Класс System.Object и его методы

Все остальные классы в .NET, даже те, которые мы сами создаем, а также базовые типы, такие как **System.Int32**, являются неявно производными от класса Object. Даже если мы не указываем класс Object в качестве базового, по умолчанию неявно класс Object все равно стоит на вершине иерархии наследования. Поэтому все типы и классы могут реализовать те методы, которые определены в классе System.Object. Рассмотрим эти методы.

ToString

Метод ToString служит для получения строкового представления данного объекта. Для базовых типов просто будет выводиться их строковое значение:

```
int i = 5;
Console.WriteLine(i.ToString()); // выведет число 5

double d = 3.5;
Console.WriteLine(d.ToString()); // выведет число 3,5
```

Для классов же этот метод выводит полное название класса с указанием пространства имен, в котором определен этот класс. И мы можем переопределить данный метод. Посмотрим на примере:

```
using System;

namespace FirstApp
{
    class Program
    {
        private static void Main(string[] args)
        {
            Person person = new Person { Name = "Tom" };
            Console.WriteLine(person.ToString());
// выведет название класса Person

            Clock clock = new Clock { Hours = 15, Minutes = 34,
Seconds = 53 };
            Console.WriteLine(clock.ToString()); // выведет 15:34:53

            Console.Read();
        }
    }
    class Clock
    {
        public int Hours { get; set; }
        public int Minutes { get; set; }
```

```
24          public int Seconds { get; set; }
25          public override string ToString()
26          {
27              return $"{Hours}:{Minutes}:{Seconds}";
28          }
29      }
30      class Person
31      {
32          public string Name { get; set; }
        }
    }
```

Для переопределения метода ToString в классе Clock, который представляет часы, используется ключевое слово **override** (как и при обычном переопределении вирутальных или абстрактных методов). В данном случае метод ToString() значение свойств Hours, Minutes, Seconds, объединенные в одну строку.

Класс Person не переопределяет метод ToString, поэтому для этого класса срабатывает стандартная реализация этого метода, которая выводит просто название класса.

Кстати в данном случае мы могли задействовать обе реализации:

```
1   class Person
2   {
3       public string Name { get; set; }
4       public override string ToString()
5       {
6           if (String.IsNullOrEmpty(Name))
7               return base.ToString();
8           return Name;
9       }
10  }
```

То есть если имя - свойство Name не имеет значения, оно представляет пустую строку, то возвращается базовая реализация - название класса. Если же имя установлено, то возвращается значение свойства Name. Для проверки строки на пустоту применяется метод String.IsNullOrEmpty().

Стоит отметить, что различные технологии на платформе .NET активно используют метод ToString для разных целей. В частности, тот же метод Console.WriteLine() по умолчанию выводит именно строковое представление объекта. Поэтому, если нам надо вывести строковое представление объекта на консоль, то при передаче объекта в метод Console.WriteLine необязательно использовать метод ToString() - он вызывается неявно:

```
1   private static void Main(string[] args)
2   {
3       Person person = new Person { Name = "Tom" };
```

```
4       Console.WriteLine(person);
5
6       Clock clock = new Clock { Hours = 15, Minutes = 34, Seconds = 53 };
7       Console.WriteLine(clock); // выведет 15:34:53
8
9       Console.Read();
10  }
```

Метод GetHashCode

Метод **GetHashCode** позволяет возвратить некоторое числовое значение, которое будет соответствовать данному объекту или его хэш-код. По данному числу, например, можно сравнивать объекты. Можно определять самые разные алгоритмы генерации подобного числа или взять реализаци базового типа:

```
1   class Person
2   {
3       public string Name { get; set; }
4
5       public override int GetHashCode()
6       {
7           return Name.GetHashCode();
8       }
9   }
```

В данном случае метод GetHashCode возвращает то хеш-код для значения свойства Name. То есть два объекта Person, которые имеют одно и то же имя, будут возвращать один и тот же хеш-код. Однако в реальности алгоритм может быть самым различным.

Получение типа объекта и метод GetType

Метод GetType позволяет получить тип данного объекта:

```
1   Person person = new Person { Name = "Tom" };
2   Console.WriteLine(person.GetType());    // Person
```

Этот метод возвращает объект **Type**, то есть тип объекта.

С помощью ключевого слова **typeof** мы получаем тип класса и сравниваем его с типом объекта. И если этот объект представляет тип Client, то выполняем определенные действия.

```
1   object person = new Person { Name = "Tom" };
2   if (person.GetType() == typeof(Person))
3       Console.WriteLine("Это реально класс Person");
```

Причем поскольку класс Object является базовым типом для всех классов, то мы можем переменной типа object присвоить объект любого типа. Однако для этого

переменной метод GetType все равно вернет тот тип, на объект которого ссылается переменная. То есть в данном случае объект типа Person.

В отличие от методов ToString, Equals, GetHashCode метод GetType не переопределяется.

Метод Equals

Метод Equals позволяет сравнить два объекта на равенство:

```
1   class Person
2   {
3       public string Name { get; set; }
4       public override bool Equals(object obj)
5       {
6           if (obj.GetType() != this.GetType()) return false;
7
8           Person person = (Person)obj;
9           return (this.Name == person.Name);
10      }
11  }
```

Метод Equals принимает в качестве параметра объект любого типа, который мы затем приводим к текущему, если они являются объектами одного класса. Затем сравниваем по именам. Если имена равны, возвращаем true, что будет говорить, что объекты равны. Однако при необходимости реализацию метода можно сделать более сложной, например, сравнивать по нескольким свойствам при их наличии.
Применение метода:

```
1   Person person1 = new Person { Name = "Tom" };
2   Person person2 = new Person { Name = "Bob" };
3   Person person3 = new Person { Name = "Tom" };
4   bool p1Ep2 = person1.Equals(person2);   // false
5   bool p1Ep3 = person1.Equals(person3);   // true
```

И если следует сравнивать два сложных объекта, как в данном случае, то лучше использовать метод Equals, а не стандартную операцию ==.