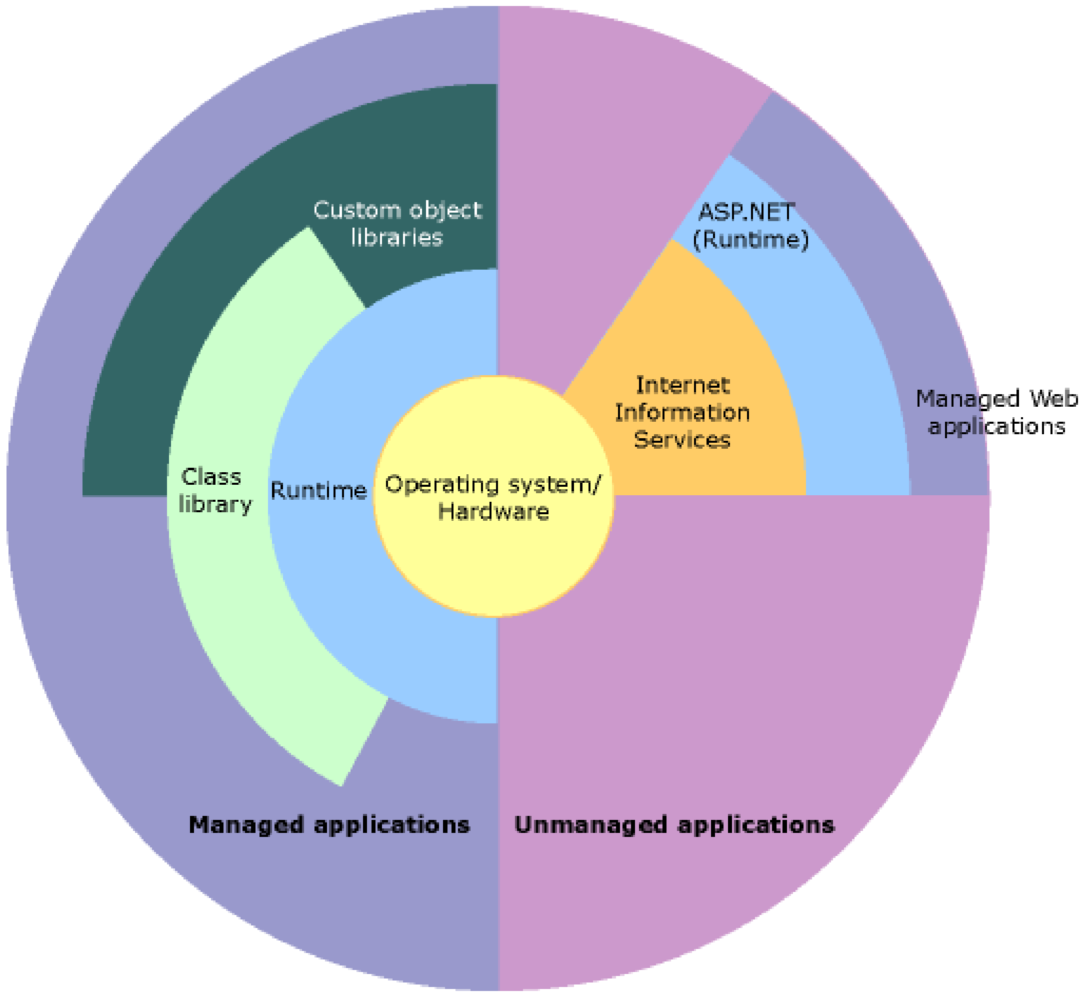
Introduction To C#

Sushant Banerjee | sushantba@cybage.com | 7221

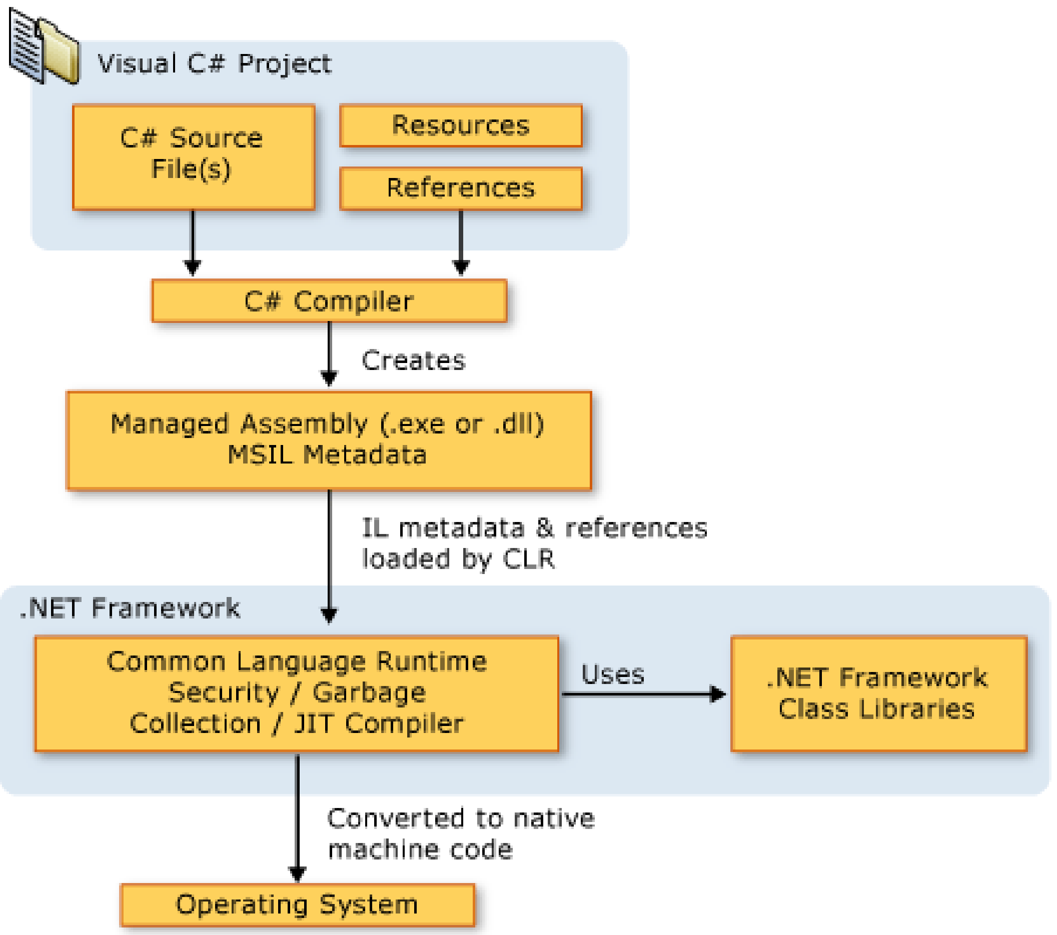
Agenda

- Overview of .NET Framework
- Understanding Compilation Process
- Common Type System
- Primitive Types
- Classes and Objects
- Statements, Expressions and Operators
- Properties and Methods
- Access Modifiers
- Static classes and Static members
- Constructors and Destructors

