.NET Day 1

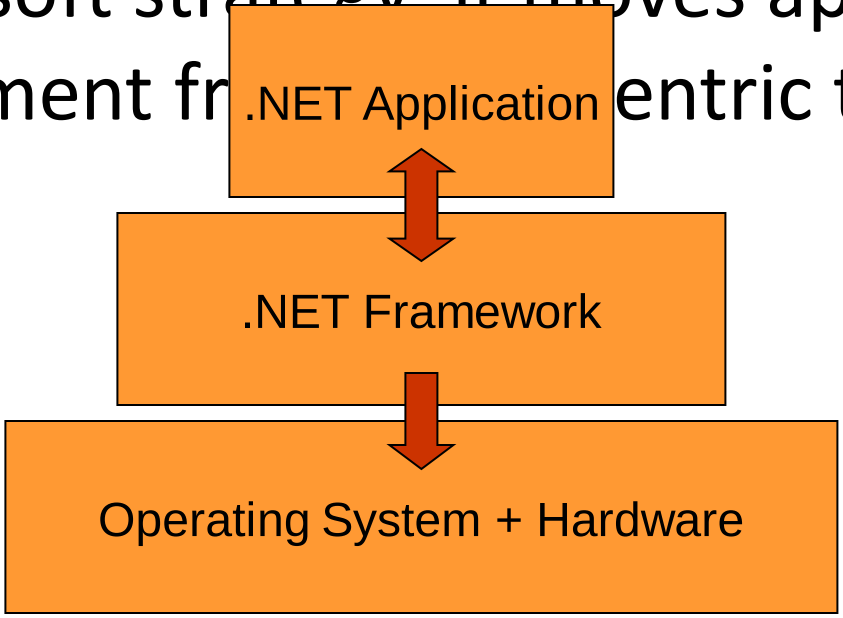
Yogesh Yadav

.NET

- .NET Framework (pronounced dot net) is a software framework developed by Microsoft that runs primarily on Microsoft Windows.
- 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)

- .Net is a new framework for developing console, window, web ,services and mobile-based applications within the Microsoft environment.
- The framework offers a fundamental shift in Microsoft strategy: it moves application development fr entric to server-centric.

