целью инкапсуляции является обеспечение согласованности внутреннего состояния объекта. В C# для инкапсуляции используются публичные свойства и методы объекта.

1. Запускаем VisualStudio

Lab1.

2. Создаем новый проект (File/New/Project)

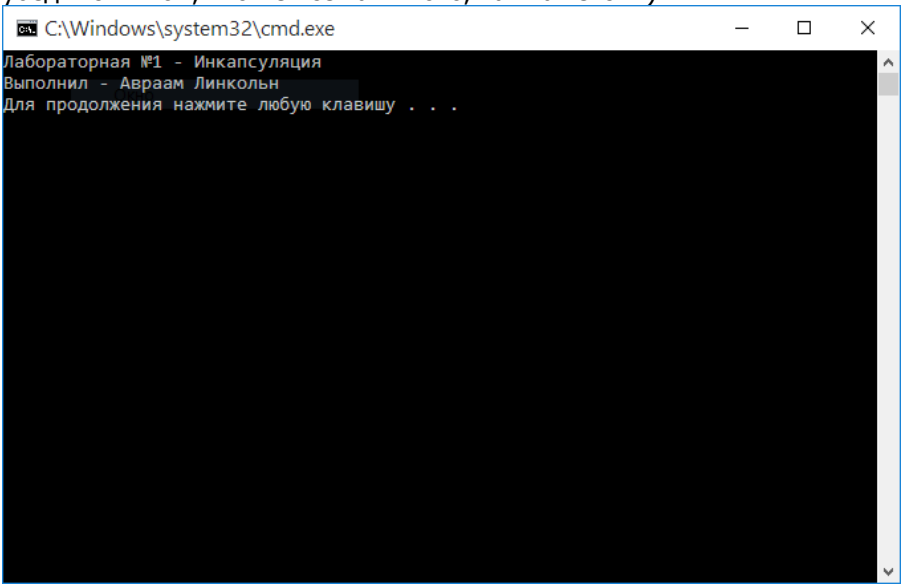
Язык - C#, тип проекта - Console Application, имя проекта -



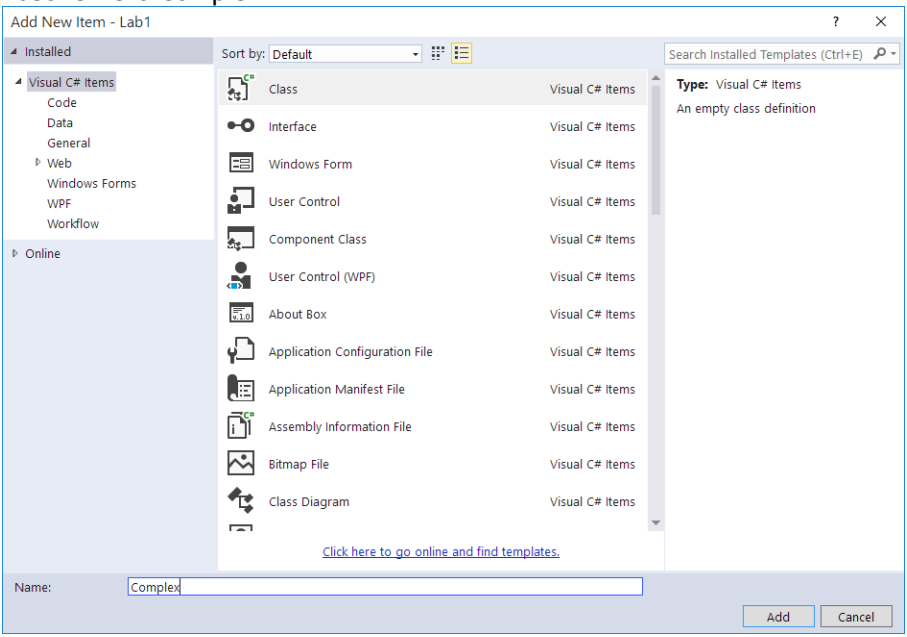
3. Выведем в консоль название лабораторной и имя автора:

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

namespace Lab1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная №1 - Инкапсуляция");
            Console.WriteLine("Выполнил - Авраам Линкольн");
        }
    }
}
```

4. Запустим (CTRL+F5) приложение с ожиданием ввода и убедимся в том, что не все так плохо, как кажется :)



5. Теперь создадим новый класс (Project/Add Class) и назовем его Complex:



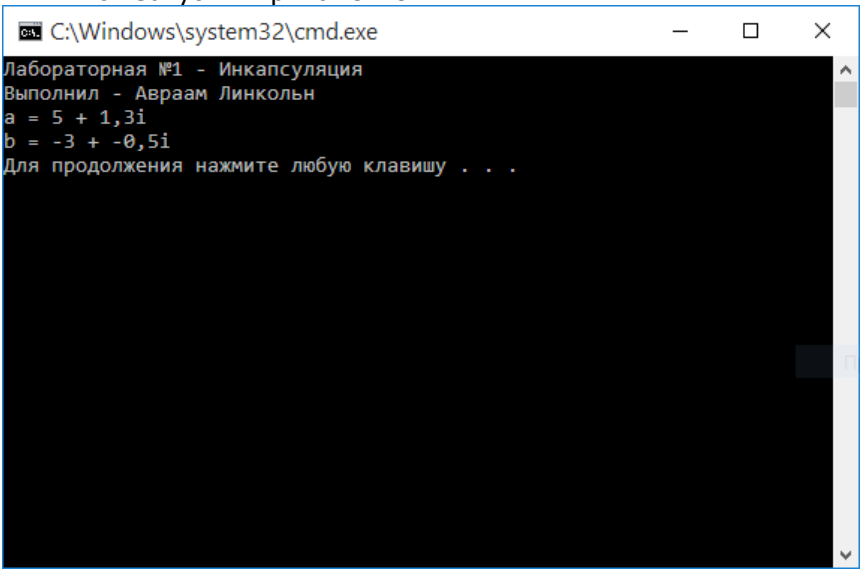
6. Создадим два публичных свойства **Real** и **Imag** типа **Double** для хранения действительной и мнимой частей комплексного числа и добавим функции(методы) сложения, вычитания, умножения и деления (**реализовать самостоятельно**).