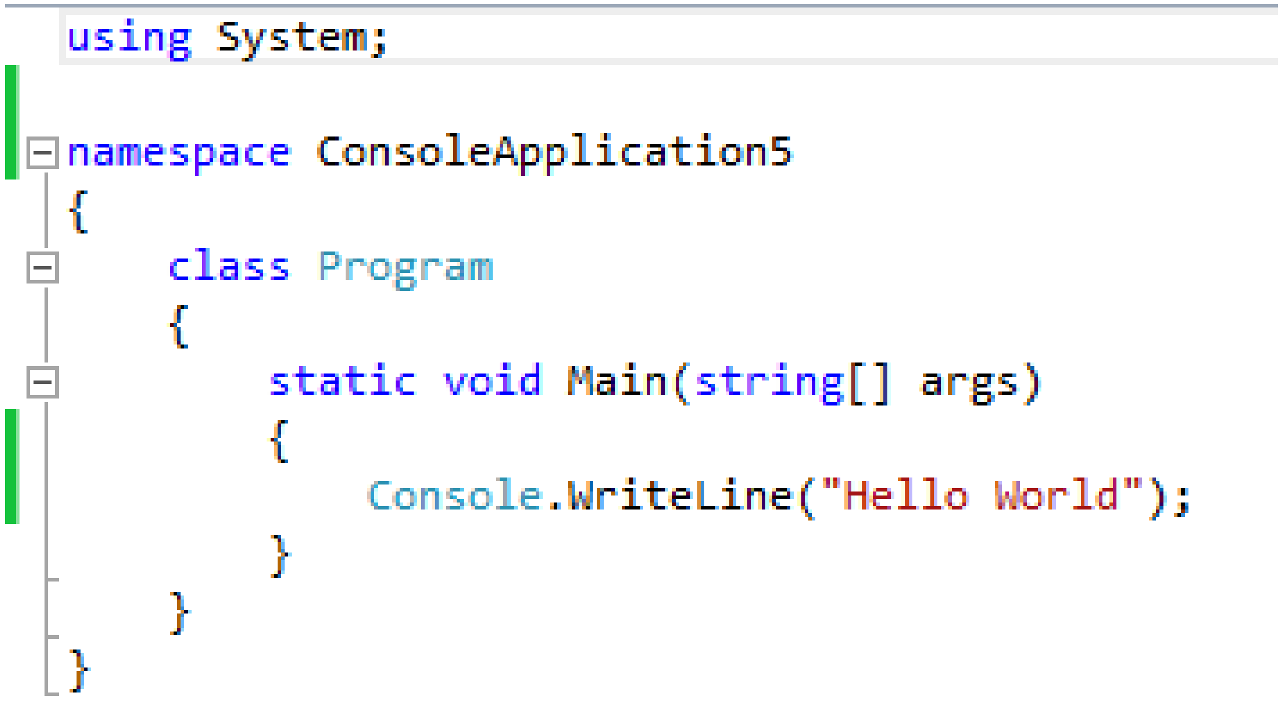
Overview of .NET Framework



Managed Execution Process


Visual C# Project
C# Source File(s)
Resources
References
C# Compiler
Creates
Managed Assembly (.exe or .dll) MSIL Metadata
IL metadata & references loaded by CLR
.NET Framework
Common Language Runtime Security / Garbage Collection / JIT Compiler
Uses
.NET Framework Class Libraries
Converted to native machine code
Operating System

