# Практическая работа №2

## 1 Теоретические сведения

### 1.1 Наследование

Наследование (inheritance) является одним из ключевых моментов ООП. Благодаря наследованию один класс может унаследовать функциональность другого класса.

Пусть у нас есть следующий класс Person, который описывает отдельного человека:

```
class Person
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    public void Display()
    {
        Console.WriteLine(Name);
    }
}
```

Но вдруг нам потребовался класс, описывающий сотрудника предприятия - класс Employee. Поскольку этот класс будет реализовывать тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником, или подклассом) от класса Person, который, в свою очередь, называется базовым классом или родителем (или суперклассом):

```
class Employee : Person
{

}
```

После двоеточия мы указываем базовый класс для данного класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же свойства, методы, поля, которые есть в классе Person. Единственное, что не передается при наследовании, это конструкторы базового класса.

Таким образом, наследование реализует отношение is-a (является), объект класса Employee также является объектом класса Person:

```
static void Main(string[] args)
{
    Person p = new Person { Name = "Tom"};
    p.Display();
    p = new Employee { Name = "Sam" };
    p.Display();
    Console.Read();
}
```

И поскольку объект Employee является также и объектом Person, то мы можем так определить переменную: Person p = new Employee().

По умолчанию все классы наследуются от базового класса Object, даже если мы явным образом не устанавливаем наследование. Поэтому выше определенные классы Person и Employee кроме своих собственных методов, также будут иметь и методы класса Object: ToString(), Equals(), GetHashCode() и GetType().

Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений:

Не поддерживается множественное наследование, класс может наследоваться только от одного класса.

При создании производного класса надо учитывать тип доступа к базовому классу - тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа internal, то производный класс может иметь тип доступа internal или private, но не public.

Однако следует также учитывать, что если базовый и производный класс находятся в разных сборках (проектах), то в этом случае производый класс может наследовать только от класса, который имеет модификатор public.

# 2 Практические задания

## 2.1 Задание 1

Реализуйте иерархию наследования классов фигур. Базовым классом в иерархии считаем класс $\mathrm{Figure}$, обладающий целочисленными полями Width, Height и объявляет метод $\mathrm{int\ GetSquare()}$. Наследниками Figure являются класс круга – Circle, класс прямоугольника – Rectangle, класс квадрата – Square. Каждый из них реализует специфику вычисления площади фигур и сохранения их ширины и высоты.

Для решения задания создайте файлы Figure, Circle, Rectangle, Square и опишите реализации классов в соответствии с заданиями. Задание считается пройденным, если после запуска команды $\mathrm{dotnet\ test}$ все тесты успешно пройдены.

# 3 Развертывание и использование проекта

Проект выполняется в Visual Studio Code с плагинами C# и Dotnet Essentials.

Если вы не знакомы с системой контроля версий, то можете скачать проект в виде zip архива.

После скачивания проекта необходимо установить зависимости проекта dotnet restore в папке проекта.

Для успешной сдачи необходимо, что бы написанный вами код проходил автоматические тесты, которые можно запустить командой dotnet test в каталоге Test.