**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

Факультет «Информатика и вычислительная техника»

Кафедра ИУ5 «Системы обработки информации и управления»

**Отчет по лабораторной работе № 2**

Задание: разработать программу, реализующую работу с классами

Студент Левин Михаил Александрович

*фамилия, имя, отчество*

Группа ИУ5-32Б

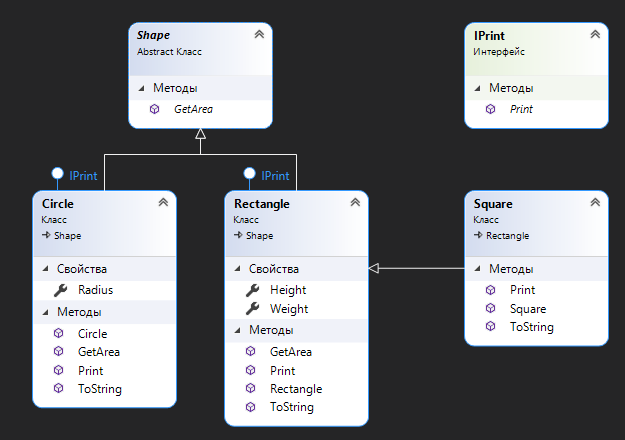
Москва, 2020 г.

# Задание

Разработать программу, реализующую работу с классами.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Абстрактный класс «Геометрическая фигура» содержит виртуальный метод для вычисления площади фигуры.
3. Класс «Прямоугольник» наследуется от «Геометрическая фигура». Ширина и высота объявляются как свойства (property). Класс должен содержать конструктор по параметрам «ширина» и «высота».
4. Класс «Квадрат» наследуется от «Прямоугольник». Класс должен содержать конструктор по длине стороны.
5. Класс «Круг» наследуется от «Геометрическая фигура». Радиус объявляется как свойство (property). Класс должен содержать конструктор по параметру «радиус».
6. Для классов «Прямоугольник», «Квадрат», «Круг» переопределить виртуальный метод Object.ToString(), который возвращает в виде строки основные параметры фигуры и ее площадь.
7. Разработать интерфейс IPrint. Интерфейс содержит метод Print(), который не принимает параметров и возвращает void. Для классов «Прямоугольник», «Квадрат», «Круг» реализовать наследование от интерфейса IPrint. Переопределяемый метод Print() выводит на консоль информацию, возвращаемую переопределенным методом ToString().

# Диаграмма классов



# Текст программы

**IPrint.cs**

using System;

namespace Lab2

{

class Circle:Shape,IPrint

{

public double Radius { get; set; }

public Circle(double r)

{

Radius = r;

}

public override double GetArea()

{

return Math.PI \* Radius \* Radius;

}

public override string ToString()

{

return $"Круг. Радиус: {Radius} Площадь: {GetArea()}";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Shape.cs**

namespace Lab2

{

/// <summary>

/// Абстрактный класс геометрическая фигура

/// </summary>

abstract class Shape

{

/// <summary>

/// Метод вычисления площади

/// </summary>

/// <returns></returns>

public abstract double GetArea();

}

}

**Rectangle.cs**

using System;

namespace Lab2

{

/// <summary>

/// Класс прямоугольник

/// </summary>

class Rectangle:Shape,IPrint

{

public double Height { get; set; }

public double Weight { get; set; }

public override double GetArea()

{

return Height \* Weight;

}

public Rectangle(double w, double h)

{

Weight = w;

Height = h;

}

public override string ToString()

{

return $"Прямоугольник. Ширина: {Weight}. Высота: {Height}. Площадь: {GetArea()}";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Square.cs**

using System;

namespace Lab2

{

/// <summary>

/// Квадрат

/// </summary>

class Square:Rectangle

{

public Square(double l):base(l,l)

{

}

public override string ToString()

{

return $"Квадрат. Длина стороны: {Weight}. Площадь: {GetArea()}";

}

public new void Print()

{

Console.WriteLine(ToString());

}

}

}

**Circle.cs**

using System;

namespace Lab2

{

class Circle:Shape,IPrint

{

public double Radius { get; set; }

public Circle(double r)

{

Radius = r;

}

public override double GetArea()

{

return Math.PI \* Radius \* Radius;

}

public override string ToString()