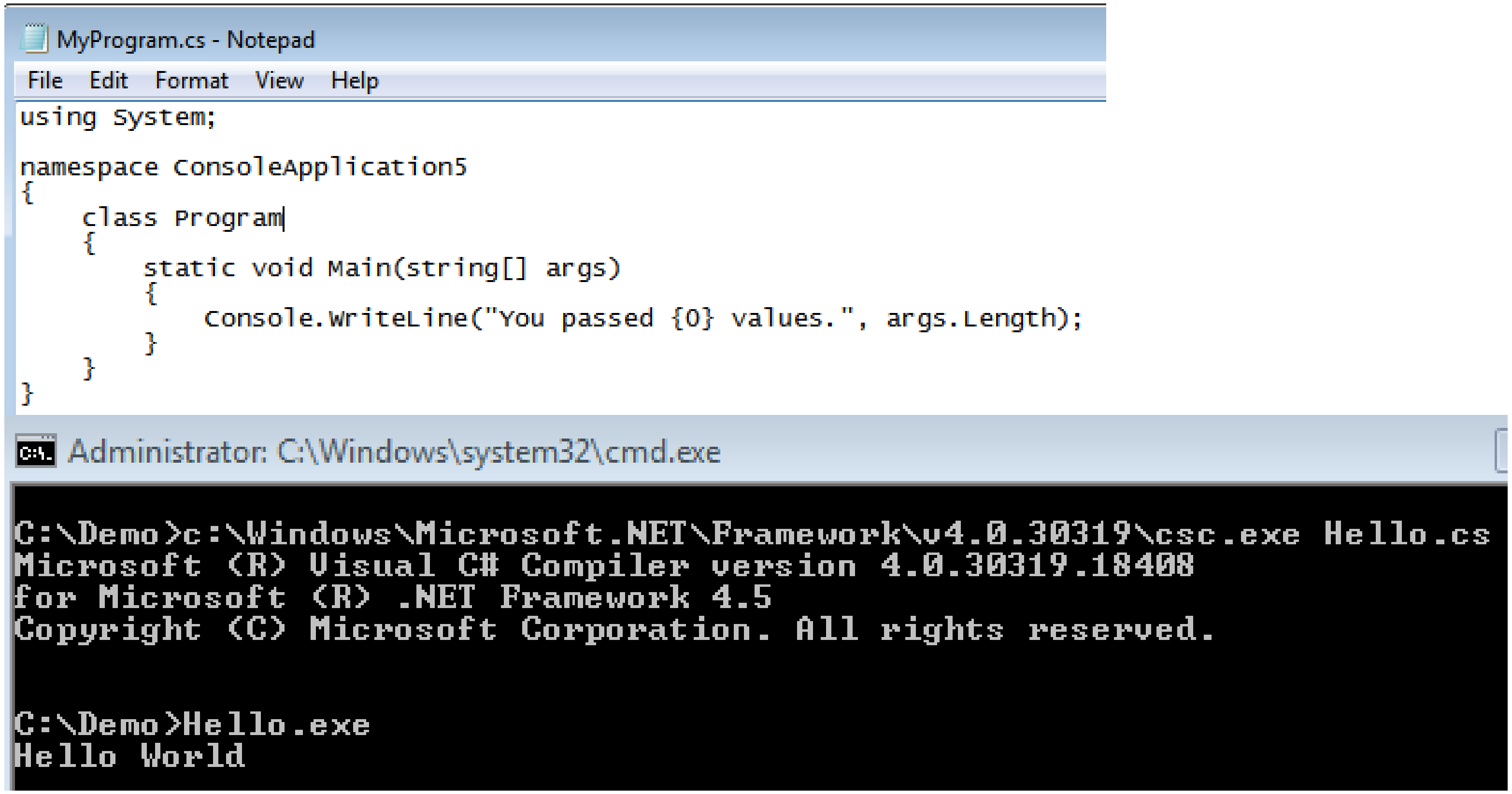
Basic Structure of a C# Program



```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

Compilation and Execution

- Compiling program
  - Using Command Prompt
  - Passing command line arguments.


MyProgram.cs - Notepad

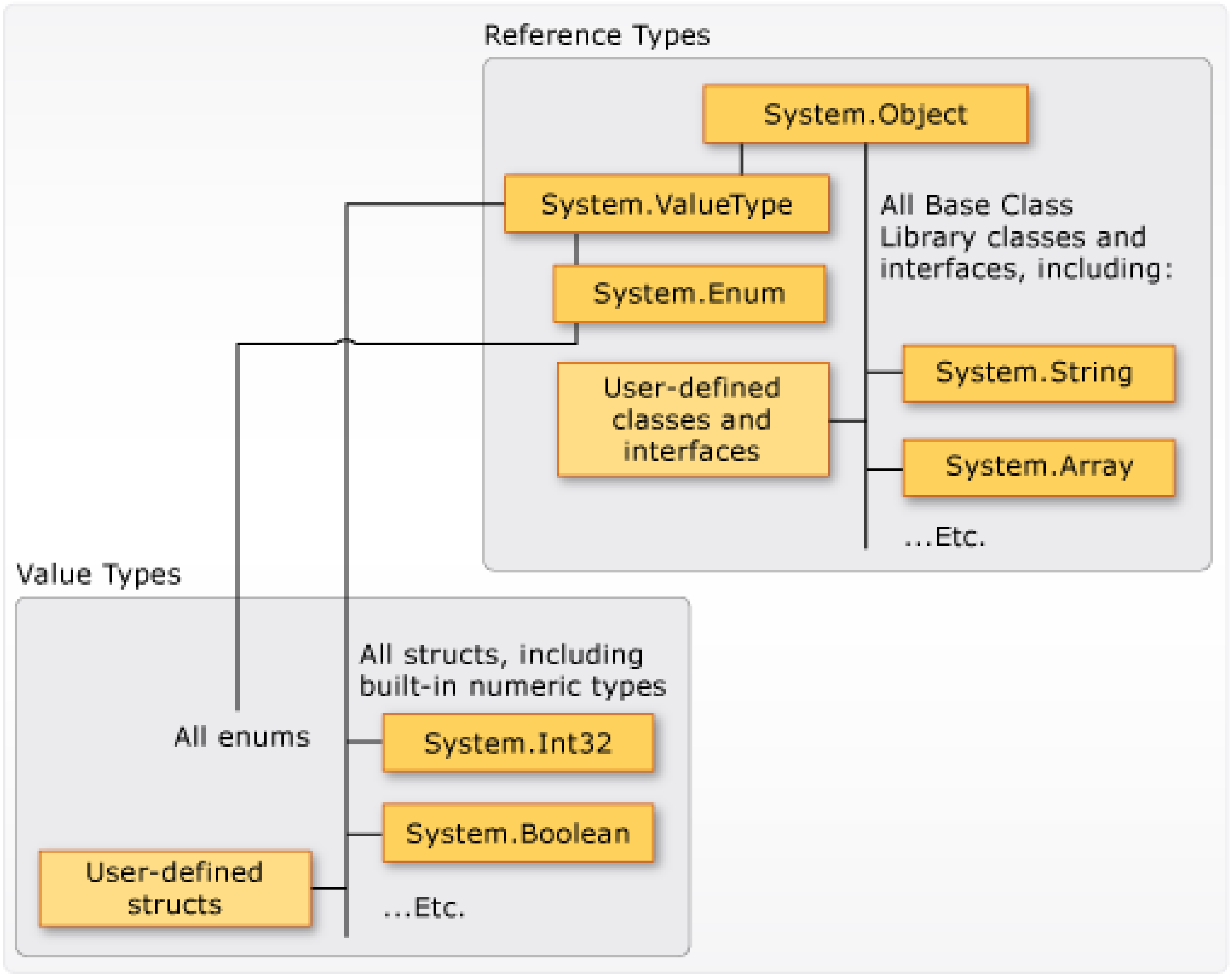
```
using System;

namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("You passed {0} values.", args.Length);
        }
    }
}
```

Administrator: C:\Windows\system32\cmd.exe

```
C:\Demo>c:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe Hello.cs
Microsoft (R) Visual C# Compiler version 4.0.30319.18408
for Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Demo>Hello.exe
Hello World
```

Demo

Common Type System (CTS)



Cybage®
Delivering Value. Scientifically.

Numeric Types - Integral

C# Type	System Type	Suffix	Size	Range
sbyte	SByte		8 bits	-128 to 127
short	Int16		16 bits	-32,768 to 32,767
int	Int32		32 bits	-2,147,483,648 to 2,147,483,647
long	Int64	L	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Numeric Types – Real Numbers

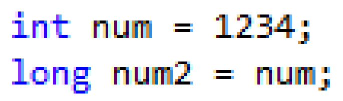
C# Type	System Type	Suffix	Size	Approximate Range	Precision
float	Single	F	32 bits	±1.5e−45 to ±3.4e38	7 digits
double	Double	D	64	±5.0e−324 to ±1.7e308	15 – 16 digits
decimal	Decimal	M	128 bits	±1.0 × 10−28 to ±7.9 × 1028	28 – 29 digits

Other Types

C# Type	System Type	Size	Range
Bool	Boolean	8 bits	True or False
char	Char	Unicode 16 bits	U+0000 to U+FFFF
string	String	2 GB	0 to 2 Billion characters

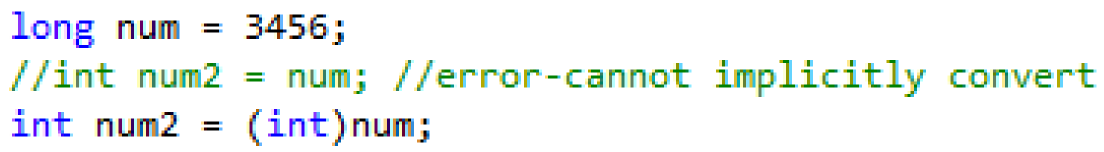
Type Conversion

- Implicit Conversion



