.NET FRAMEWORK AND C# (MCA-137)

UNIT 1

Introduction to .NET Framework

.NET – What Is It?

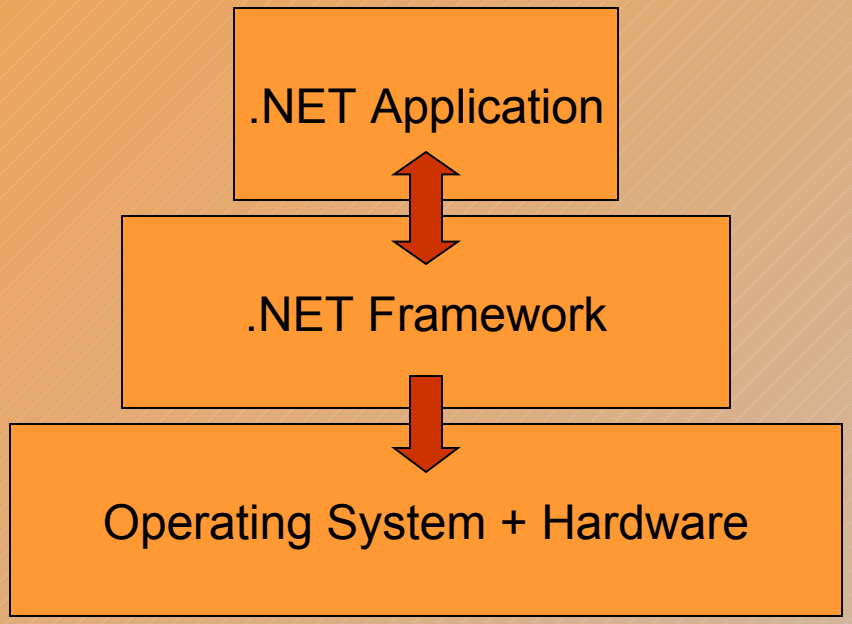
- Software platform
- Language neutral
- In other words:

  .NET is not a language (Runtime and a library for writing and executing written programs in any compliant language)

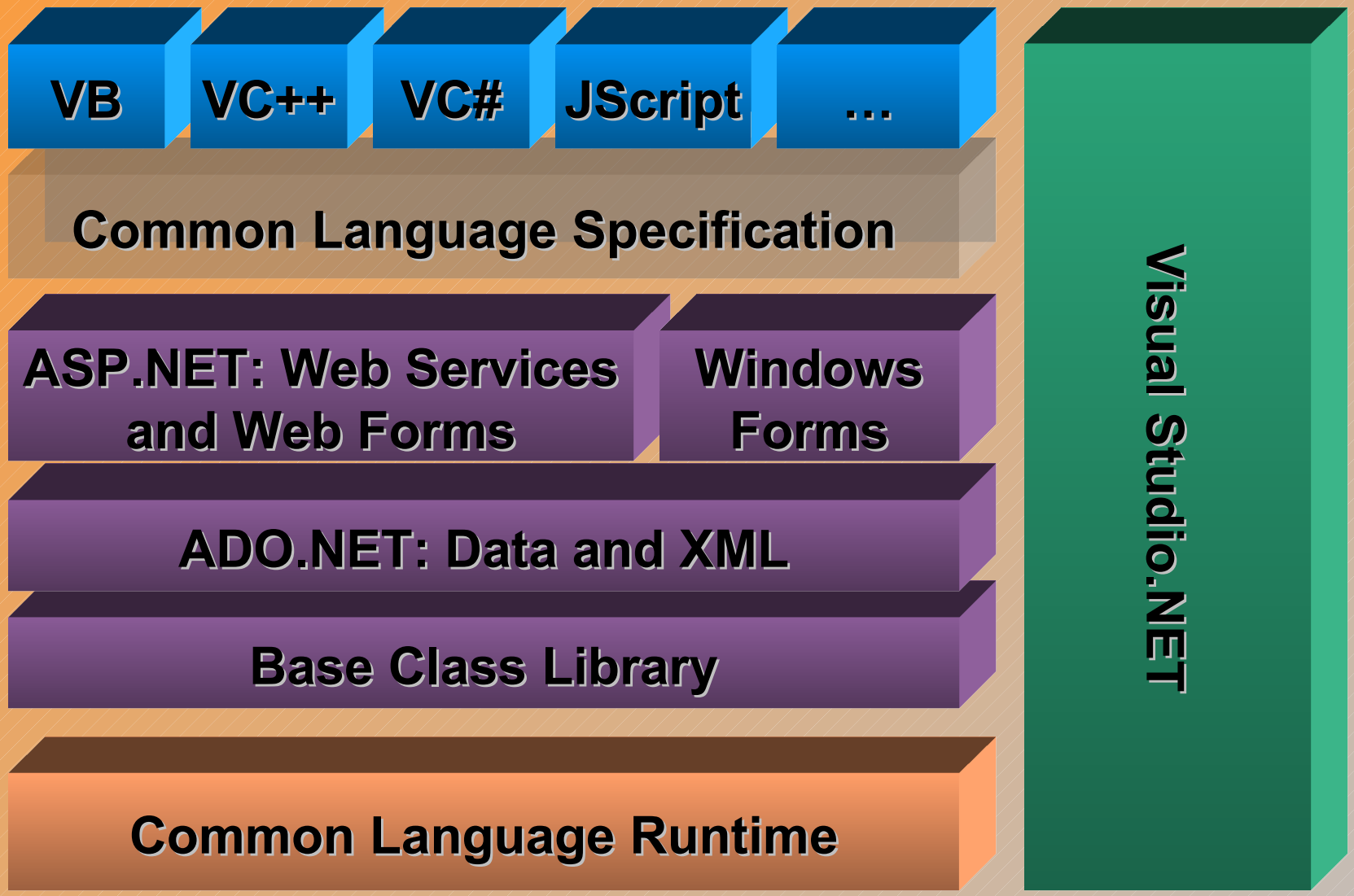
What Is .NET

- .Net is a new framework for developing web-based and windows-based applications within the Microsoft environment.
- The framework offers a fundamental shift in Microsoft strategy: it moves application development from client-centric to server-centric.

.NET – What Is It?



Framework, Languages, And Tools



The .NET Framework

.NET Framework Services

- Common Language Runtime
- Windows® Forms
- ASP.NET
  - Web Forms
  - Web Services
- ADO.NET, evolution of ADO
- Visual Studio.NET

Common Language Runtime (CLR)

- CLR works like a virtual machine in executing all languages.
- All .NET languages must obey the rules and standards imposed by CLR. Examples:
  - Object declaration, creation and use
  - Data types,language libraries
  - Error and exception handling
  - Interactive Development Environment (IDE)

Common Language Runtime

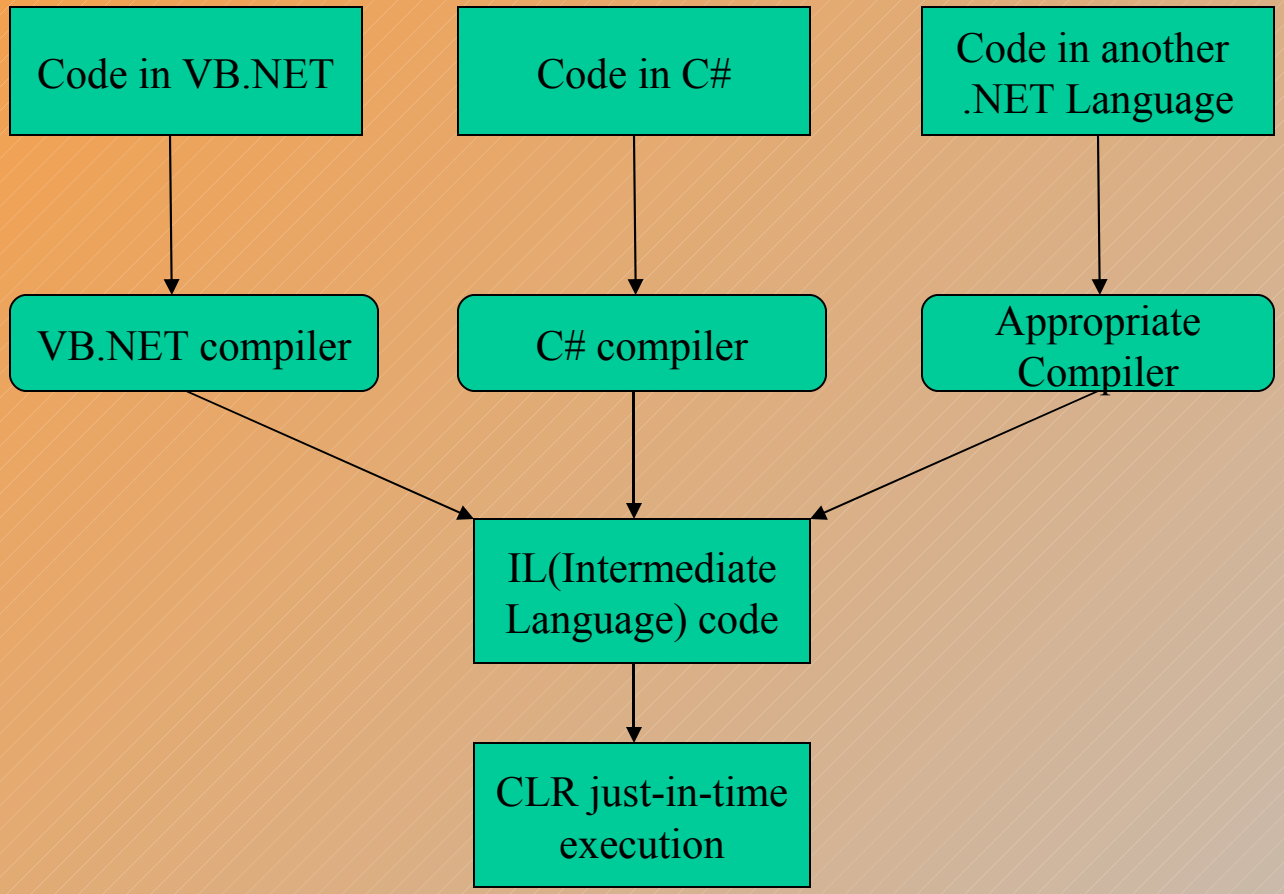
- Development
  - Mixed language applications
    - Common Language Specification (CLS)
    - Common Type System (CTS)
    - Standard class framework
    - Automatic memory management
  - Consistent error handling and safer execution
  - Potentially multi-platform
- Deployment
  - Removal of registration dependency
  - Safety – fewer versioning problems

Common Language Runtime

Multiple Language Support

- CTS is a rich type system built into the CLR
  - Implements various types (int, double, etc)
  - And operations on those types
- CLS is a set of specifications that language and library designers need to follow
  - This will ensure interoperability between languages

Compilation in .NET



Intermediate Language (IL)

- .NET languages are not compiled to machine code. They are compiled to an Intermediate Language (IL).

- CLR accepts the IL code and recompiles it to machine code. The recompilation is just-in-time (JIT) meaning it is done as soon as a function or subroutine is called.

- The JIT code stays in memory for subsequent calls. In cases where there is not enough memory it is discarded thus making JIT process interpretive.

Languages

- Languages provided by MS
  - VB, C++, C#, J#, JScript
- Third-parties are building
  - APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk…

Windows Forms

- Framework for Building Rich Clients
  - RAD (Rapid Application Development)
  - Rich set of controls
  - Data aware
  - ActiveX® Support
  - Licensing
  - Accessibility
  - Printing support
  - Unicode support
  - UI inheritance

ASP.NET

- ASP.NET,the platform services that allow to program Web Applications and Web Services in any .NET language

- ASP.NET Uses .NET languages to generate HTML pages. HTML page is targeted to the capabilities of the requesting Browser

- ASP.NET “Program” is compiled into a .NET class and cached the first time it is called. All subsequent calls use the cached version.

ASP.NET

- Logical Evolution of ASP
  - Supports multiple languages
  - Improved performance
  - Control-based, event-driven execution model
  - More productive
  - Cleanly encapsulated functionality

ASP.NET Web Forms

- Allows clean cut code
  - Code-behind Web Forms
- Easier for tools to generate
- Code within is compiled then executed
- Improved handling of state information
- Support for ASP.NET server controls
  - Data validation
  - Data bound grids

ASP.NET Web Services

- A technical definition
  - “A programmable application component accessible via standard Web protocols”

Web Services

- It is just an application…
- …that exposes its features and capabilities over the network…
- …using XML…
- …to allow for the creation of powerful new applications that are more than the sum of their parts…

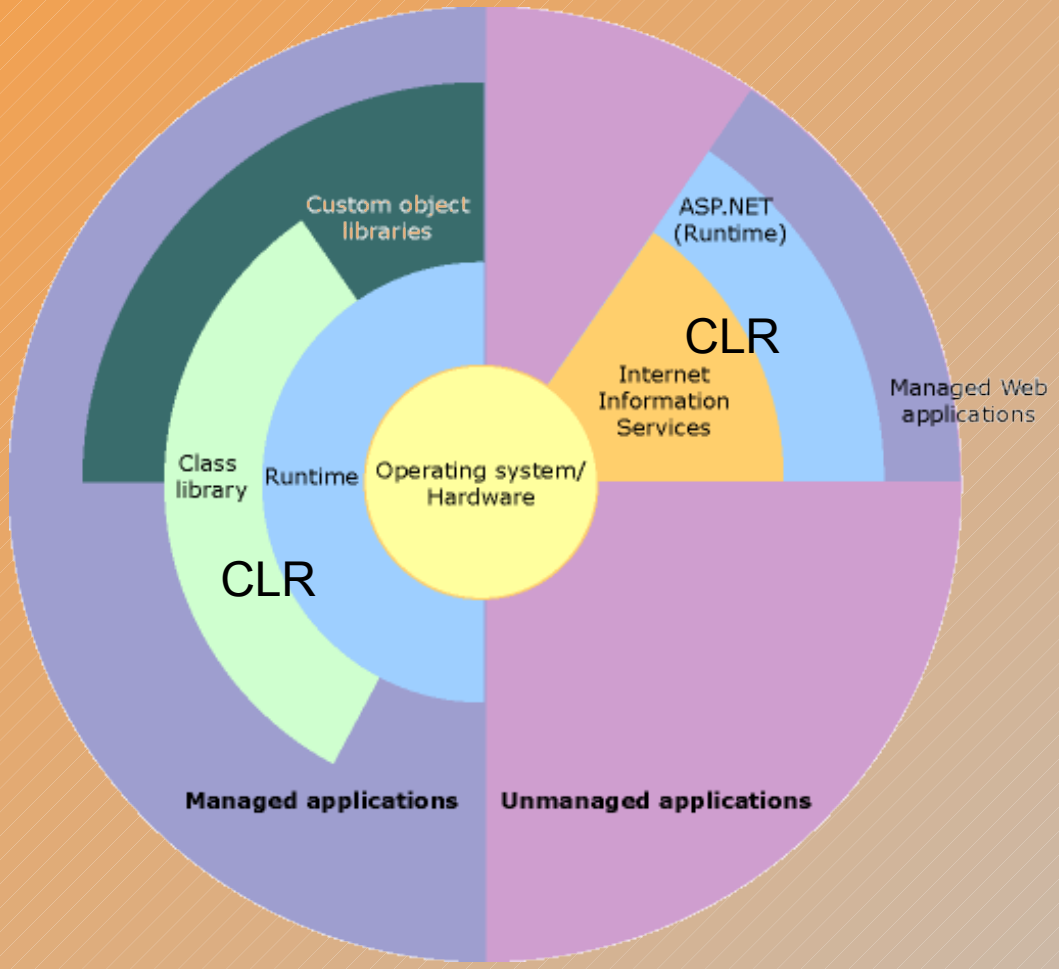
ADO.NET (Data and XML)

- New objects (e.g., DataSets)
- Separates connected / disconnected issues
- Language neutral data access
- Uses same types as CLR
- Great support for XML

Visual Studio.NET

- Development tool that contains a rich set of productivity and debugging features

.NET – Hierarchy, Another View



Summary

- The .NET Framework
  - Dramatically simplifies development and deployment
  - Provides robust and secure execution environment
  - Supports multiple programming languages

THE ORIGIN OF .NET TECHNOLOGY

The Origin of .Net Technology

- The current technology of .NET has gone through 3 significant phases of development:
  1. OLE technology
  2. COM technology
  3. .NET technology

Generations of Component Model

Phase 1 early 1990's	OLE technology	**Interprocess Communication**
Phase 2 1995	COM technology	**Intermodule Communication**
Phase 3 late1990's	.NET technology	**Intersite Communication**

OLE Technology

- Object linking and embedding technology was developed by Microsoft in early 1990's to easy interprocess communications
- OLE provides the support to do the following:
  1. To embed documents from one application to another
  2. To enable one application to manipulate objects located in another application

OLE Technology (Contd.)

- This enabled users to develop application which required inter-operability between various products such as MS Word and MS Excel

COM Technology

- Component Object Model technology was developed by Microsoft in 1995 to easy intermodule communications
- The old approach was used for developing software but when program became too large and complex, this approach to a number of problems in terms of maintainability and software testing to overcome thus Microsoft introduced the component-based model for developing software programs.

COM Technology (Contd.)

- In component-based approach a program is broken down into a number of independent components where each one offers a particular service.
- Each component can be developed and tested independently and then migrated into the main system. This technology was called component object model (COM) and the software built using this was called component ware

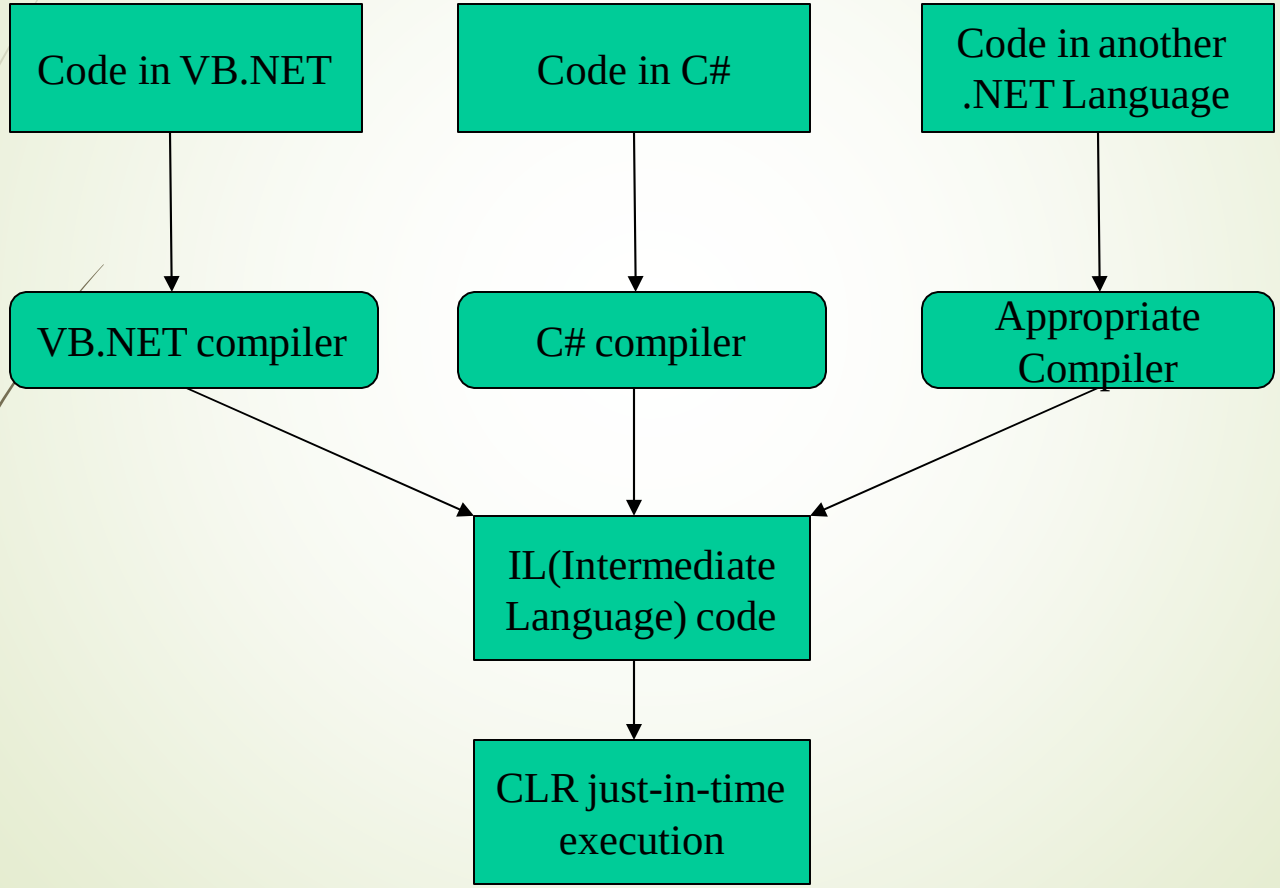
.NET Technology

- Provide new level of interoperability compared to COM technology intermodule communication in COM is replaced by IL(Intermediate language) by Microsoft in .NET technology.
- Various .NET Compilers enforce inter-operability by compiling code into IL which is automatically compatible with other IL modules.

