# Лабораторная работа №3

Наследование в C#. Абстрактные классы

## 1) Наследование

Наследование, вместе с инкапсуляцией и полиморфизмом, является одной из трех основных характеристик (или базовых понятий) объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах. Класс, члены которого наследуются, называется базовым классом, а класс, который наследует эти члены, называется производным классом. Производный класс может иметь только один непосредственный базовый класс. Однако наследование является транзитивным. Если ClassC является производным от ClassB, и ClassB является производным от ClassA, то ClassC наследует члены, объявленные в ClassB и ClassA.

При определении класса для наследования от другого класса, производный класс явно получает все члены базового класса, за исключением его конструкторов и деструкторов. Производный класс может таким образом повторно использовать код в базовом классе без необходимости в его повторной реализации. В производном классе можно добавить больше членов. Таким образом, производный класс расширяет функциональность базового класса.

Рассмотрим механизм наследования на примере классов «Человек» и «Студент»:

```
class Human
{
    public String Name { get; set; } // Свойство Имя
    public String Surname { get; set; } // Свойство Фамилия
    public int Age { get; set; } // Свойство Возраст

    public String getInfo()
    {
        return Surname + " " + Name;
    }
```

```
}

class Student : Human
{
    public bool IsSloven { get; set; } // Свойство Разгильдяй
    public String Group { get; set; } // Свойство Группа
}
```

Класс Student наследует класс Human (используется знак «двоеточие»), и все его поля и методы, помеченные модификаторами доступа protected и public. К членам с модификатором private обратиться из производного класса нельзя. Пример реализации:

```
Student student1 = new Student();
student1.Name = "Sergey";
student1.Surname = "Perchenko";
student1.IsSloven = true;
Console.WriteLine(student1.getInfo());
```

В этом примере создается экземпляр класса Student, устанавливаются свойства, и вызывается метод getInfo(), реализованный в классе Human.

**2) Ключевое слово base**

Ключевое слово base используется для обращения к членам базового класса из производного класса. Чаще всего используется для вызова конструктора базового класса:

```
class Human
{
    private String mName;
    private String mSurname;

    public int Age { get; set; }

    public Human(String name, String surname)
    {
        mName = name;
        mSurname = surname;
    }

    public String getInfo()
    {
        return mSurname + " " + mName;
    }
}

class Student : Human
{
    public bool IsSloven { get; set; } // Свойство Разгильдяй
    public String Group { get; set; } // Свойство Группа

    public Student(String name, String surname) : base (name, surname)
    {
    }
```

```
}
```

В этом примере класс Human содержит закрытые поля mName, mSurname, значения которых устанавливаются в конструкторе класса. Так как класс Human не содержит конструкторов по умолчанию (без аргументов), то при наследовании класс Student обязан вызвать единственный конструктор класса Human. В этом примере конструктор класса Student принимает 2 аргумента – имя и фамилию, и передает их конструктору базового класса – Human.

### 3) Ключевое слово new

Иногда производные классы содержат методы, имя которых совпадает с именем методов базового класса. Рассмотрим пример класса Student (класс Human такой же, как в пункте 2).

```
class Student : Human
{
    public bool IsSloven { get; set; } //Свойство Разгильдяй
    public String Group { get; set; } //Свойство Группа

    public Student(String name, String surname) : base (name, surname)
    {
    }
    public new String getInfo()
    {
        return Group;
    }
    public String getBaseInfo()
    {
        return base.getInfo();
    }
    public String getNewInfo()
    {
        return getInfo();
    }
}
```

Этот класс содержит три метода получения информации.

1) getInfo() с модификатором доступа new, который сообщает компилятору, что будет использоваться название метода базового класса. Если этот модификатор не поставить, то компилятор выдаст сообщение об ошибке.

2) getBaseInfo(). Вызывает метод базового класса, для этого используется ключевое слово base.

3) getNewInfo(). Вызывает метод getInfo() класса Student.