```
int num = 1234;
long num2 = num;
```

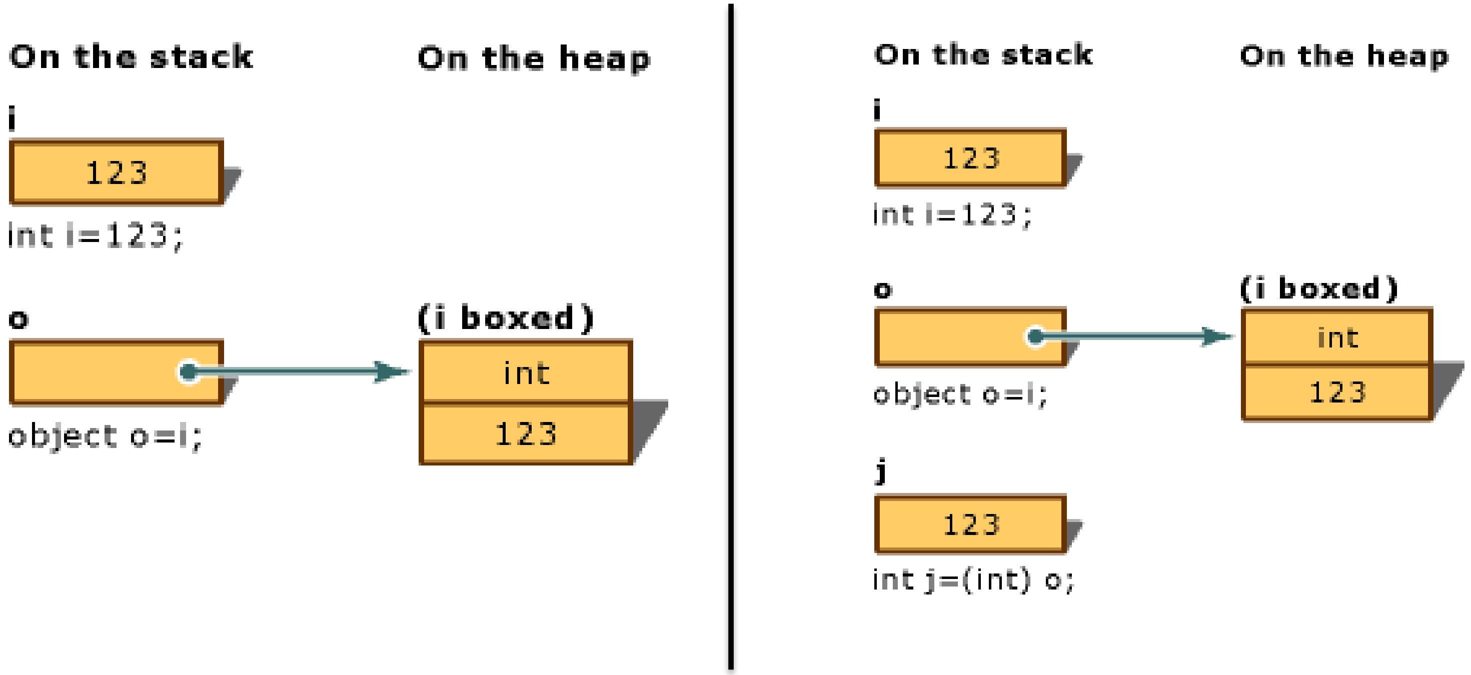
- Explicit Conversion



```
long num = 3456;
//int num2 = num; //error-cannot implicitly convert
int num2 = (int)num;
```

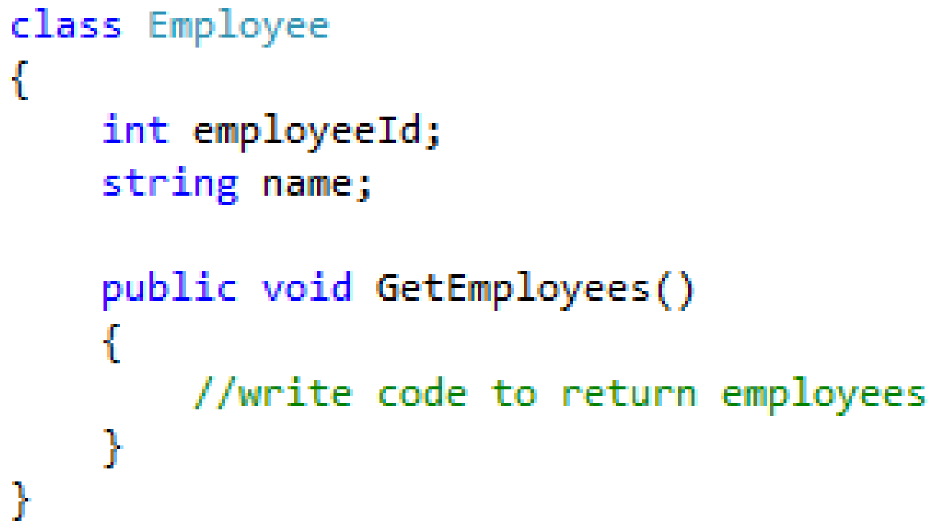
Boxing and Unboxing

- Computationally expensive process



Classes

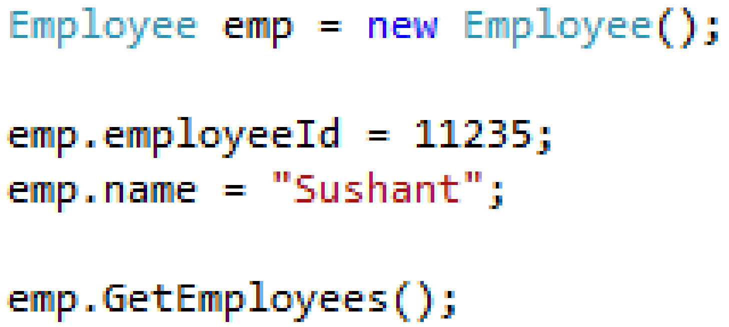
- A class is a custom reference type
- Contains members such as
  - Variables
  - Properties
  - Methods etc.



