.NET PROGRAMMING USING C#

UNIT I

Jalpa Poriya
Indus University

Index

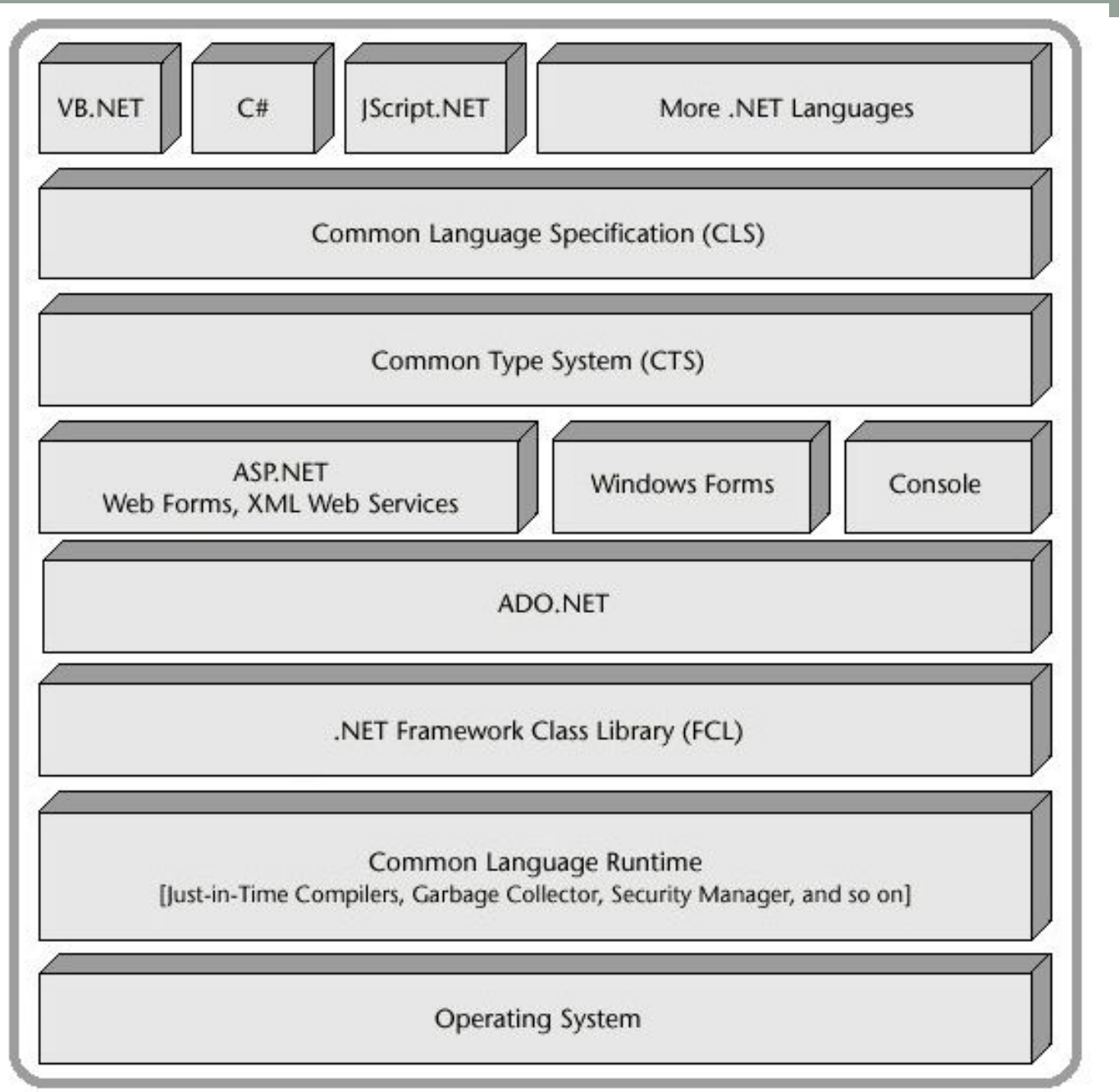
Overview of ASP.NET

(ASP) Active Server Page (.Net) Network Enabled Technology

- ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites. It allows you to use a full featured programming language such as C# or VB.NET to build web applications easily.
- ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.
- ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.
- The ASP.NET application codes can be written in any of the following languages:
  - C#
  - Visual Basic.Net
  - Jscript
  - J#

