Lesson 13 - Collections I

Arrays

C# has many more complex data types built-in that help us to store and manipulate data. One such example is the `Array` type. An `Array` stores collections of data of the **same type**. An example, let us suppose we want to store 5 number, without using arrays, you would something like this,

```
int num1 = 5;
int num2 = 10;
int num3 = 15;
int num4 = 20;
int num4 = 25;
```

Declaration

But by using arrays, we can make this much easier. Given below is the syntax on how to decalre an array of integers,

```
int[] myArray;
```

Instantiation

Given below is the syntax on how to instantiate it. The part to the right of the `=` sign indicates the instantiation and there we define the size of the `Array`. So the below code instantiates an `Array` of type `int` which can store `5` integers.

```
int[] myArray = new int[5];
```

Assignment and Indexing

We can assign values to individual elements in an array using the `index` of the element. The `index` is an position or id of the value, and it always starts from `0`. Below we assign a value to the first and last items in the `Array`.

```
int[] myArray = new int[5];  // indexes: 0, 1, 2, 3, 4
myArray[0] = 10;
myArray[4] = 100;
```

Initialization

We can provide intial values to an `Array` during declaration by using *curly* brackets, with each element separated by a *comma*.

```
int[] myArray = new int[5] { 1, 2, 3, 4, 5 };
```

We can omit the size declaration because the size can be inferred from the provided collection automatically.

```
int[] myArray = new int[] { 1, 2, 3, 4, 5 };
```

We can even omit the `new` operator during initialization,

```
int[] myArray = { 1, 2, 3, 4, 5 };
```

Examples

```
int[] intArray = { 1, 2, 3 };
string[] stringArray = { "one", "two", "three" };
bool[] boolArray = { true, false, true };
double[] doubleArray = { 1.1, 1.2, 2.1 };
```

Using arrays in loops

```
int numOfStudentsInClass = 10;
int[] classGrades = new int[numOfStudentsInClass];

Console.WriteLine("Please enter student grades:");

int i = 0;
while (i < numOfStudentsInClass)
{
        Console.Write($"Grade of student #{i + 1}: ");
        classGrades[i++] = Convert.ToInt32(Console.ReadLine());
}

i = 0;
while (i < numOfStudentsInClass)
```

```
{
        Console.WriteLine(classGrades[i++]);
}
```

Array Properties & Methods

You can access the properties and method of an `Array` and many other datatypes using the `dot` operator. *Properties* are special values stored in complex types, and *Methods* are special functionality we perform using those types. Methods have round brackets `()` and that is how we can differentiate between Properties and Methods.

```
int[] numbers = { 1, 2, 3, 4 };

int i = 0;
while (i < numbers.Length)  // length: 4, indexes: 0, 1, 2, 3
{
        Console.WriteLine(numbers[i++]);
}

Console.WriteLine($"Sum: {numbers.Sum()}");
```