Data Structure

Loy Vanich (laploy@gmail.com 084 007 5544)

What is?

- Data organization
- Management
- Storage
- Enables efficient access and modification
- Collection of data values

C# Data Structure

- **Tuple**: lightweight syntax type
- **Dictionary**: Store items as key/value
- **List**: smart array, access items by index
- **Queue**: First-in-First-Out (FIFO)
- **Stack**: Last-in-First-Out (LIFO)

Tuple

- Types define using a lightweight syntax
- Simpler syntax
- Conversions based on cardinality
- Consistent rules for
    - Copies
    - Equality tests
    - Assignments
- Do not support inheritance

Unnamed Tuples

```
var unnamed = ("one", "two");
```

Named tuples

```
var named = (first: "one", second: "two");
```

Field names for a tuple may be provided from the variables used to initialize the tuple

```
var sum = 12.5;
var count = 5;
var accumulation = (count, sum);
```

Dictionary<TKey, TValue>

- Generic class provides a mapping from a set of keys to a set of values.
- Each addition to the dictionary consists of a value and its associated key.
- Retrieving a value by using its key is very fast

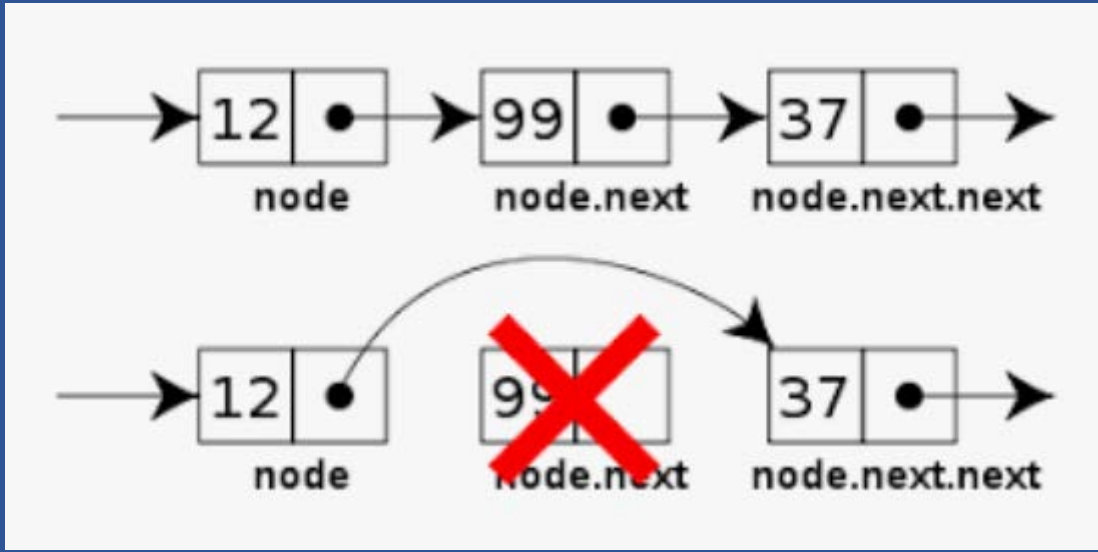
Create and add element

```
// Create a new dictionary of strings, with string keys.
//
Dictionary<string, string> openWith =
    new Dictionary<string, string>();

// Add some elements to the dictionary. There are no
// duplicate keys, but some of the values are duplicates.
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");
```

Generic List<T>

- Defined in the System.Collections.Generic
- Add, insert, remove, search etc.
- Replacement for arrays
- Grow in size on-demand.
- Accessed by index



