OOP

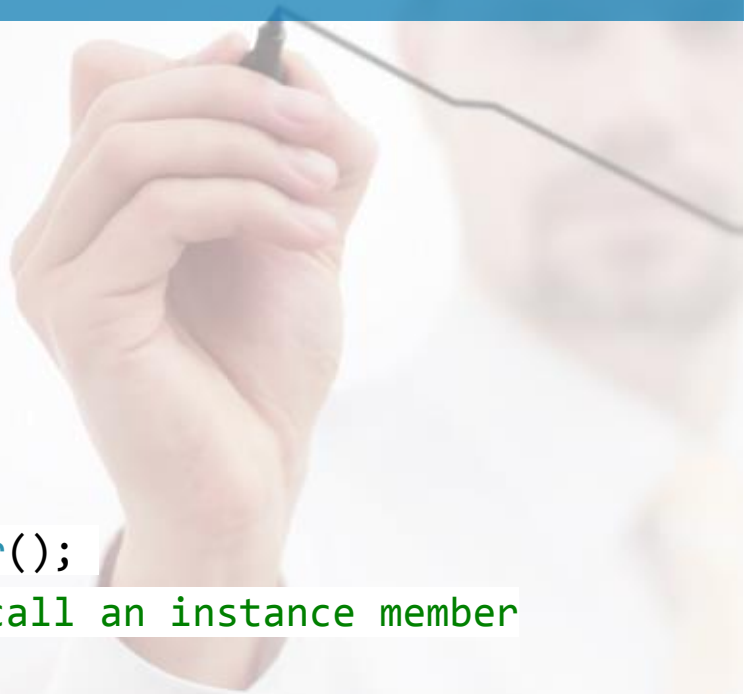
Authored by : Sushant Banerjee
Email : sushantba@cybage.com

Presented by : Sushant Banerjee
Extn. 7210

Agenda

- Classes and Objects
- Access Modifiers
- Fields
- Methods
- Arguments
- Extension Method
- Async Method
- Constructors and Destructors

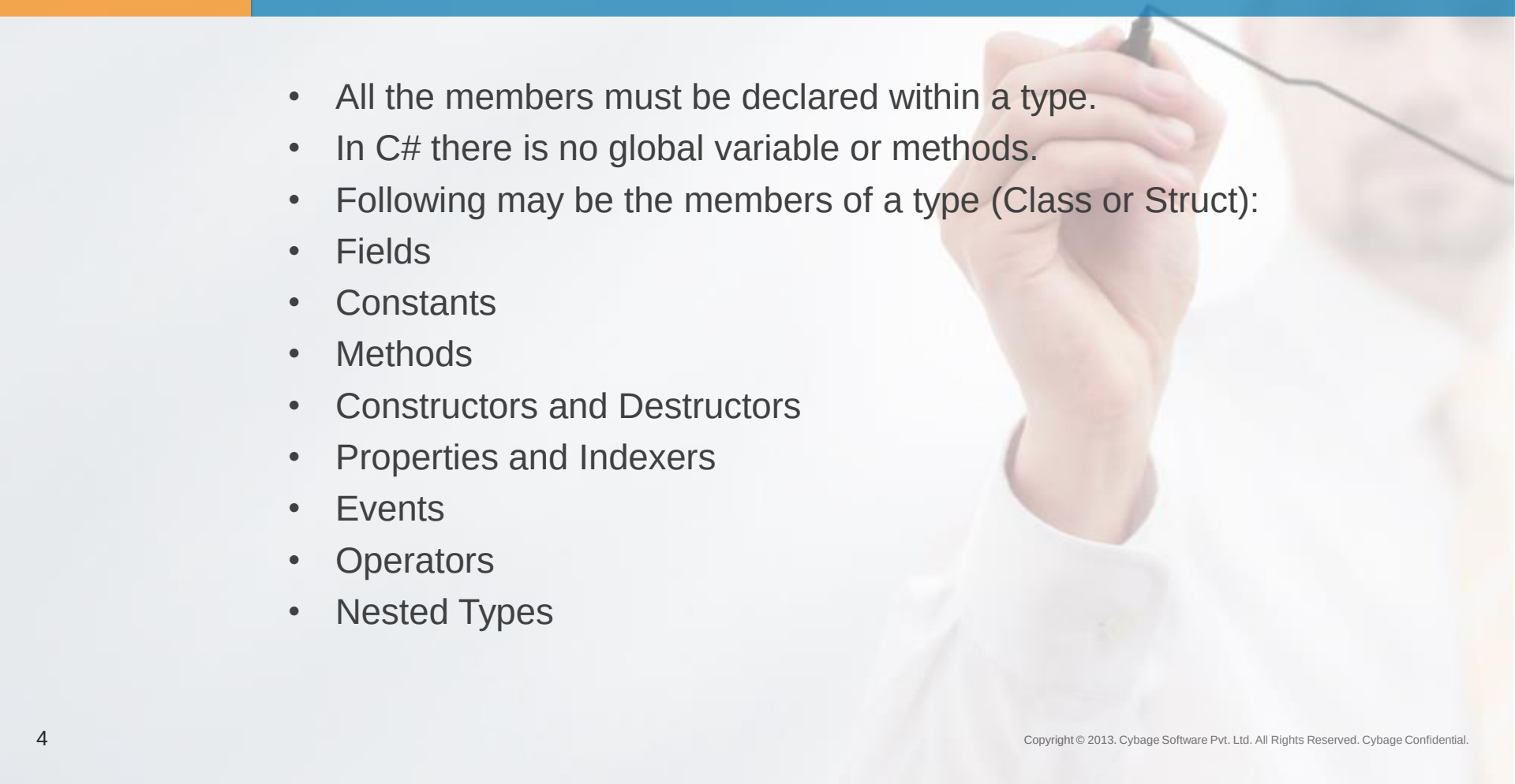
Classes and Objects



```
class Calculator
{
    public void Add()
    {
    }
}

Calculator calc= new Calculator();
calc.Add(); //using object to call an instance member
```