```
namespace Lab1
{
    class Complex
    {
        public double Real { get; set; }
        public double Imag { get; set; }

        // Сложение
        public void Add(Complex x)
        {
            Real += x.Real;
            Imag += x.Imag;
        }

        // Вычитание
        public void Substract(Complex x)
        {
            Real -= x.Real;
            Imag -= x.Imag;
        }

        // Умножение
        public void Multiply(Complex x)
        {

        }

        // Деление
        public void Divide(Complex x)
        {

        }
    }
}
```

7. В процедуре **main** создадим несколько комплексных чисел и проведем некоторые действия над ними:

```
namespace Lab1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная №1 - Инкапсуляция");
            Console.WriteLine("Выполнил - Авраам Линкольн");

            // Создадим комплексные числа и сразу присвоим значения
            // действительной и мнимой частям
            Complex a = new Complex() { Real = 3, Imag = 0.5 };
            Complex b = new Complex() { Real = 2, Imag = 0.8 };

            // К числу a прибавим число b
            a.Add(b);

            // Из числа b вычтем число a
            b.Substract(a);

            // Выведем результат
            Console.WriteLine("a = {0} + {1}i", a.Real, a.Imag);
            Console.WriteLine("b = {0} + {1}i", b.Real, b.Imag);
        }
    }
}
```

8. Запустим приложение:



Осталось только реализовать методы умножения и деления и вывести результат.

Чтобы в полной мере осознать содеянное, вернемся к определению.

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе, и скрыть детали реализации от пользователя.

Поздравляем тебя!

Не подразумевая того ты задействовал один из главных принципов объектно-ориентированного программирования - **инкапсуляцию**. Ты объединил данные комплексного числа (**Real** и **Imag**) и методы (**Add**, **Subtract**, **Multiply** и **Divide**), работающие с ними, в одном классе **Complex**.

Задание 2

По аналогии с заданием 1 создайте класс **Student**, добавить в него данные об имени, поле, цвете волос, возрасте и многом другом. Создайте метод (или функцию) сложения двух студентов. Или даже трех. Интересно, что получится из этого? Прояви как можно больше фантазии: чем больше фантазии - тем больше баллов.

Вопросы

1. Эволюция методологий программирования.
2. Парадигмы программирования.
3. Основные принципы объектного подхода
4. Средства инкапсуляции C#
5. Задачи, решаемые применением инкапсуляцией

ЛАБОРАТОРНАЯ РАБОТА №2: НАСЛЕДОВАНИЕ

Наследование — механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса или интерфейса. Потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами.

Класс, от которого произошло наследование, называется **базовым** или **родительским** (англ. base class). Классы, которые произошли от базового, называются **потомками**, **наследниками** или **производными классами** (англ. derived class).

Задание 1

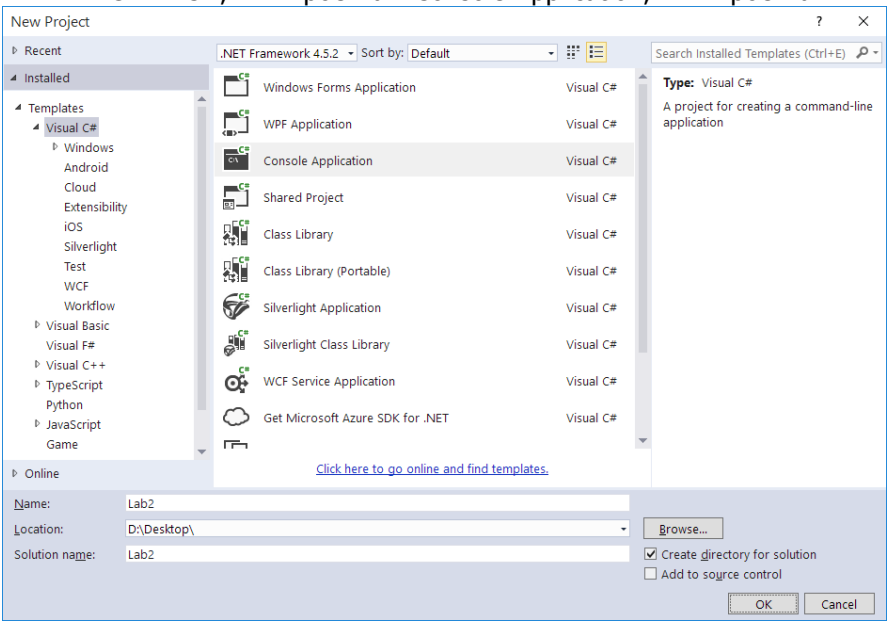
Создайте консольное приложение C#, в котором описать иерархию классов, представляющих различные геометрические фигуры. При этом будет применен один из главных принципов ООП - наследование.

1. Запускаем VisualStudio

Lab2.

2. Создаем новый проект (File/New/Project)

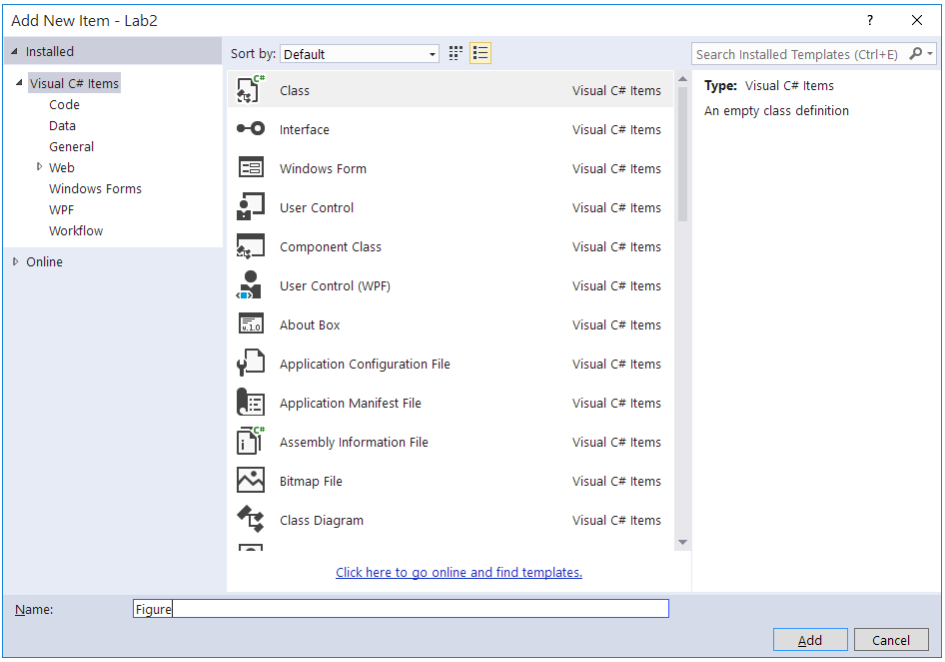
Язык - C#, тип проекта - Console Application, имя проекта -



3. Выведем в консоль название лабораторной и имя автора:

```
7   namespace Lab2
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              Console.WriteLine("Лабораторная №2 - Наследование");
14              Console.WriteLine("Выполнил - Исаак Ньютон");
15          }
16      }
17  }
```

4. Теперь создадим новый класс (Project/Add Class) и назовем его Figure:



5. Создадим публичное свойство **Name** типа **String** для хранения названия фигуры. Также создадим функцию **GetArea** для расчета площади фигуры. Функцию **GetArea** как и сам класс **Figure** пометим как **abstract** (абстрактный), так как мы описываем не конкретно какую-то фигуру, а её абстрактное представление. Таким образом, мы создали **абстрактный** класс, который будем использовать для порождения **классов-потомков** реальных фигур.

```
 7   namespace Lab2
 8   {
 9       public abstract class Figure
10       {
11           public string Name { get; set; }
12
13           public abstract double GetArea();
14       }
15   }
```

6. Создадим новый класс (Project/Add Class) **Rectangle** - прямоугольник. Родителем данного класса будет являться класс **Fugure**. Таким образом, свойство **Name** и функция **GetArea** перейдут по наследству и будут также находится внутри класса **Rectangle.**

7. Добавим публичные свойства **Width** и **Height** (ширину и высоту прямоугольника) типа **double**. А также реализуем расчет его площади. (Слово **override** говорит о том, что мы **переопределяем** функцию из родительского класса)

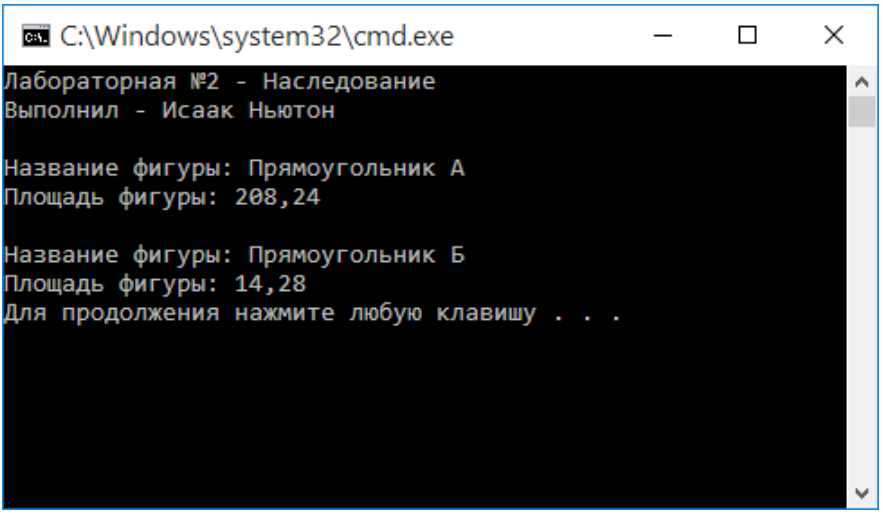
Когда мы производим наследование от абстрактного класса, мы должны обязательно раелизовать его абстрактные функции. (В нашем случае **GetArea**)

```
namespace Lab2
{
    class Rectangle : Figure
    {
        public double Width { get; set; }
        public double Height { get; set; }

        public override double GetArea()
        {
            return Width * Height;
        }
    }
}
```

8. В процедуре **main** создадим несколько прямоугольников и рассчитаем их площадь:

```
namespace Lab2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная №2 - Наследование");
            Console.WriteLine("Выполнил - Исаак Ньютон");

            // Создаем прямоугольники А и Б и на месте зададим их параметры
            Rectangle a = new Rectangle()
            {
                Name = "Прямоугольник А", Width = 15.2, Height = 13.7
            };

            Rectangle b = new Rectangle()
            {
                Name = "Прямоугольник Б", Width = 5.1, Height = 2.8
            };