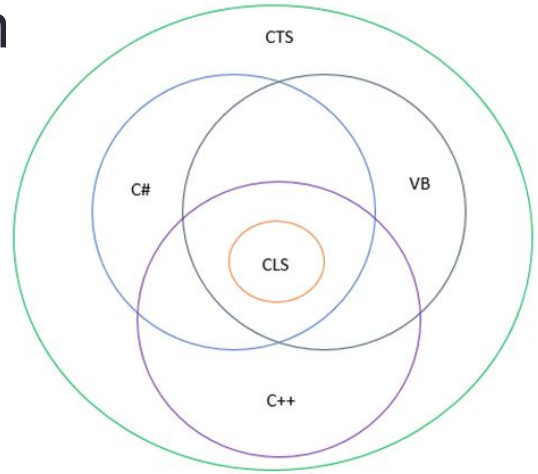
Overview of ASP.NET Framework

- The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services. The .NET Framework consists of the common language runtime and the .NET Framework class library.
- The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, while also strict type safety and other forms of code accuracy that promote security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code.
- The class library is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services


VB.NET
C#
JScript.NET
More .NET Languages
Common Language Specification (CLS)
Common Type System (CTS)
ASP.NET
Web Forms, XML Web Services
Windows Forms
Console
ADO.NET
.NET Framework Class Library (FCL)
Common Language Runtime
[Just-in-Time Compilers, Garbage Collector, Security Manager, and so on]
Operating System

CTS & CLS

- .NET Framework introduces a Common Type System, or CTS.
- The CTS specification defines all possible data types and programming constructs supported by the CLR and how they may or may not interact with each other conforming to the Common Language Infrastructure (CLI) specification.
- Because of this feature, .NET Framework supports the exchange of types and object instances between libraries and applications written usin conforming .NET language.



CTS and CLS

CLR (Common Language Runtime)

- The Common Language Runtime (CLR) serves as the execution engine of .NET Framework. All .NET programs execute under the supervision of the CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.
- The common language runtime is the foundation of the .NET Framework.
- You can think of the runtime as an agent that manages code at execution time, providing core services such as
- Memory management,
- Thread management
- Strict type safety
- Security
- Robustness.

