Day 4

Tuples

- Tuples are lightweight data structures that contain multiple fields.
- To create a tuple, simply enclose the values to be assigned to the tuple in parentheses

  Eg:- ("a", 5, "c")
- Elements in the tuple can be of same/different data types

Tuples

- Tuple can be assigned to a variable

  eg

  (string, int, string) values = ("a", 5, "c");

  OR

  var values = ("a", 5, "c");

- Now individual elements from a tuple can be accessed as

  values.item1, values.item2 and so on

Tuples

- Explicit naming can also be given to elements of tuple

  eg

  (string name, int roll) values1 = ("abc", 5);

  Or

  var values2 = (name:"abc",roll: 5);

- Now individual elements from a tuple can be accessed as

  values.name, values.roll

Boxing

- to store the data in a value type data within a reference variable is termed boxing

Eg:-

```
int myInt = 25;
// Box the int into an object reference.
object boxedInt = myInt;
```

-

Un Boxing

- Unboxing is the process of converting the value held in the object reference back into a corresponding value type on the stack.

Eg:

```
int myInt = 25;
// Box the int into an object reference.
object boxedInt = myInt;
// Unbox the reference back into a corresponding int.
int unboxedInt = (int)boxedInt;
```

Arrays

Arrays

- An array is a set of contiguous data elememts of the same type.
- Example for creating an array

```
int[ ] empid= new int[ 3 ];
//creates an integer array of 3 elemets
```

Each element of the array is identified by a number known as index. Index of first element of the array will be 0

Arrays

- Array can be initialized along with declaration.
  string [ ] empname= new string[] { “ABC", “PQR“};
- Declaration and initialization could be done this way also

  string [ ] empname= new string[2] { “ABC", “PQR“};
- Or

  string [ ] empname= { “ABC", “PQR“};
- You can create array of reference type or array of value type elements

Multidimensional Array

- Array having multiple dimensions(rows and columns)
- Example creation of multi dimensional array
- int[,] marks= new int[3,4];

jagged arrays

- Is a special 2dimensional array where number of columns in different rows may vary
- Example creating jagged array

```
int[ ][ ] jagged= new int[5][ ];
//jagged array having 5 rows
```

- Number of columns in each row is to be defined separately

Eg:- jagged[0] = new int [3];

Example accessing the array elements jagged [i][j]

System.Array Class

- Is the base class for all arrays
- Provides methods for creating, manipulating, searching, and sorting arrays,

Member of Array Class	Description
Clear()	static method sets a range of elements in the array to empty values (0 for numbers, null for object references, false for Booleans).
Reverse()	static method Can be used to reverses the contents of a one-dimensional array.
Length	This property returns the number of items within the array.
Rank	This property returns the number of dimensions of the current array.

Member of Array Class	Description
Sort()	static method sorts a one-dimensional array of intrinsic types. If the elements in the array implement the IComparer interface, you can also sort your custom types

Indices and Ranges
index from end operator (^)
The range operator (..)

Two new datatypes Index and Range
2 operators ^ and ..
Were introduced from C# v 8 to for doing array related operations

Index Data Type

- Index data type can act as index of an array

index from end operator (^)

- ^ takes a number as operand , that indicates position of the element from end
- For example
- If you want to get last element of array give position ^1

Range Operator ..

- The range operator specifies a start and end index and allows for access to a subsequence within a list.
- The start of the range is inclusive, and the end of the range is exclusive.
- Eg:

```
foreach (var itm in empname[0..3])
{
    // Prints array elements from 0 to 2
    Console.Write(itm + ", ");
}
```

Range Operator ..

- If the beginning of the range is left off, the beginning of the sequence is used.
- If the end of the range is left off, the length of the range is used.
- This does not cause an error,

Range Data Type

- We can use range data type to store range of values
- eg

```
Range r = 0..3; //the end of the range is exclusive
foreach (var itm in empname[r])
{
// Prints array elements from 0 to 2
Console.Write(itm + ", ");
}
```

Collection Classes

- In programs when you want to store group of data items , one choice is t.o use Arrays.
- But Array has a limitation that it cannot grow/shrink dynamically.
- So to better handle group of items comes collection classes.
- A collection class can come in 2 category
  - Non-generic collections and generic collections

System.Collections Namespace

- Non generic collection classes will come under System.Collections namespace.

System.Collections Class	Description	Implemented Interfaces
ArrayList	dynamically sized collection of objects listed in sequential order	IList, ICollection, IEnumerable, and ICloneable
BitArray	a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0)	ICollection, IEnumerable, and ICloneable
Hashtable	a collection of key-value pairs that are organized based on the hash code of the key	IDictionary, ICollection, IEnumerable, and ICloneable

System.Collections Class	Description	Implemented Interfaces
Queue	a standard first-in, first-out (FIFO) collection of objects	ICollection, IEnumerable, and ICloneable
SortedList	collection of key-value pairs that are sorted by the keys and are accessible by key and by index	IDictionary, ICollection, IEnumerable, and ICloneable
Stack	A last-in, first-out (LIFO) stack providing push and pop (and peek) functionality	ICollection, IEnumerable, and ICloneable

- Functionality of each of these collection depends on the interfaces it implements

System.Collections Interface	
ICollection	Defines general characteristics like, size, enumeration, and thread safety) for all nongeneric collection types
IEnumerable	Returns an object implementing the IEnumerator interface
IEnumerator	Enables collection to be accessible using foreach loop.
ICloneable	Allows the implementing object to return a copy of itself to the caller

System.Collections Interface	
IDictionary	Allows a collection to represent its contents using key-value pairs
IList	Provides behavior to add, remove, and index items in a sequential manner

Generic Classes and Methods

Generic Classes and Methods

- classes and methods not made for a specific type but can be used with any general type.
- When you declare a generic class or method you will be using a pair of angled brackets with a letter or other token sandwiched within.
- you call these tokens type parameters or placeholders.
- Eg: IEnumerable<T> read as “IEnumerable of T” or, to say it another way, “IEnumerable of type T.”

Generic Classes and Methods

- When you create an instance of a generic class you must specify the type parameter.

Constraining Type Parameters

- While creating a generic class you can specify a set of constrains on type T using a where clause
- where T : class ==T must be reference type
- where T : new() == type parameter <T> must have a default constructor.
- where T : NameOfBaseClass ==Type T must be derived from the base calss
- where T : NameOfInterface ==Type T must implement the interface

Generic Collection

- Non generic collection has 2 disadvantages
  - Not type safe
  - Less performance because of the need for unboxing
- Not type safe means they can store values of any data type as non generic collections contain elements of Sytem.Object type.
- Unboxing & Boxing

System.Collections.Generic Class	Description	Implemented Interfaces
Dictionary<TKey,TValuet>	Collection of Key value Pairs	ICollection<T>, IDictionary<TKey, TValue> IEnumerable<T>
List<T>	Sequential list of items	ICollection<T>, IEnumerable<T>, IList<T>