C# 6 COLLECTIONS AND GENERICS

COLLECTIONS AND GENERICS INTRODUCTION

COLLECTIONS AND GENERICS OVERVIEW

This presentation covers:

- Collection Types
- Arrays
- Generic Lists
- Generic Dictionaries
- Generic Collection Interfaces

TYPES OF COLLECTIONS

- Lists: allows the management of a group of typed entities like strings or integers.
  1. E.g., list of cars: "Ford", "Lincoln", "Jeep", "Chrysler", "Chevrolet".
  2. Accessed by element index beginning with 0 (zero based): e.g., element 1 is "Lincoln"
- Dictionaries: allows the management of a group of name-value pairs of entities.
  1. E.g., "CA" | "California", "TN" | "Tennessee", "GA" | "Georgia"
  2. Accessed by key: e.g., "TN" accesses "Tennessee"

ARRAYS

- A fixed size of elements that can be accessed by positional index number

0	Ford
1	Lincoln
2	Chevrolet

- OR:

0	Ford	Fusion
1	Lincoln	MKZ
2	Chevrolet	Camero

DECLARING AND INITIALIZING AN ARRAY

- Array type followed by open closed brackets and then the variable

string[] myArray;

- Could be int[], datetime[], or even a Product[] (array of objects)
- Array is a reference type that initialized by "new"

myArray = new string[3]; or string[] myArray = new string[3] or var myArray = new string[3])

- Again specify type "string' and size [3] indexed as 0, 1, 2
- If array type is string, all elements are initially null; int[] would be 0; etc.

POPULATING A SINGLE DIMENSIONAL ARRAY

- Populating an array is by variable[index] for single dimensional array:

myArray[0] = “Ford”

myArray[1] = “Lincoln”

myArray[2] = “Chevrolet”

POPULATING A MULTIDIMENSIONAL ARRAY

- Declaring a 2-dimensional array:

```
var myVar = new int[2, 2];
```

- Populating first element:

```
myVar[0, 0] = 1;

myVar[0, 1] = 10;
```

INITIALIZING ARRAYS

- C# provides simple and straightforward ways to initialize arrays at declaration time by enclosing the initial values in curly braces ({}).

```
string[] cars = new string[3] {“Ford",“Lincoln",“Chevrolet"};

int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
```

USING ARRAYS

- Use arrays when the size of the array is known at design time
- Don't use arrays when populating from a database table when the rows can increase or decrease
- Use arrays when more than one dimension is needed (size known)
- Use array built-in methods: Array.IndexOf(myArray, "Ford"); this is a static method example

myArray.SetValue("Ford", 0); this is an instance method of the myArray variable

GENERICS

- Generics allow writing code without specifying datatypes yet they are type safe validated by the compiler when compiling the program.
- For example, if you want to return different results from a specific class method that would include Boolean, integers, or string results, you could have 3 classes with methods that return 3 different results. (see non-generic classes).
- Or, you could create a generic class with a method that accepts a generic input parameter which doesn't violate the DRY (Don't Repeat Yourself) programming best practice. (See generic class)