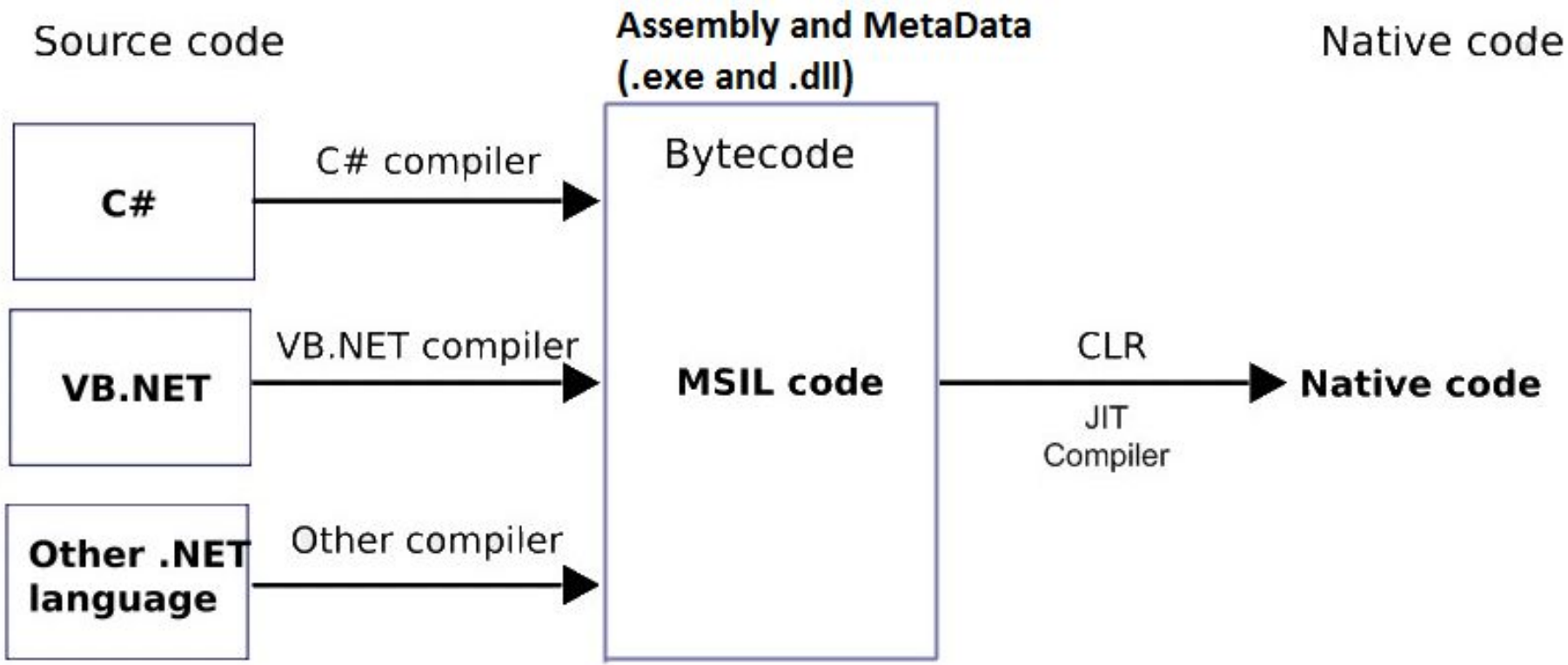
BCL / FCL / Namespaces

- The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using .NET Framework.
- The BCL provides classes that encapsulate a number of common functions, including file reading and writing, graphic rendering, database interaction, XML document manipulation, and so on. It consists of classes, interfaces of reusable types that integrate with CLR (Common Language Runtime).

BCL / FCL / Namespaces

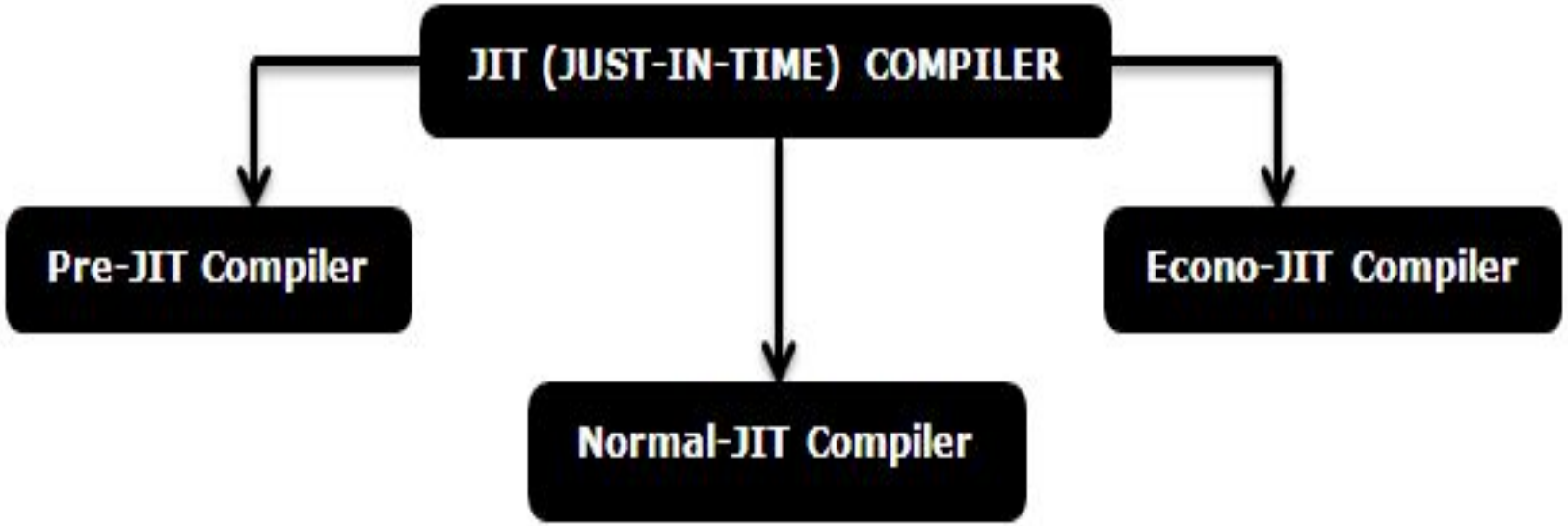
- The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using .NET Framework.
- The BCL provides classes that encapsulate a number of common functions like
  - file reading and writing
  - graphic rendering
  - database interaction
  - XML document manipulation
  - And many more
- It consists of classes, interfaces of reusable types that integrate with CLR.

