Lesson 31 - Generics

Generics

Generics allow the reuse of code across different types. For example, let's declare a method that swaps the values of its two parameters:

```
static void Swap(ref int a, ref int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Our `Swap()` method will work only for integer parameters. If we want to use it for other types, for example, doubles or strings, we have to overload it for all the types we want to use it with. Besides a lot of code repetition, it becomes harder to manage the code because changes in one method mean changes to all of the overloaded methods. Generics provide a flexible mechanism to define a generic type.

```
static void Swap<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

In the code above, `T` is the name of our **generic** type. We can name it anything we want, but `T` is a commonly used name. Our `Swap()` method now takes two parameters of type `T`. We also use the `T` type for our temp variable that is used to swap the values.

> Note the brackets in the syntax `<T>`, which are used to define a generic type.

We can use something called a tuple to shorten our method:

```
static void Swap<T>(ref T a, ref T b)
    => (b, a) = (a, b);
```

Now we can use our swap method to swap variables of different types:

```
int a = 1;
int b = 15;

Console.WriteLine($"Before Swap - A:{a}, B:{b}");
Swap<int>(ref a, ref b);
Console.WriteLine($"After Swap - A:{a}, B:{b}");

string x = "Talha";
string y = "Salman";

Console.WriteLine($"Before Swap - X:{x}, Y:{y}");
Swap<string>(ref x, ref y);
Console.WriteLine($"After Swap - X:{x}, Y:{y}");
```

When calling a generic method, we need to specify the type it will work with by using brackets. So, when `Swap<int>()` is called, the `T` type is replaced by `int`. For `Swap<string>()`, `T` is replaced by `string`.
If you omit specifying the type when calling a generic method, the compiler will use the type based on the arguments passed to the method as shown below:

```
int a = 1;
int b = 15;

Console.WriteLine($"Before Swap - A:{a}, B:{b}");
Swap(ref a, ref b);
Console.WriteLine($"After Swap - A:{a}, B:{b}");

string x = "Talha";
string y = "Salman";

Console.WriteLine($"Before Swap - X:{x}, Y:{y}");
Swap(ref x, ref y);
Console.WriteLine($"After Swap - X:{x}, Y:{y}");
```

> Multiple generic parameters can be used with a single method. For example: `Func<T, U>` takes two different generic types.

```
Utils.PrintKeyValuePair("Name", "Talha");
Utils.PrintKeyValuePair("Age", 22);
Utils.PrintKeyValuePair(1, "Family");
Utils.PrintKeyValuePair(2, "Work");
Utils.PrintKeyValuePair(3, "Gaming");

class Utils
{
    public static void PrintKeyValuePair<TKey, TValue>(TKey key, TValue value)
```

```
        => Console.WriteLine($"{key}: {value}");
}
```