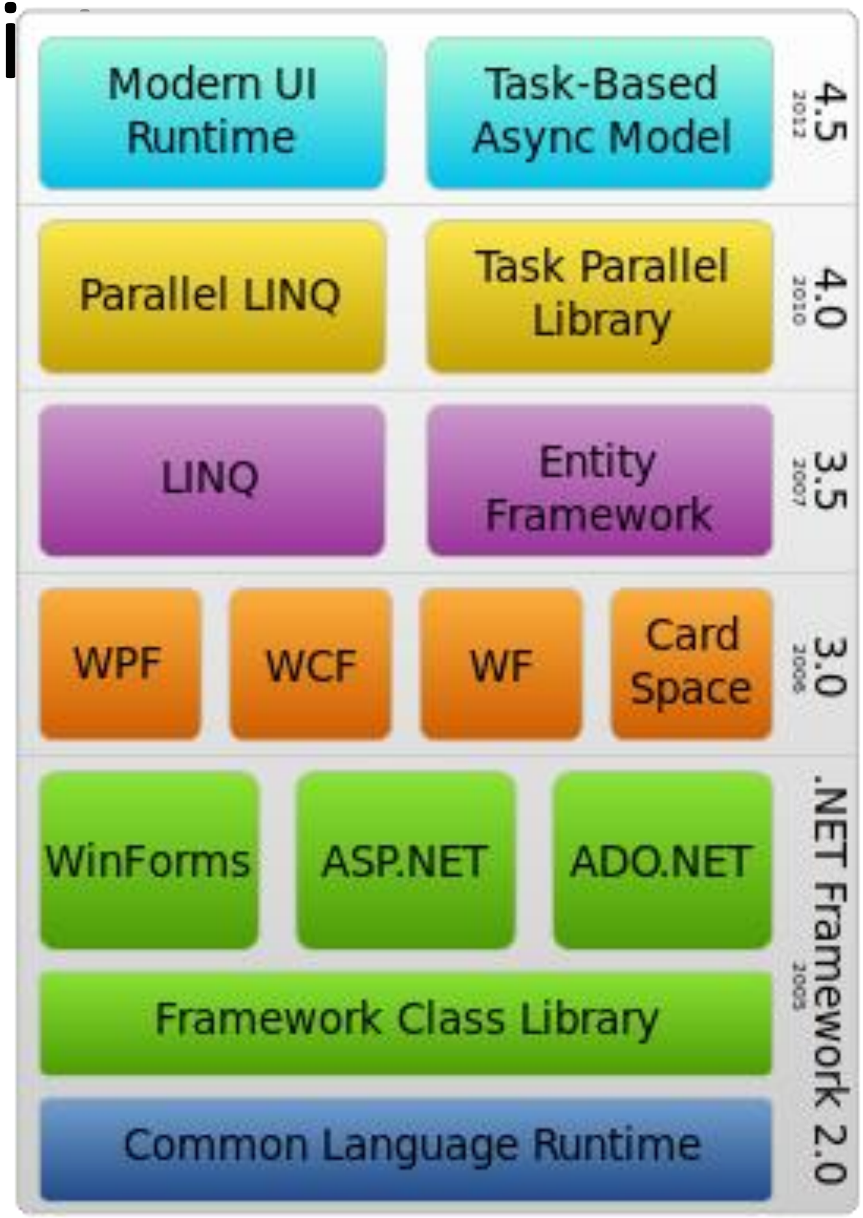


What is .NET?

- **What is .NET?**
  .Net is software development platform to build, test, run and host software applications. It is based on Virtual machine Architecture. In which High level languages like C# Vb.net Codes get compiled into CIL code that are OS, hardware independent and this Platform independent codes are converted into native or executable code through JIT compiler.

Hi

- Development began in 1998
- Beta 1 released Oct, 2000
- Beta 2 released July, 2001
- Finalized in Dec, shipping in Feb 2002
- Vista ships with .NET Framework 3.0 (Runtime)



Overview of .NET Framework release history[15][16]

Version number	CLR version	Release date	Development tool	Included in		Replaces
				Windows	Windows Server	
1.0	1.0	2002-02-13	Visual Studio .NET[17]	XP[a]	N/A	N/A
1.1	1.1	2003-04-24	Visual Studio .NET 2003[17]	N/A	2003	1.0[18]
2.0	2.0	2005-11-07	Visual Studio 2005[19]	N/A	2003, 2003 R2,[20] 2008 SP2, 2008 R2 SP1	N/A
3.0	2.0	2006-11-06	Expression Blend[21][b]	Vista	2008 SP2, 2008 R2 SP1	2.0[15]
3.5	2.0	2007-11-19	Visual Studio 2008[22]	7, 8[c], 8.1[c], 10[c]	2008 R2 SP1	2.0, 3.0[15]
4.0	4	2010-04-12	Visual Studio 2010[23]	N/A	N/A	N/A
4.5	4	2012-08-15	Visual Studio 2012[24]	8	2012	4.0[15]
4.5.1	4	2013-10-17	Visual Studio 2013[25]	8.1	2012 R2	4.0, 4.5[15]
4.5.2	4	2014-05-05	N/A	N/A	N/A	4.0, 4.5, 4.5.1[15]
4.6	4	2015-07-20	Visual Studio 2015[26]	10	2016	4.0, 4.5, 4.5.1, 4.5.2

OOPS

- If we see this world with perspective of OOPs, Every thing in this world is treated as object. With OOPs we get power to create Objects of our own and use them according to the need.

- **Class:**

  A Class is abstract model which defines a new abstract data type in a programming language, so that we can re-use abstract data type with the help of objects that are instance of class.

