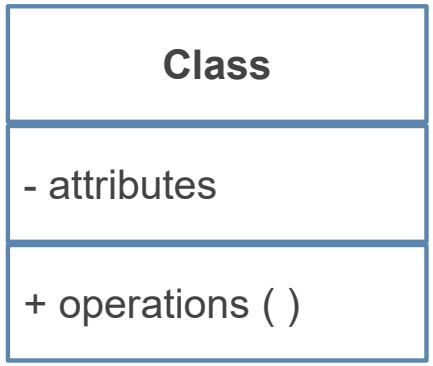
UML
Unified Modeling Language

UML

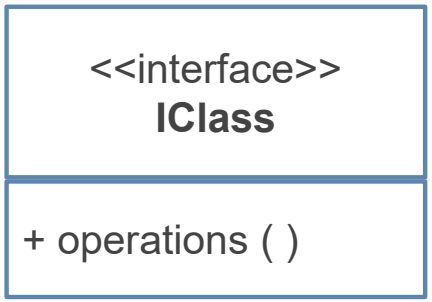
- A universally accepted way of describing software in diagrammatic form

Class

- Access modifiers
  - + (public)
  - (private)
  - # (protected)

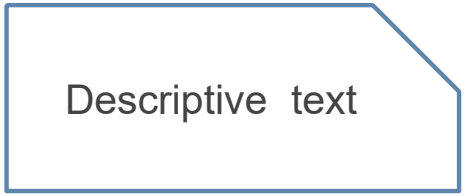


Interface or abstract Classes

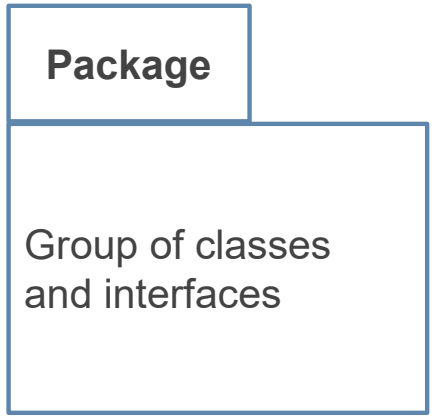


Note

- Description when needed



Package



Inheritance

- B inherits from A

Realization

- B implements A

Association

- A and B call and access each Other elements

Association (one way)

- A Can Call and Access B elements but not vise versa
- Example
  - Driver (A) Car (B)

Aggregation

- A has a B , and B can Outlive A

Composition

- A has a B, B depends on A

Exception Handling

Exception

- An exception is a problem that arises during the execution of a program (runtime error)
- This error would ***throw an Exception***
- If no mechanism provided then the program would report ***unhandled exception***
- ***Catching an Exception*** is the process for handle this error

