Занятие 10. Элементарные понятия ООП

Что такое объекты и классы

Все окружающие нас объекты – деревья, дома, автомобили, тапочки – могут быть похожими друг на друга, или же отличаться как день и ночь. Некоторые из них существуют в единственном экземпляре, например, картины художников, другие же клонированы миллиардами – например, кирпичи.

Если у объектов есть что-то общее, их можно отнести к одному классу, то есть, грубо говоря, класс – это описание схожих черт каких-то объектов. Например, уже упомянутые кирпичи называются таковыми, потому что имеют соответствующую форму, цвет, массу и прочность – все эти характеристики можно считать описанием класса «Кирпичи».

Классы могут быть более общими или менее общими, то есть частными. Общие классы описывают большее количество объектов, но при этом они менее точны. Частные же наоборот, более точны, зато количество объектов, попадающих под определение таких классов, значительно меньше. Рассмотрим на примере: три конкретных воробья, сидящие на проводе, могут быть объектами самых разных классов. В первую очередь, они входят в наиболее общий класс «Животные», потом уже в более частный «Птицы» и, наконец, в самый частный класс «Воробьи».

Однако это не означает, что их нельзя отнести к другим классам, например, «сидящие на проводе». Ну а если эти милые птички нагадили на голову, то и к классам «Мерзкие», «Противные», «Гадкие», и так далее. Голуби тоже будут объектами классов «Животные» и «Птицы», а вот к классу «воробьи» их уже не отнести, так как они у них нет свойств, присущих только воробьям.

Объекты всегда конкретны и часто материальны. Объектом класса воробей всегда будет конкретная птичка.

Классы, же, наоборот, материальными быть не могут, так как являются лишь только «описанием» группы объектов, сколько бы ни было в этой группе объектов – один или миллиард.

Система классов и объектов в нашем мире настолько очевидна, что мы редко задумываемся о ней. Программисты же испокон веков в своих программах использовали переменные, операторы и функции.

Этого было вполне достаточно для простых программ, но с увеличением сложности программного обеспечения оказалось, что гораздо удобнее переносить сущности и объекты реального мира на их виртуальные аналоги, и работать непосредственно с ними. В таком случае, программист избавлен от необходимости выдумывания своего собственного мира в каждой программе. Вместо этого он лишь переносит в свои программы кусочек нашего реального мира. Про то, как это увеличивает читабельность программ, я и не буду даже упоминать.

Объектно-ориентированное программирование (основанное на использовании объектов), оказалось очень удобным. Даже не смотря на то, что оно было придумано совсем недавно, теперь уже используется повсеместно.

Многие языки программирования со временем изменились и стали поддерживать классы и объекты, то есть стали объектно-ориентированными языками.

Преимущество языка C# перед другими объектно-ориентированными языками в том, что он изначально задумывался как полностью объектно-ориентированный язык и практически всё в нем можно с некоторой долей уверенности считать классами и объектами.

На прошлом занятии мы затронули, так называемое, структурное программирование, то есть программирование, основанное на функциях. Теперь же, мы начнем изучать программирование, которое называется объектно-ориентированным. Но, наши усилия не пропадут даром, и скоро ты увидишь почему.

Как создаются классы на языке C#

«Классы» и «объекты» в программировании, как и в реальной жизни, неразрывно связаны между собой и немного похожи на «типы данных» и «переменные». Класс, грубо говоря, это тип объекта, а сам объект является экземпляром класса и напоминает переменную, тип которой не просто целое число или массив, а некоторый класс.

Но все же это приближение недостаточно точно. Класс – это структура, содержащая в себе не только переменные различных типов, но и, самое главное, – функции для их обработки!

Переменные класса называются его *свойствами*, а функции – *методами*. Доступ к свойствам и методам объекта можно получить через точку слева направо. Например, если у нас есть объект «Джек» класса «воробей», а у всех объектов воробей есть свойство «толщина». Тогда узнать толщину нашего воробья можно, написав так: « Джек.толщина».

Вот и вся премудрость. Пришло время показать, как описываются классы и как создаются объекты.

```
public class Имя_Класса
{
      // описание свойств класса
      ...
      // описание методов класса
};
```

Описать класс довольно непросто, но, по правде сказать, программистам не так уж и часто приходится описывать свои классы. В C# уже встроено преогромное количество классов на все случаи жизни. На основе этих встроенных классов программисты создают необходимые им объекты и работают с ними. Объекты создаются следующим образом:

```
Имя_класса имя_объекта = new имя_класса();
```

Сперва, кажется, что слишком много разных имен и легко запутаться, но мы то обязательно во всем разберемся. Первые два слова запомнить легко: здесь создание объекта похоже на описание переменной, конечно если предположить, что имя_класса это тип, а объект - это переменная.

Дальше еще проще: знак равно и слово «new». Слово новое, но простое – всего-то три буквы. Оно должно быть написано всегда, когда мы создаем новый объект. Легко запомнить – всегда! Ну а в конце просто повторяем первое слово, только нужно не забыть добавить скобки и точку с запятой в самый конец. Для примера создадим из класса «Воробьи» объект «Джек»:

```
Воробьи Джек = new Воробьи();
```

По-моему ничего сверхсложного! Для тренировки неплохо было бы создать пару классов и объектов. Не потому что мы будем делать это постоянно, а просто для того чтобы попробовать, как это делается, вдруг пригодится в будущем?

Если бы мы уже стали крутыми программистами, то писали бы программы о разных серьезных и скучных вещах, но пока этого не произошло, имеем полное право заниматься ерундой.