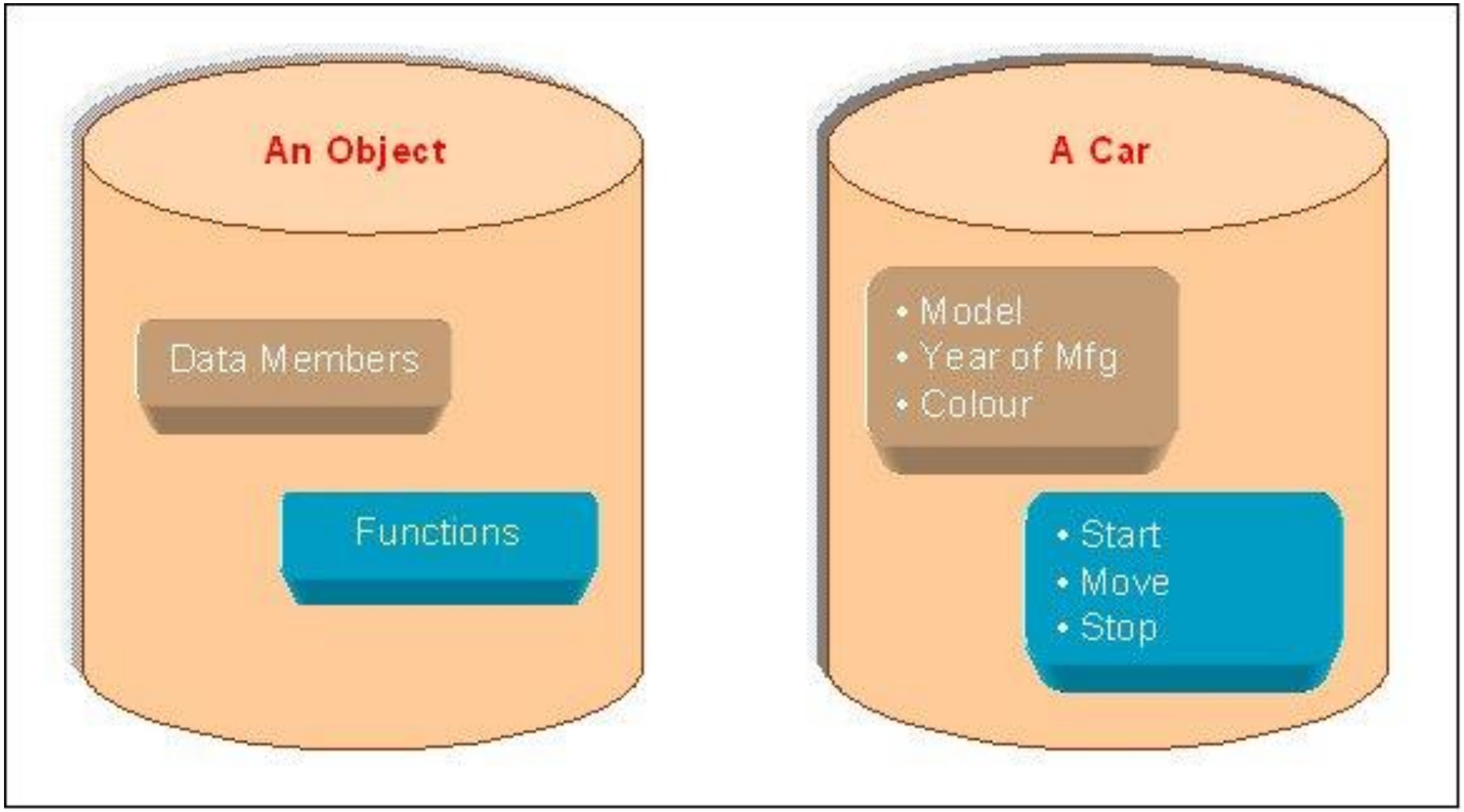
Class

- A class is the core of any modern Object Oriented Programming language such as C#.
- In OOP languages it is must to create a class for representing data.
- Class is a blueprint of an object that contains variables for storing data and functions to performing operations on these data.
- Class will not occupy any memory space and hence it is only logical

  representation of data.
- To create a class, you simply use the keyword "class" followed by the class name:
- class Employee
- {
-
- }

Object

- Objects are the basic run-time entities in an object oriented system. They may represent a person, a place or any item that the program has to handle.
  Object is a Software bundle of related variable and methods.
- Object is an instance of a class
- Class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, which is called as an object.
- When an object is created by using the keyword **new**, then memory will be allocated for the class in heap memory area, which is called as an instance and its starting address will be stored in the object in stack memory area.
- When an object is created without the keyword new, then memory will not be allocated in heap I.e. instance will not be created and object in the stack contains the value **null.**
- When an object contains null, then it is not possible to access the members of the class using that object.