Примеры вызова:

```
Student student1 = new Student("Ivan", "Ivanov");
student1.Group = "GGG-111";
Console.WriteLine(student1.getBaseInfo());
```

Результат: Ivanov Ivan

```
Console.WriteLine(student1.getInfo());
```

Результат: GGG-111

```
Console.WriteLine(student1.getNewInfo());
```

Результат: GGG-111

Язык C# поддерживает полиморфизм, то есть к производному классу можно обращаться через ссылку, имеющую тип класса родителя. В нашем случае этот выглядит так:

```
Student student1 = new Student("Ivan", "Ivanov");
student1.Group = "GGG-111";
Human human1 = student1;
Console.WriteLine(human1.getInfo());
```

Но результат выполнения этого кода отличается от приведенных выше, так как метод getInfo() вызывается у класса Human, а не Student:

Результат: Ivanov Ivan

**4) Виртуальные методы и их переопределение**

Язык C# поддерживает переопределение методов и свойств – то есть, производный класс может заменять методы и свойства базового класса. Методы, допускающие переопределение называются виртуальными. Для того, чтобы объявить виртуальный метод используется ключевое слово virtual, например, метод getInfo() класса Human:

```
class Human
{
    private String mName;
    private String mSurname;

    public int Age { get; set; } // Свойство Возраст

    public Human(String name, String surname)
    {
        mName = name;
        mSurname = surname;
    }
    public virtual String getInfo()
    {
        return mSurname + " " + mName;
    }
```

```
}
```
Для переопределения метода используется ключевое слово override. Для класса Student:

```
class Student : Human
{
    public bool IsSloven { get; set; } //Свойство Разгильдяй
    public String Group { get; set; } //Свойство Группа

    public Student(String name, String surname) : base (name, surname)
    {
    }

    public override String getInfo()
    {
        return Group;
    }
}
```
Пример реализации:

```
Student student1 = new Student("Ivan", "Ivanov");
student1.Group = "GGG-111";
Human human1 = student1;
Console.WriteLine(human1.getInfo());
```
Результат: GGG-111

То есть, несмотря на то, что метод getInfo() вызывался у ссылки на класс Human, был вызван метод getInfo() класса Student.

На самом деле все классы унаследованы от класса Object, который имеет виртуальный метод ToString(). В приведенных выше примерах более удачной реализацией было бы переопределение метода ToString(), а не введение нового метода.

### 5) Абстрактные классы

Классы могут быть объявлены абстрактными путем помещения ключевого слова abstract перед определением класса. Например:

```
public abstract class A
{
    // Class members here.
}
```

Создавать экземпляры абстрактного класса нельзя. Назначение абстрактного класса заключается в предоставлении общего определения для базового класса, которое могут совместно использовать несколько производных классов. Например, в библиотеке классов может быть определен абстрактный класс, используемый в качестве параметра для многих из ее

функций, поэтому программисты, использующие эту библиотеку, должны задать свою реализацию этого класса, создав производный класс.

Абстрактные классы могут определять абстрактные методы. Для этого перед типом возвращаемого значения метода необходимо поместить ключевое слово abstract. Например:

```
public abstract class A
{
    public abstract void DoWork(int i);
}
```

Абстрактные методы не имеют реализации, поэтому определение такого метода заканчивается точкой с запятой вместо обычного блока метода. Классы, производные от абстрактного класса, должны реализовывать все абстрактные методы.

Рассмотрим реализацию абстрактных классов на примере вычисления интеграла. Объявим абстрактный класс Integrator, содержащий абстрактный метод double f(double x), который будет содержать подынтегральное выражение. Метод double integrate(double a, double b, int N) производит интегрирование функции f(x), как предел суммы N прямоугольников, в диапазоне от a до b:

```
public abstract class Integrator
{
    protected abstract double f(double x); //Подынтегральное выражение

    public double integrate(double a, double b, double N)
    {
        double result = 0;
        double dx = (b - a) / N;
        for (double x = a; x < b; x += dx)
        {
            result += f(x);
        }
        result *= dx; return result;
    }
}
```

Создадим на базе этого класса другой класс, вычисляющий значение интеграла от функции $f(x) = \exp(-\alpha x)$:

```
public class ExpIntegral:Integrator
{
    private double mAlpha;
    public ExpIntegral(double alpha)
    {
        mAlpha = alpha;
    }
    protected override double f(double x)
    {
        return Math.Exp(-mAlpha * x);
    }
}
```

Пример реализации:

```
ExpIntegral integr = new ExpIntegral(1.0f);
Console.WriteLine(integr.integrate(0, 3, 1000000));
```

Таким образом, наследуя класс Integrator можно вычислить интеграл практически от любой функции, написав небольшое количество программного кода.