Создадим воробья и кучку хлебных крошек. Какие свойства есть у воробьев? Они же все одинаковые. Что умеют воробьи? Да тоже почти ничего: клевать крошки и чирикать. А раз они поглощают пищу и толстеют, придумаем для них свойство «толщина». Воробьи едят крошки и толщина их увеличивается. Создадим класс воробьев, в котором опишем для них свойство «толщина» и методы «чирикать» и «клевать». Метод «клевать» будет увеличивать свойство «толщина», а метод «чирикать» будет выводить на экран сообщение «Чирик!».

```
public class Sparrow    // описание класса воробей
{
      public int Thickness = 0;     // свойство «толщина»
      public void Chirp()     // метод «Чирикать»
      {
            Console.WriteLine("Чирик!");
      }

      public void Peck()      // метод «Клевать»
      {
            Thickness = Thickness + 1;    // увеличение толщины
      }
};

public static void Main()
{
      Sparrow Jack = new Sparrow();       // создаем воробья с именем «Jack»
      Jack.Chirp();           // заставляем его чирикнуть
      Console.WriteLine("Толщина:")       // смотрим насколько он толст
      Console.WriteLine(Jack. Thickness);
      Jack.Peck();            // заставляем его клюнуть крошку
      Console.WriteLine(Jack. Thickness); // проверяем его толщину
// (она изменится)
}
```

Встроенные классы .NET

Вы наверняка заметили, что точку мы использовали не в первый раз. И на самом деле, оператор «точка» присутствовала уже в нашей самой первой программе.

Там, как и во многих других местах, присутствовала строка наподобие Console.WriteLine(). Так что же такое Console, а что такое — WriteLine() и почему они пишутся через точку?

Теперь, обладая познаниями о классах, объектах и методах мы можем легко ответить на этот вопрос. «Console» — это класс, а «WriteLine» – это вызов статического метода этого класса.

Откуда же взялся этот класс «Console», если мы его не описывали? «Console», так же как и множество других классов, находится во встроенной в .NET библиотеке классов. Встроенная библиотека предоставляет нам возможность при создании наших программ пользоваться уже готовыми строительными блоками и инструментами, не придумывая их каждый раз самостоятельно.

Что же тогда такое Console? Что за кирпич? Очевидно, что этот класс специально предназначен для работы с консолью и содержит в себе свойства и методы необходимые (или просто полезные) при работе с ней. Уже хорошо знакомый метод WriteLine() печатает на консоли строку текста на экране. Кроме того мы успели повстречать еще и Console.ReadLine() – если вы помните, этот метод считывает ввод пользователя с консоли.Помимо этих двух, класс «Console» содержит еще пару десятков методов, некоторые из них полезны, а некоторые не очень.

Некоторые методы и свойства класса Console:

Beep	Издает системный звук
Clear	Очищает экран консоли
Read	Читает символы с консоли
ReadKey	Читает один символ с консоли
ReadLine	Читает строку с консоли
SetCursorPosition	Устанавливает положение курсора
Write	Печатает символы на консоль
WriteLine	Печатает строку на консоль
BackgroundColor	Цвет фона консоли
CursorVisible	Устанавливает видимость или невидимость курсора
ForegroundColor	Цвет букв консоли
KeyAvaliable	Сигнализирует о нажатой клавише
Title	Устанавливает заголовок консоли

Какие еще объекты нам встречались? Наверняка, вспомнится класс «Convert», содержащий методы для преобразования одних типов переменных в другие и наоборот. Методы класса Convert:

ToBoolean	Преобразовывает в логическую переменную
ToByte	Преобразовывает в значение типа Byte
ToChar	Преобразовывает в значение типа Char
ToDateTime	Преобразовывает в значение типа дата и время
ToDecimal	Преобразовывает в точную десятичную дробь
ToInt32	Преобразовывает в целое число
ToSingle	Преобразовывает в дробное число
ToString	Преобразовывает в строку

Довольно полезным будет познакомиться с классом Math, содержащим множество математических методов. Методы класса Math:

Abs	Возвращает модуль числа

Cos	Возвращает косинус числа
Sin	Возвращает синус числа
Tan	Возвращает тангенс числа
Sqrt	Возвращает квадратный корень числа
Pow	Возводит число в степень
Floor	Возвращает целую часть числа
Sign	Возвращает квадратный корень числа
Max	Возвращает максимальное число из двух
Min	Возвращает минимальное число из двух
Round	Возвращает округленное число

Для получения случайных чисел используются методы встроенные в класс Random:

Random	Создает объект класса Random
Next	Выдает новое случайное число
NextDouble	Выдает новое случайное число в диапазоне от 0.0 до 1.0

Демонстрация работы класса Random на примере программы, кидающая кости:

```
static void Main (String[] args)
{
      Int32 a,b;
      Random random = new Random();
      a = random.Next(1, 6);
      b = random.Next(1, 6);
      Console.WriteLine("Первая кость: {0}", a);
      Console.WriteLine("Вторая кость: {0}", b);
      Console.WriteLine("Сумма: {0}", a + b);
      Console.ReadLine();
}
```

При программировании графических приложений мы используем «Windows.Forms» и некоторые другие классы. Создавая форму, мы автоматически создаем объект «Form» из класса «Windows.Forms» и работаем с этим объектом в программе.

Стоит заметить, что методы и свойства бывают статическими и динамическими. Основное различие между ними состоит в том, что к статическим можно получить доступ через имя класса (не создавая объекта), а динамические применимы только по отношению объекту и не могут использоваться без него. Например, в случае с «Console», мы используем статические методы этого класса, а написав Form1.Close(), мы вызываем нестатический метод, закрывающий главное окно программы и прекращающий ее выполнение.