Lesson 32 - Generic Classes

Generic Classes

Generic types can also be used with classes. The most common use for generic classes is with collections of items, where operations such as adding and removing items from the collection are performed in basically the same way regardless of the type of data being stored.

Stack

One type of collection is called a `stack`. Items are "pushed", or added to the collection, and "popped", or removed from the collection. A stack is sometimes called a **Last In First Out** (`LIFO`) data structure.

```
class Stack<T>
{
    int index = 0;
    T[] innerArray = new T[100];

    public void Push(T item) => innerArray[index++] = item;

    public T Pop() => innerArray[--index];

    public T Get(int k) => innerArray[k];
}
```

The generic class stores elements in an array. As you can see, the generic type `T` is used as the type of the array, the parameter type for `Push()`, and the return type for `Pop()` and `Get()`.
Now we can create objects of our generic class, we can also use the generic class with custom types,.

> In a generic class we do not need to define the generic type for its methods, because the generic type is already defined on the class level.

```
Stack<int> intStack = new();
Stack<string> strStack = new();
```

```
intStack.Push(1);
intStack.Push(20);
intStack.Push(400);

Console.WriteLine(intStack.Get(0));

for (int i=0; i<3; i++)
    Console.WriteLine(intStack.Pop());
```

```
Stack<int> intStack = new();
Stack<string> stringStack = new();

intStack.Push(1);
intStack.Push(20);
intStack.Push(300);
intStack.Push(4000);

stringStack.Push("Talha");
stringStack.Push("Amad");
stringStack.Push("Mubashir");
stringStack.Push("John");

stringStack.PopAll();

class Stack<T>
{
    int index = 0;
    T[] innerArray = new T[100];

    public int Count { get { return index; } }
    public void Push(T item) => innerArray[index++] = item;
    public T Pop() => innerArray[--index];
    public T Get(int k) => innerArray[k];

    public void PopAll()
    {
        int count = Count;
        for (int i = 0; i < count; i++)
            Console.WriteLine(Pop());
    }
}
```

```
class DynamicArray<T>
{
    T[] Value = Array.Empty<T>();

    public int Length { get { return Value.Length; } }
    public void Add(T item)
    {
        T[] newArray = new T[Length + 1];
        for (int i = 0; i < Length; i++)
            newArray[i] = Value[i];

        newArray[Length] = item;
        Value = newArray;
    }
}
```