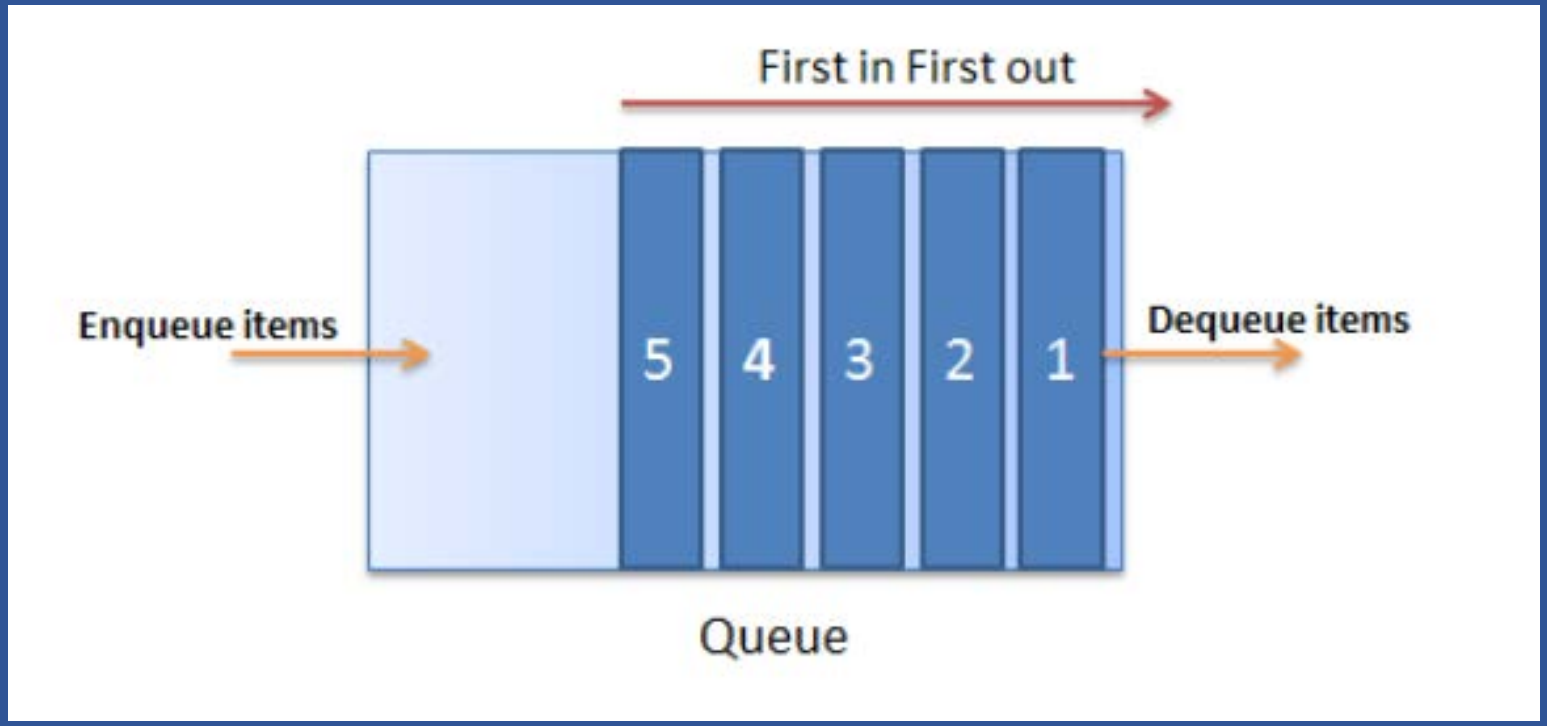

```
3  List<int> intList = new List<int>();
4
5  //Or
6
7  IList<int> intList = new List<int>();
```