{

return $"Круг. Радиус: {Radius} Площадь: {GetArea()}";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Programs.cs**

using System;

using System.Runtime.InteropServices;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите длину стороны квадрата: ");

double l = double.Parse(Console.ReadLine());

var square = new Square(l);

Console.Write("Введите ширину прямоугольника: ");

double w = double.Parse(Console.ReadLine());

Console.Write("Введите ширину высоту: ");

double h = double.Parse(Console.ReadLine());

var rectangle = new Rectangle(w,h);

Console.Write("Введите радиус круга: ");

double r = double.Parse(Console.ReadLine());

var circle = new Circle(r);

//создаём массив объектов, реализующих интерфейс IPrint

IPrint[] figures = new IPrint[] {square, rectangle, circle};

//вызываем методы Print для всех добавленных объектов

foreach (IPrint figure in figures)

{

figure.Print();

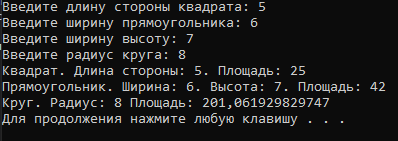
}

}

}

}

# Экранные формы



# Контрольные вопросы:

1. В чем разница между ключевыми словами «override» и «new» при переопределении виртуального метода?

В C# метод в производном классе может иметь то же имя, что и метод в базовом классе. Можно задать способ взаимодействия методов, воспользовавшись ключевыми словами new и override. Модификатор override **расширяет** метод virtual базового класса, а модификатор new **скрывает** доступный метод базового класса.

1. Как переопределить виртуальный метод?

Для переопределения виртуального метода родительского класса, необходимо в дочернем классе создать метод с таким же именем и сигнатурой с использованием ключевого слова override в сигнатуре метода.

1. Как реализуется наследование класса от класса?

Синтаксически для наследования одного класса от другого, необходимо в описании класса после его имени добавить двоеточие и написать имя родительского класса.

1. Как объявить конструктор класса в C#?

Конструкторы классов имеют отличающуюся от других методов сигнатуру, особенность которой заключается в том, что она не имеет возвращаемого типа и метод носит имя класса.

1. Как из конструктора класса вызвать конструктор базового класса?

Для вызова в конструкторе класса конструктор базового класса можно в сигнатуре метода поставить двоеточие и ключевое слово **base** (передав при необходимости в base требуемые параметры).

1. Что такое свойства и для чего они используются?

Свойство — это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного поля. Свойства можно использовать, как если бы они были членами общих данных, но фактически они представляют собой специальные методы, называемые методами доступа. Это позволяет легко получать доступ к данным и помогает повысить безопасность и гибкость методов.

1. Что такое опорная переменная свойства?

Опорная переменная свойства – это закрытое резервное поле, созданное для установки и извлечения значения свойства. Метод доступа get возвращает значение закрытого поля, а метод доступа set может выполнять определенные проверки данных до присвоения значению закрытого поля. Оба метода доступа также могут выполнять некоторые преобразования или вычисления данных до их сохранения или возвращения.

1. Как задаются области видимости для свойств и аксессоров?

Выражения get и set свойства или индексатора называются методами доступа. По умолчанию они имеют такие же уровни видимости или доступа, что и свойство или индексатор, которым они принадлежат. При необходимости можно установить область видимости аксессоров, например, сделав их видимыми только в дочерних классах, используя ключевое слово protected.

1. Что такое абстрактный класс?

Модификатор abstract указывает, что изменяемый элемент имеет отсутствующую или неполную реализацию. Модификатор abstract в объявлении класса, указывает на то, что класс предназначен только для использования в качестве базового класса для других классов и не должен быть создан сам по себе. Иными словами, абстрактные классы не могут иметь экземпляров, а служат только в роли базовых классов для реализации дочерних классов.

1. Что такое интерфейсы и для чего они используются в C#?

Интерфейс содержит определения для группы связанных функциональных возможностей, которые может реализовать неабстрактный класс или структура. Интерфейс не может объявлять данные экземпляра, такие как поля, автоматические реализуемые свойства или события, подобные свойствам.

С помощью интерфейсов можно, например, включить в класс поведение из нескольких источников. Эта возможность очень важна в C#, поскольку этот язык не поддерживает множественное наследование классов.