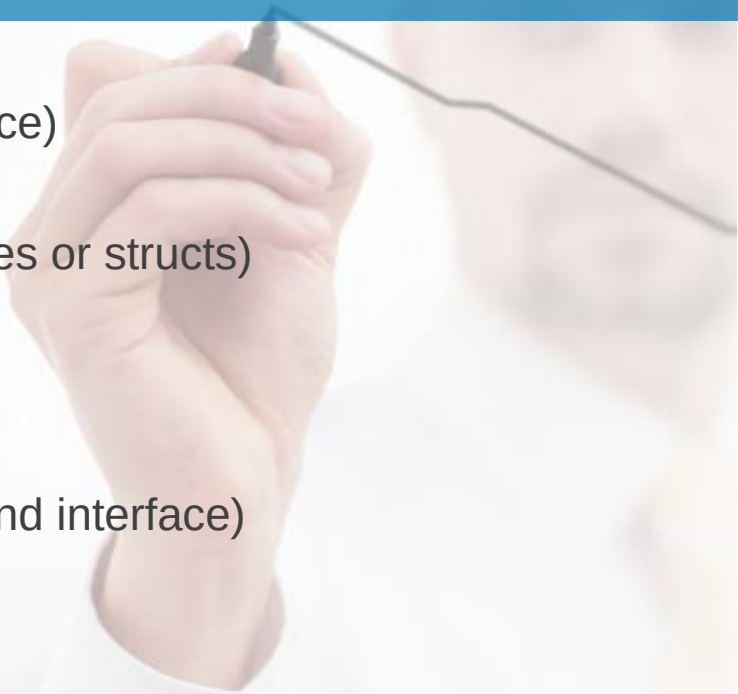
Members of a Type (Class or Struct)



- All the members must be declared within a type.
- In C# there is no global variable or methods.
- Following may be the members of a type (Class or Struct):
- Fields
- Constants
- Methods
- Constructors and Destructors
- Properties and Indexers
- Events
- Operators
- Nested Types

Access Modifiers

- Public (Default for members of interface)
- Private (Default for members of classes or structs)
- Protected
- Internal (Default for classes, structs and interface)
- Protected Internal



Fields

- Fields are variables that is member of a class or struct.
- Fields are declared directly in a class or struct.
- Variables declared inside the scope of a method is a local variable.
- Accessibility of a fields should be either private or protected.
- The data in fields should be exposed to client code through properties, methods and indexers.
- Fields are initialized immediately before the constructor of the class.