.NET Technology (Contd.)

- IL allows for true cross-language integration.
- .NET includes a host of other technologies and tools that will enable us to develop web-based application easily

Compilation in .NET



COMMON LANGUAGE RUNTIME (CLR)

Common Language Runtime (CLR) in C#

- CLR is the basic and Virtual Machine component of the .NET Framework.
- It is the run-time enviornment in the .NET Framework that runs the codes and helps in making the development process easier by providing the various services.
- Basically, it is responsible for managing the execution of .NET programs regardless of any .NET programming language.

Common Language Runtime (CLR) in C# (Contd.)

- Internally, CLR implements the VES(Virtual Execution System) which is defined in the Microsoft's implementation of the CLI(Common Language Infrastructure).
- The code that runs under the Common Language Runtime is termed as the Managed Code.
- In other words, you can say that CLR provides a managed execution environment for the .NET programs by improving the security, including the cross-language integration and a rich set of class libraries etc.

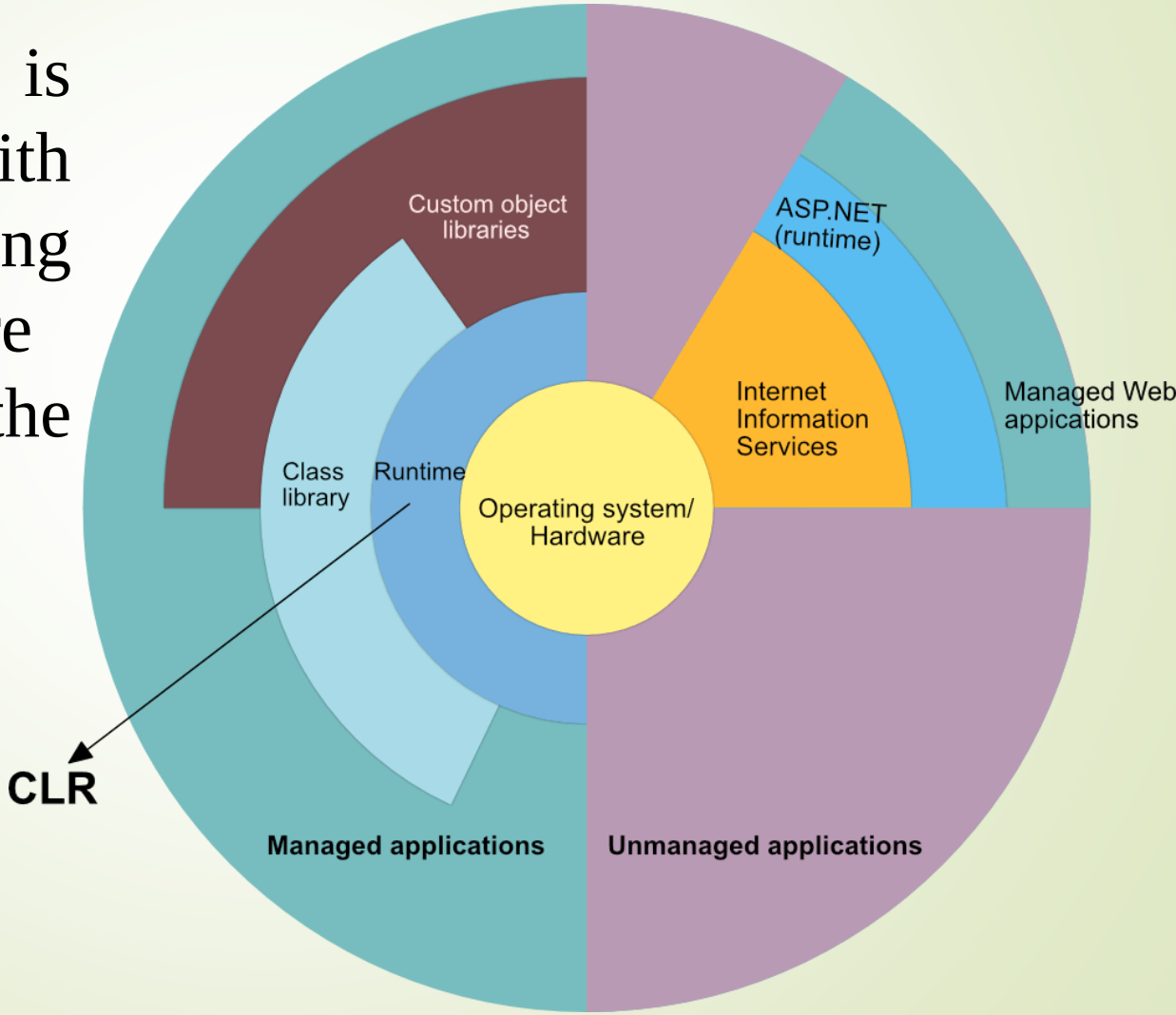
Common Language Runtime (CLR) in C# (Contd.)

- CLR is present in every .NET framework version.

CLR VERSIONS	.NET FRAMEWORK VERSIONS
1.0	1.0
1.1	1.1
2.0	2.0
2.0	3.0
2.0	3.5
4	4
4	4.5(also 4.5.1 & 4.5.2)
4	4.6(also 4.6.1 & 4.6.2)
4	4.7(also 4.7.1 & 4.7.2)

Role of CLR in the execution of a C#

- How CLR is associated with the operating system/hardware along with the class libraries.



Role of CLR in the execution of a C# (Contd.)

- Suppose you have written a C# program and save it in a file which is known as the Source Code.
- Language specific compiler compiles the source code into the MSIL(Microsoft Intermediate Language) which is also know as the CIL(Common Intermediate Language) or IL(Intermediate Language) along with its metadata.
- Metadata includes the all the types, actual implementation of each function of the program.
- MSIL is machine independent code.

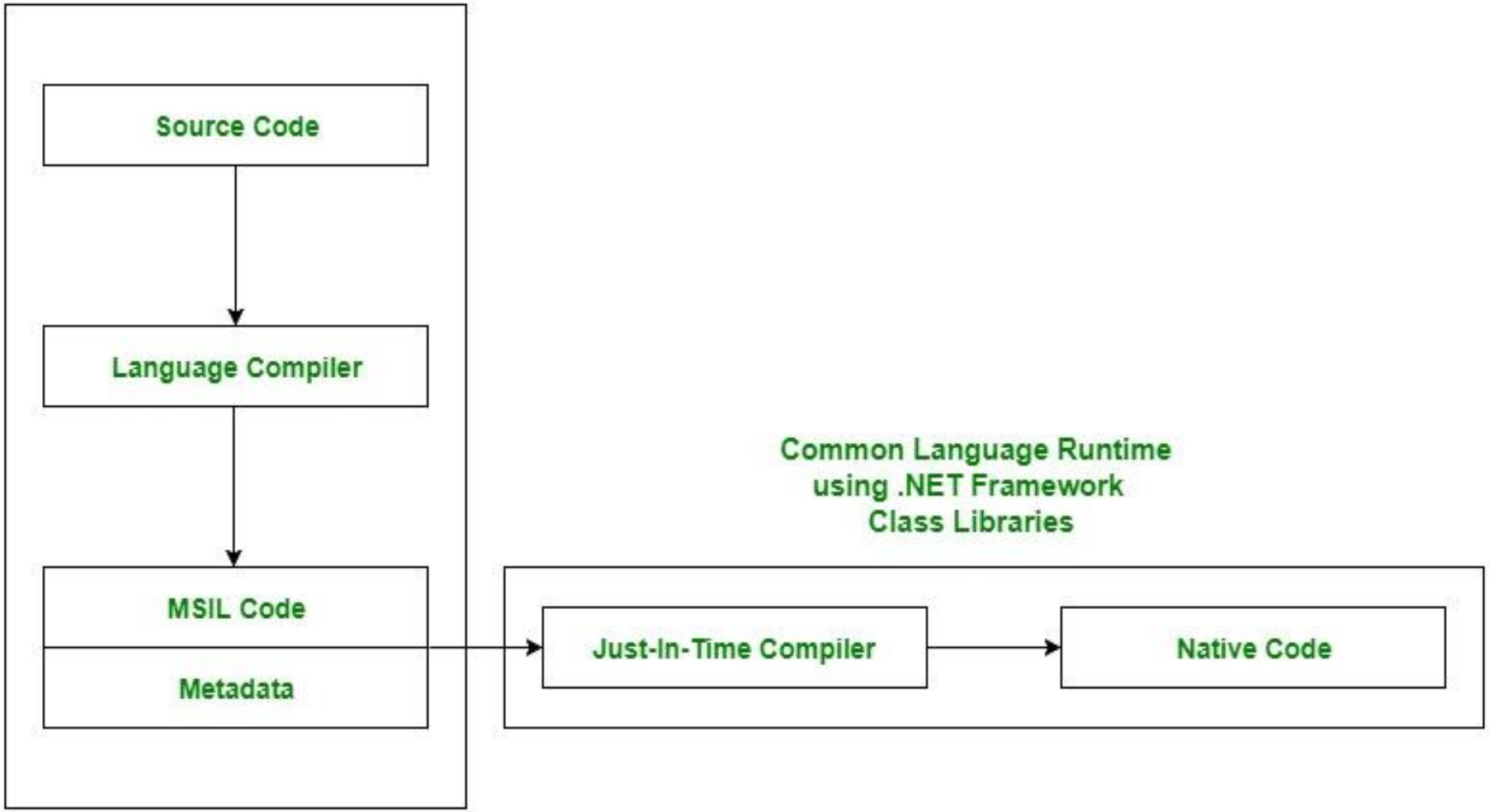
Role of CLR in the execution of a C# (Contd.)

- CLR provides the services and runtime environment to the MSIL code.
- Internally CLR includes the JIT(Just-In-Time) compiler which converts the MSIL code to machine code which further executed by CPU.
- CLR also uses the .NET Framework class libraries.
- As CLR is common so it allows an instance of a class that written in a different language to call a method of the class which written in another language.

Main Components of CLR

- Common Language Specification (CLS)
- Common Type System (CTS)
- Garbage Collection (GC)
- Just In – Time Compiler (JIT)

Main Components of CLR (Contd.)



Main Components of CLR (Contd.)

- Common Language Specification (CLS):

  It is responsible for converting the different .NET programming language syntactical rules and regulations into CLR understandable format.

  Basically, it provides the Language Interoperability. Language Interoperability means to provide the execution support to other programming languages also in .NET framework.

Main Components of CLR (Contd.)

- Common Type System (CTS):

Every programming language has its own data type system, so CTS is responsible for understanding all the data type systems of .NET programming languages and converting them into CLR understandable format which will be a common format.

Main Components of CLR (Contd.)

- Garbage Collector:

  It is used to provide the Automatic Memory Management feature. If there was no garbage collector, programmers would have to write the memory management codes which will be a kind of overhead on programmers.

- JIT(Just In Time Compiler):

  It is responsible for converting the CIL(Common Intermediate Language) into machine code or native code using the Common Language Runtime environment.

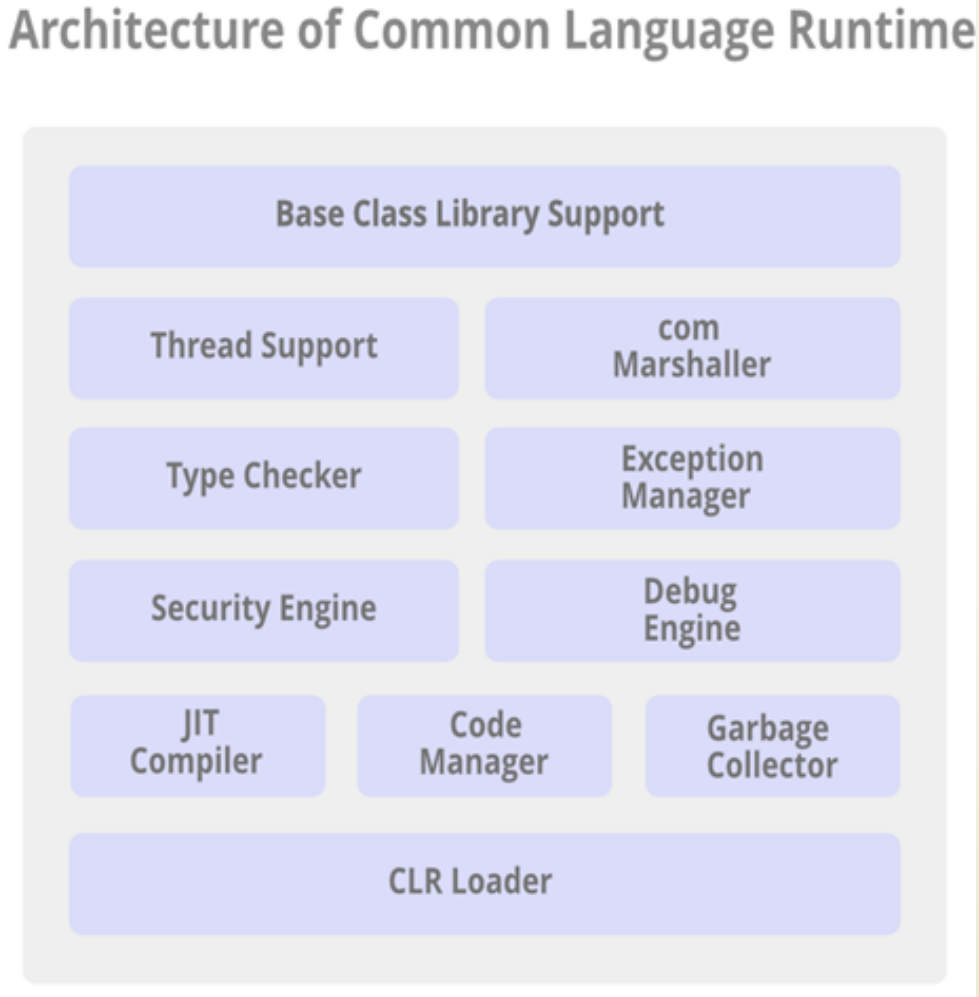
Benefits of CLR:

- It improves the performance by providing a rich interact between programs at run time.
- Enhance portability by removing the need of recompiling a program on any operating system that supports it.
- Security also increases as it analyzes the MSIL instructions whether they are safe or unsafe. Also, the use of delegates in place of function pointers enhance the type safety and security.
- Support automatic memory management with the help of Garbage Collector.

Benefits of CLR: (Contd.)

- Provides cross-language integration because CTS inside CLR provides a common standard that activates the different languages to extend and share each other's libraries.
- Provides support to use the components that developed in other .NET programming languages.
- Provide language, platform, and architecture independence.
- It allows easy creation of scalable and multithreaded applications, as the developer has no need to think about the memory management and security issues.

Architecture of Common Language Runtime (CLR)



Multiple components in the architecture of Common Language Runtime

- **Base Class Library Support:** The Common Language Runtime provides support for the base class library. The BCL contains multiple libraries that provide various features such as Collections, I/O, XML, DataType definitions, etc. for the multiple .NET programming languages.
- **Thread Support:** The CLR provides thread support for managing the parallel execution of multiple threads. The System. Threading class is used as the base class for this.

Multiple components in the architecture of Common Language Runtime(Contd.)

- **COM Marshaller:** Communication with the COM (Component Object Model) component in the .NET application is provided using the COM marshaller. This provides the COM interoperability support.
- **Type Checker:** Type safety is provided by the type checker by using the Common Type System (CTS) and the Common Language Specification (CLS) that are provided in the CLR to verify the types that are used in an application.

Multiple components in the architecture of Common Language Runtime(Contd.)

- **Exception Manager**: The exception manager in the CLR handles the exceptions regardless of the .NET Language that created them. For a particular application, the catch block of the exceptions are executed in case they occur and if there is no catch block then the application is terminated.
- **Security Engine:** The security engine in the CLR handles the security permissions at various levels such as the code level, folder level, and machine level. This is done using the various tools that are provided in the .NET framework.

Multiple components in the architecture of Common Language Runtime(Contd.)

- **Debug Engine:** An application can be debugged during the run-time using the debug engine. There are various ICorDebug interfaces that are used to track the managed code of the application that is being debugged.
- **JIT Compiler:** The JIT compiler in the CLR converts the Microsoft Intermediate Language (MSIL) into the machine code that is specific to the computer environment that the JIT compiler runs on.

Multiple components in the architecture of Common Language Runtime(Contd.)

- **Code Manager:** The code manager in CLR manages the code developed in the .NET framework i.e. the managed code. The managed code is converted to intermediate language by a language-specific compiler and then the intermediate language is converted into the machine code by the Just-In-Time (JIT) compiler.
- **Garbage Collector:** Automatic memory management is made possible using the garbage collector in CLR. The garbage collector automatically releases the memory space after it is no longer required so that it can be reallocated.

Multiple components in the architecture of Common Language Runtime(Contd.)

- **CLR Loader:** Various modules, resources, assemblies, etc. are loaded by the CLR loader. Also, this loader loads the modules on demand if they are actually required so that the program initialization time is faster, and the resources consumed are lesser.

COMMON TYPE SYSTEM (CTS)

Common Type System (CTS)

- The language interoperability, and .NET Class Framework, are not possible without all the language sharing the same data types.
- What this means is that an "int" should mean the same in VB, VC++, C# and all other .NET compliant languages. Same idea follows for all the other data types.
- This is achieved through introduction of Common Type System (CTS).

Common Type System (CTS)(Contd.)

- Common type system (CTS) is an important part of the runtimes support for cross language integration. The common type system performs the following functions:
  - Establishes a framework that enables cross-language integration, type safety, and high-performance code execution.
  - Provides an object-oriented model that supports the complete implementation of many programming languages.

Common Type System (CTS)(Contd.)