User List<T> for internal class member

User IList<T> for exposing through a library

Queue<T>

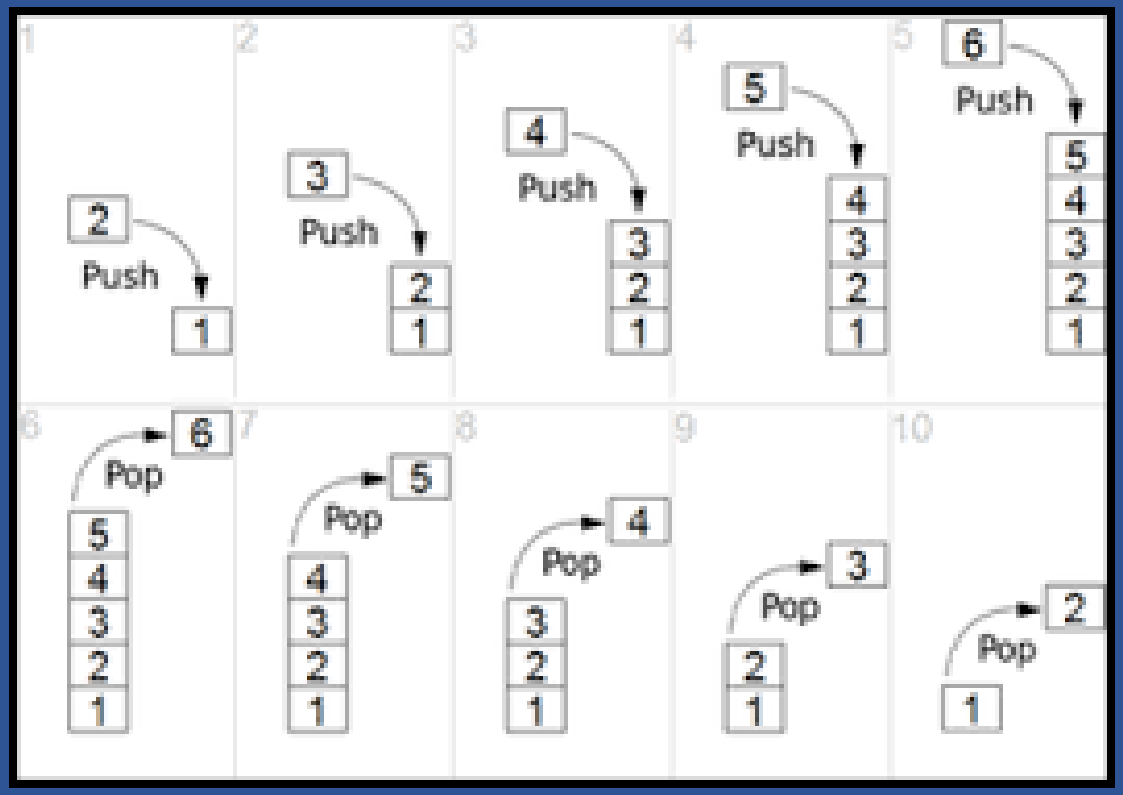
- Circular array FIFO
- Inserted at one end
- Temporary storage
- Discard an element after retrieving its value
- **Enqueue** adds an element
- **Dequeue** removes the oldest element
- **Peek** returns the oldest element




```
3   Queue queue = new Queue();
4   queue.Enqueue(3);
```

Stack<T>

- Temporary storage LIFO
- Discard an element after retrieving its value
- Three main operations
  - Push inserts an element at the top of the Stack.
  - Pop removes an element from the top of the Stack<T>
  - Peek returns an element that is at the top of the Stack<T> but does not remove it from the Stack<T>