JIT (JUST-IN-TIME) COMPILER:



JIT Types

- **Pre-JIT COMPILER**
- **Econo-JIT COMPILER**
- **Normal-JIT COMPILER**



Pre-JIT COMPILER

- Pre-JIT compiles complete source code into native code in a single compilation cycle.
- This is done at the time of deployment of the application.

Econo-JIT COMPILER

- Econo-JIT compiles only those methods that are called at runtime.
- However, these compiled methods are removed when they are not required.
-

Normal-JIT COMPILER

- Normal-JIT compiles only those methods that are called at runtime.
- These methods are compiled the first time they are called, and then they are stored in cache.
- When the same methods are called again, the compiled code from cache is used for execution.

Introduction of C#
First Program : Hello World

```
using System;
class Program
 {
        static void Main(string[] args)
        {
            //statements
            //….
            //….
            //statements
        }
 }
```

Variables

- Variable are concerned with the storage of data.
- Variable come in different flavors known as *types*.
- You can assign a valid data to a variable as per its data type.

- **Syntax of variable with c#**

  <type> <name>;
- Example

```
int num;
num = 10;
```

or

```
int num = 10;
```

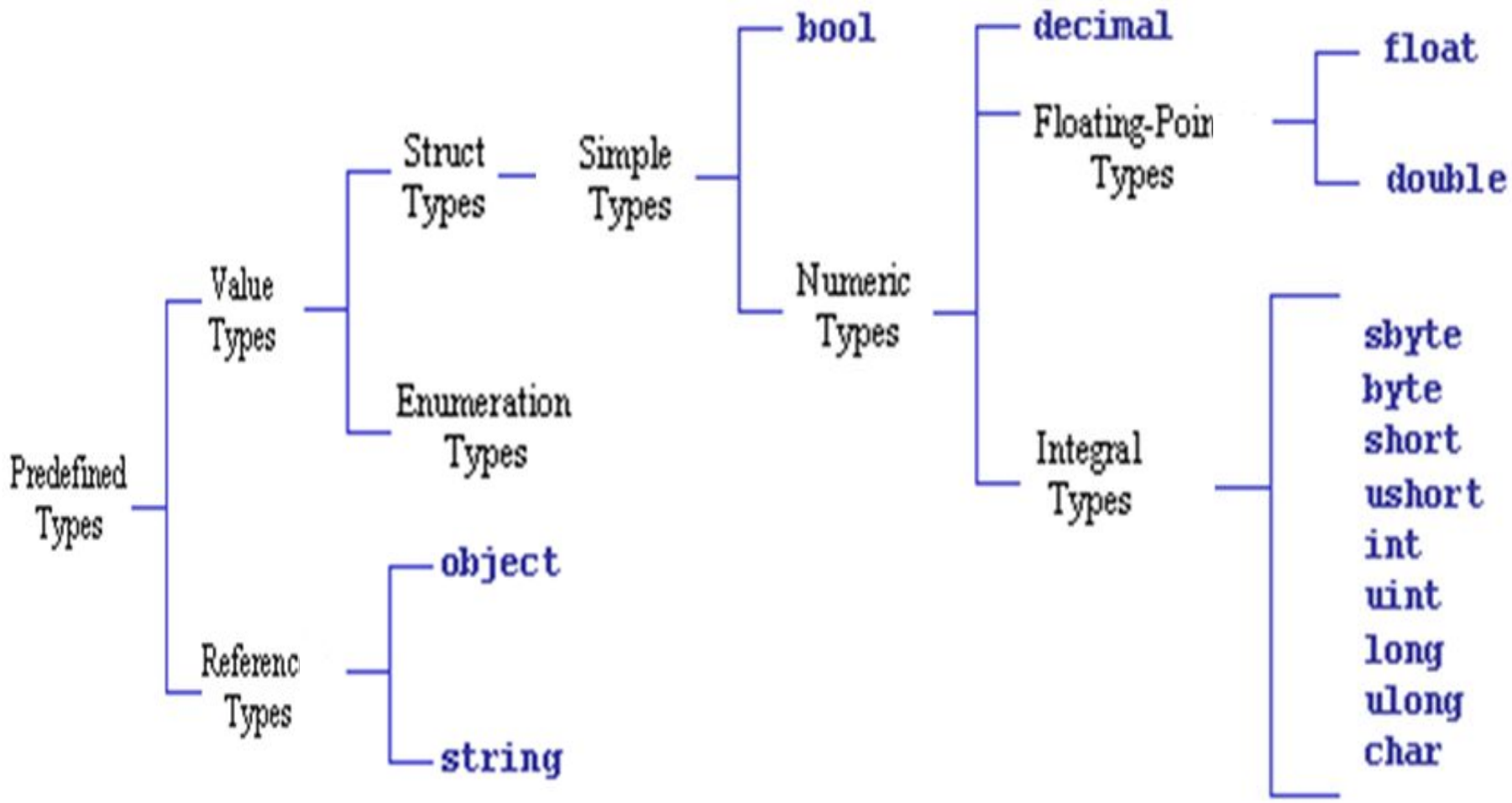
Data Types

- Data types are especially important in C# because it is a strongly typed language.
- There is no concept of a “typeless” variable in C#

- C# lets you work with two types of data

1. Value Type
2. Reference Type

Data Type Tree



Example

```
int a = 10;
uint ua = 10U;
char ans = ‘a’;
string name = “Indus University”;
float pi = 3.14f;
```

Program Control Statements

- Decision statements
  - **The if Statement**
  - **The switch Statement**
- Loop statements
  - **The for Loop**
  - **The while Loop**
  - **The do-while Loop**
  - **The foreach Loop**

The if-else-if Ladder

```
if(condition)
{   statements; }
else if(condition)
{   statement;   }