- The common type system supports two general categories of types:

1. **Value types**

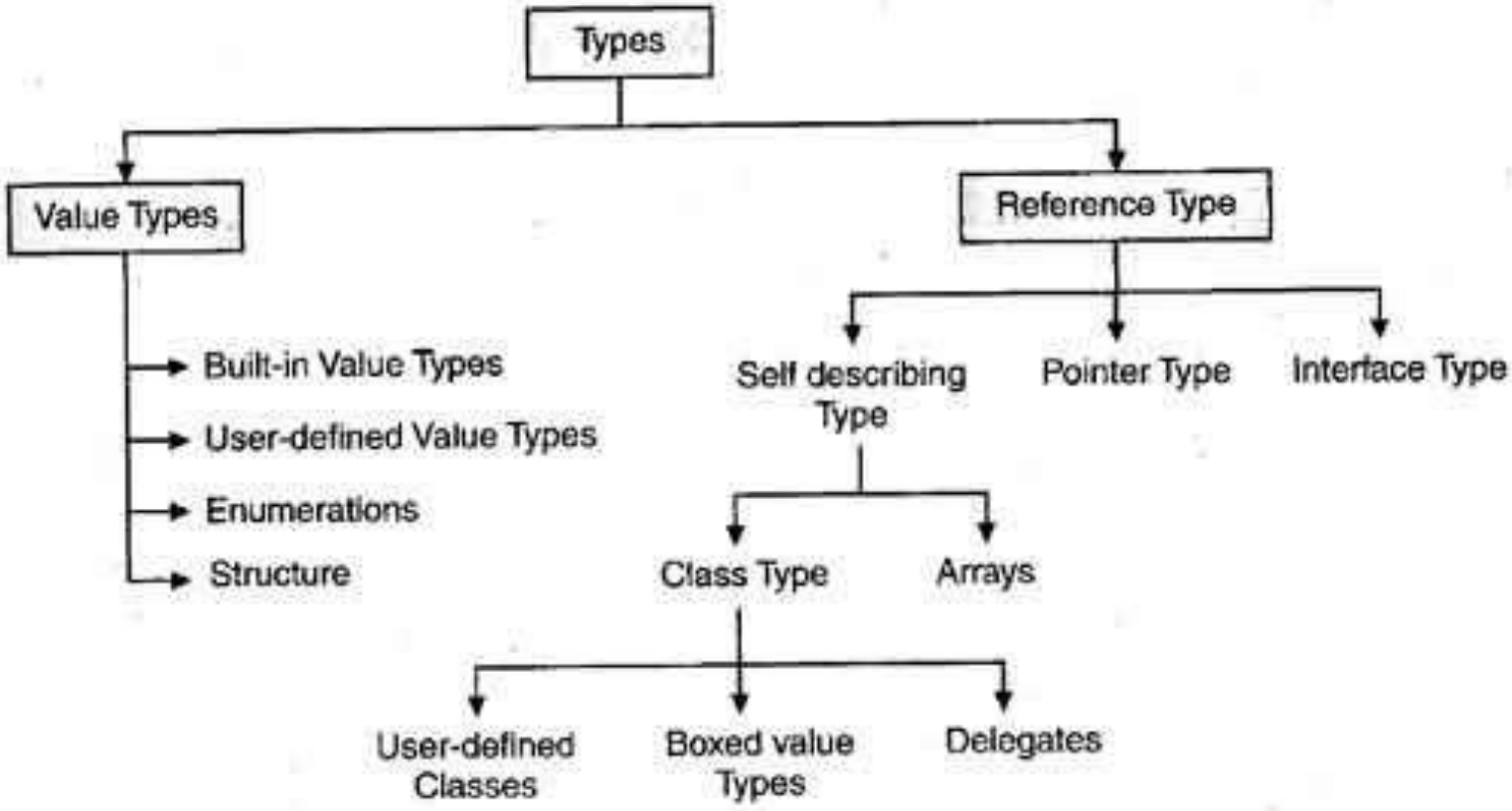
- Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure.
- Value types can be built-in, user-defined or enumerations types.

Common Type System (CTS)(Contd.)

2. **Reference types:**

- Reference types stores a reference to the value's memory address and are allocated on the heap.
- Reference types can be self-describing types, pointers types, or interface types.
- The type of a reference type can be determined from values of self-describing types.
- Self-describing types are further split into arrays and class types are user-defined classes, boxed value types, and delegates.

Common Type System (CTS)(Contd.)

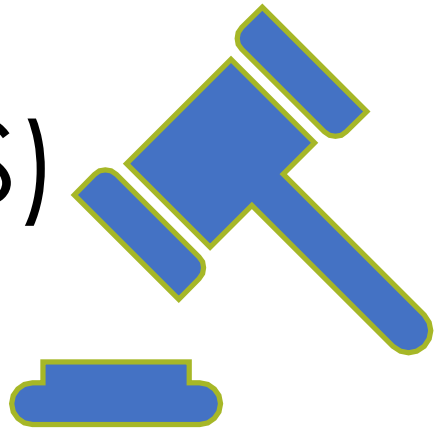


Common type system

Common Type System (CTS)(Contd.)

CTS Type Name	C# Alias	Description
System.Object	Object	Base class for all CTS types
System.String	String	String
System.Sbyte	Sbyte	Signed 8-bit byte
System.Byte	Byte	Unsigned 8-bit byte
System.Int16	Short	Signed 16-bit value
System.UInt16	Ushort	Unsigned 16-bit value
System.Int32	Int	Signed 32-bit value
System.UInt32	Uint	Unsigned 32-bit value
System.Int64	Long	Signed 64-bit value
System.UInt64	Ulong	Unsigned 64-bit value
System.Char	Char	16-bit Unicode character
System.Single	Float	IEEE 32-bit float
System.Double	Double	IEEE 64-bit float
System.Boolean	Bool	Boolean value (true/false)
System.Decimal	Decimal	128-bit data type

COMMON LANGUAGE SPECIFICATION(CLS)



Common Language Specification(CLS)

It is responsible for converting the different .NET programming language syntactical rules and regulations into CLR understandable format. Basically, it provides the Language Interoperability. Language Interoperability means to provide the execution support to other programming languages also in .NET framework.

Common Language Specification(CLS)Contd.)

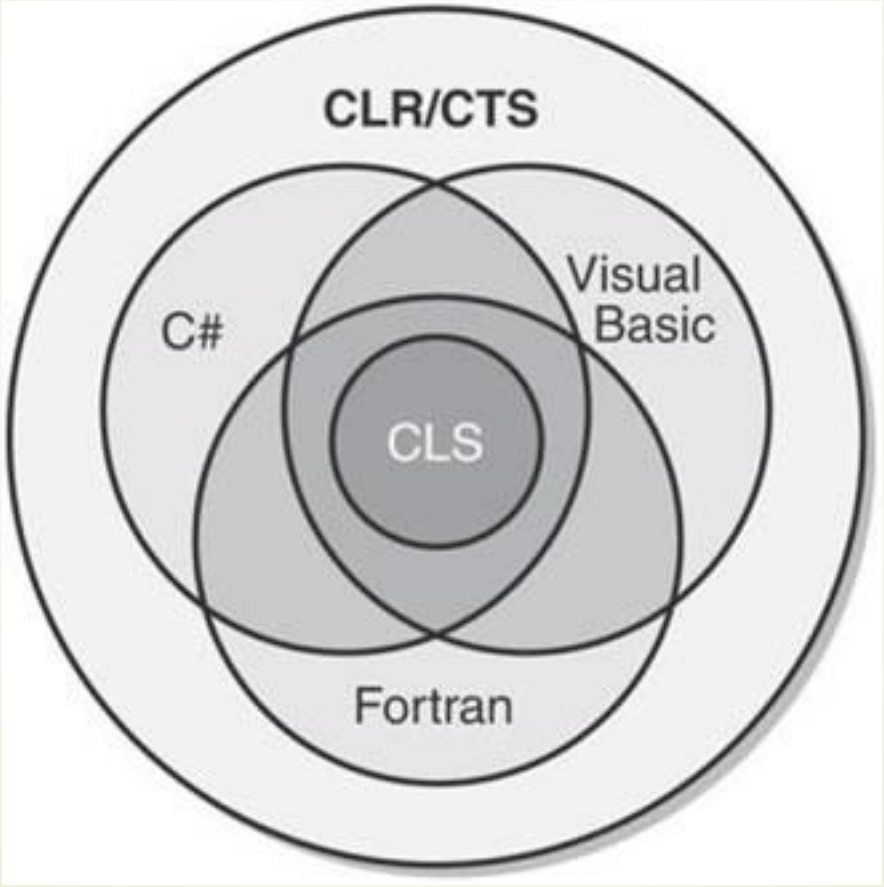
Language Interoperability can be achieved in two ways :

1. **Managed Code:** The MSIL code which is managed by the CLR is known as the Managed Code. For managed code CLR provides three .NET facilities:
   - CAS(Code Access Security)
   - Exception Handling
   - Automatic Memory Management

Common Language Specification(CLS)Contd.)

2. **Unmanaged Code:** Before .NET development the programming language like .COM Components & Win32 API do not generate the MSIL code. So these are not managed by CLR rather managed by Operating System.

Common Language Specification(CLS)Contd.)



MICROSOFT INTERMEDIATE LANGUAGE(MSIL)

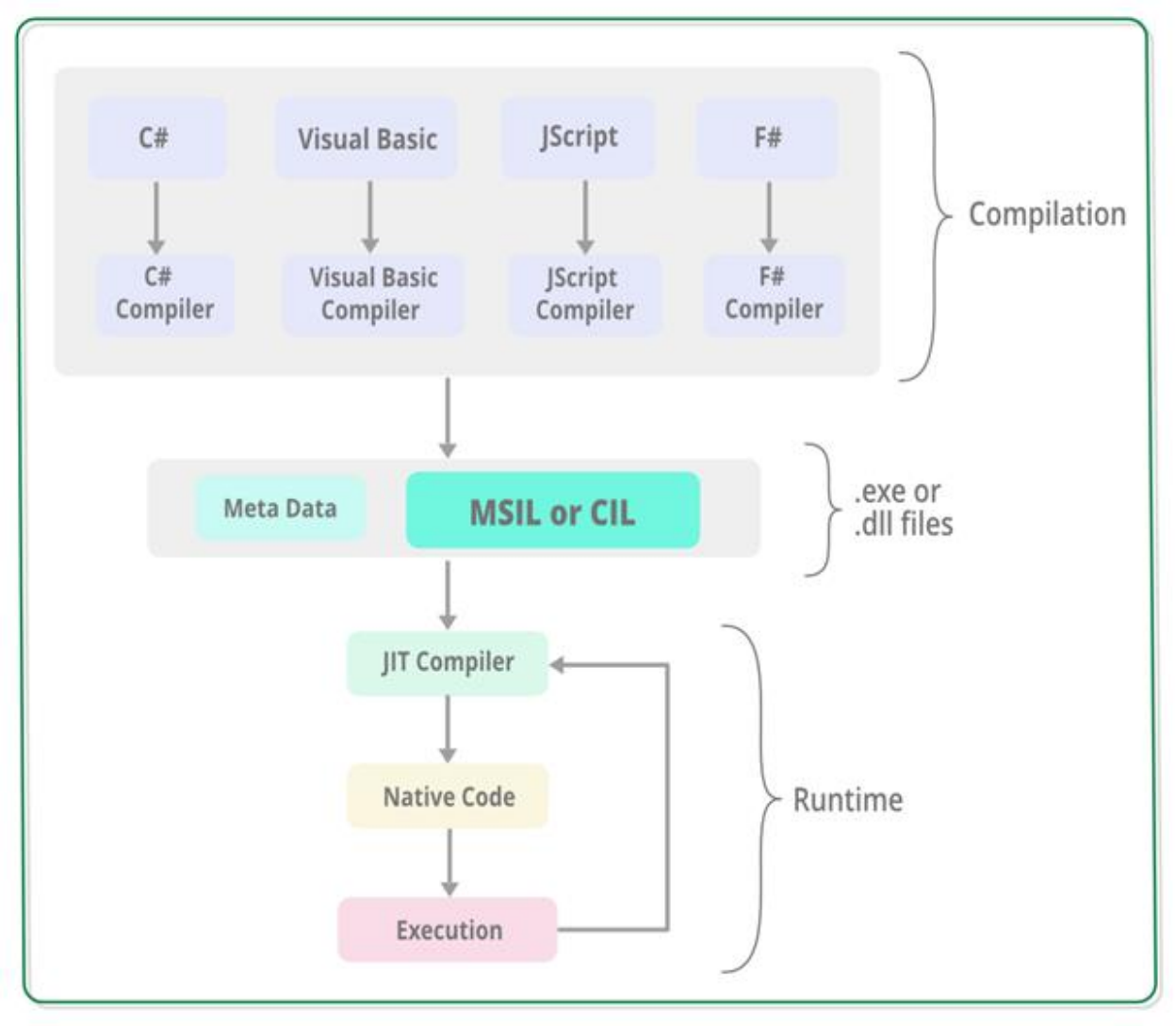
MICROSOFT INTERMEDIATE LANGUAGE(MSIL)

- The Microsoft Intermediate Language (MSIL), also known as the Common Intermediate Language (CIL) is a set of instructions that are platform independent and are generated by the language-specific compiler from the source code.
- The MSIL is platform independent and consequently, it can be executed on any of the Common Language Infrastructure supported environments such as the Windows .NET runtime.

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)

- The MSIL is converted into a particular computer environment specific machine code by the JIT compiler. This is done before the MSIL can be executed.
- The MSIL is converted into the machine code on a requirement basis i.e. the JIT compiler compiles the MSIL as required rather than the whole of it.
- Execution process in Common Language Runtime (CLR): The execution process that includes the creation of the MSIL and the conversion of the MSIL into machine code by the JIT compiler.

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)



MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)

- The source code is converted into the MSIL by a language-specific compiler in the compile time of the CLR. Also, along with the MSIL, metadata is also produced in the compilation. The metadata contains information such as the definition and signature of the types in the code, runtime information, etc.
- A Common Language Infrastructure (CLI) assembly is created by assembling the MSIL. This assembly is basically a compiled code library that is used for security, deployment, versioning, etc. and it is of two types i.e. process assembly (EXE) and library assembly (DLL).

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)

- The JIT compiler then converts the Microsoft Intermediate Language(MSIL) into the machine code that is specific to the computer environment that the JIT compiler runs on. The MSIL is converted into the machine code on a requirement basis i.e. the JIT compiler compiles the MSIL as required rather than the whole of it.
- The machine code obtained using the JIT compiler is then executed by the processor of the computer.

JUST IN TIME(JIT) COMPILATION

OVERVIEW

- Compilation vs Interpretation
- JVM and Common Language Runtime
- What is Just-In-Time Compilation?
- Why is Just-In-Time Compilation?

Compilation vs Interpretation

Compilation

Pros

> Programs run faster

Cons

> Compilation overhead

> Programs are typically bigger

> Programs are not portable

> No run-time information

Interpretation

Pros

> Programs are typically smaller

> Programs tend to be more portable

> Access to run-time information

Cons

> Programs run slower

JVM AND COMMON LANGUAGE RUNTIME

Virtual Machine: a software execution engine for a program written in a machine-independent language

– Ex., Java bytecodes, CLI, Pascal p-code, Smalltalk v-code

Step 1: syntax analysis and generate intermediate code, e.g., bytecode

Step 2: Interpret/compile the code on the VM (managed runtime) for portability, better safety checks

Microsoft Common Language Runtime: The immediate language can be shared by multiple source languages

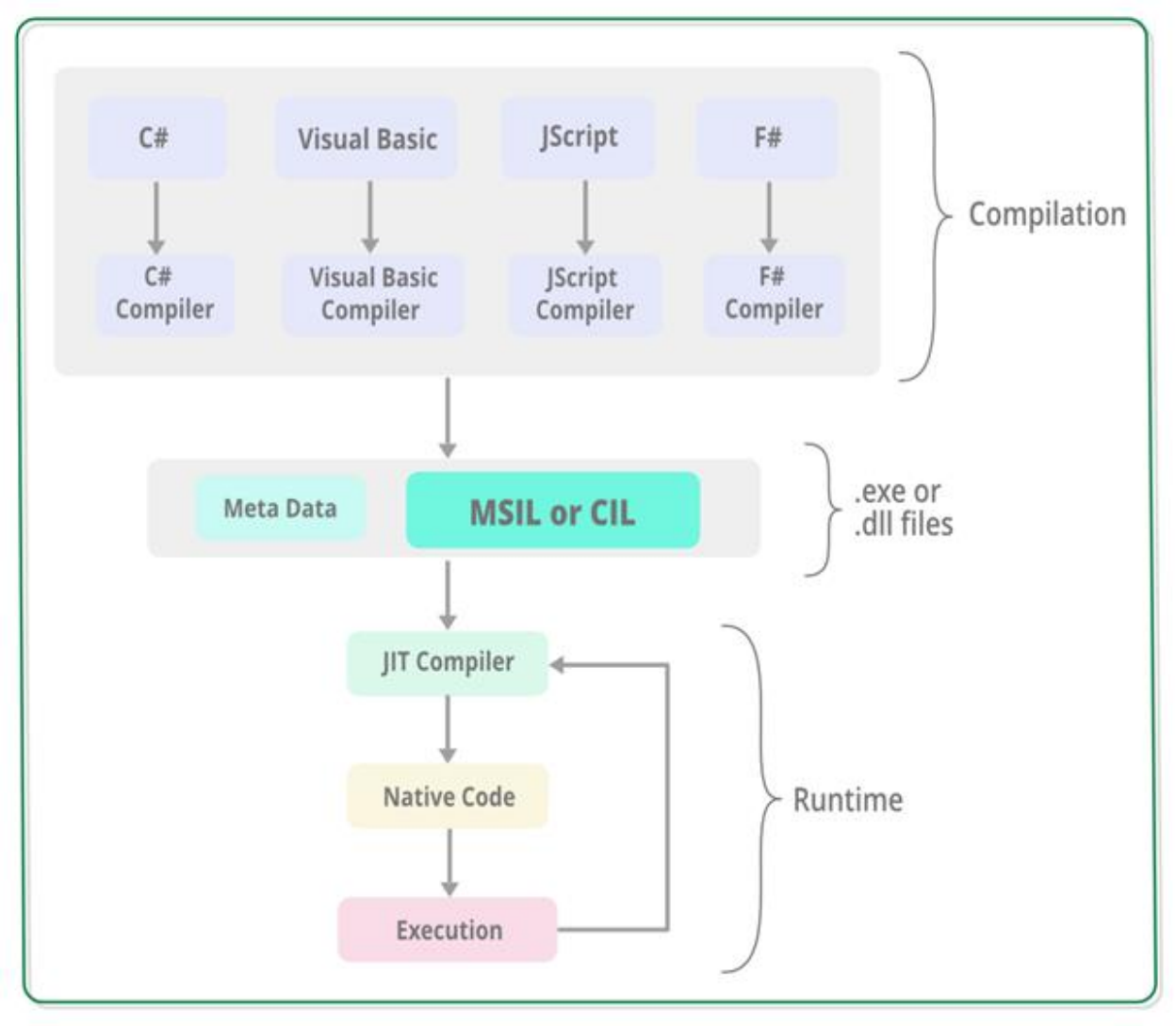
(e.g., C# and managed C++)

CIL is a CPU- and platform-independent instruction set.

JVM AND COMMON LANGUAGE RUNTIME

- The Common Language Runtime (CLR) manages the execution of code.
- CLR uses Just-In-Time (JIT) compiler to compile the CIL code to the native code for device used.
- Through the runtime compilation process CIL code is verified for safety during runtime, providing better security and reliability than natively compiled binaries.