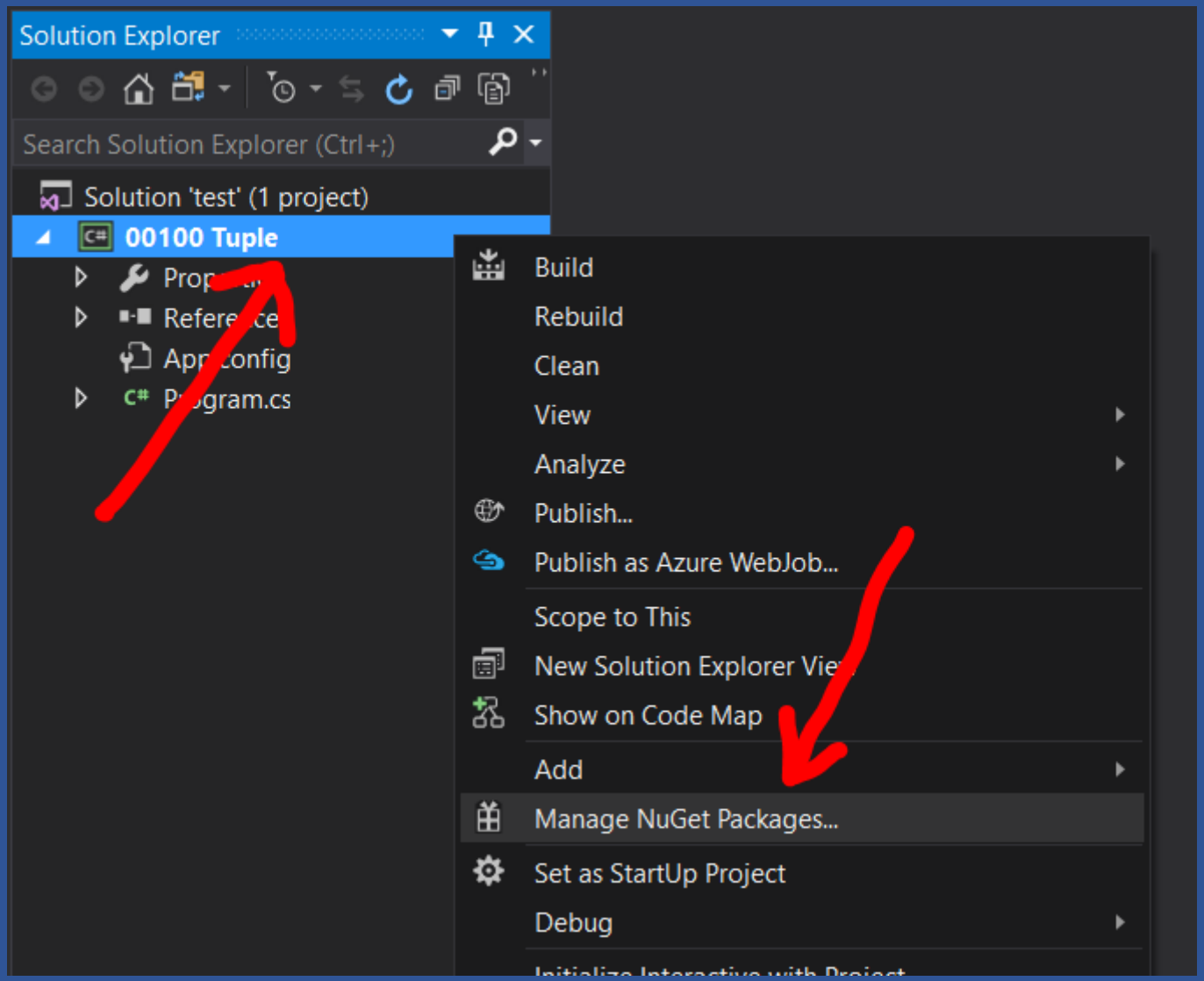



```
2
3  Stack<int> myStack = new Stack<int>();
4  myStack.Push(100);
5
```

