**Returning Objects:**

A method can return any type of data, including class types. For example, the following version of the **Rect** class includes a method called **Enlarge( )** that creates a rectangle that is proportionally the same as the invoking rectangle, but larger by a specified factor:

// Return an object.

using System;

class Rect

{

int width;

int height;

public Rect(int w, int h)

{

width = w;

height = h;

}

public int Area()

{

return width \* height;

}

public void Show()

{

Console.WriteLine(width + " " + height);

}

/\* Return a rectangle that is a specified

factor larger than the invoking rectangle. \*/

public Rect Enlarge(int factor)

{

return new Rect(width \* factor, height \* factor);

}

}

class RetObj

{

public static void Main()

{

Rect r1 = new Rect(4, 5);

Console.Write("Dimensions of r1: ");

r1.Show();

Console.WriteLine("Area of r1: " + r1.Area());

Console.WriteLine();

// Create a rectangle that is twice as big as r1.

Console.Write("Dimensions of r2: ");

r2.Show();

Console.WriteLine("Area of r2: " + r2.Area());

}

}

**The output is shown here:**

Dimensions of r1: 4 5

Area of r1: 20

Dimensions of r2: 8 10

Area of r2: 80

* When an object is returned by a method, it remains in existence until there are no more references to it. At that point, it is subject to garbage collection. Thus, an object won’t be destroyed just because the method that created it terminates.
* One application of object return types is the *class factory.*
* A class factory is a method that is used to construct objects of its class. In some situations, you may not want to give users of a class access to the class’ constructor because of security concerns or because object construction depends upon certain external factors.
* In such cases, a class factory is used to construct objects. Here is a simple example:

// Use a class factory.

using System;

class MyClass

{

int a, b; // private

// Create a class factory for MyClass.

public MyClass Factory(int i, int j)

{

MyClass t = new MyClass();

t.a = i;

t.b = j;

return t; // return an object

}

public void Show()

{

Console.WriteLine("a and b: " + a + " " + b);

}

}

class MakeObjects

{

public static void Main()

{

MyClass ob = new MyClass();

int i, j;

// Generate objects using the factory.

for(i=0, j=10; i < 10; i++, j--)

{

MyClass anotherOb = ob.Factory(i, j); // make an object

anotherOb.Show();

}

Console.WriteLine();

}

}

**The output is shown here:**

a and b: 0 10

a and b: 1 9

a and b: 2 8

a and b: 3 7

a and b: 4 6

a and b: 5 5

a and b: 6 4

a and b: 7 3

a and b: 8 2

a and b: 9 1

Let’s look closely at this example.

* **MyClass** does not define a constructor, so only the default constructor is available.
* Thus, it is not possible to set the values of **a** and **b** using a constructor.
* However, the class factory **Factory( )** can create objects in which **a** and **b** are given values.
* Moreover, since **a** and **b** are private, using **Factory( )** is the only way to set

these values.

* In **Main( )**, a **MyClass** object is instantiated, and its factory method is used inside the **for** loop to create ten other objects. The line of code that creates objects is shown here:

MyClass anotherOb = ob.Factory(i, j); // get an object

* With each iteration, an object reference called **anotherOb** is created, and it is assigned a reference to the object constructed by the factory.
* At the end of each iteration of the loop, **anotherOb** goes out of scope, and the object to which it refers is recycled.