            // Выведем информацию о прямоугольниках
            Console.WriteLine();
            Console.WriteLine("Название фигуры: {0}", a.Name);
            Console.WriteLine("Площадь фигуры: {0}", a.GetArea());
            Console.WriteLine();
            Console.WriteLine("Название фигуры: {0}", b.Name);
            Console.WriteLine("Площадь фигуры: {0}", b.GetArea());
        }
    }
}
```

9. Запустим приложение:



Чтобы в полной мере осознать содеянное, вернемся к определению.

Наследование — механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса или интерфейса. Потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами.

Поздравляем тебя!

Ты задействовал один из главных принципов объектно-ориентированного программирования - **наследование**. Ты создал класс **Rectangle** на основе существующего класса **Figure**. Добавил в класс потомок собственные свойства (**Width** и **Height**) и воспользовался родительским методом (функцией) **GetArea** (изменив его).

Задание 2

По аналогии с заданием 1 данной работы создайте классы следующих фигур: прямоугольник (уже готово), круг, квадрат, треугольник, трапеция, ромб, параллелограмм, правильный пятиугольник и правильный десятиугольник. Задать все необходимые свойства каждой геометрической фигуры и рассчитать её площадь. Информацию о каждой фигуре вывести на экран.

Вопросы

1. Что такое наследование с точки зрения ООП
2. Как в C# наследовать класс A от класса B
3. Дайте определение слову «abstract»
4. Объясните переопределение функции (метода)
5. Поясните, для чего используется слово «override»

ЛАБОРАТОРНАЯ РАБОТА №3: ПОЛИМОРФИЗМ

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

При использовании термина «полиморфизм» в сообществе ООП подразумевается полиморфизм подтипов; а использование параметрического полиморфизма называют обобщённым программированием.

Задание 1

На основе лабораторный работы №2 создайте консольное приложение, которое будет содержать классы следующих фигур: прямоугольник, круг, квадрат, треугольник, трапеция, ромб, параллелограмм, правильный пятиугольник и правильный десятиугольник. В каждый класс добавить координаты самой фигуры, её цвет и функцию определения координат центра фигуры. Нарисовать фигуры на форме и внутри каждой фигуры отобразить её площадь.

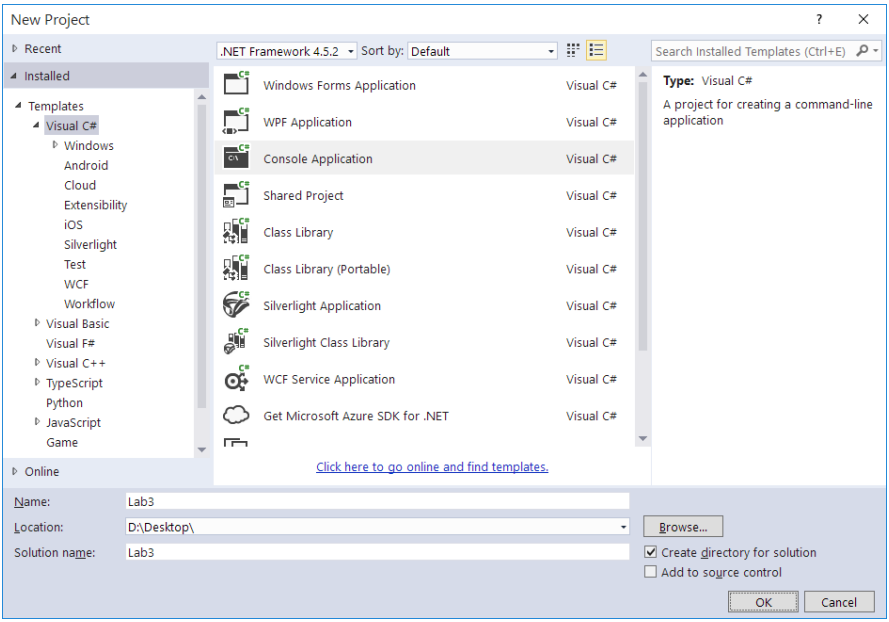
При этом будет применен один из главных принципов ООП - полиморфизм.

1. Запускаем VisualStudio

Lab3.

2. Создаем новый проект (File/New/Project)

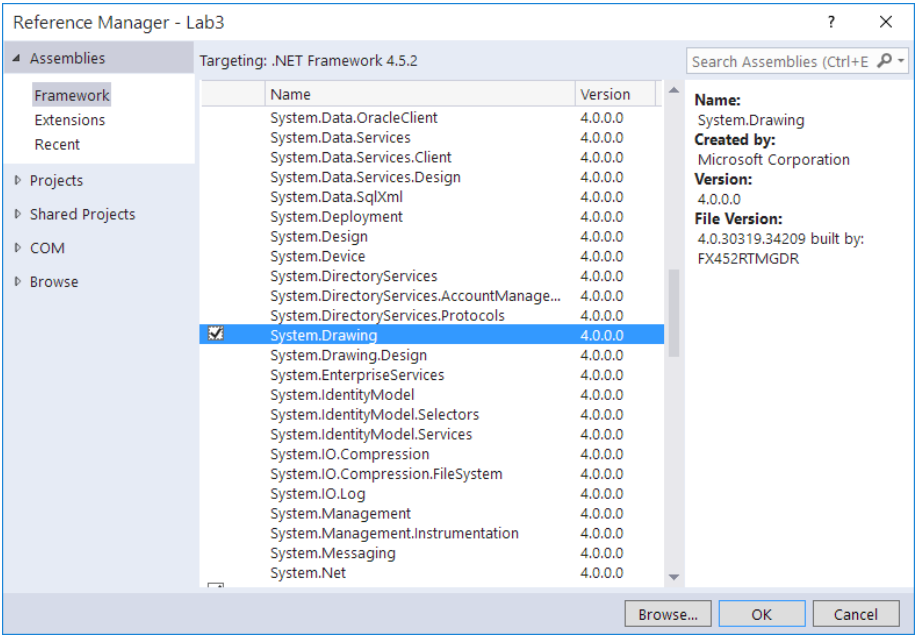
Язык - C#, тип проекта - Console Application, имя проекта –



3. Выведем в консоль название лабораторной и имя автора:

```
namespace Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная работа №3 - Полиморфизм");
            Console.WriteLine("Выполнил - Николай Коперник");
        }
    }
}
```

4. Подключим (Project/Add Reference) к проекту библиотеку **System.Drawing** для рисования фигур:



5. Создадим в классе (из лабораторной работы №2) **Figure** новую абстрактную функцию **GetCenter**, которая получает значение координат центра фигуры, поле **Position**, которое будет содержать координаты самой фигуры и поле **Color**, содержащее цвет фигуры.

```
namespace Lab3
{
    public abstract class Figure
    {
        public string Name { get; set; }

        public System.Drawing.Color Color { get; set; }

        public System.Drawing.Point Position { get; set; }

        public abstract double GetArea();

        public abstract System.Drawing.Point GetCenter();
    }
}
```

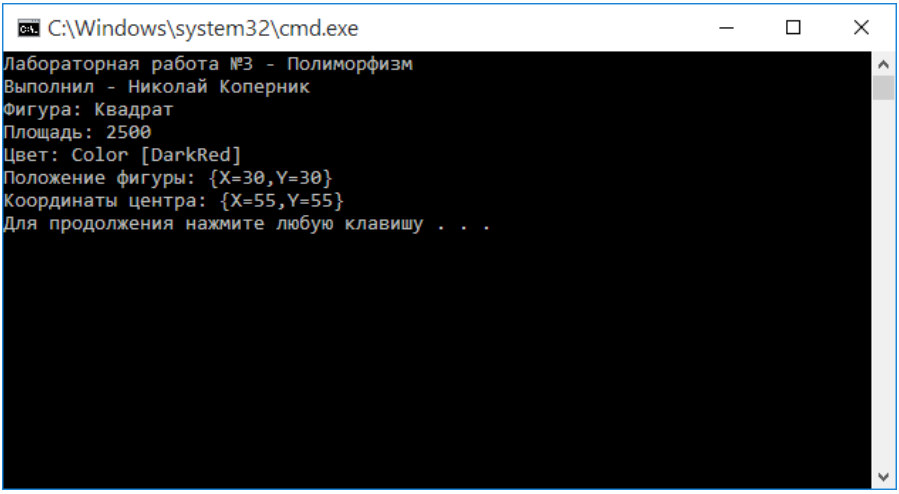
6. Реализуем функцию **GetCenter** для каждой фигуры (здесь только для **Rectangle** в качестве примера):

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

namespace Lab3
{
    class Rectangle : Figure
    {
        public double Width { get; set; }
        public double Height { get; set; }

        public override double GetArea()
        {
            return Width * Height;
        }

        public override Point GetCenter()
        {
            return new Point((int)(Position.X + Width / 2), (int)(Position.Y + Height / 2));
        }
    }
}
```

