Создание клонируемых объектов (интерфейс **ICloнable**)

```
// Наш класс - это просто точка с координатами на плоскости
public class Point

// Поля (открытые переменные)
public int x, y;
// Конструкторы
public Point() {}
public Point( int x, int y)
{
this.x = x;  this.y = y;
}
// Замещаем Object.ToString()
public override string ToString()
{
return "X: " + x + " Y: " + y;
}
```

Point — это класс, а следовательно, он относится к ссылочным типам (как мы помним из обсуждения ссылочных и структурных типов в главе 2). Если мы применим к нему оператор назначения (=), то есть метод MemberwiseClone(), настоящей копии объекта создано не будет — вместо этого появится еще одна ссылка на область, занимаемую объектом в оперативной памяти. Но очень часто бывает нужно создавать настоящие, действительно отдельные копии объекта (deep copy — глубокое копирование). Для того чтобы можно было применять глубокое копирование к объектам нашего класса при помощи стандартных методов, наш класс должен реализовывать интерфейс ICloneable.

В интерфейсе ICloneable предусмотрен один-единственный метод— Clone().

Реализация этого метода, конечно же, зависит от того, какие внутренние данные определены в вашем классе. Однако смысл работы этого метода для всех классов будет одним и тем же: будет создан новый объект, и всем переменным этого объекта копии будут присвоены значения соответствующих переменных исходного объекта. Давайте научим наш объект Point клонироваться:

```
// Реализуем в классе Point поддержку глубокого копирования
// через интерфейс ICloneable
public class Point : ICloneable
{
// Данные о состоянии объекта
public int x, y;
// Конструкторы
public Point(){}
public Point(int x, int y) {this.x = x; this.y = y;}
// Реализуем единственный метод ICloneable
public object Clone()
{
        return new Point(this.x, this.y);
}
public override string ToString()
{
        return "X: " + x + " Y: " + y;
 }
```

```
Point p3 = new Point(100, 100);
Point p4 = (Po1nt) p3.Clone();
// Меняем p4.x (при этом p3.x не изменится)
p4.x = 0;

// Проверяем, так ли это:
Console.WriteLine(“Deep copying using Clone()");
Console.WriteLine(p3);
Console.WriteLine(p4);
```

Создание сравниваемых объектов (интерфейс **IComparable**)

Еще один распространенный интерфейс I C o m p a r a b l e , также определенный в пространстве имен Systern, позволяет производить сортировку объектов, основываясь на специально определенном внутреннем ключе. Формальное определение этого интерфейса выглядит следующим образом:

```
// Этот интерфейс позволяет определять место объекта среди других // аналогичных объектов
interface IComparable
{
int CompareTo(object );
}
```

```
// Создаем массив объектов Car
Car[] myAutos = new Car[5];
myAutos[0] = new Car(123, "Rusty"):
myAutos[1] = new Car(6, "Mary");
myAutos[2] = new Car(83, "Viper");
myAutos[3] = new Car(13, "NoName");
myAutos[4] = new Car(9873, "Chocky");
```

Если попробовать запустить этот код на выполнение, будет сгенерировано исключение ArgurnentException со следующим комментарием:

At least one object must implement IComparable («По крайней мере в одном объекте должен быть реализован IComparable »).

Таким образом, чтобы можно было стандартным способом производить сортировку ваших пользовательских объектов, они должны реализовывать интерфейс IComparable. Поскольку этот интерфейс состоит из единственного метода, вся соль заключается в том, как будет реализован этот метод. Видимо, наиболее важное решение, которое мы должны принять, — это определить, по значению какой внутренней переменной будет производиться сортировка. Для нашего типа Car самая подходящая переменная — это идентификатор автомобиля.

```
public class Car : IComparable
{
// Реализация IComparable
        int IComparable.CompareTo(object o)
        {
        Car temp = Car(o);
        if ( this.CarID > temp.CarID)
                return 1;
        if( this.CarID < temp.CarID)
                return -1;
        else
                return 0;
        }
}
```

Значения, возвращаемые методом CompareTo()

Любое число меньше нуля :
Значение идентификатора у текущего объекта меньше, чем у принимаемого в качестве параметра

Нуль:
Значения идентификаторов у текущего и принимаемого объекта равны

Любое число больше нуля:
Значения идентификатора у текущего объекта больше, чем у принимаемого

```
public static int Main(string[] args)
{
// Создаем нассив объектов Car
        Car[] myAutos = new Car[5];
        myAutos[0] = new Car(123, "Rusty");
        myAutos[1] = new Car(6,,"Magy"};
        myAutos[2] = new Car(83, "Viper"};
        myAutos[3] = new Car(13, "NoName"):
        myAutos[4] = new Car(9873, "Chucky");
// Выводим информацию об автомобилях из неупорядоченного массива
// на системную консоль
        Console.WriteLine(“Here is the unordered set of cars:"):
        foreach(Car c in MyAutos)
                Console.WriteLine(c.ID + “   " + c,PetName);
// А теперь используем возможности только что реализованного
// IComparable
        Array.Sort(myAutos):
// Выводим информацию уже из упорядоченного массива
        Console.WriteLine(“Here is the ordered set of cars:"):
        foreach(Car c in myAutos)
        Console.WriteLine(c.ID + "   " c.PetName);
        return 0;
}
```

КОНЕЦ...