Exercise 3 – Extending Types with Extension Methods

Extension methods provide a way for developers to extend the functionality of existing types by defining new methods that are invoked using the normal instance method syntax. Extension methods are static methods that are declared by specifying the keyword `this` as a modifier on the first parameter of the methods. In this exercise, imagine the Order class was precompiled and a library was provided. New features can still be built on top of the Order class. To do this a new class is created to hold these extension methods. In addition, you will extend the `List<T>` generic class, adding an extension method to append the list to another.

Task 1 – Declaring Extension Methods

This task begins by extending a class using C# 2.0 and then utilizes the C# 3.0 Extension Methods feature.

1. Add a new static class, **Extensions**, to the **NewLanguageFeatures** namespace:

```
namespace NewLanguageFeatures
{
    public static class Extensions
    {
    }
    public class Customer
    …
```

2. In this new class add a method, **Compare** that given two customers checks to see if all the properties of each are the same, and if so returns true.

```
public static class Extensions
{
    public static bool Compare(Customer customer1, Customer customer2)
    {
        if (customer1.CustomerID == customer2.CustomerID &&
            customer1.Name == customer2.Name &&
            customer1.City == customer2.City )
        {
            return true;
        }
        return false;
    }
}
```

3. Rewrite **Main** to compare a new Customer with all the others in the list to see if it is present already:

```
static void Main(string[] args)
{
    var customers = CreateCustomers();
    var newCustomer = new Customer(10)
    {
        Name = "Diego Roel",
        City = "Madrid"
    };

    foreach (var c in customers)
```

```
    {
        if (Extensions.Compare(newCustomer, c))
        {
            Console.WriteLine("The new customer was already in the list");
            return;
        }
    }

    Console.WriteLine("The new customer was not in the list");
}
```

4. Press **Ctrl+F5** to build and run the application, which displays “The new customer was not in the list”. Press any key to terminate the application.

With C# 3.0, you can now define an extension method that can be invoked using instance method syntax. An extension method is declared by specifying the keyword `this` as a modifier on the first parameter of the method.

5. Add the modifier `this` to the first parameter accepted by **Compare**:

```
public static class Extensions
{
    public static bool Compare(this Customer customer1, Customer customer2)
    {
        …
```

6. In the **Main** method, change the invocation of **Compare** to use the instance method syntax, making **Compare** appear as a method of the Order class:

```
foreach (var c in customers)
{
    if (newCustomer.Compare(c))
    {
        Console.WriteLine("The new customer was already in the list");
        return;
    }
    …
```

7. Press **Ctrl+F5** to build and run the application again and verify that it displays the same output, then press any key to close the console window and terminate the application.

Extension methods are only available if declared in a static class and are scoped by the associated namespace. They then appear as additional methods on the types that are given by their first parameter.

Task 2 – Using Extension Methods with Generic Types

Extension methods can be added to any type, including the generic types such as List<T> and Dictionary<K, V>.

8. Add an extension method, **Append**, to the **Extensions** class that combines all elements of two List<T> objects into a single List<T>:

```
public static class Extensions
{
    public static List<T> Append<T>(this List<T> a, List<T> b)
    {
        var newList = new List<T>(a);
        newList.AddRange(b);
        return newList;
    }
    public static bool Compare(this Customer customer1, Customer customer2)
    {
        …
```

9. return to the **Main** method and use the **Append** method to append the **addedCustomers** list to the **customers** list. Check this new list to see if the **newCustomer** is in the updated list:

```
static void Main(string[] args)
{
    var customers = CreateCustomers();
    var addedCustomers = new List<Customer>
    {
        new Customer(9)  { Name = "Paolo Accorti", City = "Torino" },
        new Customer(10) { Name = "Diego Roel", City = "Madrid" }
    };
    var updatedCustomers = customers.Append(addedCustomers);
    var newCustomer = new Customer(10)
    {
        Name = "Diego Roel",
        City = "Madrid"
    };
    foreach (var c in updatedCustomers)
    {
        …
```

10. Press **Ctrl+F5** to build and run the application, which now displays “The new customer was already in the list”. Press any key to terminate the application.

Extension methods provide an elegant way to extend types with functionality you develop, making the extensions appear to be part of the original types. Extension methods enable new functionality to be added to an already compiled class. This includes user created classes as well as .NET classes such as `List<T>`.