else
{   statement;   }
```

The switch Statement

```
switch(expression)
{
        case constant1:
            statement sequence
            break;
        default:
            statement sequence
            break;
}
```

The for Loop

Syntax:

```
for(initialization; condition; iteration)
{
        statement;
}
```

The while Loop

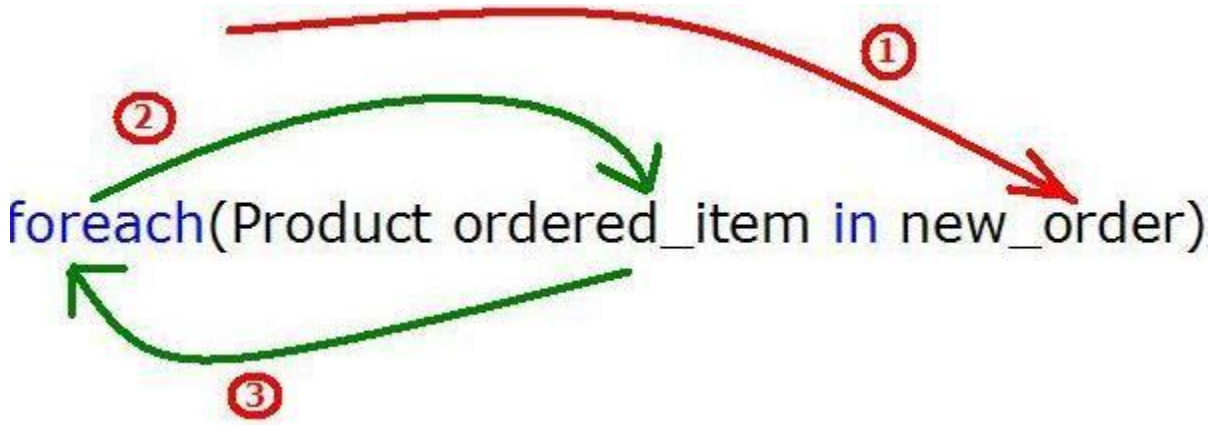
- **Syntax**
- **The while loop**
- while(*condition) statement;*

- **The do-while Loop**

```
do {
        statements;
} while(condition);
```

The foreach Loop

- The **foreach loop is used to cycle through the elements of a *collection.***
- A collection is a group of objects. C# defines several types of collections, of which one is an array.
- foreach(*type loopvar in collection) statement;*



OOP with C#

- Class
- Object
- Inheritance
- Polymorphism (overloading , overriding)

Class

- A class is like a blueprint.
- It defines the data and behavior of a type.
- classes support *inheritance*, a fundamental characteristic of object-oriented programming.

- General syntax of class

```
<access specifier> class <class_name>
{
   // member variables
    <access specifier> <data type> variable1;
   // member methods
   <access specifier> <return type> method1(parameter_list) { // method
     body }
}
```

Object

- Object is an instance of class.
- Syntax:

  <class_name> <object_name> = new <class_name>();

Access Modifiers

- Public
  - There are no restrictions on accessing public members.
- Private
  - Private members are accessible only within the body of the class or the struct in which they are declared.
- Protected
  - A protected member is accessible within its class and by derived class instances.
- Internal
  - Internal types or members are accessible only within files in the same assembly.
- Protected internal
  - Both internal accessibility (all parts of this program can use the member) and protected accessibility (all derived classes can use the member).