An Object
Data Members
Functions
A Car
• Model
• Year of Mfg
• Colour
• Start
• Move
• Stop

- class Employee
- {

- }
- Syntax to create an object of class Employee:-

- Employee objEmp = new Employee();

Abstraction

- Abstraction is "To represent the essential feature without representing the back ground details."

- Abstraction lets you focus on what the object does instead of how it does it.

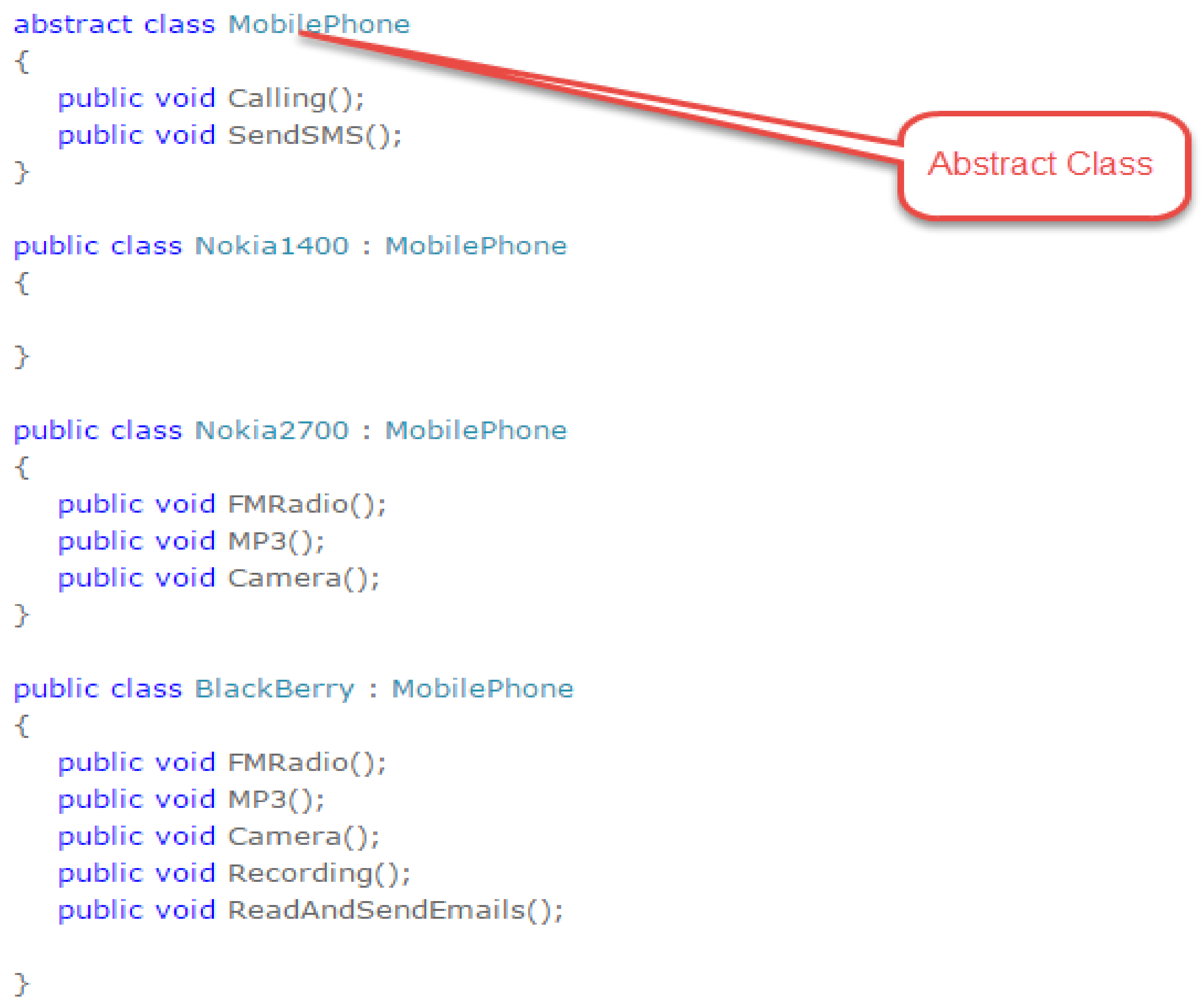
- Abstraction provides you a generalized view of your classes or object by providing relevant information.