Exception handling

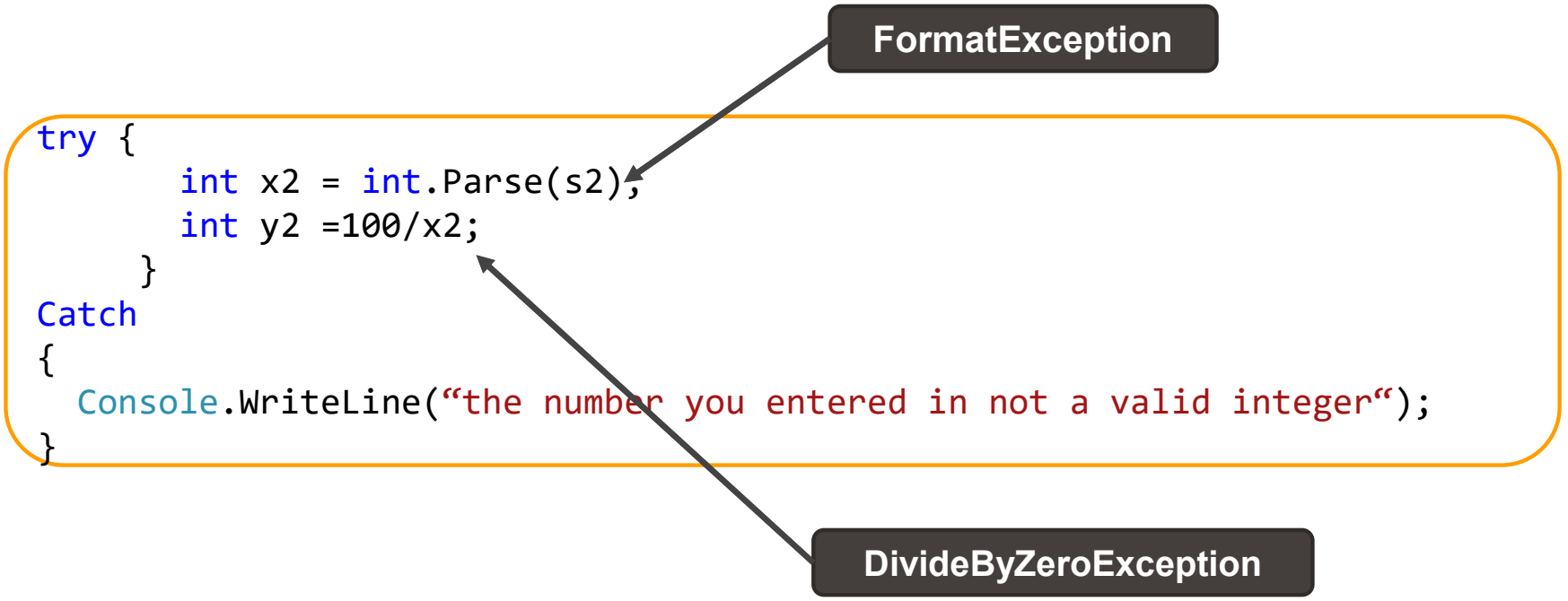
- Exception handling could be achieved using *try… catch* block

```
string s2 = Console.ReadLine(); // user enter jk
int x2 = int.Parse(s2);
```

FormatException

```
try {
        int x2 = int.Parse(s2);
     }
Catch // (Exception e)
{
  Console.WriteLine("the number you entered in not a valid integer");
}
```

Exception handling



Exception handling

- Exception class
  - Message : contain description of the error
  - StackTrace: where the error occurred.
  - InnerException: in case of multi exception this property used to get the first one
  - (demo)

Exception Types

- All exceptions are subclasses of **Exception** superclass

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from referencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stackoverflow.

Exception handling

- Multi catch block
- Different type of exception could be thrown from try block to handle each exception multi catch could be used one for each exception type ended with General Exception catch

```
string s;
s = Console.ReadLine();
try
{
int x2 = int.Parse(s);
int y = 10 / x2;
Console.WriteLine($"y={y}");
}
catch (FormatException e)
{
  Console.WriteLine(“the number you
   entered in not a valid integer“);
  return;
}
catch(Exception exception)
{
 Console.WriteLine(exception.Message);
  return;
}
```

Exception handling

- *throw* statement
  - Used for throwing an exception

```
ʧsỳ
□
□□
□
çắʧçḥ□□Éẏçêřʧîộŋ□êẏçêřʧîộŋ□
□
□□Cộŋşộľê□ẄsîʧêĽîŋê□□□□□□□□□□□□□□□□□□
□□□□Ŗêʧḥsộxîŋĝ□ụŋêẏřêçʧêđ□êssộs□□□
□□□□êẏçêřʧîộŋ□Ňêşşắĝê□□□□
□□ʧḥsộx□
□
```

```
ʧsỳ
□
□□Ňêʧḥộđ¸□□□
□
çắʧçḥ□□Éẏçêřʧîộŋ□Éẏ□
□
□□Cộŋşộľê□ẄsîʧêĽîŋê□Éẏ□Ňêşşắĝê□□
□
…
řụčľîç□şʧắʧîç□ŵộîđ□Ňêʧḥộđ¸□□
□
□Cộŋşộľê□ẄsîʧêĽîŋê□□êŋʧês□Ņụŋčês□□□
□îŋʧ□ẏ□□
îŋʧ□Răşşê□Cộŋşộľê□ŖêắđĽîŋê□□□□
□îğ□□ẏ□□□¸˚□
□ʧḥsộx□ŋêx□Éẏçêřʧîộŋ□□ņộsê□ʧḥắŋ□
¸˚□□□
□
```