Constructors in C#

- A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.
- A constructor will have exact same name as the class and it does not have any return type.
- if you need a constructor can have parameters. Such constructors are called **parameterized constructors**.

Syntax

```
class <class_name>
{
    <constructor>(<parameters>)
    {
    }
}
```

Destructors in C#

- A **destructor** is a special member function of a class that is executed whenever an object of its class goes out of scope.
- A **destructor** will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters.

Destructor

```
class <class_name>
{
    <constructor>(<parameters>)
    {
    }
    ~<destructor()
    {
    }
}
```

Inheritance

- **Single Inheritance**
- **Multilevel Inheritance**
- **Hierarchical Inheritance**
- **Hybrid Inheritance**

Inheriting a class

- Inheritance provides a powerful and natural mechanism for organizing and structuring software program
- A class that is inherited is called a *base class.*
- *The class that does* the inheriting is called a *derived class.*
- Syntax*:*

```
class derived-class-name : base-class-name
{
    // body of class
}
```

- **C# does not support multiple inheritance.**

Overloading a method

- *Overloading* is what happens when you have two methods with the same name but different signatures.
- The return type of a method is *not* considered to be part of a method's signature.
- Syntax :

```
class a
{
    public void Message()
    {   }
    public void Message(string msg)
    {   }
}
```

Overriding a method

- *Overriding* is what happens when you have two methods with the same name same signatures but in different class. That class must be in inheritance.

```
class a
{
        public void Message()
        {     }
}
class b:a
{
        public void Message()
        {     }
}
```

Error Handling
Exception using try - catch - finally

- An exception is a problem that arises during the execution of a program.
- A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.
- Exceptions provide a way to transfer control from one part of a program to another.

Keywords

- try :
- A block which has possibility of run time error.
- catch :
- Programmer can handle exception through catch block.
- One or more catch is possible.
- finally :
- Finally block run always
- throw :
- It is possible to throw an exception manually by using the **throw** statement.

```
try
{
        intTemp = 100/x;

}
catch(DivideByZeroException e)
{
         Console.WriteLine("Division by Zero occurs");
        //or
        //Console.WriteLine(e.Message);
}
```

- ArrayList
- Hashtable
- BitArray
- Stack
- Queue
- SortedList

ArrayList Collection

- An ArrayList is a dynamic array and implements the IList interface. Each element is accessible using the indexing operator.
- While the traditional array has a fixed number of elements, in the case of Array List, elements can be added or removed at run time.

ArrayList

- **Why ArrayList**
  - Drawback of Array…
  - Collection of similar data types
  - Fix size
  - Fix dimension
  - Not like a List.
- **ArrayList over Array**
  - It is a dynamic array.
  - An ArrayList automatically expands as data is added.
  - ArrayList can hold data of multiple data types.

- Methods
  - **Add** : int Add(Object value)
  - **AddRange** : void AddRange(ICollection c)
  - **Clear** : void Clear()
  - **Contains** : bool Contains(Object item)
  - **Insert** : void Insert(int index, Object value)
  - **Remove** : void Remove(Object obj)
  - **RemoveAt** : void RemoveAt(int index)
  - **Sort** : void Sort()
  - **TrimToSize()**

Garbage Collection in .Net framework

- The Garbage collection is very important technique in the .Net framework to free the unused managed code objects in the memory and free the space to the process.
- The garbage collection (GC) is new feature in Microsoft .net framework.
- When we have a class that represents an object in the runtime that allocates a memory space in the heap memory.
- All the behavior of that objects can be done in the allotted memory in the heap.
- Once the activities related to that object is get finished then it will be there as unused space in the memory.
- It is very important part in the .Net framework. Now it handles this object clear in the memory implicitly. It overcomes the existing explicit unused memory space clearance.

What is a .Net Assembly?

- The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file.
- All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.
- There are two kind of assemblies in .NET;
  - Private
  - Shared

- Private Assembly:

1)Private assembly can be used by only one application.

2)Private assembly will be stored in the specific application's directory or sub-directory.

3)Private assembly doesn't have any version constraint.

- Public Assembly:

1)Public assembly can be used by multiple applications.

2)Public assembly is stored in Global Assembly Cache.

3)Public assembly is also termed as shared assembly.