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)



MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)

- The source code is converted into the MSIL by a language-specific compiler in the compile time of the CLR. Also, along with the MSIL, metadata is also produced in the compilation. The metadata contains information such as the definition and signature of the types in the code, runtime information, etc.
- A Common Language Infrastructure (CLI) assembly is created by assembling the MSIL. This assembly is basically a compiled code library that is used for security, deployment, versioning, etc. and it is of two types i.e. process assembly (EXE) and library assembly (DLL).

MICROSOFT INTERMEDIATE LANGUAGE (MSIL)(Contd.)

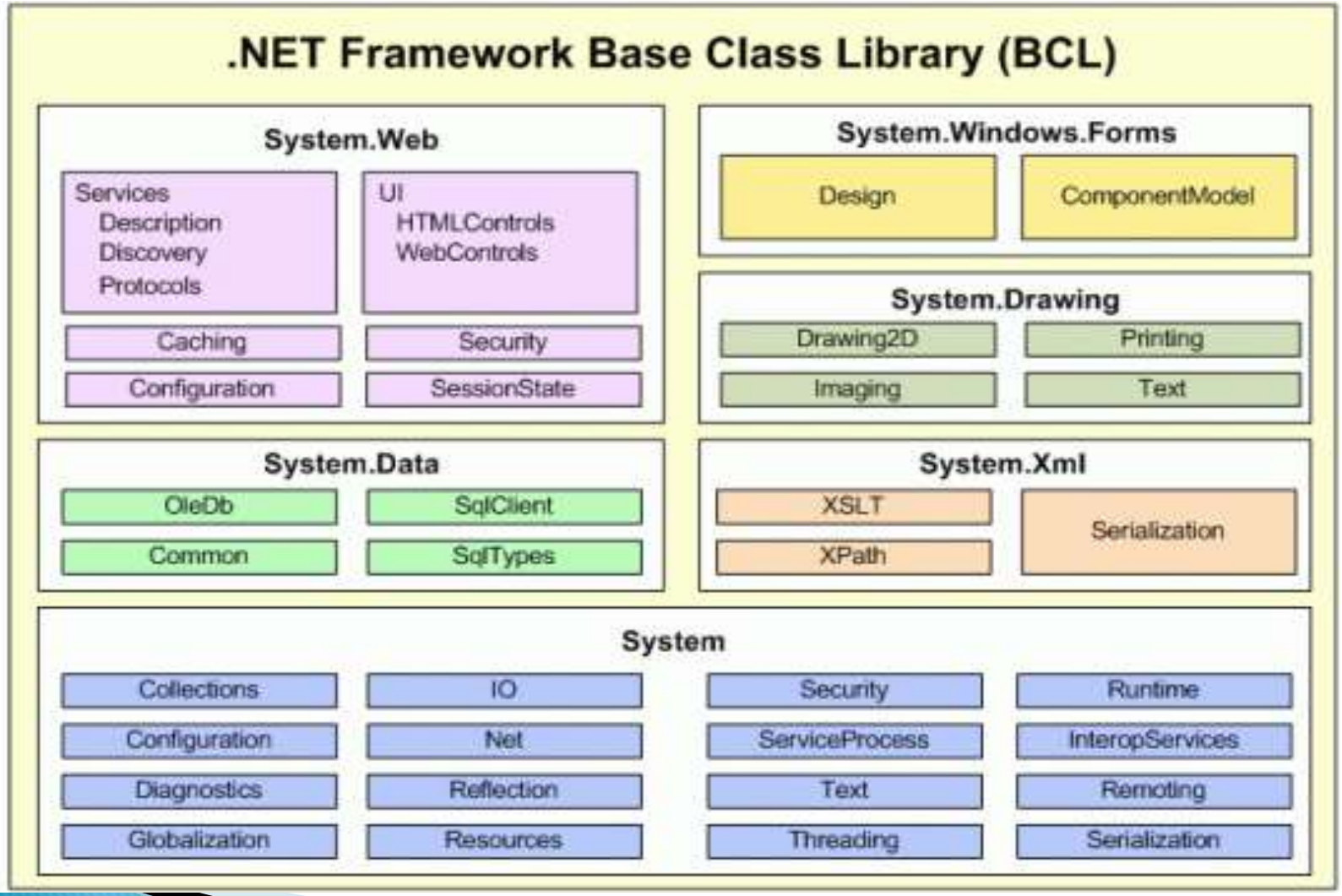
- The JIT compiler then converts the Microsoft Intermediate Language(MSIL) into the machine code that is specific to the computer environment that the JIT compiler runs on. The MSIL is converted into the machine code on a requirement basis i.e. the JIT compiler compiles the MSIL as required rather than the whole of it.
- The machine code obtained using the JIT compiler is then executed by the processor of the computer.

FCL: Framework Class Library

FCL

"The .NET Framework class library is a library of classes, interfaces, and value types that provide access to system functionality. It is the foundation on which .NET Framework applications, components, and controls are built."

Base Class Library



System Library

The System Library contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

I/O library

- Input/output library in `System.IO` namespace
- Compiled into `mscorlib.dll` assembly
- Support provided for:
  - file and directory management
  - text files
  - binary files

Part 2

- Database access…

Database library

- Database access provided by `System.Data.*` namespaces
- Compiled into `System.Data.dll` assembly
- Known collectively as *ADO.NET*
  - native support for SQL Server and Oracle
  - support for other databases via older *OleDB* technology
  - requires a knowledge of SQL

- Core namespaces:
  - general: **`System.Data`**, **`System.Data.Common`**
  - SQL Server: **`System.Data.SqlClient`**
  - Oracle: **`System.Data.OracleClient`**
  - OleDB: **`System.Data.OleDb`**

Part 3

- Data structures…

Collections library

- Data structures in .NET are generally known as *Collections*
- Located in the namespace `System.Collections`
- Compiled into `mscorlib.dll` assembly
- Defined in terms of `object` for generic use

- Core classes:
  - **`Array`**
  - **`ArrayList`**
  - **`Hashtable`**
  - **`Stack`**
  - **`Queue`**

- Thank You

C -Sharp Language (C#)

What is C#

- C# is pronounced as "C-Sharp".
- It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.
- By the help of C# programming language, we can develop different types of secured and robust applications:
- Window applications
- Distributed applications
- Web applications
- Web service applications
- Database applications etc.

What is C# (Contd.)

- C# is approved as a standard by ECMA (European Computer Manufacturers Association) and International Organization for Standardization (ISO).
- C# is designed for CLI (Common Language Infrastructure). CLI is a specification that describes executable code and runtime environment.
- C# programming language is influenced by C++, Java, Eiffel, Modula-3, Pascal etc. languages.

Java vs C#

No.	Java	C#
1)	*Java is a* **high level, robust, secured and object-oriented programming** *language developed by Oracle.*	*C# is an* **object-oriented programming** *language developed by Microsoft that runs on .Net Framework.*
2)	*Java programming language is designed to be run on a Java platform, by the help of* **Java Runtime Environment (JRE).**	*C# programming language is designed to be run on the* **Common Language Runtime (CLR).**
3)	*Java type safety is safe.*	*C# type safety is unsafe.*
4)	*In java, built-in data types that are passed by value are called* **primitive types.**	*In C#, built-in data types that are passed by value are called* **simple types.**

Java vs C# (Contd.)

No.	Java	C#
5)	*Arrays in Java are direct specialization of* **Object.**	*Arrays in C# are specialization of* **System.**
6)	*Java does not support* **conditional compilation.**	*C# supports conditional compilation using preprocessor directives.*
7)	*Java doesn't support goto statement.*	*C# supports goto statement.*
8)	*Java doesn't support* **structures and unions.**	*C# supports structures and unions.*

C# History

- C# is pronounced as "C-Sharp".
- It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.
- Anders Hejlsberg is known as the founder of C# language.

C# History (Contd.)

- It is based on C++ and Java, but it has many additional extensions used to perform component-oriented programming approach.
- C# has evolved much since their first release in the year 2002.
- It was introduced with .NET Framework 1.0 and the current version of C# is 5.0.

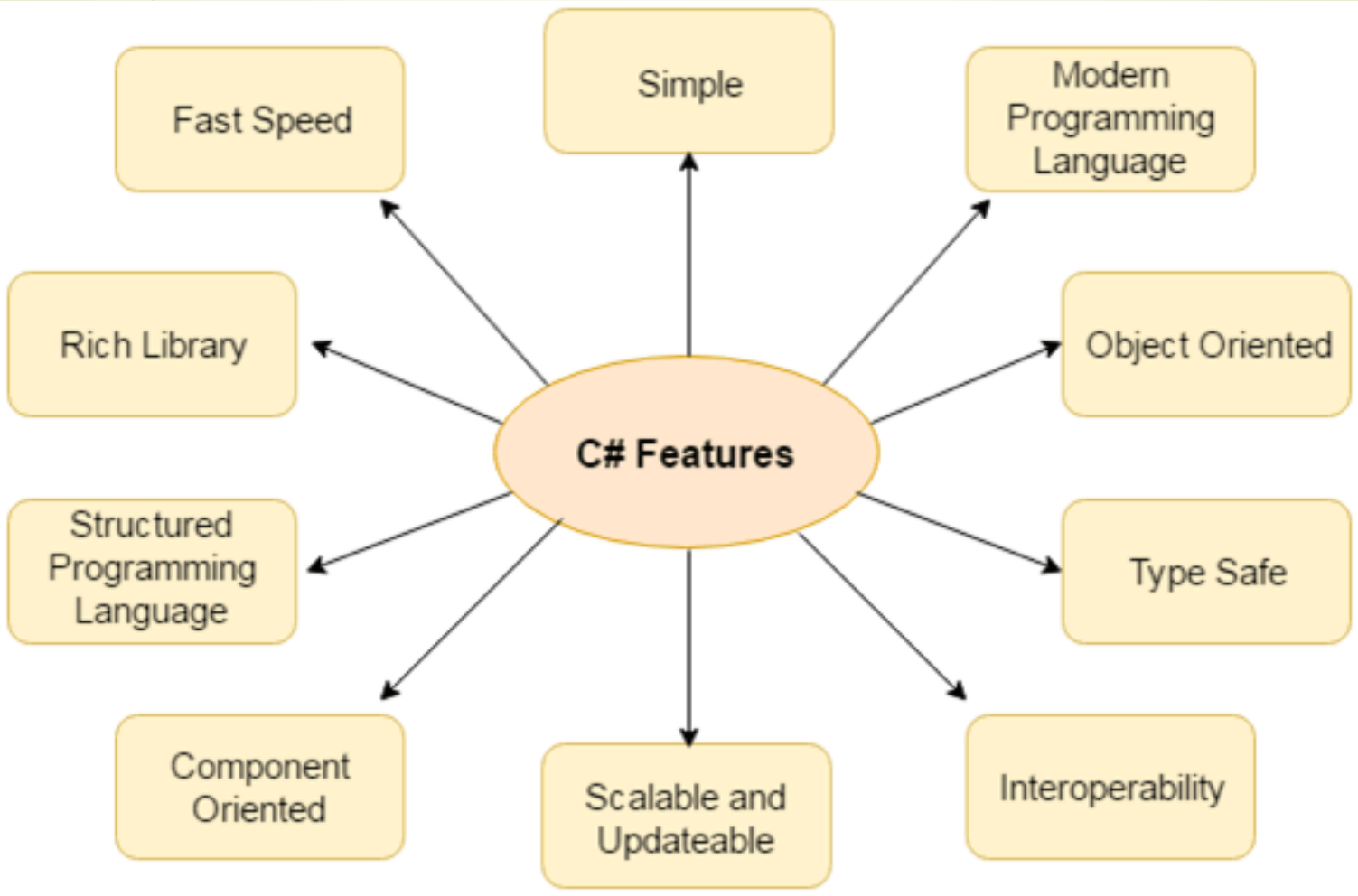
C# Version History

Version	Features	Year of release
C# 1.0	Basic Features	2002
C# 2.0	Generics, Partial types, Anonymous methods, Nullable types, Static classes	2005
C# 3.0	Var, LINQ, Lambda expression, Auto-implemented properties, Anonymous types, Extension methods	2007
C# 4.0	Dynamic binding, Named and Optional arguments	2010
C# 5.0	Asynchronous methods, Caller info attributes	2012
C# 6.0	Auto-property initializers, Null-propagating operator, Exception filters, Using static members, ...	2015

C# Features

1. Simple
2. Modern programming language
3. Object oriented
4. Type safe
5. Interoperability
6. Scalable and Updateable
7. Component oriented
8. Structured programming language
9. Rich Library
10. Fast speed

C# Features


C# Features
Simple
Modern Programming Language
Object Oriented
Type Safe
Interoperability
Scalable and Updateable
Component Oriented
Structured Programming Language
Rich Library
Fast Speed

C# Features (Contd.)

1. **Simple**

C# is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

2. **Modern Programming Language**

C# programming is based upon the current trend and it is very powerful and simple for building scalable, interoperable and robust applications.

C# Features (Contd.)

3. Object Oriented

C# is object oriented programming language. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

4. Type Safe

C# type safe code can only access the memory location that it has permission to execute. Therefore it improves a security of the program.

C# Features (Contd.)

5. Interoperability

Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

6. Scalable and Updateable

C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.

C# Features (Contd.)

7. Component Oriented

C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.

8. Scalable and Updateable

C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

C# Features (Contd.)

9. Rich Library

C# provides a lot of inbuilt functions that makes the development fast.

10. Fast Speed

The compilation and execution time of C# language is fast.

C# Example: Hello World

In C# programming language, a simple "hello world" program can be written by multiple ways. Let's see the top 4 ways to create a simple C# example:

- Simple Example
- Using System
- Using public modifier
- Using namespace

C# Example (Contd.)

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

Output:

Hello World!

C# Example(Contd.)

Description

class: is a keyword which is used to define class.

Program: is the class name. A class is a blueprint or template from which objects are created. It can have data members and methods. Here, it has only Main method.

static: is a keyword which means object is not required to access static members. So it saves memory.

C# Example(Contd.)

Description

void: is the return type of the method. It does't return any value. In such case, return statement is not required.

Main: is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.

C# Example(Contd.)

Description

string[] args: is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.

System.Console.WriteLine("HelloWorld!"): Here, System is the namespace. Console is the class defined in System namespace. The WriteLine() is the static method of Console class which is used to write the text on the console.

C# Example: Using System

If we write using *System* before the class, it means we don't need to specify System namespace for accessing any class of this namespace. Here, we are using Console class without specifying System.Console.

C# Example: Using System (Contd.)

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

Output:

Hello World!

C# Example: Using public modifier

We can also specify public modifier before class and Main() method. Now, it can be accessed from outside the class also.

C# Example: Using public modifier(Contd.)

```
using System;
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
```

Output:

Hello World!

C# Example: Using namespace

We can create classes inside the namespace. It is used to group related classes. It is used to categorize classes so that it can be easy to maintain.