7. Создадим экземпляр класса **Rectangle** в файле **Program.cs** и выведем всю информацию о нем.

```
namespace Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Лабораторная работа №3 - Полиморфизм");
            Console.WriteLine("Выполнил - Николай Коперник");

            Figure f = new Rectangle()
            {
                Name = "Квадрат",
                Color = System.Drawing.Color.DarkRed,
                Position = new System.Drawing.Point(30, 30),
                Width = 50, Height = 50
            };

            Console.WriteLine("Фигура: " + f.Name);
            Console.WriteLine("Площадь: " + f.GetArea());
            Console.WriteLine("Цвет: " + f.Color);
            Console.WriteLine("Положение фигуры: " + f.Position);
            Console.WriteLine("Координаты центра: " + f.GetCenter());
        }
    }
}
```

И получим:



8. Создадим форму и нарисуем на ней несколько прямоугольников, для этого:

- подключим библиотеку **System.Windows.Forms** и добавим соответствующее пространство имен;

```
6    using System.Windows.Forms;
```

- создадим новую форму с заголовком **Лабораторная №3 – Полиморфизм,** размером **800x600**, и расположим её в **центре экрана;**

```
Form frm = new Form()
{
    Text = "Лабораторная №3 - Полиморфизм",
    Size = new System.Drawing.Size(800, 600),
    StartPosition = FormStartPosition.CenterScreen
};

Application.Run(frm);
```

- для того, чтобы нарисовать фигуру на форме, нужно добавить в класс **Figure** абстрактную функцию **Draw** и определить эту функцию в классе **Rectangle;**

```
public override void Draw(Graphics gr)
{
    // Рисуем прямоугольник
    gr.DrawRectangle(new Pen(Color), Position.X, Position.Y, (int)Width, (int)Height);

    // Рисуем информацию о координатах его центра
    gr.DrawString(GetCenter().ToString(), new Font("Arial", 9), Brushes.Black, GetCenter());
}
```

- создадим массив фигур и разместим в нем несколько разных прямоугольников;

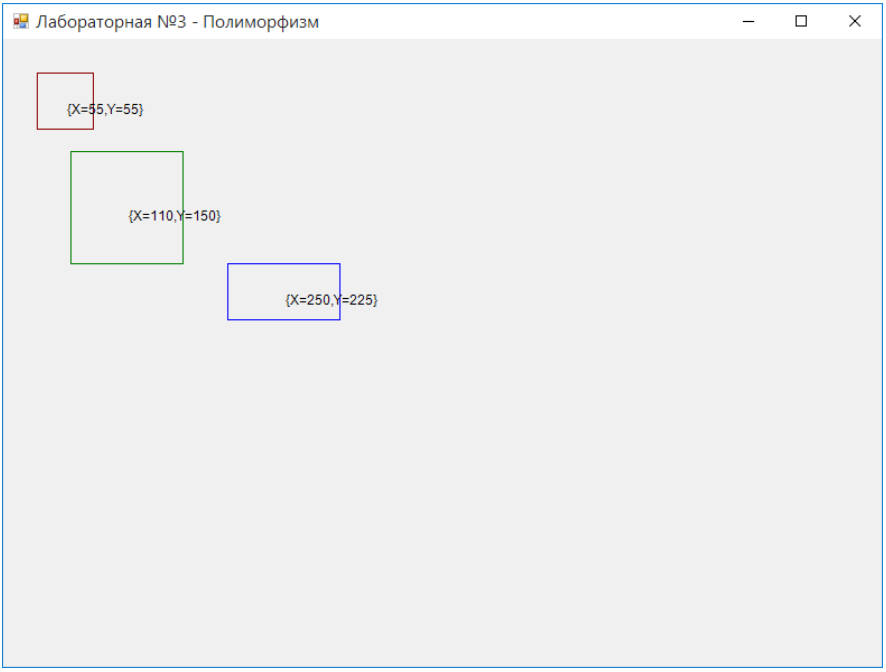
```
10      class Program
11      {
12          public static Figure[] figures = {
13              new Rectangle()
14              {
15                  Name = "Квадрат #1",
16                  Color = System.Drawing.Color.DarkRed,
17                  Position = new System.Drawing.Point(30, 30),
18                  Width = 50, Height = 50
19              },
20
21              new Rectangle()
22              {
23                  Name = "Квадрат #2",
24                  Color = System.Drawing.Color.Green,
25                  Position = new System.Drawing.Point(60, 100),
26                  Width = 100, Height = 100
27              },
28
29              new Rectangle()
30              {
31                  Name = "Прямоугольник #1",
32                  Color = System.Drawing.Color.Blue,
33                  Position = new System.Drawing.Point(200, 200),
34                  Width = 100, Height = 50
35              }};
```

- чтобы отобразить фигуру на форме и воспользоваться созданной функцией **Draw** нужно задействовать событие **Paint**;

```
37          static void Main(string[] args)
38          {
39              Console.WriteLine("Лабораторная работа №3 - Полиморфизм");
40              Console.WriteLine("Выполнил - Николай Коперник");
41
42              Form frm = new Form()
43              {
44                  Text = "Лабораторная №3 - Полиморфизм",
45                  Size = new System.Drawing.Size(800, 600),
46                  StartPosition = FormStartPosition.CenterScreen
47              };
48
49              frm.Paint += Frm_Paint;
50
51              Application.Run(frm);
52          }
53
54          private static void Frm_Paint(object sender, PaintEventArgs e)
55          {
56              foreach (Figure f in figures)
57              {
58                  f.Draw(e.Graphics);
59              }
60          }
61      }
62  }
```

- в событии **Paint** с помощью цикла **foreach** перебираем все фигуры из массива **figures** и вызываем функцию **Draw**, для отображения фигур на форме. В этот момент как раз и происходит реализация **полиморфизма** (в массиве **figures** могут быть не только прямоугольники, а все производные от класса **Figure**).

В результате получим:



Осталось только реализовать приведенные выше функции для всех фигур и вывести их на форму.

Чтобы в полной мере осознать содеянное, вернемся к определению.

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Поздравляем тебя!