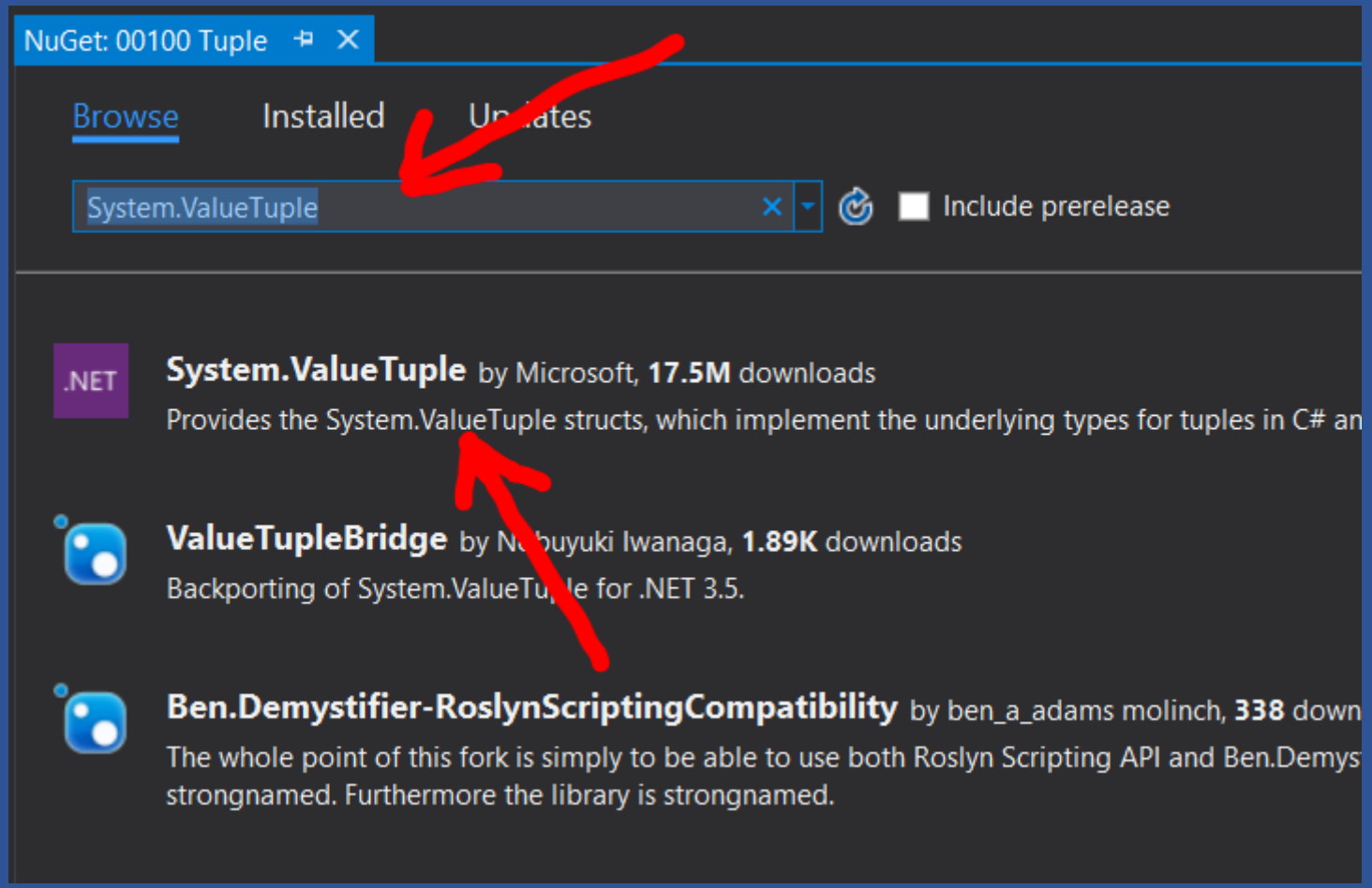
Example 00100: Tuple

- Create New Consol App Project “Test”
- Create New Project Consol App “00100 Tuple”

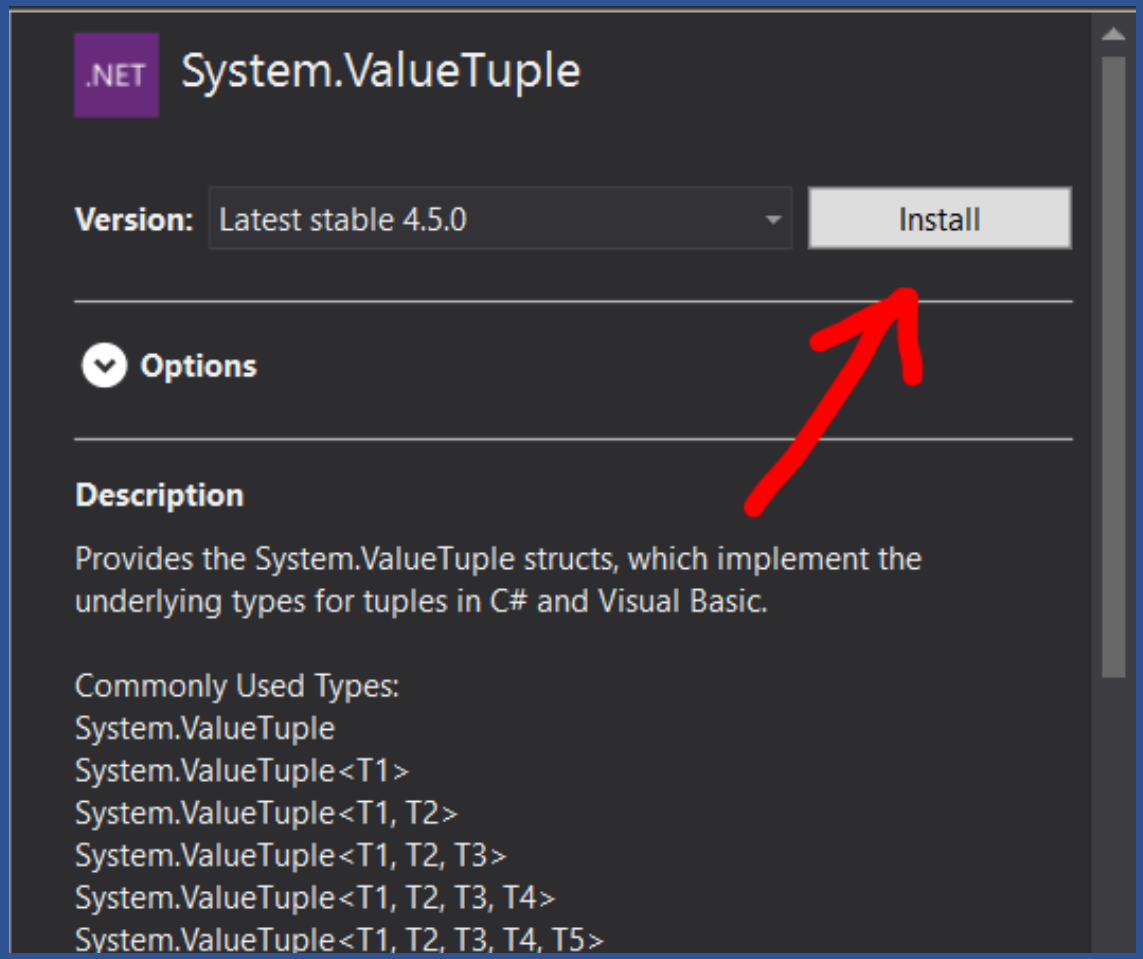
Right-Click project's Name / Manage Nuget packages...