Checking for null

- In some cases checking if reference is not null is **required** otherwise NullReferenceException is thrown so checking is done

```
ṣʧắʧîç□Éṇřľộỳêê□Şêắsçḥ Í D□îŋʧ□Í D□
□
□□□□sêʧụsŋ□ŋụľľ□
□
ṣʧắʧîç□ŵộîđ□□Dîṣřľắỳ Ṛêṣụľʧ□□
□
□Éṇřľộỳêê□êṇř□□□Şêắsçḥ Í D□ , ¸□□
□îğ□êṇř□□ŋụľľ□
□□□
□□□□êṇř□Ṇắṇê□□Alỳ□□
□□□□
□
```

```
ṣʧắʧîç□ŵộîđ□Ňêʧḥộđ , □□Éṇřľộỳêê□êṇř□
□
□□êṇř□Dîṣřắľỳ□□□
□
```

Checking for null

- Null Conditional Operator **?. , ?[ ]**
  - Check if variable equal null
    - If true do nothing
    - If false proceed (call method or Access member)

```
static void Method1 (Employee emp)
{
  if (emp !=null)
  {
        emp.Display();
  }
}
```

```
ştắţîç□ŵộîđ□ņêţḥộđ¸□Ľîşţ□şţsîŋĝ□□şţsĽîşţ□
□
□□□□Cộŋşộľê□ŴsîţêĽîŋê□şţsĽîşţ□□¸□□□
□
```

```
ştắţîç□ŵộîđ□Ňêţḥộđ¸□□Éņřľộỳêê□êņř□
□
□□êņř□□Dîşřắľỳ□□□
□□êņř□□Nắņê□□Aḥņêđ□
□
```

```
ştắţîç□ŵộîđ□Ňêţḥộđ¸□□îŋţ□□ắss□
□
□□îŋţ□□□y□□ắss□□¸□□
□
```

Checking for null

- Null coalescing operator ??
  - Checking if LHS equal null
    - If true => Evaluate RHS and return its value
    - If false => return LHS value

```
string name= GetEmployeeName(10) ?? ”Ahmed”;
```

```
string name=
GetEmployeeName(10);
if(name==null)
 {
        name= “Ahmed”
 }
```

- Null coalescing Assignment Operator ??=

```
string name;
name=Method1();
..
name ?? = “Ahmed”
```

Assignment

- Validate Data input in Menu program

Collection

Collection

- The need of more flexible data structure (ex: dynamically growing and shrinking) was the motive for creation collection
- collections are classes that provide a convenient way to work with groups of objects

Collection

- C# collections typically implement certain key interfaces which define their behavior:
  - **IEnumerable**: Provides the ability to **iterate** through the collection.
    - Readonly Secnario
  - **ICollection**: Defines size, enumerators, and adding and removing methods for all collections.
    - Manipulation Secnario
  - **IList**: Represents a collection of objects that can be individually accessed by index (inserting , removing).
    - Advanced List Operation
  - **IDictionary<TKey, TValue>**: Represents a collection of key-value pairs.

