Inheritance(cont.)

- abstract class
  - Partially designed class
  - Class for design purpose only

abstract class
Geoshape
Int dim1,dim2
abstract CalcArea( )

```
abstract class Geoshape
{
    protected int dim1, dim2;
     …
    public abstract float CalcArea();
}
```

Inheritance(cont.)

- Interface
  - Considered as a contract
  - Support inheritance
    - Ex: Iqueryable :IEnumerable

```
interface Imyinter
{
 int prop { set; get; }
        void mymethod();
}
```

Inheritance

- Implement interface
  - Implicitly
    - Through class reference
    - Through interface reference

```
class myclass : Imyinter
    {
        void mymethod()
        {…..}
    }
```

  - Explicitly
    - No access modifier
    - Through interface reference only
  - Used in case of multiple implementation

```
class myclass : Imyinter
    {
        void Imyinter.mymethod()
        {…..}
    }
```

Var vs Object

- Var strongly type
  - Object reference could referee to object of any type
- Var read only (immutable)
  - Object will not access members(except that contained in Object Class)
- Var can’t be used as method parameter (Local variable)
  - Object can
- Anonymous type associated with var contain ToString method override

Assignments

- menu program
- Sort an Array of Employee (hard coded way)
  - Using Array. Sort(Array)
  - by implementing IComparable interface in Employee class
    - Implementing CompareTo(Object) method
- Sort Array of Employee (Dynamic Way)
  - Using Array. Sort(array, IComparer)
  - By implementing the way of sorting in classes that implements Icomparer Interface
    - Implementing compare (Object, Object) method in these classes

Exception handling

- Handling exception

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

FormatException

- Try catch

```
try {
        int x2 = int.Parse(s2);
    }
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

Exception handling

- Try catch finally

```
string s;
s = Console.ReadLine();
try
{
    int x2 = int.Parse(s);
    int y = 10 / x2;
    Console.WriteLine($"y={y}");
}
catch (FormatException e)
{
    Console.WriteLine(e.Message);
}
catch(Exception e)
{
    Console.WriteLine(e.Message);
}
finally
{
    Console.WriteLine("Thank You!!");
}
```

Exception Types

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.

Assignments

- Adding handling exception for employee data input

Class(cont.) adv. topic

- Garbage collector and memory management
- GC and resource management
  - Finalizer (destructor)
    - Time of invoking destructor
  - Dispose method and IDisposable interface
    - Runtime error and exception
      - Try catch finally
    - *using* statement
      - Clean resources twice
        - System.GC.SuppressFinalize(**this**);

Finalize(destructor) and Dispose

- Resources cant be handled by GC (Responsible for Memory only)
    - File handlers
    - window handlers
    - network sockets
    - database connections
- Finalize method used to clean these resources
    - Implemented through class destructor
    - Disadvantage
        - Unknown time of call

Dispose method

- Dispose method by Implementing IDisposable interface
  - Implementing IDisposable Interface

```
public class class1 : IDisposable
```

  - Preventing distructor from being called

```
GC.SuppressFinalize(this);
```

  - Called directly

```
employee emp = new employee();
emp.Dispose();
```

  - Called Through *using* statement

```
using ( employee emp = new employee() )
{
    //scope of emp variable
}
```