System.ValueTuple



Press Install



Wait until Install complete

```
Installing System.ValueTuple 4.5.0.
Adding package 'System.ValueTuple.4.5.0' to folder 'D:\Temp\test\packages'
Added package 'System.ValueTuple.4.5.0' to folder 'D:\Temp\test\packages'
Added package 'System.ValueTuple.4.5.0' to 'packages.config'
Successfully installed 'System.ValueTuple 4.5.0' to 00100 Tuple
Executing nuget actions took 2.17 sec
Time Elapsed: 00:00:03.0791319
========== Finished ==========
```

Add Code to program.cs

Example 00110: Dictionary

```
Dictionary<string, string> openWith =
    new Dictionary<string, string>();

// Add some elements to the dictionary. There are no
// duplicate keys, but some of the values are duplicates.
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");
```

Example 0120 : List

demonstrates how to add, remove, and insert a simple business object in a List<T>

```
public class Part : IEquatable<Part>
{
    public string PartName { get; set; }
    public int PartId { get; set; }

    public override string ToString()...
    public override bool Equals(object obj)...
    public override int GetHashCode()...
    public bool Equals(Part other)...
    // Should also override == and != operators.

}
public class Example...
```

Example 0130 : List

demonstrates several properties and methods of the List<T>

```
List<string> dinosaurs = new List<string>();

Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);

dinosaurs.Add("Tyrannosaurus");
dinosaurs.Add("Amargasaurus");
dinosaurs.Add("Mamenchisaurus");
dinosaurs.Add("Deinonychus");
dinosaurs.Add("Compsognathus");
Console.WriteLine();
foreach (string dinosaur in dinosaurs)
{
    Console.WriteLine(dinosaur);
}
```

Example 0138 : Queue

Demo basic queue

```
Queue<string> myQueue = new Queue<string>();
myQueue.Enqueue("Quick");
myQueue.Enqueue("Brow");
myQueue.Enqueue("Fox");
myQueue.Enqueue("Jump");
myQueue.Enqueue("Over");
foreach (var q in myQueue)
    Console.Write(q + " ");
Console.WriteLine();
while(myQueue.Count > 0)
{
    myQueue.Dequeue();
    foreach (var q in myQueue)
        Console.Write(q + " ");
    Console.WriteLine();
}
```

Example 0140 : Queue

demonstrates several methods of the Queue<T> generic class

```
Queue<string> numbers = new Queue<string>();
numbers.Enqueue("one");
numbers.Enqueue("two");
numbers.Enqueue("three");
numbers.Enqueue("four");
numbers.Enqueue("five");

// A queue can be enumerated without disturbing its contents.
foreach (string number in numbers)
{
    Console.WriteLine(number);
}
```

Example 0150 : Stack

demonstrates several methods of the Stack<T> generic class

```
Stack<string> numbers = new Stack<string>();
numbers.Push("one");
numbers.Push("two");
numbers.Push("three");
numbers.Push("four");
numbers.Push("five");

// A stack can be enumerated without disturbing its contents.
foreach (string number in numbers)
{
    Console.WriteLine(number);
}
```