```
class Employee
{
    int employeeId;
    string name;

    public void GetEmployees()
    {
        //write code to return employees
    }
}
```

Objects

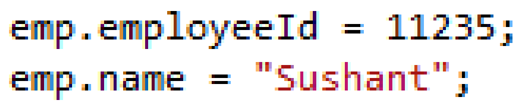
- An object is a block memory allocated based on the class
- Objects are created using “new” keyword
- An object is used to access members of the class.



```
Employee emp = new Employee();

emp.employeeId = 11235;
emp.name = "Sushant";

emp.GetEmployees();
```

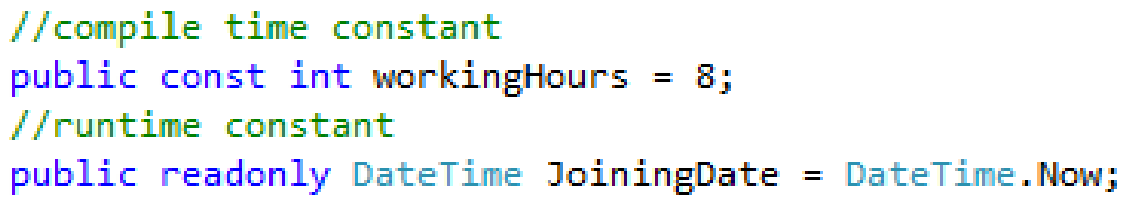
Variables, Constants and Readonly

- Values of a variable may vary



```
emp.employeeId = 11235;
emp.name = "Sushant";
```

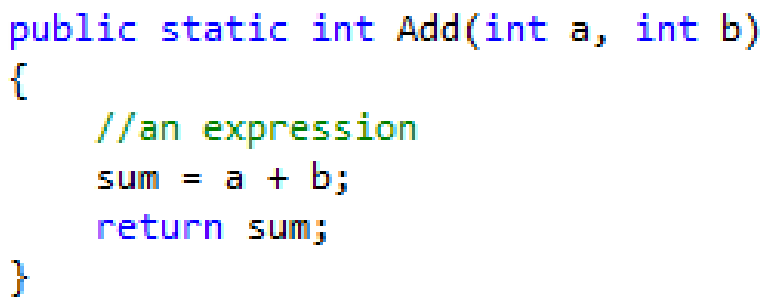
- Values of a constant is fixed



```
//compile time constant
public const int workingHours = 8;
//runtime constant
public readonly DateTime JoiningDate = DateTime.Now;
```

Statements and Expressions

- Selection statements
  - If, else, switch, case
- Iteration statements
  - Do, for, foreach, in, while
- Jump statements
  - Break, continue, default, goto, return
- Expressions are sequence of one or more operands and operators
- Expressions can be evaluated to a single value.



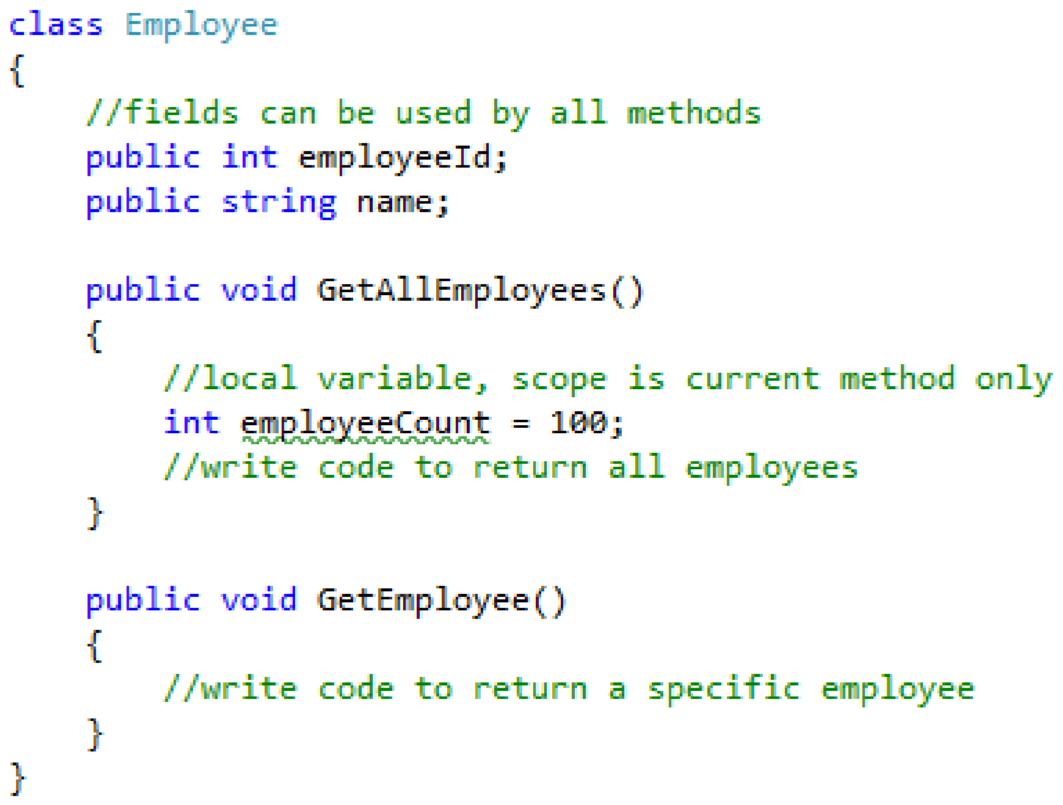
```
public static int Add(int a, int b)
{
    //an expression
    sum = a + b;
    return sum;
}
```

Operators

- An operator is used along with operands to create expressions
- Unary operators
  - X++, X--, ++X, --X
- Binary operators or Arithmetic operators
  - X + Y, X – Y, X * Y, X / Y, X % Y
- Relational or Comparison operators
  - X > Y, X < Y, X >= Y, X <= Y, X == Y, X != Y
- Conditional AND – X && Y
- Condition OR – X || Y

A Field

- A field is a variable declared in class level
- A field initialized immediately before the constructor.



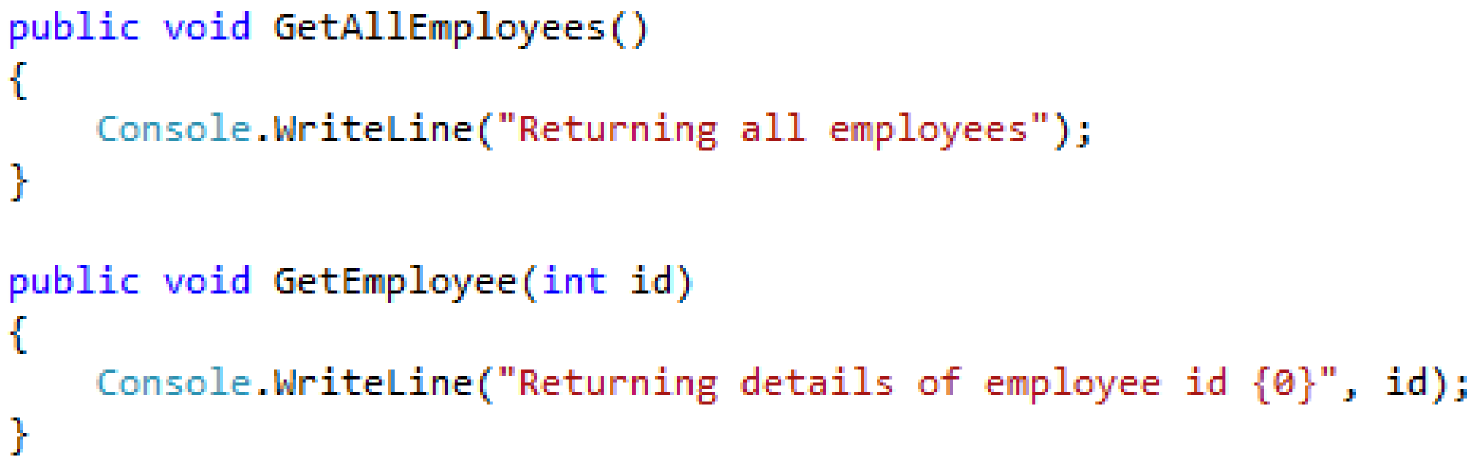
```
class Employee
{
    //fields can be used by all methods
    public int employeeId;
    public string name;