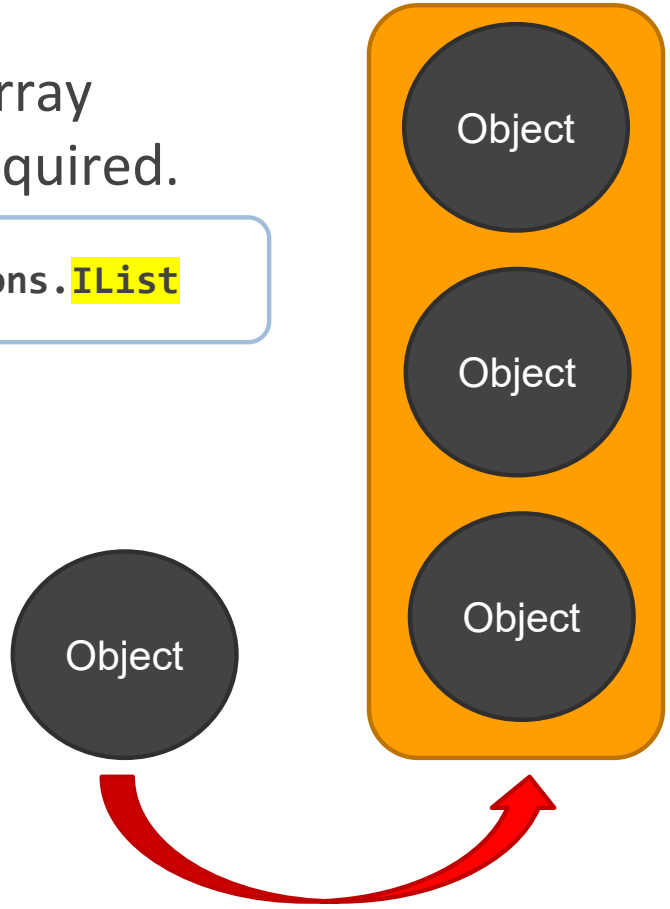
ArrayList

- Implements the IList interface using an array whose size is dynamically increased as required.

```
public class ArrayList : ICloneable, System.Collections.IList
```

- Methods
  - Add(Object)
  - Insert(Index,Object)
  - Remove(Object)
  - RemoveAt(index)
  - RemoveRange(start index, end index)
  - Clear()

```
ArrayList arlist = new ArrayList();
arlist.Add(10);
```



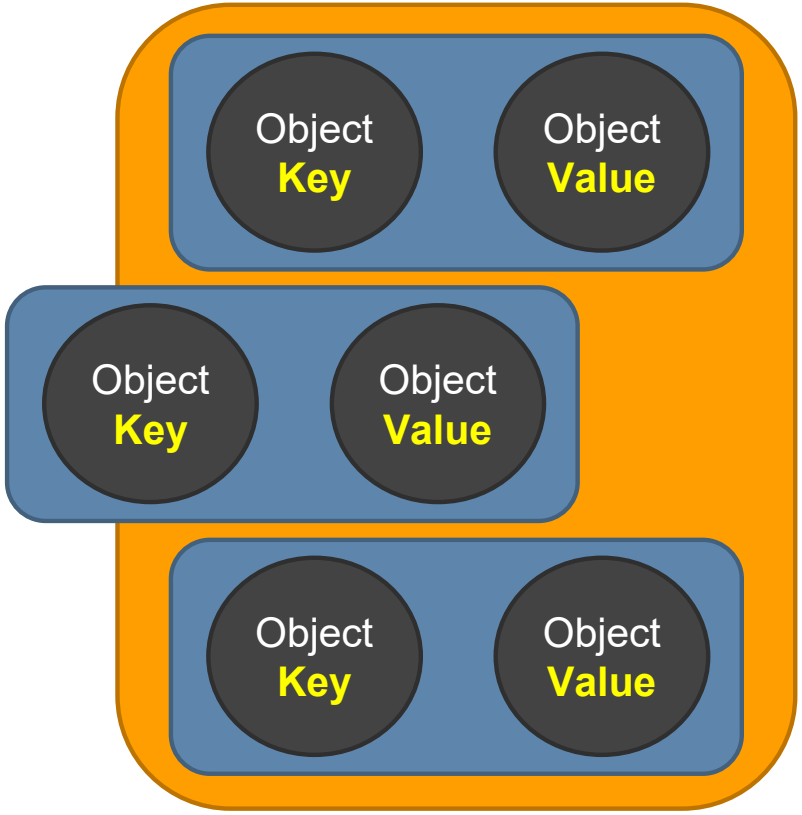
ArrayList

- Methods
  - TrimToSize( )
  - Sort()
  - Reverse()
  - Object[] ToArray()
  - int indexOf(Object)
  - Contains(Object) ➔ Object.Equals()
  - [int index] indexer
- Properties
  - Capacity
  - Count

SortedList

```
public class SortedList : ICloneable, System.Collections.IDictionary
```

- Represents a collection of key/value pairs that are sorted by the keys and are accessible by key and by index.
- Collection of (key-value) pair where key Is unique
- Internally maintains two arrays
  - One for keys one for values
- Auto sorted by
  - key's implementation of *IComparable*
  - using *IComparer* Implementation
    - Passed to Constructor