- Abstraction is the process of hiding the working style of an object, and showing the information of an object in understandable manner.

- **Real world Example of Abstraction:** -
- Suppose you have an object Mobile Phone.
- Suppose you have 3 mobile phones as following:-
- Nokia 1400 (Features:- Calling, SMS)
- Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)
- Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)

- Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS."

so that, for mobile phone object you will have abstract class like following:-



```
abstract class MobilePhone
{
    public void Calling();
    public void SendSMS();
}

public class Nokia1400 : MobilePhone
{

}

public class Nokia2700 : MobilePhone
{
    public void FMRadio();
    public void MP3();
    public void Camera();
}

public class BlackBerry : MobilePhone
{
    public void FMRadio();
    public void MP3();
    public void Camera();
    public void Recording();
    public void ReadAndSendEmails();

}
```

Abstraction means putting all the variables and methods in a class which are necessary.

For example: - Abstract class and abstract method.

Abstraction is the common thing.

example:

If somebody in your collage tell you to fill application form, you will fill your details like name, address, data of birth, which semester, percentage you have got etc.

If some doctor gives you an application to fill the details, you will fill the details like name, address, date of birth, blood group, height and weight.

See in the above example what is the common thing?

Age, name, address so you can create the class which consist of common thing that is called abstract class.

That class is not complete and it can inherit by other class.

Encapsulation

Wrapping up data member and method together into a single unit (i.e. Class) is called Encapsulation.

- Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.

- Encapsulation is like your bag in which you can keep your pen, book etc. It means this is the property of encapsulating members and functions.

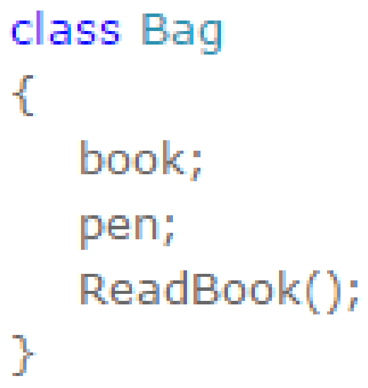
- **Abstraction** focuses on the outside view of an object (i.e. the interface)**Encapsulation** (information hiding) prevents clients from seeing its inside view, where the behavior of the abstraction is implemented.

- Encapsulation means hiding the internal details of an object, i.e. how an object does something.