Static Vs. Instance Fields

- A field can be optionally declared as static.
- A static field is part of a class and not instance and is accessible by class name, not by instance name.
- An instance field is part of the instance and different instances get a copy of the same field.
- Only one copy of a static member exists, regardless of the how many instances of the class are created.
- Change made in one instance field by an object of the class is not visible to the other instances of the class.

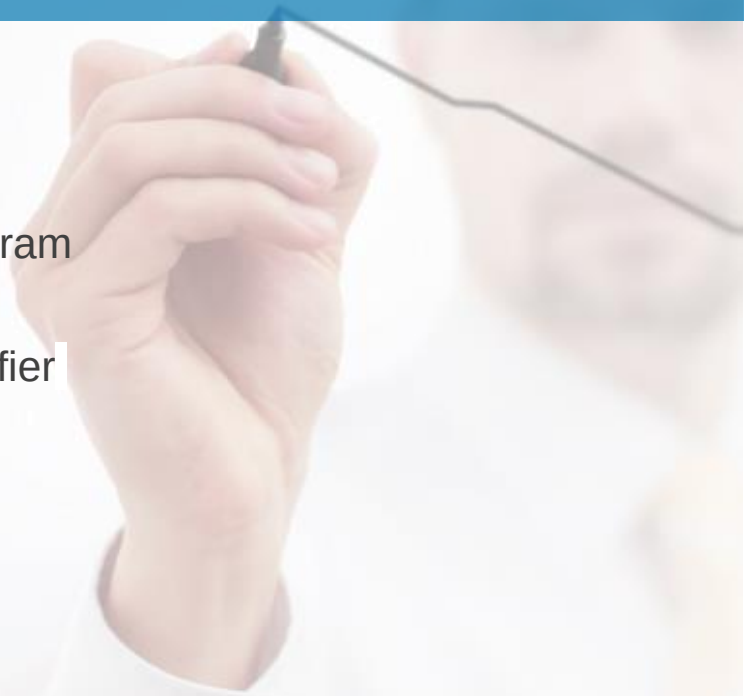
Read-Only Fields

- The readonly keyword can be used to declare a field.
- Value of the readonly field can not be changed.
- A readonly field can be initialized either in the
  - Declaration statement in the class level or
  - Inside the constructor of the same class

```
class Employee
 {
      readonly int empid;
      public Employee()
      {
          empid = 20;
      }
 }
```

Constants

- Constants are known at compile time
- Can’t be changed for the life of the program
- Constants are declared with const modifier



```
public const int empid = 123;
empid = 456;//compile time error, cannot change empid.
```

Readonly Vs. Constants

- The readonly fields can be initialized either in the declaration statement or in the constructor.
- The const fields must only be initialized in the declaration statement.
- A const field is compile time constant, the readonly field can be used as runtime constant.

```
readonly string dt = DateTime.Now.ToString();//compile successfully
const string dt2 = DateTime.Now.ToString();//compile time error
```

Understanding Methods

- Method Signature
- Return Type of Methods
- Parameters and Arguments
- Using **ref** and **out** keywords

Method Signature

- Signature of the method includes only name of the method and the data type of its parameters, not return type and parameter name.

```
public void Add(int num1, int num2)
{
   //statements to execute
}
```

Return type of a method

- Return type of the method can be void or any other type.
- If return type is void the method will not return any value.
- A method uses return keyword to return data to the caller.

```
public int Add(int num1, int num2)
{
    return num1 + num2;
}
```

Parameters Vs. Arguments

- A method can have parameters to accept input.
- A method definition contains names and types of parameters.
- When calling code calls the method provides concrete values as arguments for each parameter.

```
public int Add(int a, int b) //parameters
{
    return a + b;
}

Employee emp = new Employee();
emp.Add(5, 10); //arguments
```

Passing by value Vs. by reference

- While calling methods you pass arguments from calling code either
  - By value or
  - By Reference (ref keyword)

```
static void SwapStrings(ref string s1, ref string s2)
SwapStrings(ref str1, ref str2); // Passing strings by reference
```