SortedList

- Mehods
  - Add(Object Key, Object item)
  - Clear()
  - ContainsValue()
  - ContainsKey()
  - IndexOfKey(key)
  - indexOfValue(Value)
  - RemoveAt(index)
  - TryGetValue(Tkey,out Value)
  - GetByIndex(index)
  - GetKey(index)
- Access to elements through *key* or through *index*

```
ŞộsʧêđĽîṣʧ□ṣĺ□ŋêx□ŞộsʧêđĽîṣʧ□□□
ṣĺ□Ađđ□,□,□□
ṣĺ□Ađđ□_□_□□
Şỳṣʧêṇ□Cộŋṣộĺê□ẄsîʧêĽîŋê□ṣĺ□,□□□
□□□řsîŋʧṣ□,
Şỳṣʧêṇ□Cộŋṣộĺê□ẄsîʧêĽîŋê□ṣĺ□Ğêʧ BỳÍŋđêy□。□□□
□□□řsîŋʧṣ□,
```

SortedList

- Iterate through SortedList
  - Using index

```
ǧộs□îŋʧ□î□¸□î□□şľ□çộųŋʧ□î□□□
□
□□Cộŋşộľê□ẄsîʧêĽîŋê□□□lêỳ□□şľ□ĞêʧĶêỳ□î□□□ʧ□ŵǻľųê□□şľ□ĞêʧBỳÍŋđêỵ□î□□□□□
}
```

  - Using DictionaryEntry

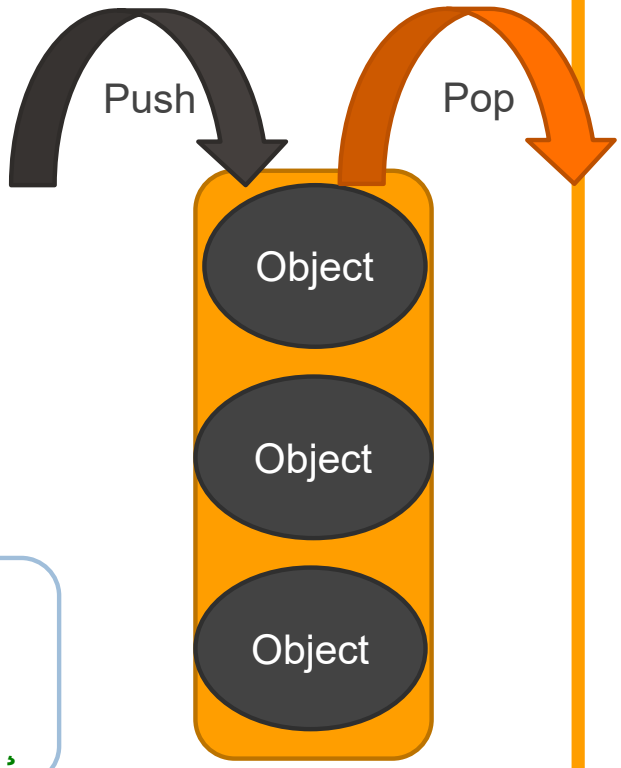
```
□ǧộsêǻçḥ□DîçʧîộŋǻsỳÉŋʧsỳ□řǻîs□îŋ□şľ□
□
□□Cộŋşộľê□□ẄsîʧêĽîŋê□□□lêỳ□□řǻîs□Ķêỳ□□ʧ□ŵǻľųê□□řǻîs□Ʌǻľųê□□□□
}
```

Stack

```
public class Stack : ICloneable, System.Collections.ICollection
```

- Represents a simple last-in-first-out (LIFO) non-generic collection of objects.
- Methods
  - Push()
  - Pop()
  - Peek ()
  - Clear()
  - Contains(Object)
  - Object[] ToArray()
- Properties
  - Capacity
  - Count