C# Example: Using namespace (Contd.)

```
using System;
namespace ConsoleApplication1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Output:

Hello World!

C# Variables

- A variable is a name of memory location.
- It is used to store data.
- Its value can be changed and it can be reused many times.
- It is a way to represent memory location through symbol so that it can be easily identified.

C# Variables (Contd.)

The basic variable type available in C# can be categorized as:

Variable Type	Example
Decimal types	*decimal*
Boolean types	*True or false value, as assigned*
Integral types	*int, char, byte, short, long*
Floating point types	*float and double*
Nullable types	*Nullable data types*

C# Variables (Contd.)

Let's see the syntax to declare a variable:

type variable_list;

The example of declaring variable is given below:

int i, j;

double d;

float f;

char ch;

Here, i, j, d, f, ch are variables and int, double, float, char are data types.

C# Variables (Contd.)

We can also provide values while declaring the variables as given below:

int i=2,j=4; //declaring 2 variable of integer type

float f=40.2;

char ch='B';

Rules for defining variables

- A variable can have alphabets, digits and underscore.
- A variable name can start with alphabet and underscore only. It can't start with digit.
- No white space is allowed within variable name.
- A variable name must not be any reserved word or keyword e.g. char, float etc.

Rules for defining variables

Valid variable names:

int x;

int _x;

int k20;

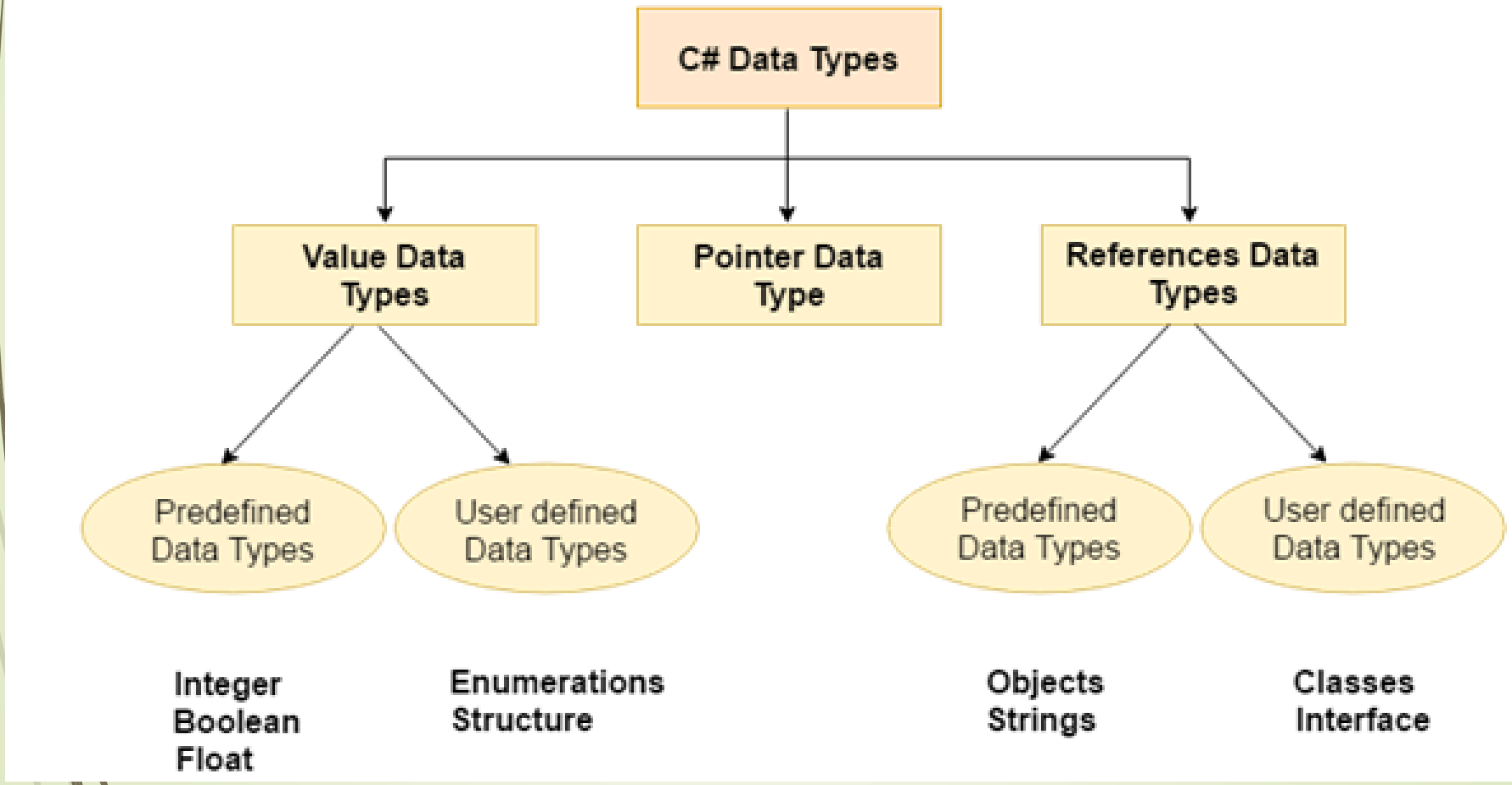
Invalid variable names:

int 4;

int x y;

int double;

Data Types


C# Data Types
Value Data Types
Pointer Data Type
References Data Types
Predefined Data Types
User defined Data Types
Predefined Data Types
User defined Data Types
Integer
Boolean
Float
Enumerations
Structure
Objects
Strings
Classes
Interface

Data Types (Contd.)

There are 3 types of data types in C# language.

Types	Data Types
Value Data Type	*short, int, char, float, double etc*
Reference Data Type	*String, Class, Object and Interface*
Pointer Data Type	*Pointers*

Value Data Types

The value data types are integer-based and floating-point based. C# language supports both signed and unsigned literals.

There are 2 types of value data type in C# language.

1) Predefined Data Types - such as Integer, Boolean, Float, etc.

2) User defined Data Types - such as Structure, Enumerations, etc.

Value Data Types (Contd.)

The memory size of data types may change according to 32 or 64 bit operating system.

Let's see the value data types. It size is given according to 32 bit OS.

Value Data Types (Contd.)

Data Types	Memory Size	Range
char	*1 byte*	*-128 to 127*
signed char	*1 byte*	*-128 to 127*
unsigned char	*1 byte*	*0 to 127*
short	*2 byte*	*-32,768 to 32,767*
signed short	*2 byte*	*-32,768 to 32,767*

Value Data Types (Contd.)

unsigned short	*2 byte*	*0 to 65,535*
int	*4 byte*	*-2,147,483,648 to -2,147,483,647*
signed int	*4 byte*	*-2,147,483,648 to -2,147,483,647*
unsigned int	*4 byte*	*0 to 4,294,967,295*
long	*8 byte*	*?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807*

Value Data Types (Contd.)

signed long	*8 byte*	*?9,223,372,036,854,775,808 to 9,223,372,036,854,775,807*
unsigned long	*8 byte*	*0 - 18,446,744,073,709,551,615*
float	*4 byte*	*$1.5 * 10^{-45}$ - $3.4 * 10^{38}$, 7-digit precision*
double	*8 byte*	*$5.0 * 10^{-324}$ - $1.7 * 10^{308}$, 15-digit precision*
decimal	*16 byte*	*at least $-7.9 * 10^{?28}$ - $7.9 * 10^{28}$, with at least 28-digit precision*

Reference Data Type

The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.

If the data is changed by one of the variables, the other variable automatically reflects this change in value.

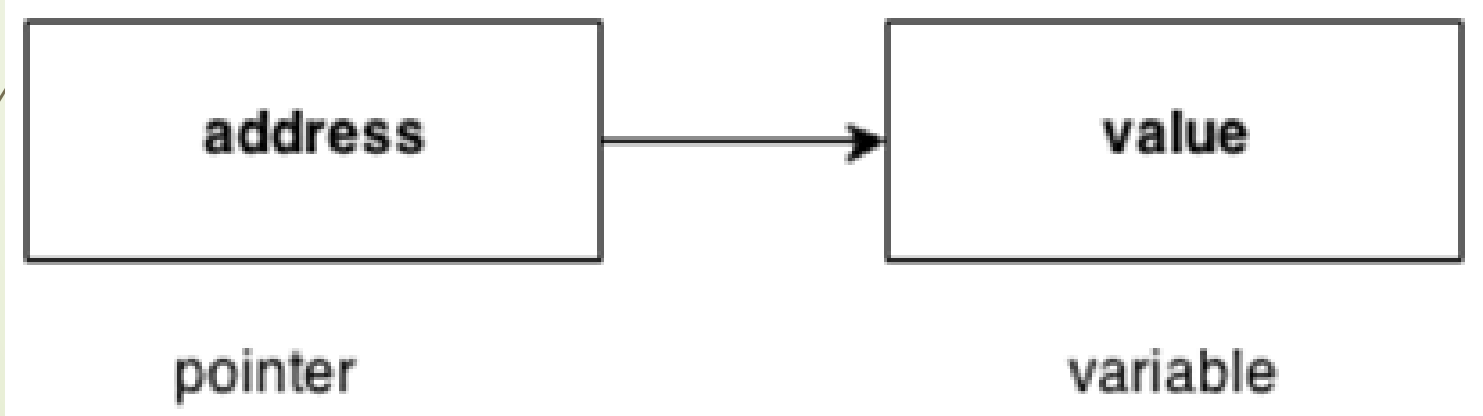
There are 2 types of reference data type in C# language.

1) Predefined Types - such as Objects, String.

2) User defined Types - such as Classes, Interface.

Pointer Data Type (Contd.)

The pointer in C# language is a variable, it is also known as locator or indicator that points to an address of a value.



Pointer Data Type

Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	*Address operator*	*Determine the address of a variable.*
** (asterisk sign)*	*Indirection operator*	*Access the value of an address.*

Pointer Data Type (Contd.)

Declaring a pointer

The pointer in C# language can be declared using * (asterisk symbol).

```
int * a;  //pointer to int
char * c; //pointer to char
```

C# operators

An operator is simply a symbol that is used to perform operations. There are following types of operators to perform different types of operations in C# language.

Arithmetic Operators

Relational Operators

Logical Operators

Bitwise Operators

Assignment Operators

Unary Operators

Ternary Operators

Misc Operators

C# operators

	Operator	Type
Binary Operator	+, -, *, /, %	Arithmetic Operators
	<,<=, >, >=, ==, !=	Relational Operators
	&&, ||, !	Logical Operators
	&, |, <<, >>, ~, ^	Bitwise Operators
	=, +=, -=,*=, /=, %=	Assignment Operators
Unary Operator	++, --	Unary Operator
Ternary Operator	?:	Ternary or Conditional Operator

Precedence of Operators in C#

- The precedence of operator specifies that which operator will be evaluated first and next. The associativity specifies the operators direction to be evaluated, it may be left to right or right to left.
- Let's understand the precedence by the example given below:

  int data= 10+ 5*5

- The "data" variable will contain 35 because * (multiplicative operator) is evaluated before + (additive operator).

Precedence of Operators in C#

Category (By Precedence)	Operator(s)	Associativity
Unary	*+ - ! ~ ++ -- (type)* & sizeof*	*Right to Left*
Additive	*+ -*	*Left to Right*
Multiplicative	*% / **	*Left to Right*
Relational	*< > <= >=*	*Left to Right*
Shift	*<< >>*	*Left to Right*
Equality	*== !=*	*Right to Left*
Logical AND	*&*	*Left to Right*

Precedence of Operators in C#

Category (By Precedence)	Operator(s)	Associativity
Logical OR	|	*Left to Right*
Logical XOR	^	*Left to Right*
Conditional OR	||	*Left to Right*
Conditional AND	&&	*Left to Right*
Null Coalescing	??	*Left to Right*
Ternary	?:	*Right to Left*
Assignment	= *= /= %= += - = <<= >>= &= ^= |= =>	*Right to Left*

C# Keywords

- A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

- In C# keywords cannot be used as identifiers. However, if we want to use the keywords as identifiers, we may prefix the keyword with @ character.

C# Keywords (Contd.)

abstract	base	as	bool	break	catch	case
byte	*char*	*checked*	*class*	*const*	*continue*	*decimal*
private	*protected*	*public*	*return*	*readonly*	*ref*	*sbyte*
explicit	*extern*	*false*	*finally*	*fixed*	*float*	*for*
foreach	*goto*	*if*	*implicit*	*in*	*in (generic modifier)*	*int*
ulong	*ushort*	*unchecked*	*using*	*unsafe*	*virtual*	*void*

C# Keywords (Contd.)

null	*object*	*operator*	*out*	*out (generic modifier)*	*override*	*params*
default	*delegate*	*do*	*double*	*else*	*enum*	*event*
sealed	*short*	*sizeof*	*stackalloc*	*static*	*string*	*struct*
switch	*this*	*throw*	*true*	*try*	*typeof*	*uint*
abstract	*base*	*as*	*bool*	*break*	*catch*	*case*
volatile	*while*					

C# Literals

Outline

- Integer Literals
- Floating-point Literals
- Character Literals
- String Literals
- Null Literals
- Boolean Literals

C# Literals

- The fixed values are called as Literal.
- Literal is a value which is used by the variables.
- Values can be either an integer, float or string etc.
  - // Here 100 is a constant/literal.

    int x = 100;

C# Literals (Contd.)

- Literals can be of following types:
  - Integer Literals
  - Floating-point Literals
  - Character Literals
  - String Literals
  - Null Literals
  - Boolean Literals

Integer Literals

- A literal of integer type is know as the integer literal.
- It can be octal, decimal or hexadecimal constant.
- No prefix is required for the decimal numbers.
- A suffix can also be used with the integer literals like U or u are used for unsigned numbers while l or L are used for long numbers.

Integer Literals (Contd.)

- For Integral data types (byte, short, int, long), we can specify literals in 3 ways:
- Decimal literals (Base 10) : In this form the allowed digits are 0-9.
  - int x = 101;
- Octal literals (Base 8) : In this form the allowed digits are 0-7.
  - // The octal number should be prefix with 0.

    int x = 0146;

Integer Literals (Contd.)

- Hexa-decimal literals (Base 16) : In this form the allowed digits are 0-9 and characters are a-f. We can use both uppercase and lowercase characters. As we know that c# is a case-sensitive programming language but here c# is not case-sensitive.
- // The hexa-decimal number should be prefix

  // with 0X or 0x.
- int x = 0X123Face;

Integer Literals Example

- 07778 // invalid: 8 is not an octal digit
- 045uu // invalid: suffix (u) is repeated
- 456 // valid decimal literal
- 02453 // valid octal literal
- 0x65d // valid hexadecimal literal
- 12356 // valid int literal
- 304U // valid unsigned int literal
- 3078L // valid long literal
- 965UL // valid unsigned long literal

Integer Literals Program

```
// C# program to illustrate the use of Integer Literals
using System;
  class IntergerLiteral {
    // Main method
    public static void Main(String []args)     {
        // decimal-form literal                    // octal-form literal
        int a = 101;                                   int b = 0145;
                                                     // Hexa-decimal form literal
                                                       int c = 0xFace;
                                                     Console.WriteLine(a);
                                                       Console.WriteLine(b);
                                                       Console.WriteLine(c);     } }
```

Output

101
145
64206

Floating-point Literals

- Floating-point Literals: The literal which has an integer part, a decimal point, a fractional part and an exponent part is known as the floating point literal. These can be represented either in decimal form or exponential form.
- Examples:
  - Double d = 3.14145 // Valid
  - Double d = 312569E-5 // Valid
  - Double d = 125E // invalid: Incomplete exponent
  - Double d = 784f // valid
  - Double d = .e45 // invalid: missing integer or fraction

Floating-point Literals Program

```
// C# program to illustrate the use of
// floating-point literals
using System;
 class FloatLiteral {
     // Main Method
   public static void Main(String []args)  {
      // decimal-form literal
      double a = 101.230;
      // It also acts as decimal literal
      double b = 0123.222;
      Console.WriteLine(a);
      Console.WriteLine(b);    } }
```

Output:

101.23
123.222

Note: By default, every floating-point literal is of double type and hence we can't assign directly to float variable. But we can specify floating-point literal as float type by suffixed with f or F. We can specify explicitly floating point literal as the double type by suffixed with d or D, of course, this convention is not required.

Character Literals

- For character data types we can specify literals in 3 ways:

- Single quote : We can specify literal to char data type as single character within single quote.
  - char ch = 'a';
- Unicode Representation : We can specify char literals in Unicode representation ‘\uxxxx’. Here xxxx represents 4 hexadecimal numbers.
  - char ch = '\u0061';// Here /u0061 represent a.

Character Literals

- Escape Sequence : Every escape character can be specify as char literals.
  - char ch = '\n';

ESCAPE SEQUENCE	MEANING
\\	\ character
\'	‘ character
\?	? character
\"	" character
\b	Backspace
\a	Alert or Bell
\n	New Line
\f	Form Feed
\r	Carriage Return
\v	Vertical Tab
\xhh…	Hexadecimal number of one or more digits

Character Literals Example

```
// C# program to illustrate the use of char literals
using System;
 class CharLiteral {
// Main Method
public static void Main(String []args)     {
        // character literal within single quote
       char ch = 'a';
                                                    // Unicode representation
                                                          char c = '\u0061';
                                                           Console.WriteLine(ch);
                                                          Console.WriteLine(c);
                                                          // Escape character literal
                                                          Console.WriteLine("Hello\n\nWorld\t!");
                                                    } }
```

Output:

```
a
a
Hello

World    !
```

String Literals

- Literals which are enclosed in double quotes(“”) or starts with @”” are known as the String literals.

- Examples:

- String s1 = "Hello World!";
- String s2 = @"Hello World!";

String Literals

```
// C#  program to illustrate the use of String literals
using System;
  class StringLiteral {
    // Main Method
   public static void Main(String []args)     {
     String s = "Hello World!";
       String s2 = @"Hello World!";
                                                    // If we assign without "" then it
                                                          // treats as a variable
                                                          // and causes compiler error
                                                          // String s1 = World;
                                                         Console.WriteLine(s);
                                                          Console.WriteLine(s2); } }
```

Output:

```
Hello World!
Hello World!
```

Boolean Literals

- Only two values are allowed for Boolean literals i.e. true and false.
- Example:
  - bool b = true;
  - bool c = false

Boolean Literals Program

```
// C# program to illustrate the use
// of boolean literals
using System;
  class BoolLiteral {
     // Main Method
    public  static  void  Main(String  []args)
{
      const bool b = true;
       bool c = false;
```

Output:

True
False

```
// these will give compile time error
      // bool d = 0;
      // bool e = 1;
      // Console.WriteLine(d);
      // Console.WriteLine(e);
       Console.WriteLine(b);
      Console.WriteLine(c);   } }
```

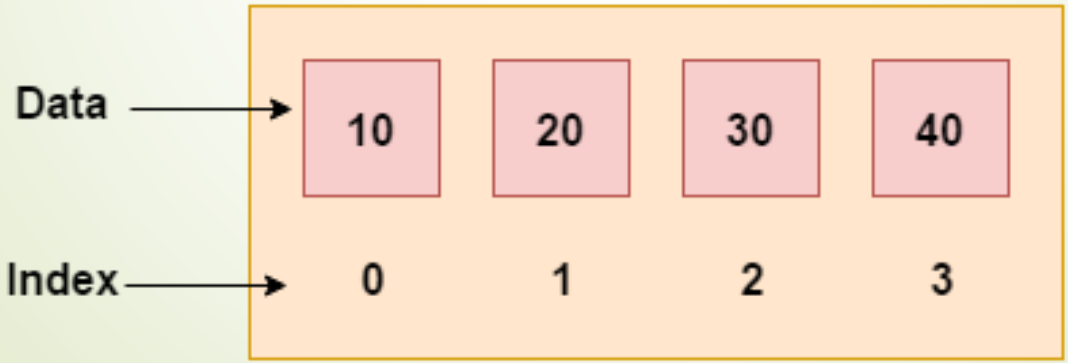
C# Array

Outline

- C# Arrays
- C# Array to Function
- C# Multidimensional Array
- C# Jagged Arrays
- C# Params
- C# Array class
- C# Command Line Args

C# Arrays

- Like other programming languages, array in C# is a group of similar types of elements that have contiguous memory location.
- In C#, array is an object of base type System.Array.
- In C#, array index starts from 0.
- We can store only fixed set of elements in C# array.



Advantages of C# Array

- Code Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

Disadvantages of C# Array

- Fixed size

C# Array Types

- There are 3 types of arrays in C# programming:
  - Single Dimensional Array
  - Multidimensional Array
  - Jagged Array

C# Single Dimensional Array

- To create single dimensional array, you need to use square brackets [] after the type.
  - int[] arr = new int[5];//creating array
- You cannot place square brackets after the identifier.
  - int arr[] = new int[5];//compile time error

Example of C# array

- Declare, initialize and traverse array

```
using System;
public class ArrayExample
{
    public static void Main(string[] args)
    {
        int[] arr = new int[5];//creating array
        arr[0] = 10;//initializing array
        arr[2] = 20;
        arr[4] = 30;
          //traversing array
        for (int i = 0; i < arr.Length; i++)
        {
            Console.WriteLine(arr[i]);
        }
    }
}
```

Output:

```
10
0
20
0
30
```

C# Array Example: Declaration and Initialization at same time

- There are 3 ways to initialize array at the time of declaration.
  - int[] arr = new int[5]{ 10, 20, 30, 40, 50 };
- We can omit the size of array.
  - int[] arr = new int[]{ 10, 20, 30, 40, 50 };
- We can omit the new operator also.
  - int[] arr = { 10, 20, 30, 40, 50 };

C# Array Example: Declaration and Initialization at same time

- Let's see the example of array where we are declaring and initializing array at the same time.

```
using System;
public class ArrayExample
{
   public static void Main(string[] args)
   {
      int[] arr = { 10, 20, 30, 40, 50 };//Declaration and Initialization of array

      //traversing array
      for (int i = 0; i < arr.Length; i++)
      {
         Console.WriteLine(arr[i]);
      }
   }
}
```

Output:

10
20
30
40
50

C# Passing Array to Function

- In C#, to reuse the array logic, we can create function.
- To pass array to function in C#, we need to provide only array name.
- functionname(arrayname);//passing array

C# Passing Array to Function Example: print array elements

- Let's see an example of C# function which prints the array elements.

```
using System;
public class ArrayExample  {
   static void printArray(int[] arr) {
      Console.WriteLine("Printing array elements:");
      for (int i = 0; i < arr.Length; i++)  {
           Console.WriteLine(arr[i]);
     }}
   public static void Main(string[] args)  {
      int[] arr1 = { 25, 10, 20, 15, 40, 50 };
      int[] arr2 = { 12, 23, 44, 11, 54 };
      printArray(arr1);//passing array to function
      printArray(arr2);
  }  }
```

Output:

```
Printing array elements:
25
10
20
15
40
50
Printing array elements:
12
23
44
11
54
```

C# Passing Array to Function Example: Print minimum number

- Let's see an example of C# array which prints minimum number in an array using function.

```
using System;
public class ArrayExample  {
   static void printMin(int[] arr)  {
      int min = arr[0];
      for (int i = 1; i < arr.Length; i++)  {
         if (min > arr[i])  {
            min = arr[i];
         }  }
      Console.WriteLine("Minimum element is: " + min); }
   public static void Main(string[] args)  {
      int[] arr1 = { 25, 10, 20, 15, 40, 50 };
      int[] arr2 = { 12, 23, 44, 11, 54 };
      printMin(arr1);//passing array to function
      printMin(arr2);
   }  }
```

Output:

```
Minimum element is: 10
Minimum element is: 11
```

C# Passing Array to Function Example: Print maximum number

- Let's see an example of C# array which prints maximum number in an array using function.

