The ICloneable Interface

As you might recall , System.Object defines a method named MemberwiseClone(). This method is used to obtain a *shallow copy* of the current object

Object users do not call this method directly, as it is protected. However, a given object may call this method itself during the *cloning* process.

To illustrate, create a new Console Application named CloneablePoint that defines a

class named Point:

// A class named Point.

public class Point

{

public int X {get; set;}

public int Y {get; set;}

public Point(int xPos, int yPos) { X = xPos; Y = yPos;}

public Point(){}

// Override Object.ToString().

public override string ToString()

{ return string.Format("X = {0}; Y = {1}", X, Y ); }

}

Given what you already know about reference types and value types, you are aware

that if you assign one reference variable to another, you have two references pointing to the same object in memory.

Thus, the following assignment operation results in two references to the same Point object on the heap; modifications using either reference affect the same object on the heap:

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Object Cloning \*\*\*\*\*\n");

// Two references to same object!

Point p1 = new Point(50, 50);

Point p2 = p1;

p2.X = 0;

Console.WriteLine(p1);

Console.WriteLine(p2);

Console.ReadLine();

}

When you want to give your custom type the ability to return an identical copy of itself to the caller, you may implement the standard ICloneable interface. this type

defines a single method named Clone():