Ты задействовал один из главных принципов объектно-ориентированного программирования - **полиморфизм**. Таким образом ты задействовал объекты с общим интерфейсом, объявленным в классе **Figure** и отобразил на форме эти фигуры, используя функцию **Draw**. При этом не важно какие именно фигуры используются: прямоугольники, окружности, треугольники. Главное, чтобы они имели общего предка **Figure**.

Задание 2

По аналогии с заданием 1 данной работы создайте классы следующих фигур: прямоугольник (уже готово), круг, квадрат, треугольник, трапеция, ромб, параллелограмм, правильный пятиугольник и правильный десятиугольник. Задать все необходимые свойства каждой геометрической фигуры и рассчитать её площадь. Информацию о каждой фигуре вывести на экран.

Вопросы

1. Что такое полиморфизм
2. Что такое перегрузка функций / перегрузка методов
3. Какие бывают формы полиморфизма
4. Что позволяет выполнять полиморфизм-перегрузка

ЛАБОРАТОРНАЯ РАБОТА №4: ИНТЕРФЕЙСЫ

Интерфейс (interface) — это коллекция общедоступных методов и свойств, сгруппированных для инкапсуляции конкретной функциональности.

Определив интерфейс, его можно реализовать в классе, т.е. данный класс будет осуществлять поддержку все члены и свойства указанного интерфейса. Интерфейсы могут лишь определять члены, не реализовывать их. Эту функцию выполняют классы, реализующие данный интерфейс.

Для реализации интерфейса в классе необходимо предоставить реализации методов, описанных в этом интерфейсе. Один и тот же интерфейс можно реализовать в нескольких классах по-разному. При этом, каждый из них должен поддерживать один и тот же набор методов данного интерфейса. На примере взаимодействия с интерфейсами закрепим еще раз один из основных принципов ООП - полиморфизм интерфейс один, а методов множество.

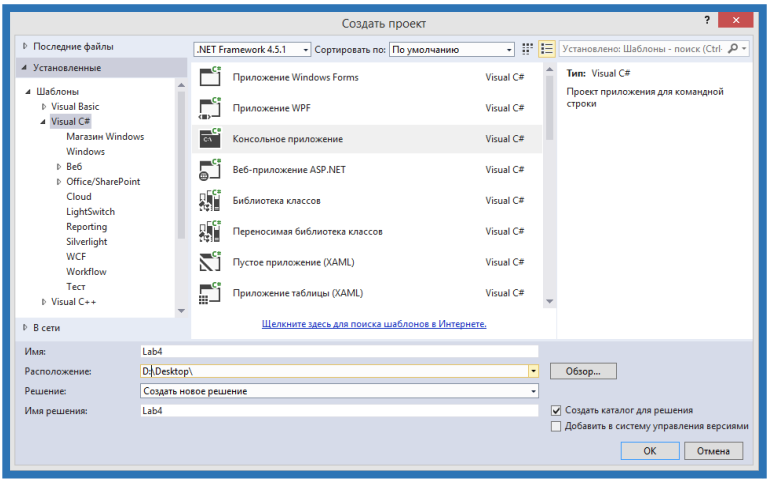
Задание 1

Создадим приложение, реализующее работу с интерфейсом.

1. Запускаем VisualStudio

2. Создаем новый проект (File/New/Project)

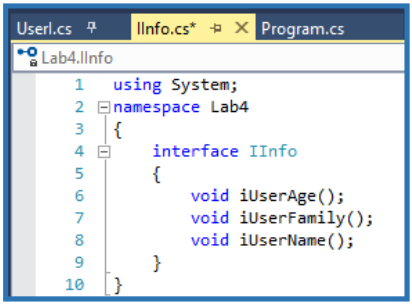
Язык - C#, тип проекта - Console Application, имя проекта – Lab4.



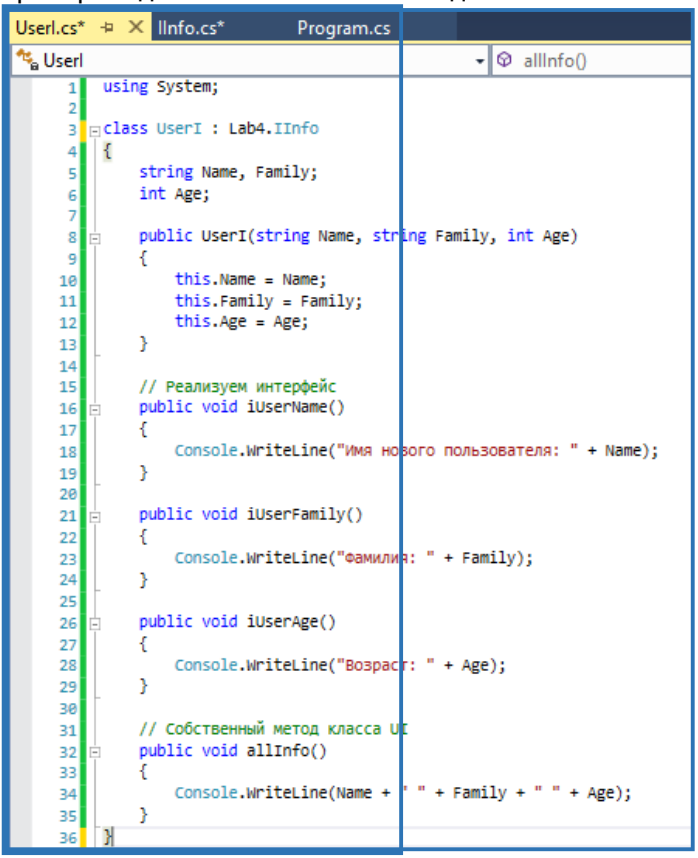
3. Выведем в консоль название лабораторной и имя автора:

```
7  namespace Lab4
8  {
9      class Program
10     {
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Лабораторная работа №4 - интерфейсы");
14             Console.WriteLine("Выполнил Станислав Лем");
15         }
16     }
17 }
```