```
using System;
public class ArrayExample  {
   static void printMax(int[] arr)  {
      int max = arr[0];
      for (int i = 1; i < arr.Length; i++)  {
         if (max < arr[i])  {
            max = arr[i];
         }  }
      Console.WriteLine("Maximum element is: " + max);   }
   public static void Main(string[] args)
      int[] arr1 = { 25, 10, 20, 15, 40, 50 };
      int[] arr2 = { 12, 23, 64, 11, 54 };
      printMax(arr1);//passing array to function
      printMax(arr2);   }  }
```

Output:

```
Maximum element is: 50
Maximum element is: 64
```

C# Multidimensional Arrays

- The multidimensional array is also known as rectangular arrays in C#.
- It can be two dimensional or three dimensional.
- The data is stored in tabular form (row * column) which is also known as matrix.
- To create multidimensional array, we need to use comma inside the square brackets. For example:
  - int[,] arr=new int[3,3];//declaration of 2D array
  - int[,,] arr=new int[3,3,3];//declaration of 3D array

C# Multidimensional Array Example

- Let's see a simple example of multidimensional array in C# which declares, initializes and traverse two-dimensional array.

```
using System;
public class MultiArrayExample  {
   public static void Main(string[] args)  {
      int[,] arr=new int[3,3];//declaration of 2D array
      arr[0,1]=10;//initialization
      arr[1,2]=20;
      arr[2,0]=30;
      //traversal
      for(int i=0;i<3;i++){
         for(int j=0;j<3;j++){
            Console.Write(arr[i,j]+" ");  }
         Console.WriteLine();//new line at each row
      }  }  }
```

Output:

```
0 10 0
0 0 20
30 0 0
```

C# Multidimensional Array Example: Declaration and initialization at same time

- There are 3 ways to initialize multidimensional array in C# while declaration.
  - int[,] arr = new int[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
- We can omit the array size.
  - int[,] arr = new int[,]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
- We can omit the new operator also.
  - int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

C# Multidimensional Array Example

- Let's see a simple example of multidimensional array which initializes array at the time of declaration.

```
using System;
public class MultiArrayExample  {
   public static void Main(string[] args)      {
      int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };//declaration and initializati
on
      //traversal
      for(int i=0;i<3;i++){
         for(int j=0;j<3;j++){
            Console.Write(arr[i,j]+" ");  }
         Console.WriteLine();//new line at each row
      }
   }
}
```

Output:

```
1 2 3
4 5 6
7 8 9
```

C# Jagged Arrays

- In C#, jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.
- Declaration of Jagged array
  - Let's see an example to declare jagged array that has two elements.
  - int[][] arr = new int[2][];

Initialization of Jagged array

- Let's see an example to initialize jagged array. The size of elements can be different.
  - arr[0] = new int[4];
  - arr[1] = new int[6];
- Initialization and filling elements in Jagged array
- Let's see an example to initialize and fill elements in jagged array.
- arr[0] = new int[4] { 11, 21, 56, 78 };
- arr[1] = new int[6] { 42, 61, 37, 41, 59, 63 };

C# Jagged Array Example

- Let's see a simple example of jagged array in C# which declares, initializes and traverse jagged arrays.

```
public class JaggedArrayTest  {
   public static void Main()      {
      int[][] arr = new int[2][];// Declare the array
     arr[0] = new int[] { 11, 21, 56, 78 };// Initialize the array
      arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };
       // Traverse array elements
      for (int i = 0; i < arr.Length; i++)  {
         for (int j = 0; j < arr[i].Length; j++) {
            System.Console.Write(arr[i][j]+" ");
         }
         System.Console.WriteLine();
      }
   }
}
```

Output:

```
11 21 56 78
42 61 37 41 59 63
```

Initialization of Jagged array upon Declaration

- Let's see an example to initialize the jagged array while declaration.

```
int[][] arr = new int[3][]{
        new int[] { 11, 21, 56, 78 },
        new int[] { 2, 5, 6, 7, 98, 5 },
        new int[] { 2, 5 }
        };
```

C# Jagged Array Example 2

- Let's see a simple example of jagged array which initializes the jagged arrays upon declaration.

```
public class JaggedArrayTest {
   public static void Main()  {
      int[][] arr = new int[3][]{
      new int[] { 11, 21, 56, 78 },
      new int[] { 2, 5, 6, 7, 98, 5 },
      new int[] { 2, 5 }
      };
       // Traverse array elements
      for (int i = 0; i < arr.Length; i++)   {
         for (int j = 0; j < arr[i].Length; j++)  {
            System.Console.Write(arr[i][j]+" ");
         }
         System.Console.WriteLine();
      }   } }
```

Output:

```
11 21 56 78
2 5 6 7 98 5
2 5
```

C# Params

- In C#, params is a keyword which is used to specify a parameter that takes variable number of arguments.
- It is useful when we don't know the number of arguments prior.
- Only one params keyword is allowed and no additional parameter is permitted after params keyword in a function declaration.

C# Params Example 1

```
using System;
namespace AccessSpecifiers  {
    class Program   {
        // User defined function
        public void Show(params int[] val) // Params Paramater   {
            for (int i=0; i<val.Length; i++)  {
                Console.WriteLine(val[i]);
            }
        }
        // Main function, execution entry point of the program
        static void Main(string[] args)  {
            Program program = new Program(); // Creating Object
            program.Show(2,4,6,8,10,12,14); // Passing arguments of variable
length
        }
    }
}
```

Output:

```
2
4
6
8
10
12
14
```

C# Params Example 2

In this example, we are using object type params that allow entering any number of inputs of any type.

```
using System;
namespace AccessSpecifiers {
   class Program {
      // User defined function
      public void Show(params object[] items)
      // Params Paramater {
         for (int i = 0; i < items.Length; i++) {
            Console.WriteLine(items[i]);
         }
      }
      // Main function, execution entry point of the program
      static void Main(string[] args)        {
         Program program = new Program(); // Creating Object
         program.Show("Ramakrishnan Ayyer","Ramesh",101, 20.50,"Peter",
'A'); // Passing arguments of variable length } } }
```

Output:

Ramakrishnan Ayyer
Ramesh
101
20.5
Peter
A

C# Array class

- C# provides an Array class to deal with array related operations.
- It provides methods for creating, manipulating, searching, and sorting elements of an array.
- This class works as the base class for all arrays in the .NET programming environment.

C# Array class Signature

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public abstract class Array : ICloneable, IList, ICollection,
IEnumerable, IStructuralComparable, IStructuralEquatable
```

Note: In C#, Array is not part of collection but considered as collection because it is based on the IList interface.

C# Array Properties

Property	Description
IsFixedSize	*It is used to get a value indicating whether the Array has a fixed size or not.*
IsReadOnly	*It is used to check that the Array is read-only or not.*
IsSynchronized	*It is used to check that access to the Array is synchronized or not.*
Length	*It is used to get the total number of elements in all the dimensions of the Array.*
LongLength	*It is used to get a 64-bit integer that represents the total number of elements in all the dimensions of the Array.*
Rank	*It is used to get the rank (number of dimensions) of the Array.*
SyncRoot	*It is used to get an object that can be used to synchronize access to the Array.*

C# Array Methods

Method	Description
AsReadOnly<T>(T[])	*It returns a read-only wrapper for the specified array.*
BinarySearch(Array,Int32,Int32,Object)	*It is used to search a range of elements in a one-dimensional sorted array for a value.*
BinarySearch(Array,Object)	*It is used to search an entire one-dimensional sorted array for a specific element.*
Clear(Array,Int32,Int32)	*It is used to set a range of elements in an array to the default value.*
Clone()	*It is used to create a shallow copy of the Array.*
Copy(Array,Array,Int32)	*It is used to copy elements of an array into another array by specifying starting index.*
CopyTo(Array,Int32)	*It copies all the elements of the current one-dimensional array to the specified one-dimensional array starting at the specified destination array index*
CreateInstance(Type,Int32)	*It is used to create a one-dimensional Array of the specified Type and length.*
Empty<T>()	*It is used to return an empty array.*

C# Array Methods

Method	Description
Finalize()	*It is used to free resources and perform cleanup operations.*
Find<T>(T[],Predicate<T>)	*It is used to search for an element that matches the conditions defined by the specified predicate.*
IndexOf(Array,Object)	*It is used to search for the specified object and returns the index of its first occurrence in a one-dimensional array.*
Initialize()	*It is used to initialize every element of the value-type Array by calling the default constructor of the value type.*
Reverse(Array)	*It is used to reverse the sequence of the elements in the entire one-dimensional Array.*
Sort(Array)	*It is used to sort the elements in an entire one-dimensional Array.*
ToString()	*It is used to return a string that represents the current object.*

C# Array Example

```
using System;
namespace CSharpProgram  {
   class Program     {
      static void Main(string[] args)  {
         // Creating an array
         int[] arr = new int[6] { 5, 8, 9, 25, 0, 7 };
         // Creating an empty array
         int[] arr2 = new int[6];
         // Displaying length of array
         Console.WriteLine("length of first array: "+arr.Length);
         // Sorting array
         Array.Sort(arr);
         Console.Write("First array elements: ");
         // Displaying sorted array
         PrintArray(arr);
         // Finding index of an array element
```

C# Array Example

```
Console.WriteLine("\nIndex position of 25 is "+Array.IndexOf(arr,25));
        // Coping first array to empty array
        Array.Copy(arr, arr2, arr.Length);
        Console.Write("Second array elements: ");
        // Displaying second array
        PrintArray(arr2);
        Array.Reverse(arr);
        Console.Write("\nFirst Array elements in reverse order: ");
        PrintArray(arr);  }
    // User defined method for iterating array elements
    static void PrintArray(int[] arr)
    {
        foreach (Object elem in arr)
        {
            Console.Write(elem+" ");
        }  }  } }
```

Output:
length of first array: 6
First array elements: 0 5 7 8 9 25
Index position of 25 is 5
Second array elements: 0 5 7 8 9 25
First Array elements in reverse order: 25 9 8 7 5 0

C# Command Line Arguments

- Arguments that are passed by command line known as command line arguments.
- We can send arguments to the Main method while executing the code.
- The string args variable contains all the values passed from the command line.

C# Command Line Arguments Example

```
using System;
namespace CSharpProgram{
   class Program {
      // Main function, execution entr
y point of the program
      static void Main(string[] args) //
string type parameters   {
         // Command line arguments
          Console.WriteLine("Argumen
t length: "+args.Length);
         Console.WriteLine("Supplied
Arguments are:");
         foreach (Object obj in args)
         {
            Console.WriteLine(obj);
         }
      }
   }
}
```

Compile: csc Program.cs

Execute: Program.exe Hi there, how are you?

After executing the code, it produces the following output to the console.

Output:

```
Argument length: 5
Supplied Arguments are:
Hi
there,
how
are
you?
```

C# String

C# String

- In C#, string is an object of System.String class that represent sequence of characters.

- We can perform many operations on strings such as concatenation, comparision, getting substring, search, trim, replacement etc.

string vs String

- In C#, string is keyword which is an alias for System.String class. That is why string and String are equivalent. We are free to use any naming convention.
- string s1 = "hello";//creating string using string keyword
- String s2 = "welcome";//creating string using String class

C# String Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "hello";

        char[] ch = { 'c', 's', 'h', 'a', 'r', 'p' };
        string s2 = new string(ch);

        Console.WriteLine(s1);
        Console.WriteLine(s2);
    }
}
```

Output:

hello
csharp

C# String methods

Method Name	Description
Clone()	*It is used to return a reference to this instance of String.*
Compare(String, String)	*It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.*
CompareOrdinal(String, String)	*It is used to compare two specified String objects by evaluating the numeric values of the corresponding Char objects in each string..*
CompareTo(String)	*It is used to compare this instance with a specified String object. It indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.*
Concat(String, String)	*It is used to concatenate two specified instances of String.*
Contains(String)	*It is used to return a value indicating whether a specified substring occurs within this string.*
Copy(String)	*It is used to create a new instance of String with the same value as a specified String.*

C# String methods

Method Name	Description
CopyTo(Int32, Char[], Int32, Int32)	*It is used to copy a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.*
EndsWith(String)	*It is used to check that the end of this string instance matches the specified string.*
Equals(String, String)	*It is used to determine that two specified String objects have the same value.*
Format(String, Object)	*It is used to replace one or more format items in a specified string with the string representation of a specified object.*
GetEnumerator()	*It is used to retrieve an object that can iterate through the individual characters in this string.*
GetHashCode()	*It returns the hash code for this string.*
GetType()	*It is used to get the Type of the current instance.*
GetTypeCode()	*It is used to return the TypeCode for class String.*

C# String methods

Method Name	Description
IndexOf(String)	*It is used to report the zero-based index of the first occurrence of the specified string in this instance.*
Insert(Int32, String)	*It is used to return a new string in which a specified string is inserted at a specified index position.*
Intern(String)	*It is used to retrieve the system's reference to the specified String.*
IsInterned(String)	*It is used to retrieve a reference to a specified String.*
IsNormalized()	*It is used to indicate that this string is in Unicode normalization form C.*
IsNullOrEmpty(String)	*It is used to indicate that the specified string is* **null** *or an Empty string.*
IsNullOrWhiteSpace(String)	*It is used to indicate whether a specified string is* **null***, empty, or consists only of white-space characters.*
Join(String, String[])	*It is used to concatenate all the elements of a string array, using the specified separator between each element.*

C# String methods

Method Name	Description
LastIndexOf(Char)	*It is used to report the zero-based index position of the last occurrence of a specified character within String.*
LastIndexOfAny(Char[])	*It is used to report the zero-based index position of the last occurrence in this instance of one or more characters specified in a Unicode array.*
Normalize()	*It is used to return a new string whose textual value is the same as this string, but whose binary representation is in Unicode normalization form C.*
PadLeft(Int32)	*It is used to return a new string that right-aligns the characters in this instance by padding them with spaces on the left.*
PadRight(Int32)	*It is used to return a new string that left-aligns the characters in this string by padding them with spaces on the right.*
Remove(Int32)	*It is used to return a new string in which all the characters in the current instance, beginning at a specified position and continuing through the last position, have been deleted.*
Replace(String, String)	*It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.*

C# String methods

Method Name	Description
Split(Char[])	*It is used to split a string into substrings that are based on the characters in an array.*
StartsWith(String)	*It is used to check whether the beginning of this string instance matches the specified string.*
Substring(Int32)	*It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.*
ToCharArray()	*It is used to copy the characters in this instance to a Unicode character array.*
ToLower()	*It is used to convert String into lowercase.*
ToLowerInvariant()	*It is used to return convert String into lowercase using the casing rules of the invariant culture.*

C# String methods

Method Name	Description
ToString()	*It is used to return instance of String.*
ToUpper()	*It is used to convert String into uppercase.*
Trim()	*It is used to remove all leading and trailing white-space characters from the current String object.*
TrimEnd(Char[])	*It Is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.*
TrimStart(Char[])	*It is used to remove all leading occurrences of a set of characters specified in an array from the current String object.*

C# String Clone()

- The C# Clone() method is used to clone a string object. It returns another copy of same data. The return type of Clone() method is object.
- Signature
  - public object Clone()
- Parameters
  - It does not take any parameter.
- Returns
  - It returns a reference.

C# String Clone()

- The C# Clone() method is used to clone a string object. It returns another copy of same data. The return type of Clone() method is object.
- Signature
  - public object Clone()
- Parameters
  - It does not take any parameter.
- Returns
  - It returns a reference.

C# String Clone () method example

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello ";
            string s2 = (String)s1.Clone();
            Console.WriteLine(s1);
            Console.WriteLine(s2);
        }
    }
```

Output:

```
Hello
Hello
```

C# String Compare()

- The C# Compare() method is used to compare first string with second string lexicographically. It returns an integer value.
- If both strings are equal, it returns 0. If first string is greater than second string, it returns 1 else it returns -1
- Rule
  - s1==s2 returns 0
  - s1>s2 returns 1
  - s1<s2 returns -1

C# String Compare()

- Signatures
  - public static int Compare(String first, String second)
  - public static int Compare(String, Int32, String, Int32, Int32)
  - public static int Compare(String, Int32, Int32, String, Int32, Boolean)
  - public static int Compare(String, Boolean, Int32, Int32, String, Int32, CultureInfo)
  - public static int Compare(String, CultureInfo, Int32, Int32, String, Int32, CompareOptions)
  - public static int Compare(String, Int32, Int32, String, Int32, StringComparison)
  - public static int Compare(String, String, Boolean)
  - public static int Compare(String, String, Boolean, CultureInfo)
  - public static int Compare(String, String, CultureInfo, CompareOptions)
  - public static int Compare(String, String, StringComparison)

C# String Compare()

- Parameters
  - first: first argument represents string which is to be compared with second string.
  - second: second argument represents string which is to be compared with first string.

- Return
  - It returns an integer value.

C# String Compare() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "hello";
         string s2 = "hello";
         string s3 = "csharp";
         string s4 = "mello";

         Console.WriteLine(string.Compare(s1,s2));
         Console.WriteLine(string.Compare(s2,s3));
         Console.WriteLine(string.Compare(s3,s4));
      }
   }
```

Output:

0
1
-1

C# String CompareOrdinal()

- The C# CompareOrdinal() method compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.

- If both strings are equal, it returns 0. If first string is greater than second string, it returns positive number in difference else it returns negative number.

C# String CompareOrdinal()

- Rule
  - s1==s2 returns 0
  - s1>s2 returns positive number in difference
  - s1<s2 returns negative number in difference
- Signature
  - public static int CompareOrdinal(String first, String second)
  - public static int CompareOrdinal(String, Int32, String, Int32, Int32)

C# String CompareOrdinal()

- Parameters
  - first: first argument represents string which is to be compared with second string.
  - second: second argument represents string which is to be compared with first string.

- Return
  - It returns an integer value.

C# String CompareOrdinal() Method Example

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "hello";
            string s2 = "hello";
            string s3 = "csharp";
            string s4 = "mello";

            Console.WriteLine(string.CompareOrdinal(s1,s2));
            Console.WriteLine(string.CompareOrdinal(s1,s3));
            Console.WriteLine(string.CompareOrdinal(s1,s4));
        }
    }
```

Output:

```
0
5
-5
```

C# String CompareTo()

- The C# CompareTo() method is used to compare String instance with a specified String object.

- It indicates whether this String instance precedes, follows, or appears in the same position in the sort order as the specified string or not.

C# String CompareTo()

- Signature
  - public int CompareTo(String str)
  - public int CompareTo(Object)
- Parameters
  - str: it is a string argument which is used to compare.
- Return
  - It returns an integer value.

C# String CompareTo() Method Example

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "hello";
            string s2 = "hello";
            string s3 = "csharp";
            Console.WriteLine(s1.CompareTo(s2));
            Console.WriteLine(s2.CompareTo(s3));
        }
    }
```

Output:

0
1

C# String Concat()

- The C# Concat() method is used to concatenate multiple string objects. It returns concatenated string. There are many overloaded methods of Concat().
- Signature
  - public static string Concat(String, String)
  - public static string Concat(IEnumerable<String>)
  - public static string Concat(Object)
  - public static string Concat(Object, Object)
  - public static string Concat(Object, Object, Object)
  - public static string Concat(Object, Object, Object, Object)
  - public static string Concat(params Object[])
  - public static string Concat(String, String, String)
  - public static string Concat(String, String, String,?String)
  - public static string Concat(params String[])
  - [ComVisibleAttribute(false)]
  - public static string Concat<T>(IEnumerable<T>)

C# String Concat()

- Parameters
  - It takes two String object arguments.

- Return
  - It returns a string object.

C# String Concat() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello ";
         string s2 = "C#";
         Console.WriteLine(string.Concat(s1,s2));
      }
   }
```

Output:

Hello C#

C# String Contains()

- The C# Contains() method is used to return a value indicating whether the specified substring occurs within this string or not. If the specified substring is found in this string, it returns true otherwise false.

