Exercise 4 – Easy Initialization with Object and Collection Initializers

In C# 3.0, when declaring an object or collection, you may include an initializer that specifies the initial values of the members of the newly created object or collection. This new syntax combines object creation and initialization into a single step.

Task 1 – Using Object Initializers

An object initializer consists of a sequence of member initializers, contained in { } braces and separated by commas. Each member initializer assigns a value to a field or property of the object. For example, recall the Point class from Exercise 1:

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }
}
```

An object of type Point can be initialized using an object initializer like this:

```
Point p = new Point { X = 3, Y = 99 };
```

This concise initialization syntax is semantically equivalent to invoking the instance constructor and then performing assignments for each of the properties.

1. Replace the code in **Main** with the following:

```
static void Main(string[] args)
{
    Customer c = new Customer(1) { Name = "Maria Anders", City = "Berlin" };
    Console.WriteLine(c);
}
```

2. Press **Ctrl+F5** to build and run the application. The customer is printed to the console window. Now press any key to terminate the application.

For this object, the object initializer invokes the object's constructor, and then initializes the named fields to the specified values. Not all fields of the object need to be specified in the list. If not specified, the fields will have their default value. Also note if invoking the object's parameterless constructor, the parentheses are optional. For example you could have written `Customer c = new Customer() {…} as Customer c = new Customer {…}`.

Task 2 – Using Collection Initializers

With C# 3.0, any object that implements `System.Collections.Generic.IEnumerable<T>` and has a public Add method can have its values initialized with a collection initializer.

Using the **Point** class, let's create a shape that is made up of a collection of points.

```
List<Point> Square = new List<Point>
{
    new Point { X=0, Y=5 },
```

```
        new Point { X=5, Y=5 },
        new Point { X=5, Y=0 },
        new Point { X=0, Y=0 }
    };
```

3. For the remainder of the lab a collection of Customers will be needed. Create a list of Customers in a new method **CreateCustomers**.

```
static void Main(string[] args)
{
    Customer c = new Customer(1) { Name = "Maria Anders", City = "Berlin" };
    Console.WriteLine(c);
}
static List<Customer> CreateCustomers()
{
    return new List<Customer>
    {
        new Customer(1) { Name = "Maria Anders",      City = "Berlin"    },
        new Customer(2) { Name = "Laurence Lebihan", City = "Marseille" },
        new Customer(3) { Name = "Elizabeth Brown",  City = "London"    },
        new Customer(4) { Name = "Ann Devon",        City = "London"    },
        new Customer(5) { Name = "Paolo Accorti",    City = "Torino"    },
        new Customer(6) { Name = "Fran Wilson",      City = "Portland"  },
        new Customer(7) { Name = "Simon Crowther",   City = "London"    },
        new Customer(8) { Name = "Liz Nixon",        City = "Portland"  }
    };
}
```

4. In **Main,** replace the code with the following:

```
static void Main(string[] args)
{
    List<Customer> customers = CreateCustomers();

    Console.WriteLine("Customers:\n");
    foreach (Customer c in customers)
        Console.WriteLine(c);
}
```

5. Press **Ctrl+F5** to build and run the application and print the list of customers. Press any key to terminate the application.

When calling CreateCustomers, the list is initialized empty and each object is created separately. Each object is then added to the list using the Add method.