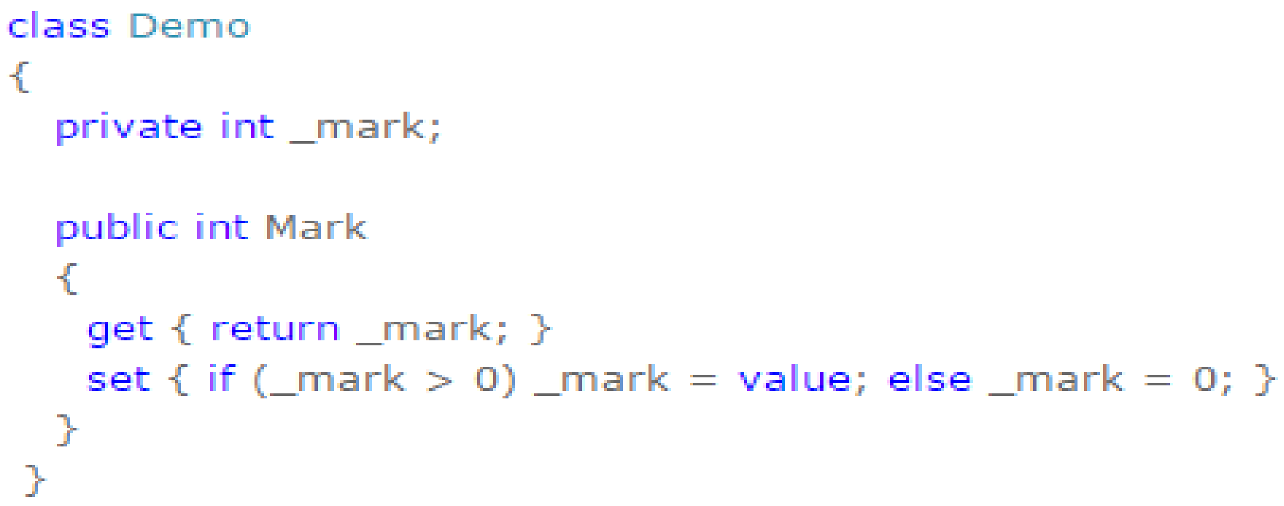
- Encapsulation prevents clients from seeing it inside view, where the behaviour of the abstraction is implemented.

- Encapsulation is a technique used to protect the information in an object from the other object.

- Hide the data for security such as making the variables as private, and expose the property to access the private data which would be public.
- So, when you access the property you can validate the data and set it.



```
class Bag
{
    book;
    pen;
    ReadBook();
}
```



```
class Demo
{
  private int _mark;

  public int Mark
  {
    get { return _mark; }
    set { if (_mark > 0) _mark = value; else _mark = 0; }
  }
 }
```

- Let's take example of Mobile Phone and Mobile Phone Manufacturer
- Suppose you are a Mobile Phone Manufacturer and you designed and developed a Mobile Phone design(class), now by using machinery you are manufacturing a Mobile Phone(object) for selling, when you sell your Mobile Phone the user only learn how to use the Mobile Phone but not that how this Mobile Phone works.

- This means that you are creating the class with function and by making object (capsule) of it you are making availability of the functionality of you class by that object and without the interference in the original class.

Inheritance:

- When a class acquire the property of another class is known as inheritance.
- Inheritance is process of object reusability.
- For example, A Child acquire property of Parents.

```
public class ParentClass
    {
        public ParentClass()
        {
            Console.WriteLine("Parent Constructor.");
        }

        public void print()
        {
            Console.WriteLine("I'm a Parent Class.");
        }
    }

    public class ChildClass : ParentClass
    {
        public ChildClass()
        {
            Console.WriteLine("Child Constructor.");
        }

        public static void Main()
        {
            ChildClass child = new ChildClass();

            child.print();
        }
    }
```

Output:

```
Parent Constructor.
Child Constructor.
I'm a Parent Class.
```

Polymorphism

- Polymorphism means **one name many forms**.
- One function behaves different forms.
- In other words, "Many forms of a single object is called Polymorphism.“

Example-1:

A Teacher behaves to student.

A Teacher behaves to his/her seniors.

Here teacher is an object but attitude is different in different situation.

Example-2:

Person behaves SON in house at the same time that person behaves EMPLOYEE in office.

Example-3:

Your mobile phone, one name but many forms

As phone

As camera

As mp3 player

As radio

Difference between Abstraction and Encapsulation

Abstraction	Encapsulation
1. Abstraction solves the problem in the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction is used for hiding the unwanted data and giving relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. **Abstraction**- Outer layout, used in terms of design. For Example:- Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. **Encapsulation**- Inner layout, used in terms of implementation. For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits.