- Signature
  - public bool Contains(String str)

C# String Contains()

- Parameters
  - str: it is a string object which is used to check occurrence in the calling string.

- Return
  - It returns boolean value either true or false.

C# String Contains() method Example

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello ";
            string s2 = "He";
            string s3 = "Hi";
            Console.WriteLine(s1.Contains(s2));
            Console.WriteLine(s1.Contains(s3));

        }
    }
```

Output:

True
False

C# String Copy()

- The C# Copy() method is used to create a new instance of String with the same value as a specified String. It is a static method of String class. Its return type is string.
- Signature
  - public static string Copy(String str)
- Parameter
  - str: it takes a string argument which is used to create a copy of specified string.
- Return
  - It returns string object.

C# String Copy() Method Example

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello ";
            string s2 = string.Copy(s1);
            Console.WriteLine(s1);
            Console.WriteLine(s2);
        }
    }
```

Output:

```
Hello
Hello
```

C# String CopyTo()

- The C# CopyTo() method is used to copy a specified number of characters from the specified position in the string. It copies the characters of this string into a char array.
- Signature
  - public void CopyTo(int index, char[] ch, int start, int end)

C# String CopyTo()

- Parameter
  - index: it is an integer type parameter. It is an index of string.
  - ch: it is a char type array.
  - start: it is start index of char type array.
  - end: it is end index of char type array.

C# String CopyTo() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello C#, How Are You?";
         char[] ch = new char[15];
         s1.CopyTo(10,ch,0,12);
         Console.WriteLine(ch);
      }
   }
```

Output:

How Are You?

C# String EndsWith()

- The C# EndsWith() method is used to check whether the specified string matches the end of this string or not. If the specified string is found at the end of this string, it returns true otherwise false.
- Signature
  - public bool EndsWith(String str)
  - public bool EndsWith(String, Boolean, CultureInfo)
  - public bool EndsWith (String, StringComparison)?

C# String EndsWith()

- Parameters
  - str: it is a string object which is used to check the whether a specified string ends with it.

- Return
  - It returns boolean value either true or false.

C# String EndsWith() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "Hello";
        string s2 = "llo";
        string s3 = "C#";
        Console.WriteLine(s1.EndsWith(s2));
        Console.WriteLine(s1.EndsWith(s3));
    }
}
```

Output:

True
False

C# String Equals()

- The C# Equals() method is used to check whether two specified String objects have the same value or not. If both strings have same value, it return true otherwise false.

- In other words, it is used to compare two strings on the basis of content.

C# String Equals()

- Signature
  - public bool Equals(String str)
  - public static bool Equals(String, String)
  - public override bool Equals(Object)
  - public static bool Equals(String, String, StringComparison)
  - public bool Equals(String, StringComparison)
- Parameter
  - str: it is a string object.
- Return
  - It returns boolean value either true or false

C# String Equals() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello";
         string s2 = "Hello";
         string s3 = "Bye";
         Console.WriteLine(s1.Equals(s2));
         Console.WriteLine(s1.Equals(s3));
          Console.WriteLine(string.Equals(s1, s2));
      }
   }
```

Output:

True
False
True

C# String Format()

- The C# Format() method is used to replace one or more format items in the specified string with the string representation of a specified object.
- Signature
  - public static string Format (String first, Object second)
  - public static string Format(IFormatProvider, String, Object)
  - public static string Format(IFormatProvider, String, Object, Object)
  - public static string Format(IFormatProvider, String, Object, Object, Object)
  - public static string Format(IFormatProvider, String, Object[])
  - public static string Format(String, Object, Object)
  - public static string Format(String, Object, Object, Object)
  - public static string Format(String, params Object[])

C# String Equals()

- Parameters
  - first : it is a string type argument.
  - second: it is object type argument.
- Return
  - It returns a formatted string.

C# String Format() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {

      string s1 = string.Format("{0:D}", DateTime.Now);
      Console.WriteLine(s1);
    }
}
```

Output:

Saturday, December 17, 2016

C# String GetEnumerator()

- The C# GetEnumerator() method is used to convert string object into char enumerator. It returns instance of CharEnumerator. So, you can iterate string through loop.
- Signature
  - public CharEnumerator GetEnumerator()
- Parameter
  - It does not take any argument.
- Return
  - It returns System.CharEnumerator.

C# String GetEnumerator() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
      string s2 = "Hello C#";
      CharEnumerator ch = s2.GetEnumerator();
      while(ch.MoveNext()){
        Console.WriteLine(ch.Current);
      }
    }
}
```

Output:

H
e
l
l
o

C
#

C# String GetHashCode()

- The C# GetHashCode() method is used to get hash code of this string. It returns an integer value.
- Signature
  - public override int GetHashCode()
- Parameters
  - It does not take any parameter (argument).
- Return
  - It returns hash code of a string object.

C# String GetHashCode() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
        string s1 = "Hello C#";
        Console.WriteLine(s1.GetHashCode());
      }
   }
```

Output:

718576468

C# String GetType()

- The C# GetType() method is used to get type of current object. It returns the instance of Type class which is used for reflection.
- Signature
  - public Type GetType()
- Parameters
  - It does not take any parameter.
- Return
  - It returns object of Type class.

C# String GetType() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
      string s1 = "Hello C#";
      Console.WriteLine(s1.GetType());
    }
}
```

Output:

System.String

C# String GetTypeCode()

- The C# GetTypeCode() method is used to get type code of string. It returns the instance of TypeCode which is used to specify the type of an object.
- Signature
  - public TypeCode GetTypeCode()
- Parameters
  - It does not take any parameter.
- Return
  - It returns type code of string class.

C# String GetTypeCode() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
        string s1 = "Hello C#";
        Console.WriteLine(s1.GetTypeCode());
      }
   }
```

Output:

String

C# String IndexOf()

- The C# IndexOf() is used to get index of the specified character present in the string. It returns index as an integer value.
- Signature
  - public int IndexOf(Char ch)
  - public int IndexOf(Char, Int32)
  - public int IndexOf(Char, Int32, Int32)
  - public int IndexOf(String)
  - public int IndexOf(String, Int32)
  - public int IndexOf(String, Int32, Int32)
  - public int IndexOf(String, Int32, Int32, StringComparision)
  - public int IndexOf(String, Int32, StringComparision)
  - public int IndexOf(String, StringComparision)

C# String IndexOf()

- Parameters
  - ch: it is a character type parameter.

- Return
  - It returns an integer value.

C# String IndexOf() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
      string s1 = "Hello C#";
      int index = s1.IndexOf('e');
      Console.WriteLine(index);
    }
}
```

Output:

1

C# String Insert()

- The C# Insert() method is used to insert the specified string at specified index number. The index number starts from 0. After inserting the specified string, it returns a new modified string.

- Signature
  - public string Insert(Int32 first, String second)

C# String Insert()

- Parameters
  - first: It is used to pass as an index.
  - second: It is used to insert the given string at specified index.

- Return
  - It returns a new modified string.

C# String Insert() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
        string s1 = "Hello C#";
        string s2 = s1.Insert(5,"-");
        Console.WriteLine(s2);
      }
   }
```

Output:

Hello- C#

C# String Intern(String str)

- The C# Intern() method is used to retrieve reference to the specified String. It goes to intern pool (memory area) to search for a string equal to the specified String. If such a string exists, its reference in the intern pool is returned. If the string does not exist, a reference to specified String is added to the intern pool, then that reference is returned.
- Signature
  - The signature of intern method is given below:
  - public static string Intern(String str)
- Parameters
  - str: it is a parameter of type string.

C# String Intern() Method Example

```
using System;

    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello C#";
            string s2 = string.Intern(s1);
            Console.WriteLine(s1);
            Console.WriteLine(s2);
        }
    }
```

Output:

```
Hello C#
Hello C#
```

C# String IsInterned()

- The C# IsInterned() method is used to get reference of the specified string.
- The difference between Intern() and IsInterned() is that Intern() method interns the string if it is not interned but IsInterned() doesn't do so. In such case, IsInterned() method returns null.
- Signature
  - public static string IsInterned(String str)

C# String IsInterned()

- Parameter
  - str: it is a string type parameter.

- Return
  - It returns a reference.

C# String IsInterned() Method Example

```
using System;

  public class StringExample
  {
     public static void Main(string[] args)
     {
       string s1 = "Hello C#";
        string s2 = string.Intern(s1);
        string s3 = string.IsInterned(s1);
        Console.WriteLine(s1);
        Console.WriteLine(s2);
        Console.WriteLine(s3);
      }
  }
```

Output:

```
Hello C#
Hello C#
Hello C#
```

C# String Intern() vs IsInterned() Example

```
using System;

  public class StringExample
  {
    public static void Main(string[] args)
    {
      string a = new string(new[] {'a'});
      string b = new string(new[] {'b'});
      string.Intern(a); // Interns it
      Console.WriteLine(string.IsInterned(a) != null);//True
      string.IsInterned(b); // Doesn't intern it
      Console.WriteLine(string.IsInterned(b) != null);//False
    }
  }
```

Output:

True
False

C# String IsNormalized()

- The C# IsNormalized() method is used to check whether the string is in Unicode normalization form. It returns boolean value.

- Signature
  - public bool IsNormalized()
  - public bool IsNormalized(NormalizationForm)

C# String IsNormalized()

- Parameter
  - It does not take any parameter.

- Return
  - It returns boolean.

C# String IsNormalized() Method Example

```
using System;
using System.Text;

    public class StringExample
    {
        public static void Main(string[] args)
        {
          string s1 = "Hello C#";
           bool b1 = s1.IsNormalized();
           Console.WriteLine(s1);
           Console.WriteLine(b1);
        }
    }
```

Output:

Hello C#
True

C# String Normalize()

- The C# Normalize() method is used to get a new string whose textual value is same as this string, but whose binary representation is in Unicode normalization form.

- Signature
  - public string Normalize()
  - public string Normalize(NormalizationForm)

C# String Normalize()

- Parameter
  - First method does not take any parameter but second method takes a parameter of Normalization type.
- Return
  - It returns normalized string.

C# String Normalize() Method Example.

```
using System;
using System.Text;
 public class StringExample
    {
      public static void Main(string[] args)
      {
         string s1 = "Hello C#";
         string s2 = s1.Normalize();
         Console.WriteLine(s2);
      }
   }
```

Output:

Hello C#

C# String IsNullOrEmpty()

- The C# IsNullOrEmpty() method is used to check whether the specified string is null or an Empty string. It returns a boolean value either true or false.

- Signature
  - public static bool IsNullOrEmpty(String str)

C# String IsNullOrEmpty()

- Parameter
  - str: it is a string parameter which is used to check string.

- Return
  - It returns boolean value.

C# String IsNullOrEmpty() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
      string s1 = "Hello C#";
      string s2 = "";
      bool b1 = string.IsNullOrEmpty(s1);
      bool b2 = string.IsNullOrEmpty(s2);
      Console.WriteLine(b1);
      Console.WriteLine(b2);
    }
}
```

Output:

```
False
True
```

C# String IsNullOrWhiteSpace()

- The C# IsNullOrWhiteSpace() method is used to check whether the specified string is null, or consists only of white-space characters. It returns boolean value either True or False.

- Signature
  - public static bool IsNullOrWhiteSpace(String str)

C# String IsNullOrWhiteSpace()

- Parameter
  - str: it is a string parameter which is used to check null, white-space in string.

- Return
  - It returns boolean value.

C# String IsNullOrWhiteSpace() Method Example

```
using System;
   public class StringExample
    {
       public static void Main(string[] args)
       {
          string s1 = "Hello C#";
          string s2 = "";
          string s3 = " ";
          bool b1 = string.IsNullOrWhiteSpace(s1);
          bool b2 = string.IsNullOrWhiteSpace(s2);
          bool b3 = string.IsNullOrWhiteSpace(s3);
          Console.WriteLine(b1);      // returns False
          Console.WriteLine(b2);      // returns True
          Console.WriteLine(b3);      // returns True
     }
    }
```

Output:

```
False
True
True
```

C# String Join()

- The C# Join() methods is used to concatenate the elements of an array, using the specified separator between each element. It returns a modified string.
- Signature
  - [ComVisibleAttribute(false)]
  - public static string Join(String first, params String[] second)
  - [ComVisibleAttribute(false)]
  - public static string Joint(String, params Object[])
  - [ComVisibleAttribute(false)]
  - public static string Join (String, IEnumerable<String>)
  - [ComVisibleAttribute(false)]
  - public static string Join(String, String[], Int32, Int32)
  - [ComVisibleAttribute(false)]
  - public static string Join<T>(String, IEnumerable <T>)

C# String Join()

- Parameter
- first: it is a string type parameter.
- second: it is a string array.

- Return
- It returns a string.

C# String Join() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string[] s1 = {"Hello","C#","by",“World"};
         string s3 = string.Join("-",s1);
         Console.WriteLine(s3);
      }
   }
```

Output:

Hello-C#-by-World

C# String LastIndexOf()

- The C# LastIndexOf() method is used to find index position of the last occurrence of a specified character within String.
- Signature
- public int LastIndexOf(Char ch)
- public int LastIndexOf(Char, Int32)
- public int LastIndexOf(Char, Int32, Int32)
- public int LastIndexOf(String)
- public int LastIndexOf(String, Int32)
- public int LastIndexOf(String, Int32, Int32)
- public int LastIndexOf(String, Int32, Int32, StringComparison)
- public int LastIndexOf(String, Int32, StringComparison)
- public int LastIndexOf(String, StringComparison)

C# String LastIndexOf()

- Parameter
- ch: it is a character type parameter which is used to find last occurrence of given character within string.

- Return
- It returns integer value.

C# String LastIndexOf() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello C#";
         int index = s1.LastIndexOf('l');
         Console.WriteLine(index);
     }
   }
```

Output:

3

C# String IndexOf() vs LastIndexOf() Example

The IndexOf() method returns the index number of the first matched character whereas the LastIndexOf() method returns index number of the last matched character.

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello C#";
         int first = s1.IndexOf('l');
         int last = s1.LastIndexOf('l');
         Console.WriteLine(first);
         Console.WriteLine(last);
    }
   }
```

Output:

2
3

C# String LastIndexOfAny()

- The C# LastIndexOfAny() method is used to find index position of the last occurrence of one or more characters specified in this string.
- Signature
- public int LastIndexOfAny(Char[] ch)
- public int LastIndexOfAny(Char[], Int32)
- public int LastIndexOfAny(Char[], Int32, Int32)

C# String LastIndexOfAny()

- Parameter
- ch: it is a character type array.

- Return
- It returns integer value.

C# String LastIndexOfAny() Method Example

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "abracadabra";
        char[] ch = {'r','b'};
        int index = s1.LastIndexOfAny(ch);//Finds 'r' at the last
        Console.WriteLine(index);
    }
}
```

Output:

9

C# String LastIndexOfAny() Method Example 2

```
using System;
public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "abracadabra";
        char[] ch = {'t','b'};
        int index = s1.LastIndexOfAny(ch);//Finds 'b' at the last
        Console.WriteLine(index);
    }
}
```

Output:

8

C# String PadLeft()

- The C# PadLeft() method is used to get a new string that right-aligns the characters in this string if the string length is less than the specified length.

- For example, suppose you have "hello C#" as the string which has 8 length of characters and you are passing 10 for the padleft, it shifts the string at right side after two whitespaces. It means PadLeft() method provides padding to the string for the specified length. It is used for formatting string content.

C# String PadLeft()

- Signature
  - public string PadLeft(Int32 length)
  - public string PadLeft(Int32, Char)
- Parameter
  - length: it is an integer type parameter which is used to pass padding.
- Return
  - It returns string.

C# String PadLeft() Method Example

```
using System;

    public class StringExample
    {
        public static void Main(string[] args)
        {
        string s1 = "Hello C#";// 8 length of characters
        string s2 = s1.PadLeft(10);
//(10-8=2) adds 2 whitespaces at the left side
        Console.WriteLine(s2);
      }
    }
```

Output:

_ _Hello C#

C# String PadRight()

- The C# PadRight() method is used to get a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.

- Signature
  - public string PadRight(Int32 length)
  - public string PadRight(Int32, Char)

C# String PadRight()

- Parameter
  - length: it is an integer type parameter.

- Return
  - It returns a string.

C# String PadRight() Method Example

```
using System;

  public class StringExample
  {
    public static void Main(string[] args)
    {
      string s1 = "Hello C#";// 8 length of characters
      string s2 = s1.PadRight(15);
      Console.Write(s2);//padding at right side (15-8=7)
      Console.Write("World");//will be written after 7 white spaces

    }
  }
```

Output:

Hello C# World

C# String Remove()

- The C# Remove() method is used to get a new string after removing all the characters from specified beginIndex till given length. If length is not specified, it removes all the characters after beginIndex.
- Signature
  - public string Remove(Int32 beginIndex)
  - public string Remove(Int32 beginIndex, Int32 length)

C# String Remove()

- Parameter
  - index: it is an integer type parameter.

- Return
  - It returns a string.

C# String Remove() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello C#";
         string s2 = s1.Remove(2);
         string s3 = s1.Remove(2,2);
         Console.WriteLine(s2);
         Console.WriteLine(s3);
      }
   }
```

Output:

He
Heo C#

C# String Remove() Method Example 2

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
        string s1 = "abcdefghijk";
        string s2 = s1.Remove(4, 5);
        Console.WriteLine(s2);
       }
    }
```

Output:

abcdjk

C# String Replace()

- The C# Replace() method is used to get a new string in which all occurrences of a specified Unicode character in this string are replaced with another specified Unicode character.

- There are two methods of Replace() method. You can replace string also.

C# String Replace()

- Signature
  - public string Replace(Char first, Char second)
  - public string Replace(String firstString, String secondString)
- Parameter
  - first: it is a first parameter of char type.
  - second: it is a second parameter of char type.
- Return
  - It returns a string.

C# String Replace() Method Example

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello F#";
            string s2 = s1.Replace('F','C');
            Console.WriteLine(s2);
        }
    }
```

Output:

Hello C#

C# String Replace() Method Example 2

Output:

Cheers C#, Cheers .Net, Cheers Class

```
using System;
    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello C#, Hello .Net, Hello Class";
            string s2 = s1.Replace("Hello","Cheers");
            Console.WriteLine(s2);
        }
    }
```

C# String Split()

- The C# Split() method is used to split a string into substrings on the basis of characters in an array. It returns string array.
- Signature
- public string[] Split(params Char[] ch)
- public string[] Split(Char[], Int32)
- [ComVisibleAttribute(false)]
- public string[] Split(Char[], Int32, StringSplitOptions)
- [ComVisibleAttribute(false)]
- public string[] Split(Char[], StringSplitOptions)
- [ComVisibleAttribute(false)]
- public string[] Split(String[], Int32, StringSplitOptions)
- public string[] Split(String[], StringSplitOptions)

C# String Split()

- Parameter
  - ch: it is a character type array.

- Return
  - It returns array of string

C# String Split() Method Example

```
using System;
   public class StringExample
   {
      public static void Main(string[] args)
      {
         string s1 = "Hello C Sharp";
         string[] s2 = s1.Split(' ');
         foreach (string s3 in s2)
         {
          Console.WriteLine(s3);
         }
       }
    }
```

Output:

Hello
C
Sharp

C# String StartsWith()

- The C# StartsWith() method is used to check whether the beginning of this string instance matches the specified string.

- Signature
  - public bool StartsWith(String str)
  - public bool StartsWith(String, Boolean, CultureInfo)
  - public bool StartsWith(String, StringComparison)

C# String StartsWith()

- Parameter
- str: it is string type parameter which is used to check beginning of string.