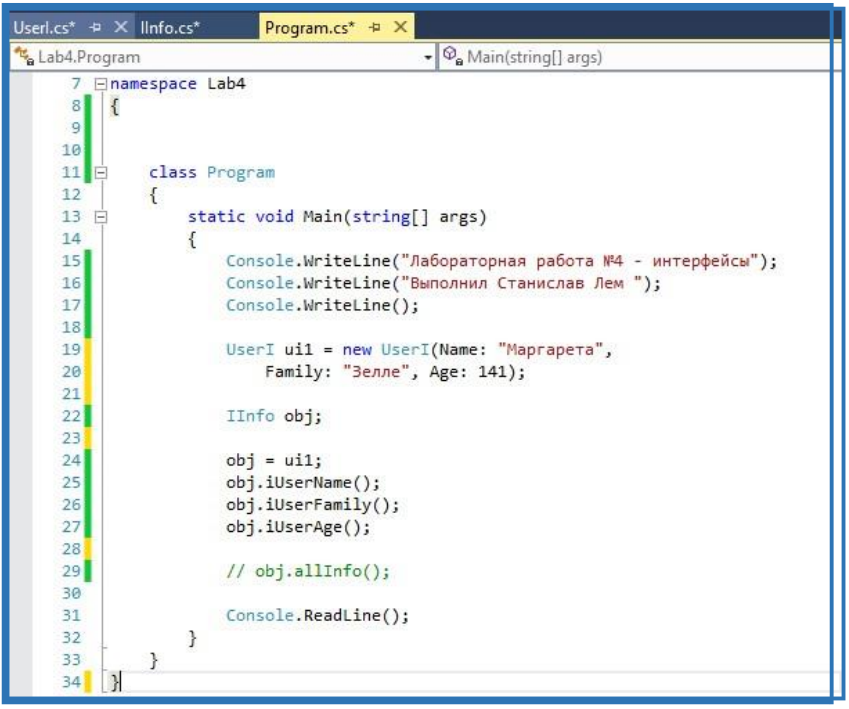
4. Определим интерфейс IInfo



5. Создадим класс UserI.cs, в котором реализуем описанный выше интерфейс IInfo (строки 15-30). В строке 32-35 показан пример создания собственного метода в классе UserI:



6. В Program.cs будем использоват разработанные интерфейс и класс. Создадим экземпляр класса (строка 19) и ссылку на интерфейс IInfo (строка 20). Далее используем ссылку на объект ***ui1.*** При этом нельзя вызывать собственный метод класса (строка 28).



Чтобы в полной мере осознать содеянное, вернемся к определению.

Интерфейс (interface) — это коллекция общедоступных методов и свойств, сгруппированных для инкапсуляции конкретной функциональности

Поздравляем тебя!

Ты научился создавать переменные ссылочного интерфейсного типа, т.е. выполнил работу с интерфейсными ссылками. При этом переменной ссылки на интерфейс доступны лишь те методы, которые объявлены в ее интерфейсе.

Задание 2

Представим, что на планете Земля остались лишь два вида игроков: футболисты и пианисты. Каждая из групп играет по-своему. Необходимо создайте интерфейс, в котором существует описание метода, характеризующее игру и классы, базирующиеся на различиях в играх. Реализацию необходимо выполнить именно в этих классах.

Вопросы

1. Как создайте интерфейс
2. Как вызвать метод интерфейса
3. Назовите причины использования интерфейсов
4. Можно ли реализовать наследование от разных интерфейсов, имеющих один и тот же метод.

САМОСТОЯТЕЛЬНАЯ РАБОТА

Задание 1

1. Создайте абстрактный класс «Геометрические фигуры» с функциями: вычисления площади, вычисления периметра, вывода информации о фигуре на экран.

2. В данном классе реализуйте метод CompareTo и отсортируйте объекты на основе значений их площади.

3. Создайте производные классы: «Прямоугольник», «Круг», «Треугольник».

Задание 2

1. Создайте абстрактный класс «Издательство» с функциями, вывода информации об издании, дате публикации(год), является ли данное издание искомым.

2. В абстрактном классе Edition реализовать метод CompareTo так, чтобы можно было отсортировать каталог изданий по фамилии автора.

3. Создайте производные классы: Book (название, фамилия автора, год издания, издательство), Article (название, фамилия автора, название журнала, его номер и год издания), OnlineResource (название, фамилия автора, ссылка, аннотация).

Задание 3

1. Создайте абстрактный класс «Транспорт» с функциями, позволяющими вывести на экран информацию о транспортном средстве, а также определить марку транспортного средства.

2. В абстрактном классе «Транспорт» реализуйте метод CompareTo для сортировки автомобилей по их марке.

3. Создайте производные классы: «Машина» (марка, номер, скорость, грузоподъемность), «Мотоцикл» (марка, номер, скорость, грузоподъемность, наличие коляски, при этом если коляска отсутствует, то грузоподъемность равна 0), Truck (марка, номер, скорость, грузоподъемность, наличие прицепа, при этом если есть прицеп, то грузоподъемность увеличивается в два раза).

Задание 4

1. Создайте абстрактный класс «Функция» с функциями вычисления значения по формуле y=f(x) в заданной точке, а также функцией, выводящей информацию о виде функции на экран.

2. В данном классе реализуйте метод CompareTo, выполняющий сортировку функций по коэффициенту a.

3. Создайте производные классы: «Уравнение прямой» (y=ax+b), «Уравнение квадратичное» (y=ax2 +bx+c), «Уравнение гиперболы» (y=a/x).

Задание 5

1. Создайте абстрактный класс «Гость» с функциями, позволяющими вывести на экран информацию о человеке, его возраст и пол.

2. В данном классе реализуйте метод CompareTo для сортировки по возрасту.

3. Создайте производные классы: «Студент» (фамилия, дата рождения, факультет, курс), «Преподаватель» (фамилия, дата рождения, факультет, должность, стаж). «Родитель» (фамилия, дата рождения, фамилия ребенка, курс)

Задание 6

1. Создайте абстрактный класс «Товары» с выводом информации о товаре, поставщике и дате производства.

2. В данном классе реализовать метод CompareTo для сортировки товаров по цене

3. Создайте производные классы: «Молочные продукты» (название, цена, дата производства, срок годности), «Хлебобулочные изделия» (название, цена, количество шт, дата производства, срок годности).

Задание 7

1. Создайте абстрактный класс «Товары» с выводом информации о товаре, поставщике и дате производства

2. В данном классе реализуйте метод CompareTo так, чтобы можно было отсортировать данные по размеру.

3. Создайте производные классы: «Обувь» (название, цена, производитель, материал, размер), «Одежда» (название, цена, производство, размер).

Задание 8

1. Создайте абстрактный класс «Справочник» с функциями, позволяющими вывести на экран информацию о записях в телефонном справочнике, а также определить соответствие записи критерию поиска.

2. В данном классе реализуйте метод CompareTo для сортировки по номеру телефона.