    public void GetAllEmployees()
    {
        //local variable, scope is current method only
        int employeeCount = 100;
        //write code to return all employees
    }

    public void GetEmployee()
    {
        //write code to return a specific employee
    }
}
```

Demo

Cybage Delivering Value. Scientifically.

Methods

- A method is a block of code
- Can perform a task when called
- Excepts parameters and return values.



```
public void GetAllEmployees()
{
    Console.WriteLine("Returning all employees");
}

public void GetEmployee(int id)
{
    Console.WriteLine("Returning details of employee id {0}", id);
}
```

Methods and Modifiers

- Methods express behavior of a class
- Keywords change that behavior
  - Public
  - Private
  - Virtual
  - Static
- Keywords also controls arguments
- Parameter modifiers
  - None
  - Out
  - Ref
  - Params

Optional Parameters

- Allows you to omit arguments
- If necessary, pass it and ignore default value
- Place at the end of the parameters list
- Must be know at compile time

```
 void SomeMethod(string city = “Pune”)
{
   //method logic
}
```

Named Parameters

- Passing arguments by position is not must any more
- Allows you to place arguments in any order
- Place named arguments after all positional arguments
- Useful when using along with optional arguments
- Call a method using following syntax:

```
SomeMethod(ParameterName : Value)
```

Nullable Types

- Numeric types cannot be assigned null values
- Use ? Operator to make it nullable
- It’s a way to set no value to a numeric type
- Use ?? Operator to assign some value if null

```
// Nullable data field.
public int? numericValue = null;

//If GetEmpId returns null, assign 100 as default
int myData = emp.GetEmpId() ?? 100;
```

Demo

String Type

- The string type represents an immutable sequence of unicode characters
- Verbatim String solves the below problem.

```
string path = "C:\\Windows\\Microsoft.NET\\Framework64";
```

- Verbatim string literal is prefixed with @ and does not support escape sequence

```
string path = @"C:\Windows\Microsoft.NET\Framework64";
```

String Concatenation

- The + operator is used to concatenate string values.

```
string name = "Sushant" + " Banerjee";
```

- If one of the value is nonstring, ToString method is called on that value.

```
string username = "Sushant" + "0510"; //Sushant0510
```

- Using the + operator to concatenate string values creates a new string to store the new value each time.

- Which ads lot of memory overhead and risk of running without memory.

String concatenation - The problem

- The below code will create 100 string instances

```
string mystring = "";
for (int i = 0; i < 100; i++)
{
//doing string concatenation
//mystring = mystring + "--" + i.ToString();

//below line of code is same as above
mystring += "--" + i.ToString();
}
Console.WriteLine(mystring);
```

StringBuilder Class – The Solution

- To solve this problem use `System.Text.StringBuilder` class.
- In case of larger string concatenation it is recommended to use StringBuilder class.
- The advantage is all the manipulation is made in the same StringBuilder instance instead of creating new instance each time.

```
StringBuilder mystring = new StringBuilder();
```

Using StringBuilder Class

```
StringBuilder mystring = new StringBuilder();

for (int i = 0; i < 100; i++)
{
//doing string concatenation
//mystring = mystring + "--" + i.ToString();

//below line of code is same as above
//mystring += "--" + i.ToString();
mystring.Append("--");
mystring.Append(i);
}
Console.WriteLine(mystring);
```

Cybage® Delivering Value. Scientifically.

String Manipulation Methods

Method	Description
Contains, StartsWith, EndsWith	To search specific word in a string
IndexOf	Returns index of specific character or string in a string value
Substring	Extracts part of the string
Insert, Remove, Replace	To insert or remove characters
TrimStart, TrimEnd, Trim	To remove whitespace characters
ToUpper, ToLower	Returns uppercase or lowercase string
Split and Join	Split a sentence into array of words and join does opposite

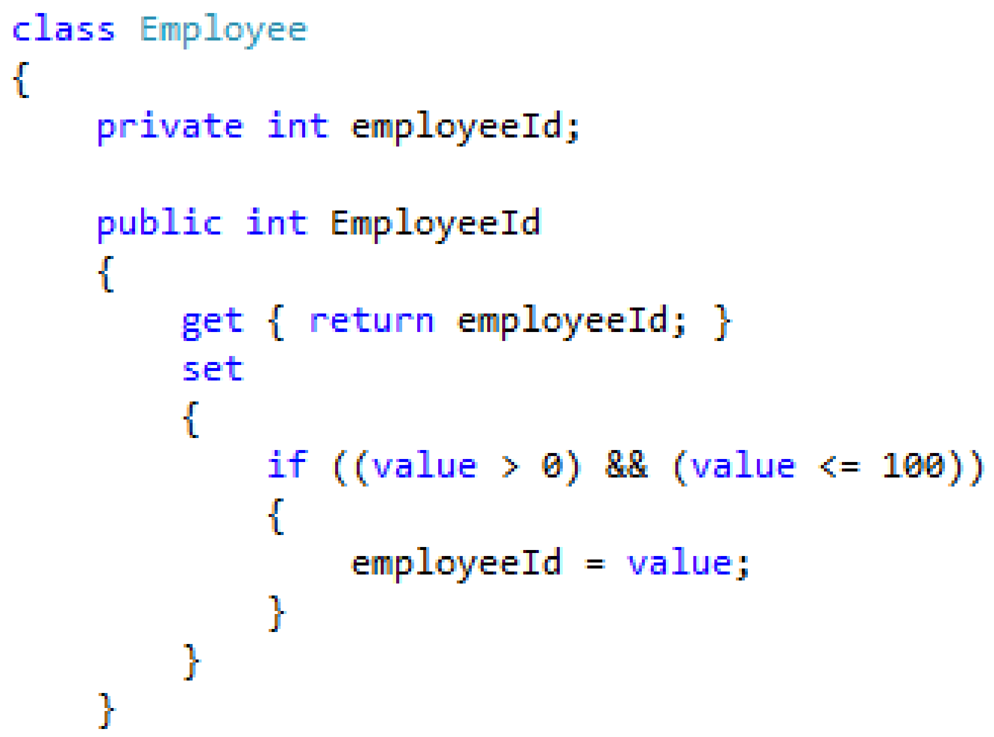
Working With DateTime

- Creating DateTime Object
- Using DateTime Methods
- Using DateTime Properties
- Using TimeSpan

Demo

Properties

- A property combines features of fields and methods
- Automatic properties are used when no need to validate data.