- Return
- It returns boolean value.

C# String StartsWith() Method Example

```
using System;

public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "Hello C Sharp";
        bool b1 = s1.StartsWith("h");
        bool b2 = s1.StartsWith("H");
        Console.WriteLine(b1);
        Console.WriteLine(b2);
    }
}
```

Output:

```
False
True
```

C# String SubString()

- The C# SubString() method is used to get a substring from a String. The substring starts at a specified character position and continues to the end of the string.

- Signature
  - public string Substring(Int32 index)
  - public string Substring(Int32, Int32)

C# String SubString()

- Parameter
  - index: it is an integer type parameter which is used to pass index to get a substring.

- Return
  - It returns a string.

C# String SubString() Method Example

```
using System;

    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello C Sharp";
            string s2 = s1.Substring(5);
            Console.WriteLine(s2);
        }
    }
```

Output:

C Sharp

C# String ToCharArray()

- The C# ToCharArray() method is used to get character array from a string object.

- Signature
  - public char[] ToCharArray()
  - public char[] ToCharArray(Int32, Int32)

C# String ToCharArray()

- Parameter
  - First method does not take any parameter while second method takes two integer parameters.

- Return
  - It returns a character array.

C# String ToCharArray() Method Example

```
using System;

public class StringExample
{
    public static void Main(string[] args)
    {
      string s1 = "Hello C#";
      char[] ch = s1.ToCharArray();
      foreach(char c in ch){
      Console.WriteLine(c);
     }
    }
}
```

Output:

```
H
e
l
l
o

C
#
```

C# String ToLower()

- The C# ToLower() method is used to convert a string into lowercase. It returns a string in lower case.

- Signature
  - public string ToLower()
  - public string ToLower(CultureInfo)

C# String ToLower()

- Parameter
  - First method does not take any parameter.
- Return
  - It returns a string.

C# String ToLower() Method Example

```
using System;

    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello C#";
            string s2 = s1.ToLower();
            Console.WriteLine(s2);
        }
    }
```

Output:

hello c#

C# String ToLowerInvariant()

- The C# ToLowerInvariant() method is used to converted a string into lowercase using the casing rules of the invariant culture.

- Signature
  - public string ToLowerInvariant()

C# String ToLowerInvariant()

- Parameter
  - It does not take any parameter.
- Return
  - It returns a string.

C# String ToLowerInvariant() Method Example

```
using System;

    public class StringExample
    {
        public static void Main(string[] args)
        {
            string s1 = "Hello C#";
            string s2 = s1.ToLowerInvariant();
            Console.WriteLine(s2);
        }
    }
```

Output:

hello c#

C# String ToString()

- The C# ToString() method is used to get instance of String.

- Signature
  - public override string ToString()
  - public string ToString(IFormatProvider)

C# String ToLowerInvariant()

- Parameter
  - It does not any parameter.

- Return
  - It returns a string object.

C# String ToString() Method Example

```
using System;

  public class StringExample
  {
     public static void Main(string[] args)
     {
        string s1 = "Hello C#";
        int a = 123;
        string s2 = s1.ToString();
        string s3 = a.ToString();
        Console.WriteLine(s2);
        Console.WriteLine(s3);
     }
  }
```

Output:

```
Hello C#
123
```

C# String ToUpper()

- The C# ToUpper() method is used to convert string into uppercase. It returns a string.

- Signature
  - public string ToUpper()
  - public string ToUpper(CultureInfo)

C# String ToLowerInvariant()

- Parameter
  - First method does not take any parameter.
- Return
  - It returns a string

C# String ToUpper() Method Example

```
using System;

public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "Hello C#";
        string s3 = s1.ToUpper();
        Console.WriteLine(s3);
    }
}
```

Output:

HELLO C#

C# String ToUpperInvariant()

- The C# ToUpperInvariant() method is used to convert string into uppercase string using the casing rules of the invariant culture.

- Signature
  - public string ToUpperInvariant()

C# String ToUpperInvariant()

- Parameter
  - It does not take any parameter
- Return
  - It returns a string.

C# String ToUpperInvariant() Method Example

```
using System;

  public class StringExample
  {
     public static void Main(string[] args)
     {
        string s1 = "Hello C#";
        string s3 = s1.ToUpperInvariant();
        Console.WriteLine(s3);
      }
  }
```

Output:

HELLO C#

C# String Trim()

- The C# Trim() method is used to remove all leading and trailing white-space characters from the current String object.

- Signature
  - public string Trim()
  - public string Trim(params Char[])

C# String Trim()

- Parameter
  - First method does not take any parameter. Second method takes a char array as parameter.

- Return
  - It returns a string.

C# String Trim() Method Example

```
using System;

public class StringExample
{
    public static void Main(string[] args)
    {
        string s1 = "Hello C#";
        string s2 = s1.Trim();
        Console.WriteLine(s2);
    }
}
```

Output:

Hello C#

C# String TrimEnd()

- The C# TrimEnd() method is used to remove all trailing occurrences of a set of characters specified in an array from the current String object.

- Signature
  - public string TrimEnd(params Char[] ch)

C# String TrimEnd()

- Parameter
  - ch: It takes a char array as parameter.

- Return
  - It returns a string.

C# String TrimEnd() Method Example

```
using System;

  public class StringExample
  {
     public static void Main(string[] args)
     {
      string s1 = "Hello C#";
      char[] ch = {'#'};
      string s2 = s1.TrimEnd(ch);
      Console.WriteLine(s2);
      }
  }
```

Output:

Hello C

C# String TrimStart()

- The C# TrimStart() method id used to remove all leading occurrences of a set of characters specified in an array from the current String object.

- Signature
  - public string TrimStart(params Char[] ch)

C# String TrimStart()

- Parameter
  - ch: it is a char array type parameter.

- Return
  - It returns a string

C# String TrimStart() Method Example

```
using System;
   public class StringExample
   {
       public static void Main(string[] args)
       {
        string s1 = "Hello C#";
        char[] ch = {'H'};
        string s2 = s1.TrimStart(ch);
        Console.WriteLine(s2);
       }
   }
```

Output:

ello C#

C# Object and Class

Outline

- C# Object and Class
- C# Constructor
- C# Destructor
- C# this
- C# static
- C# static class
- C# static constructor
- C# Structs
- C# Enum

C# Object and Class

- Since C# is an object-oriented language, program is designed using objects and classes in C#.

- C# Object
  - In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
  - In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.

C# Object and Class (Contd.)

- Object is a runtime entity, it is created at runtime.
- Object is an instance of a class. All the members of the class can be accessed through object.
- Let's see an example to create object using new keyword.
- Student s1 = new Student();//creating an object of Student
- In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class. The new keyword allocates memory at runtime.

C# Object and Class (Contd.)

- C# Class
  - In C#, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.
  - Let's see an example of C# class that has two fields only.

```
public class Student   {
    int id;//field or data member
    String name;//field or data member
 }
```

Output:
101 Sonoo Jaiswal

C# Object and Class Example

- Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

Output:

101
Anders Hejlsberg

```
using System;
  public class Student    {
      int id;//data member (also instance variable)
      String name;//data member(also instance variable)
  public static void Main(string[] args)  {
          Student s1 = new Student();//creating an object of Student
          s1.id = 101;
          s1.name = "Anders Hejlsberg";
          Console.WriteLine(s1.id);
          Console.WriteLine(s1.name);
       }
  }
```

C# Class Example 2: Having Main() in another class

- Let's see another example of class where we are having Main() method in another class. In such case, class must be public.

```
using System;
  public class Student  {
      public int id;
      public String name;
  }
  class TestStudent{
     public static void Main(string[] args)  {
         Student s1 = new Student();
         s1.id = 101;
         s1.name = "Anders Hejlsberg";
         Console.WriteLine(s1.id);
         Console.WriteLine(s1.name);
       }
  }
```

Output:

```
101
Anders Hejlsberg
```

C# Class Example 3: Initialize and Display data through method

- Let's see another example of C# class where we are initializing and displaying object through method.

```
using System;
  public class Student  {
      public int id;
      public String name;
      public void insert(int i, String n)  {
         id = i;
         name = n;
      }
      public void display()  {
         Console.WriteLine(id + " " + name);
      }
  }
```

```
class TestStudent{
    public static void Main(string[] args)
{
        Student s1 = new Student();
        Student s2 = new Student();
        s1.insert(101, "Ajeet");
        s2.insert(102, "Tom");
        s1.display();
        s2.display();
    }
}
```

Output:

101 Ajeet
102 Tom

C# Class Example 4: Store and Display Employee Information

```
using System;
  public class Employee
   {
      public int id;
      public String name;
      public float salary;
      public void insert(int i, String n,
float s)  {
         id = i;
         name = n;
         salary = s;
      }
      public void display()  {
         Console.WriteLine(id + " " +
name+" "+salary);
      }
   }
```

```
class TestEmployee{
      public static void Main(string[] args)
        {
          Employee e1 = new Employee();
           Employee e2 = new Employee();
            e1.insert(101, “Ram",890000f);
          e2.insert(102, “Shyam", 490000f);
          e1.display();
          e2.display();
        }
    }
```

Output:

101 Ram 890000
102 Shyam 490000

C# Constructor

- In C#, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C# has the same name as class or struct.

- There can be two types of constructors in C#.

  - Default constructor
  - Parameterized constructor

C# Default Constructor Example: Having Main() within class

```
using System;
  public class Employee
   {
      public Employee()
      {
         Console.WriteLine("Default Constructor Invoked");
      }
      public static void Main(string[] args)
      {
         Employee e1 = new Employee();
         Employee e2 = new Employee();
      }
   }
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C# Default Constructor Example: Having Main() in another class

- Let's see another example of default constructor where we are having Main() method in another class.

```
using System;
  public class Employee
   {
      public Employee()
      {
         Console.WriteLine("Default Constructor Invoked");
      }
   }
  class TestEmployee{
     public static void Main(string[] args)
      {
         Employee e1 = new Employee();
         Employee e2 = new Employee();
      }
   }
```

Output:

```
Default Constructor Invoked
Default Constructor Invoked
```

C# Parameterized Constructor

- A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

```
using System;
  public class Employee {
     public int id;
     public String name;
     public float salary;
     public Employee(int i,
     String n,float s)  {
        id = i;
        name = n;
        salary = s;  }
     public void display()  {
        Console.WriteLine(id + " " +
          name+" "+salary);
     }  }
```

```
  class TestEmployee{
      public static void Main(string[] args
)
      {
          Employee e1 = new Employee(1
01, “Ram", 890000f);
          Employee e2 = new Employee(1
02, “Shyam", 490000f);
          e1.display();
          e2.display();
      }
  }
```

Output:

101 Ram 890000
102 Shyam 490000

C# Destructor

- A destructor works opposite to constructor, It destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

Note:

- C# destructor cannot have parameters. Moreover, modifiers can't be applied on destructors.
- Destructor can't be public. We can't apply any modifier on destructors.

C# Constructor and Destructor Example

- Let's see an example of constructor and destructor in C# which is called automatically.

```
using System;
  public class Employee  {
      public Employee()  {
          Console.WriteLine("Constructor Invoked");
      }
      ~Employee()  {
          Console.WriteLine("Destructor Invoked");
      }
  }
  class TestEmployee{
      public static void Main(string[] args)  {
          Employee e1 = new Employee();
          Employee e2 = new Employee();
      }
  }
```

Output:
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked

C# this

- In c# programming, this is a keyword that refers to the current instance of the class.
- here can be 3 main usage of this keyword in C#.
  - It can be used to refer current class instance variable. It is used if field names (instance variables) and parameter names are same, that is why both can be distinguish easily.
  - It can be used to pass current object as a parameter to another method.
  - It can be used to declare indexers.

C# this example

```
using System;
  public class Employee  {
      public int id;
      public String name;
      public float salary;
      public Employee(int id, String name,float salary)  {
         this.id = id;
         this.name = name;
         this.salary = salary;  }
      public void display()  {
         Console.WriteLine(id + " " + name+" "+salary);
      } }
  class TestEmployee{
     public static void Main(string[] args)  {
         Employee e1 = new Employee(101, “Ram", 890000f);
         Employee e2 = new Employee(102, “Shyam", 490000f);
         e1.display();
         e2.display();
      } }
```

Output:
101 Ram 890000
102 Shyam 490000

C# static

- In C#, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C#, static can be field, method, constructor, class, properties, operator and event.
- Note: Indexers and destructors cannot be static.
- Advantage of C# static keyword
  - Memory efficient: Now we don't need to create instance for accessing the static members, so it saves memory. Moreover, it belongs to the type, so it will not get memory each time when instance is created

C# Static Field

- A field which is declared as static, is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.

- It is used to refer the common property of all objects such as rateOfInterest in case of Account, companyName in case of Employee etc.

C# static field example

```
using System;
  public class Account  {
      public int accno;
      public String name;
      public static float rateOfInterest=8.8f;
      public Account(int accno, String name)  {
         this.accno = accno;
         this.name = name;
      }
      public void display()  {
         Console.WriteLine(accno + " " + name + " " + rateOfInterest);
      }  }
  class TestAccount{
     public static void Main(string[] args)  {
       Account a1 = new Account(101, “Ram");
         Account a2 = new Account(102, “Shyam");
         a1.display();
         a2.display();
      }  }
```

Output:

101 Ram 8.8
102 Shyam 8.8

C# static field example 2: changing static field

```
using System;
  public class Account  {
      public int accno;
      public String name;
      public static float rateOfInterest=8.8f;
      public Account(int accno, String name)  {
          this.accno = accno;
          this.name = name; }
      public void display()  {
          Console.WriteLine(accno + " " + name + " " + rateOfInterest);
      }  }
  class TestAccount{
      public static void Main(string[] args)  {
          Account.rateOfInterest = 10.5f;//changing value
          Account a1 = new Account(101, “Ram");
          Account a2 = new Account(102, “Shyam");
          a1.display();
          a2.display();  } }
```

Output:

```
101 Ram 10.5
102 Shyam 10.5
```

C# static field example 3: Counting Objects

```
using System;
  public class Account  {
      public int accno;
      public String name;
      public static int count=0;
      public Account(int accno, String name)  {
          this.accno = accno;
          this.name = name;
          count++;  }
      public void display()  {
          Console.WriteLine(accno + " " + name);  }  }
  class TestAccount{
     public static void Main(string[] args)  {
          Account a1 = new Account(101, “Ram");
          Account a2 = new Account(102, “Shyam");
          Account a3 = new Account(103, “Rohan");
          a1.display();  a2.display();  a3.display();
          Console.WriteLine("Total Objects are: "+Account.count);
     }  }
```

Output:

101 Ram
102 Shyam
103 Rohan
Total Objects are: 3

C# static class

- The C# static class is like the normal class but it cannot be instantiated. It can have only static members. The advantage of static class is that it provides you guarantee that instance of static class cannot be created.
- Points to remember for C# static class
  - C# static class contains only static members.
  - C# static class cannot be instantiated.
  - C# static class is sealed.
  - C# static class cannot contain instance constructors.

C# static class example

```
using System;
  public static class MyMath
   {
      public static float PI=3.14f;
      public static int cube(int n){return n*n*n;}
   }
  class TestMyMath{
     public static void Main(string[] args)
      {
         Console.WriteLine("Value of PI is: "+MyMath.PI);
         Console.WriteLine("Cube of 3 is: " + MyMath.cube(3));
      }
   }
```

Output:

Value of PI is: 3.14
Cube of 3 is: 27

C# static constructor

- C# static constructor is used to initialize static fields. It can also be used to perform any action that is to be performed only once. It is invoked automatically before first instance is created or any static member is referenced.

- Points to remember for C# Static Constructor
  - C# static constructor cannot have any modifier or parameter.
  - C# static constructor is invoked implicitly. It can't be called explicitly.

C# Static Constructor example

- Let's see the example of static constructor which initializes the static field rateOfInterest in Account class.

```
using System;
  public class Account  {
      public int id;
      public String name;
      public static float rateOfInterest;
      public Account(int id, String name)  {
          this.id = id;
          this.name = name;  }
      static Account()  {
          rateOfInterest = 9.5f;
      }
      public void display()  {
          Console.WriteLine(id + " " + name+" "+rateOfInterest);
      }
  }
```

```
class TestEmployee{
    public static void Main
(string[] args) {
        Account a1 = new
Account(101, “Ram");
        Account a2 = new
Account(102, “Shyam");
        a1.display();
        a2.display();
    }
}
```

Output:

```
101 Ram 9.5
102 Shyam 9.5
```

C# Structs

- In C#, classes and structs are blueprints that are used to create instance of a class. Structs are used for lightweight objects such as Color, Rectangle, Point etc.

- Unlike class, structs in C# are value type than reference type. It is useful if you have data that is not intended to be modified after creation of struct..

C# Struct Example

Let's see a simple example of struct Rectangle which has two data members width and height.

```
using System;
public struct Rectangle  {
   public int width, height;
  }
public class TestStructs  {
   public static void Main()
   {
      Rectangle r = new Rectangle();
      r.width = 4;
      r.height = 5;
      Console.WriteLine("Area of Rectangle is: " + (r.width * r.height));
    }
}
```

Output:

Area of Rectangle is: 20

C# Struct Example: Using Constructor and Method

Let's see another example of struct where we are using constructor to initialize data and method to calculate area of rectangle.

```
using System;
public struct Rectangle  {
   public int width, height;
   public Rectangle(int w, int h) {
      width = w;
      height = h;   }
   public void areaOfRectangle() {
    Console.WriteLine("Area of Rectangle is: "+(width*height)); }  }
public class TestStructs  {
   public static void Main() {
      Rectangle r = new Rectangle(5, 6);
      r.areaOfRectangle();
   }
}
```

Output:

Area of Rectangle is: 30

Note: Struct doesn't support inheritance. But it can implement interfaces

C# Enum

- Enum in C# is also known as enumeration. It is used to store a set of named constants such as season, days, month, size etc. The enum constants are also known as enumerators. Enum in C# can be declared within or outside class and structs.
- Enum constants has default values which starts from 0 and incremented to one by one. But we can change the default value.
- Points to remember
  - enum has fixed set of constants
  - enum improves type safety
  - enum can be traversed

C# enum example changing start index

```
using System;
public class EnumExample
{
    public enum Season { WINTER=10, SPRING, SUMMER, FALL }

    public static void Main()
    {
        int x = (int)Season.WINTER;
        int y = (int)Season.SUMMER;
        Console.WriteLine("WINTER = {0}", x);
        Console.WriteLine("SUMMER = {0}", y);
    }
}
```

Output:

WINTER = 10
SUMMER = 12

C# enum example for Days

```
using System;
public class EnumExample
{
    public enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    public static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Mon;
        int z = (int)Days.Sat;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Mon = {0}", y);
        Console.WriteLine("Sat = {0}", z);
    }
}
```

Output:

Sun = 0
Mon = 1
Sat = 6

C# enum example: traversing all values using getNames()

```
using System;
public class EnumExample
{
    public enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    public static void Main()
    {
        foreach (string s in Enum.GetNames(typeof(Days)))
        {
            Console.WriteLine(s);
        }
    }
}
```

Output:

```
Sun
Mon
Tue
Wed
Thu
Fri
Sat
```

C# enum example: traversing all values using getValues()

```
using System;
public class EnumExample
{
    public enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    public static void Main()
    {
        foreach (Days d in Enum.GetValues(typeof(Days)))
        {
            Console.WriteLine(d);
        }
    }
}
```

Output:

```
Sun
Mon
Tue
Wed
Thu
Fri
Sat
```