3. Создайте производные классы: «Друзья» (фамилия, ад-

рес, номер телефона), «Организация» (название, адрес, телефон, факс, контактное лицо), «Коллеги» (фамилия, адрес, номер телефона, дата рождения).

Задание 9

1. Создайте абстрактный класс «Клиент» с функциями, позволяющими вывести на экран информацию о клиентах аптеки, а также определить соответствие клиента критерию поиска.
2. В данном классе реализовать метод CompareTo сортирующий клиентов по дате последней покупки
3. Создайте производные классы: «Юридическое лицо»(общая сумма покупок, фамилия, дата последней покупки, номер накладной на отгрузку), «Физическое лицо» (общая сумма покупок, фамилия, дата последней покупки).

Задание 10

1. Создайте абстрактный класс «Программное обеспечение» с методами, выводящими информацию о программном обеспечении, дате обновления лицензии.
2. В данном классе реализовать метод CompareTo для сортировки ПО по названию.
3. Создайте производные классы: «Свободное ПО» (название, производитель), «Демо-версии» (название, производитель, дата установки, срок бесплатного использования), «Платное ПО» (название, производитель, цена, дата установки, срок использования).

ИТОГОВОЕ ЗАДАНИЕ

В ходе выполнения итоговой работы необходимо разработать приложение, тестирующее систему классов, спроектированную и реализованную для решения конкретной задачи из некоторой предметной области. При этом отчет должен содержать:

- анализ задания;
- разработка библиотеки классов;
- разработка тестового приложения;
- оформление отчета по результатам работы.

Описание классов необходимо оформить в виде отдельного модуля. Иерархия классов должна включать минимум четыре класса, один из которых – абстрактный.

Варианты работ

Библиотека классов, реализующих комплексную арифметику.
Библиотека классов, реализующих векторы в n-мерном пространстве.
Реализация строк и операций над ними, включая работу с регулярными выражениями.
Реализация различных типов графов и операций над ними.
Система классов для обеспечения работы с абонентами телефонной компании.
Система классов для обеспечения работы деканата.
Система классов, описывающих сотрудников предприятия/организации с их функциями (сотрудник, менеджер, …).
Система классов, описывающих различные транспортные средства.
Моделирование замкнутой биологической системы (корм, травоядное, хищник).
Моделирование муравейника (несколько типов муравьёв, источники питания, внешние раздражители,…).
Моделирование дорожного движения на заданной карте дорог.
Моделирование компьютерной сети (стационарной).
Моделирование планетарной системы.
Система классов для реализации матричной арифметики.
Библиотека классов для реализации игрового приложения.

ПРАКТИЧЕСКИЕ ВОПРОСЫ

1. Что выведет код, представленный ниже?

```
class Foo
{
    protected class Quux
    {
        public Quux()
        {
            Console.WriteLine("Foo.Quux()");
        }
    }
}

class Bar : Foo
{
    new class Quux
    {
        public Quux()
        {
            Console.WriteLine("Bar.Quux()");
        }
    }
}

class Baz : Bar
{
    public Baz()
    {
        new Quux();
    }
}
void Main()
{
    new Baz();
}
```

2. Что выведет код, представленный ниже

```
class Foo<T>
{
    public static int Bar;
}
void Main()
{
    Foo<int>.Bar++;
    Console.WriteLine(Foo<double>.Bar);
}
```

38. Что выведет код, представленный ниже

```
class Foo
{
    public Foo()
    {
        Quux();
    }
    public virtual void Quux()
    {
        Console.WriteLine("Foo.Quux()");
    }
}
class Bar : Foo
{
    protected string name;
    public Bar()
    {
        name = "Bar";
    }
    public override void Quux()
    {
        Console.WriteLine("Bar.Quux(), " + name);
    }
    public void Quux(params object[] args)
    {
        Console.WriteLine("Bar.Quux(params object[])");
    }
}
class Baz : Bar
{
    public Baz()
    {
    name = "Baz";
    Quux();
    ((Foo)this).Quux();
    }
}
void Main()
{
    new Baz();
}
```

39. Что выведет код, представленный ниже

```
class Foo
{
    public virtual void Quux(int a)
    {
        Console.WriteLine("Foo.Quux(int)");
    }
}
class Bar : Foo
{
    public override void Quux(int a)
    {
        Console.WriteLine("Bar.Quux(int)");
    }
    public void Quux(object a)
    {
        Console.WriteLine("Bar.Quux(object)");
    }
}
class Baz : Bar
{
    public override void Quux(int a)
    {
        Console.WriteLine("Baz.Quux(int)");
    }
    public void Quux<T>(params T[] a)
    {
        Console.WriteLine("Baz.Quux(params T[])");
    }
}
void Main()
{
    new Bar().Quux(42);
    new Baz().Quux(42);
}
```

5. Что выведет код, представленный ниже

```
void Foo(object a)
{
    Console.WriteLine("object");
}
void Foo(object a, object b)
{
    Console.WriteLine("object, object");
}
void Foo(params object[] args)
{
    Console.WriteLine("params object[]");
}
void Foo<T>(params T[] args)
{
    Console.WriteLine("params T[]");
}
class Bar { }
void Main()
{
    Foo();
    Foo(null);
    Foo(new Bar());
    Foo(new Bar(), new Bar());
    Foo(new Bar(), new object());
}
```

СПИСОК ЛИТЕРАТУРЫ

1. Курс UniTrain-I "Автоматическое управление температурой, скоростью и светом", www.unitrain-i.com.

2. В.А. Бесекерский, Е.П. Попов «Теория автоматического управления», СПБ, Изд-во «Профессия», 2003.-752с.

3. Л.Д. Певзнер «Практикум по теории автоматического управления»: Учеб. пособие-М.: Высш. шк., 2006.-590с.

4. Современные системы управления/ Р. Дорф, Р. Бишоп. Пер. с англ. Б.И. Копылова.- М.: Лаборатория Базовых Знаний, 2002.-832 с.:ил.

5. Ватсон, Б. C# 4.0 на примерах / Б. Ватсон. – СПб. : БХВ-Петербург, 2011. – 608 с. : ил.

6. Материалы свободной энциклопедии «Википедиа». – http://ru.wikipedia.org/wiki/ (режим доступа – свободный, дата обращения: 28.10.2016)

7. Д.Э. Кнут. Искусство программирования, том 3. Пер. с англ. М.: Издательский дом "Вильямс", 2007. (и др. издания)