```
class Employee
{
    private int employeeId;

    public int EmployeeId
    {
        get { return employeeId; }
        set
        {
            if ((value > 0) && (value <= 100))
            {
                employeeId = value;
            }
        }
    }
```

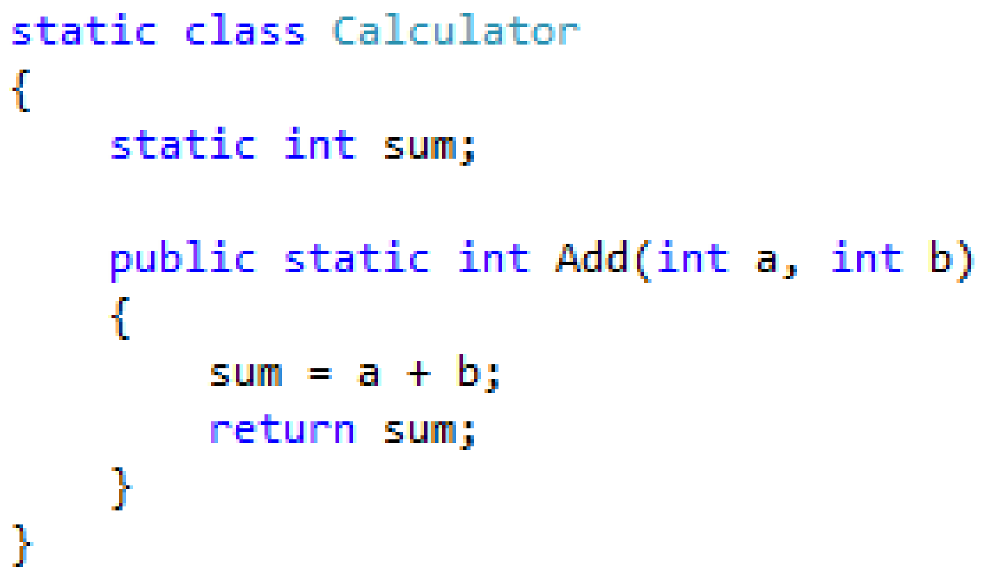
Access Modifiers

- Public
  - Accessible from any code either in the same assembly or by another assembly
  - Default value for the members of an interface
- Private
  - Accessible only from the same class
  - Default value for members of a class
- Protected
  - Accessible only from the same class or child classes
- Internal
  - Accessible from only the same assembly
  - Default value for any class
- Protected Internal
  - Accessible from the same assembly or from child classes of another assembly.

Demo

Static Classes and Members

- Static classes contain only static members
- Static classes can not be instantiated and inherited
- Static members are accessed using class name
- Only one copy of static member exists.



```
static class Calculator
{
    static int sum;

    public static int Add(int a, int b)
    {
        sum = a + b;
        return sum;
    }
}
```

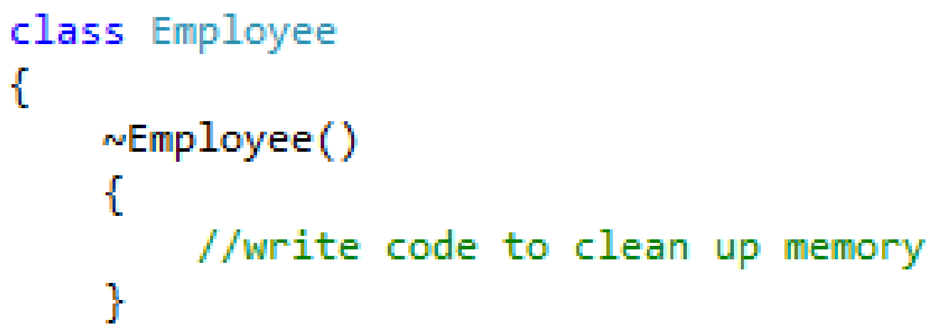
Constructors

- A special method in a class
- Called automatically when you instantiate a class
- Used to assign default values to fields
- Can be instance, private or static.

```
class Employee
{
    private int _employeeId;
    private string _name;
    //default constructor
    Employee()
    {
        _employeeId = 0;
    }
    //constructor with parameters
    Employee(int empId, string name)
    {
        _employeeId = empId;
        _name = name;
    }
```

Destructors

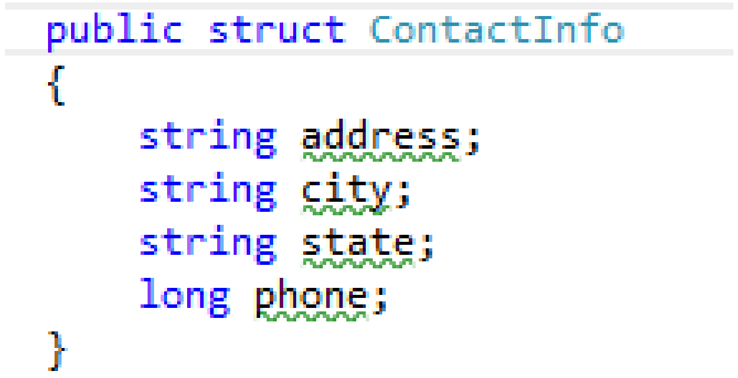
- A destructor is used to clean up memory
- A class can have only one destructor
- A destructor does not take any modifiers and parameters
- A destructor is called when an object is eligible for destruction
- A call to destructor is determined by Garbage Collector.