The out keyword

- The out keyword causes arguments to be passed by reference.
- The out keyword works similar to the ref keyword, the difference is the ref keyword requires variable be initialized before it is passed.
- While using out keyword it is not necessary to initialize the variable before passing.
- The out keyword is useful when we want a method should return multiple values.

```
public static void Square(out int a)
{
    a = 5 * 5;
}
int num1;
Calculator.Square(out num1);
```

Arguments

- Positional Arguments
- Named Arguments (New in C# 4.0)
- Required Arguments
- Optional Arguments (New in C# 4.0)

Extension Methods (C# 3.0)

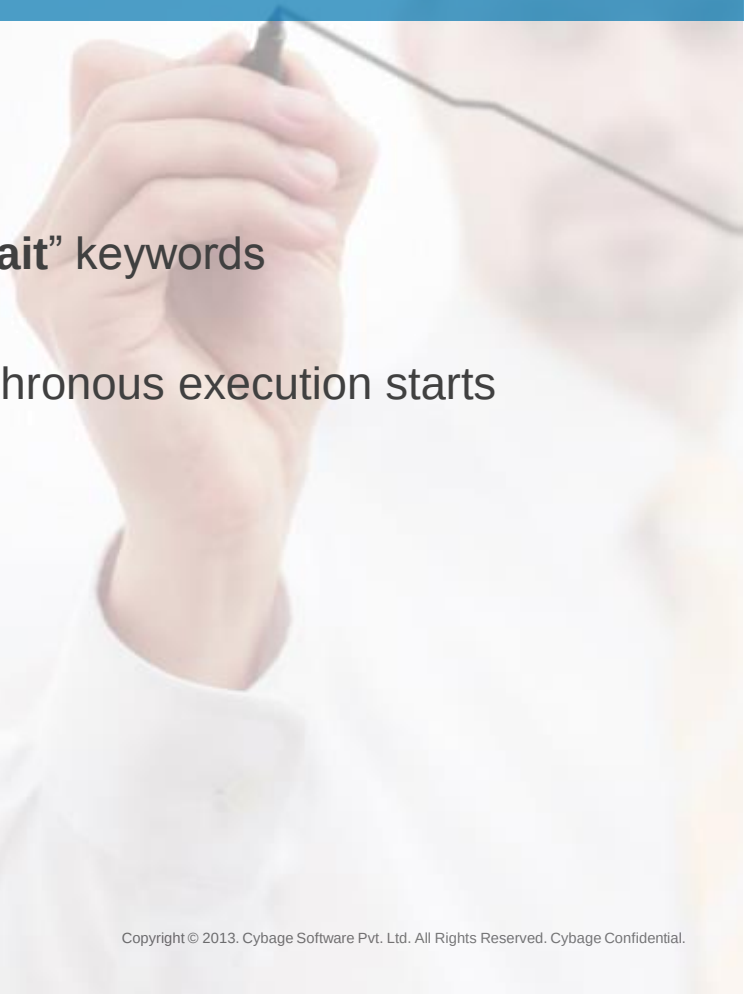
- Add new methods to existing types
- No need to modify original type
- These are static but look like instance
- The parameter preceded by “this” modifier

Extension Methods Cont'd

```
public static class CustomExtension
{
   public static int CountWords(this string str)
   {
       return str.Split(new char[] {' '},
  StringSplitOptions.RemoveEmptyEntries).Length;
   }
}
 //calling extension method
 string str = "I love reading";
 Console.WriteLine(str.CountWords());
```

Async Method (C# 5.0)

- Asynchronous Process
- Async method uses “**async**” and “**await**” keywords
- It is “**await**” keyword where the asynchronous execution starts



Async Method

```
public async Task<DateTime> ReturnServerDateTime(){
  for (int i = 0; i < 5; i++)
  {
      await Task.Delay(1000);
  }
  return DateTime.Now;
}
public async void ShowServerDateTime(){
   var serverDateTime = await ReturnServerDateTime();
 Console.WriteLine("Server's Datetime : {0}",
  serverDateTime);
}
```

Constructors

- Default Constructor
- Instance Constructors
- Private Constructors
- Static Constructor

Destructors

```
class Calculator
{
    ~Calculator()
    {
        //cleanup code
    }
}
```

Bibliography, Important Links

- C# 5.0 in a Nutshell – Published by O'Reilly
- www.msdn.com – Library
- http://en.wikipedia.org

Any Questions?

Thank you!