### 6) Задания к лабораторной работе №3

1) Создать абстрактный базовый класс Figure с абстрактными методами вычисления площади и периметра. Создать производные классы: Rectangle (Прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра.

2) Создать абстрактный базовый класс Currency (валюта) для работы с денежными суммами. Определить абстрактные методы перевода в рубли и вывода на экран. Реализовать производные классы Dollar (доллар) и Euro (евро) со своими функциями перевода и получения текстового значения.

3) Создать абстрактный базовый класс Triangle для представления треугольника с абстрактными функциями вычисления площади и периметра. Поля данных должны включать две стороны и угол между ними. Определить классы-наследники: прямоугольный треугольник, равнобедренный треугольник, равносторонний треугольник со своими методами вычисления площади и периметра.

4) Создать абстрактный базовый класс Equation (Уравнение) с абстрактными методами вычисления корней и вывода результата на экран. Определить производные классы Linear (линейное уравнение) и Square (квадратное уравнение) с собственными методами вычисления корней и вывода на экран.

5) Создать абстрактный класс Body (тело) с абстрактными функциями вычисления площади поверхности и объема. Создать производные классы: Parallelepiped (Параллелепипед) и Ball (шар) со своими функциями площади поверхности и объема.

6) Реализовать абстрактный класс "помещение". Помещение предполагается в виде прямоугольного параллелепипеда, и обязательно имеет длину, ширину и высоту. В абстрактном классе должен быть описан метод вычисления общей площади помещения и метод вычисления объёма. Определить классы наследники: склад (просто помещение, но высоту через свойство нельзя задать менее 3-х метров), офис (нежилое помещение, помимо основного помещения имеется санузел тоже имеющий площадь, но без указания ширины и высоты), квартира (помимо основной комнаты, должна указываться площадь кухни, ванны, туалета и прихожей). В классе "квартира" переопределённый метод вычисления площади помещения рассчитывает всю площадь квартиры, помимо него должен быть метод вычисления жилой площади (только комнаты). Стоит так же учитывать, что высота любых помещений почти не бывает менее 2-х метров, а площадь – менее 1 кв. м.

7) Создать абстрактный класс "Литературный источник" (практически каждый источник имеет хотя бы одного автора, название, и год издания). Унаследовать от него класса "Книга одного автора", "Книга двух авторов", "Книга трёх авторов", "Книга четырёх и более авторов", «Статья из периодического издания (журнала)". В абстрактном классе должна быть абстрактная функция получения библиографического описания.

8) Создать абстрактный класс "язык" (language) для представления языка имеющего алфавит (массив символов), содержащий абстрактный метод Hello() (возвращающий написание слова "привет" на том или ином языке) и виртуальный метод Alphabet() (возвращающий строку, в которой перечислены все буквы алфавита, разделённые запятыми). Создать три производных от него класса описывающих языки (набор языков на выбор студента, но русского среди них не должно быть).

9) Реализовать абстрактный класс очки (Glasses). В классе должны быть абстрактные методы MakeStronger() (сделать посильнее) и MakeWeaker (сделать послабее), приводящие в классах наследниках к усилению или ослаблению основных характеристик. Должны быть реализованы наследующие его классы, описывающие очки для коррекции зрения (имеют определённую оптическую силу, измеряемую в диоптриях), солнцезащитные

очки (имеющие коэффициент затемнения).

### 7) Контрольные вопросы

1) Для чего используется наследование? Как создать класс наследующий базовый?

2) Возможно ли в C# унаследовать класс сразу от нескольких базовых классов?

3) Для чего используется ключевое слово base?

4) Для чего используется ключевое слово new?

5) Для чего используются виртуальные методы?

6) Что такое абстрактные классы и методы? Каковы особенности их использования в C#?

7) Как переопределить метод? Какие методы могут быть переопределены?

8) Обязательно ли в классе наследнике переопределять все абстрактные методы базового класса?