```
class Employee
{
    ~Employee()
    {
        //write code to clean up memory
    }
```

Structs

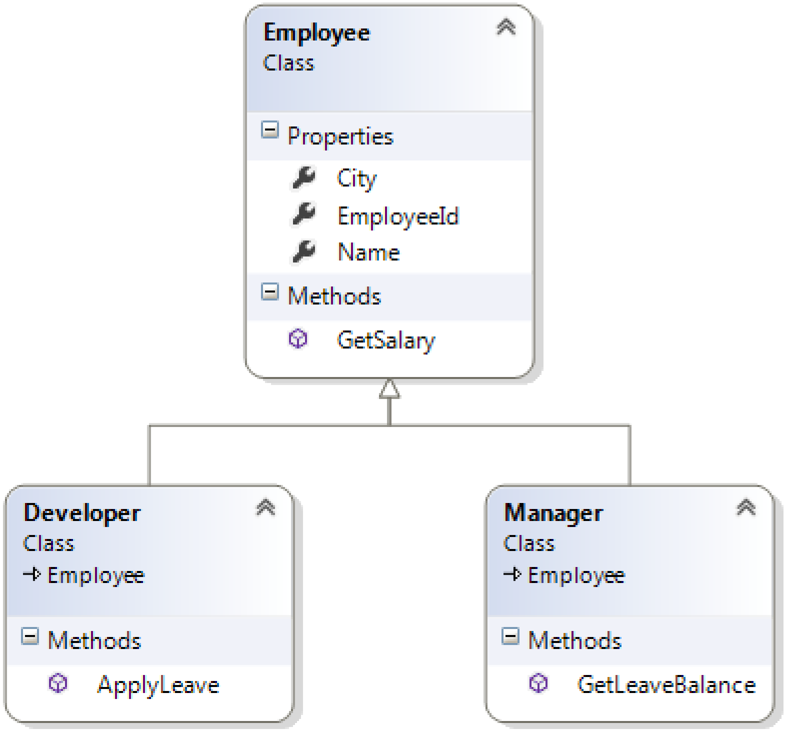
- Structs are very similar to classes
- Supports limited features than classes
- Structs are value types and do not support
  - Default constructor but only parameterized constructors
  - Can't inherit from a struct or class
  - Can't initialize fields unless they are const or static
- Structs can implement interfaces.



```
public struct ContactInfo
{
    string address;
    string city;
    string state;
    long phone;
}
```

Inheritance

- Derived class inherits all members from base class except constructors and destructors
- Derived class may reuse, extend or modify behavior of base class.



Polymorphism

- It’s a Greek word means “Many-Shaped”
- There are two aspects of polymorphism
  - Method Overloading
  - Method Overriding

- To achieve polymorphism we use
  - Virtual methods
  - Override virtual method in child class
  - The “new” keyword if used hides base class method
  - The “base” keyword can be used to call base class method from derived class

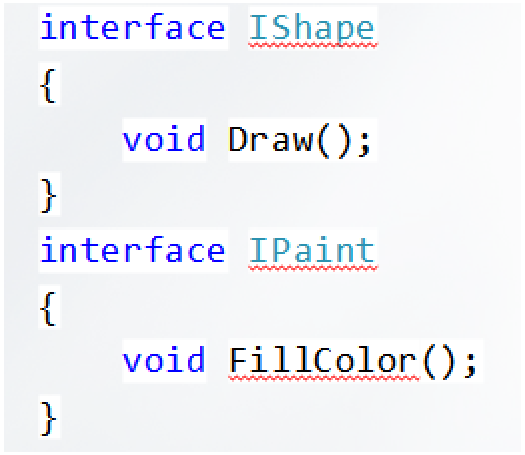
- The “sealed” keyword prevents further inheritance
- The “abstract” keyword ideally define a base class.

Interfaces

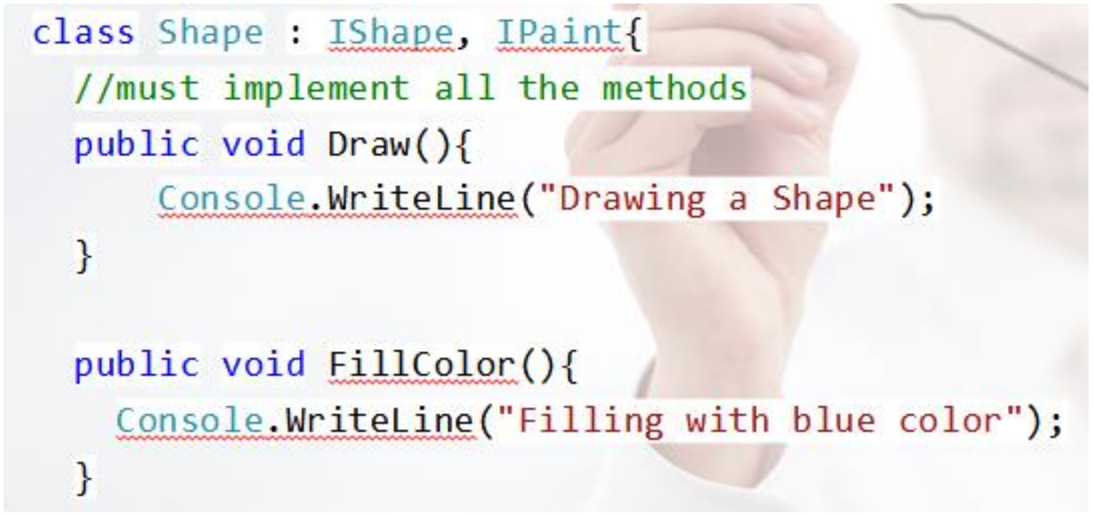
- An interface can contain only definition of related behavior.
- A class or struct can implement the interface.
- An interface includes only method definition not implementation.
- Interfaces can contain methods, properties, indexers and events.
- An interface can’t contain constants, fields, operators, instance constructors, destructors or types.
- Interface members can not be static.

Interfaces

- Interface members are by default public and they can't include any access modifiers.
- If a class or struct implement an interface, it must provide implementation of all the members of the interface.



```
interface IShape
{
      void Draw();
}
interface IPaint
{
      void FillColor();
}
```



```
class Shape : IShape, IPaint{
   //must implement all the methods
   public void Draw(){
         Console.WriteLine("Drawing a Shape");
   }

   public void FillColor(){
      Console.WriteLine("Filling with blue color");
   }
```

Demo

Summary

- Classes and objects
- Methods and properties
- Access modifiers
- Static vs. instance
- Inheritance and polymorphism

Bibliography, Important Links

- https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx
- https://msdn.microsoft.com/en-us/library/ff926074.aspx

Delivering Value. Scientifically.

Any Questions?

Thank you!