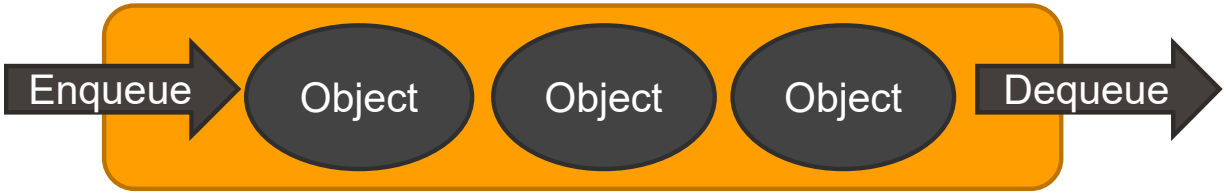
```
Ş₮ǎçl□ş□ŋêx□Ş₮ǎçl□□□
ş□Rụṣḥ□¸□□
ş□Rụṣḥ□¸□□
Cộŋşộľê□Ẅsî₮ĝêĽîŋê□ş□Rộř□□□□□□□řsîŋ₮ş□¸
```



Queue

```
public class Queue : ICloneable, System.Collections.ICollection
```

- Represents a first-in, first-out collection of objects.
- Methods
  - Enqueue()
  - Dequeue()
  - Peek ()
  - Clear()
  - Contains(Object)
  - Object[] ToArray()

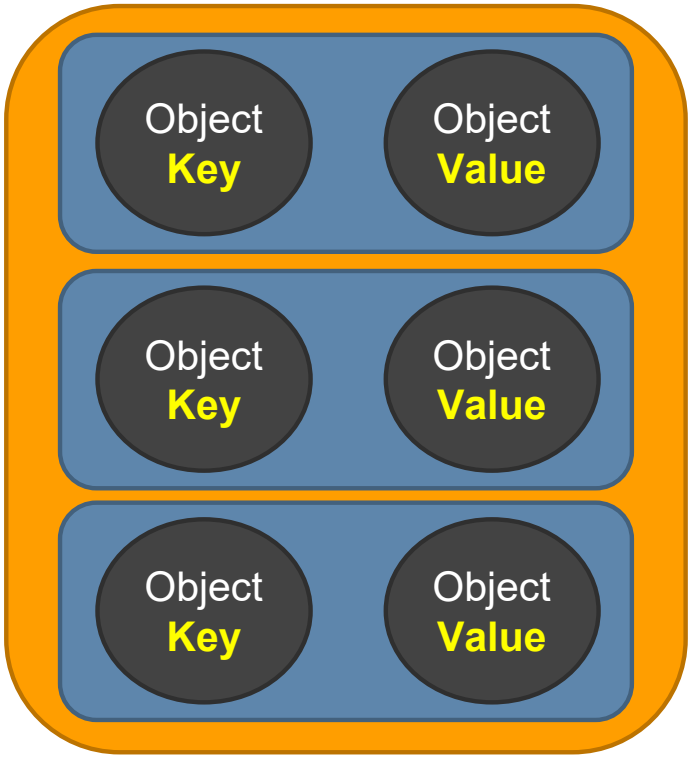


- Properties
  - Capacity
  - Count

```
Queue q = new Queue();
q.Enqueue(1);
q.Enqueue(2);
Console.WriteLine(q.Dequeue().ToString()); // prints 1
```

Hashtable

- Store Data in Key-value format, where keys are unique and used in indexer
  - Ex: Dictionary (word – meaning)
- Methods
  - void Add(object key, object value)
  - void Clear()
  - bool ContainsKey(object key)
  - bool ContainsValue(object value)
  - void Remove(object key);



Hashtable

- Properties
  - Count
  - Item[Key]
  - Keys
  - values

```
Hashtable ht = new Hashtable();
ht.Add("One",1 );
ht.Add("Two", 2);
ht.Add("three", 3);
Console.WriteLine(ht["three"].ToString()); // print 3
```

```
 foreach(DictionaryEntry node in ht)
{
   Console.WriteLine(node.ToString()); // print 3
}
```

```
foreach (var k in ht.Keys)
{
   Console.WriteLine(k.ToString());
}
```

Assignment

- Modify Menu program to use ArrayList instead of Array of Employees
  - New
    - Add one employee at a time
  - Display
    - Display all Employees
  - Search
    - Search employee by (Id , name)
  - Sort
    - Sort Employee using Sort(IComparer)