Take an example of Mobile Phone:-

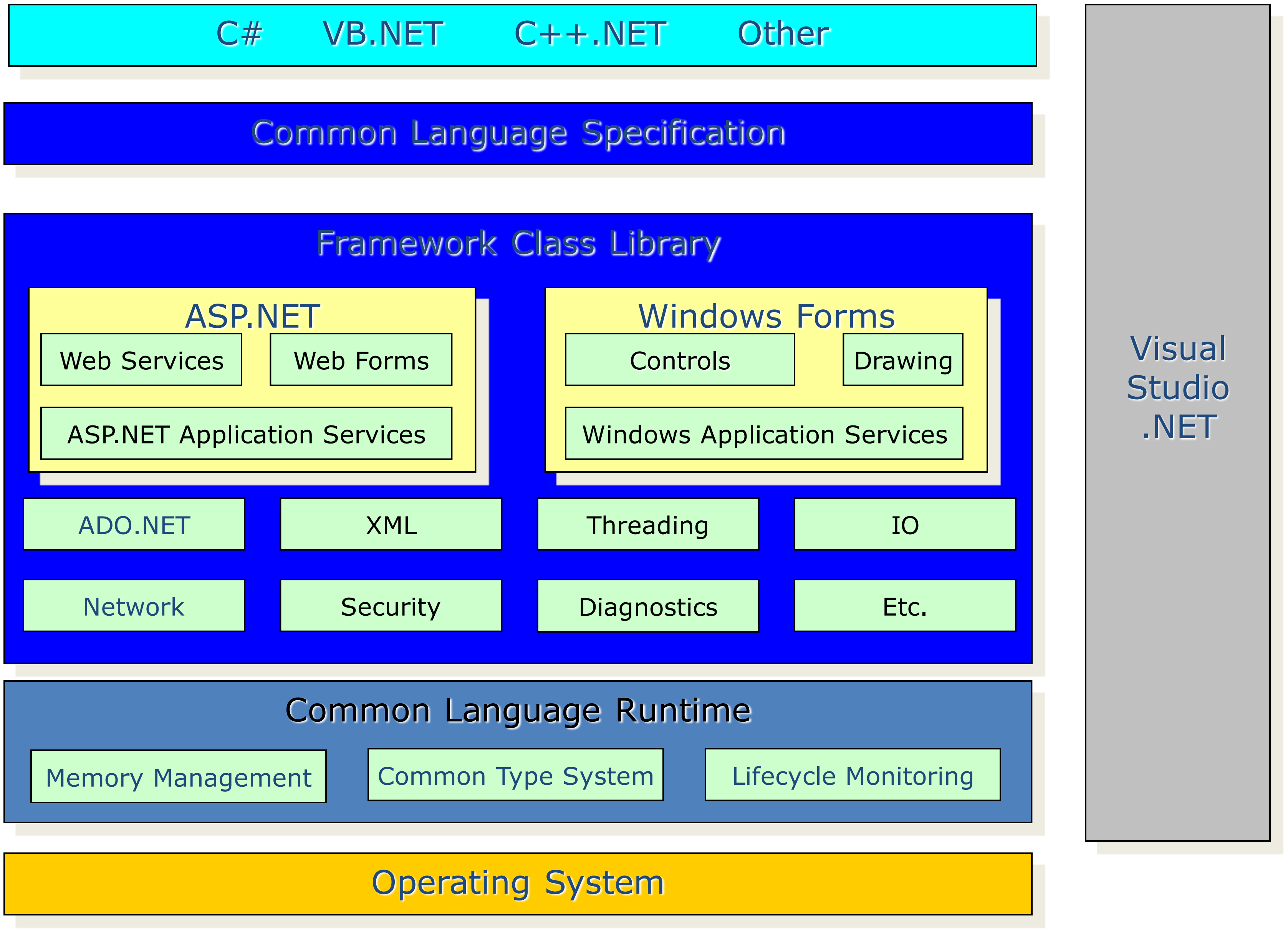
- You have a Mobile Phone, you can dial a number using keypad buttons. Even you don't know how these are working internally. This is called Abstraction. You have the only information that is needed to dial a number. But not its internal working of mobile.

- But how the Mobile Phone internally working?, how keypad buttons are connected with internal circuit? is called Encapsulation.

Summary:

- "Encapsulation is accomplished by using Class. - Keeping data and methods that accesses that data into a single unit"
- "Abstraction is accomplished by using Interface. - Just giving the abstract information about what it can do without specifying the back ground details"
- "Information/Data hiding is accomplished by using Modifiers - By keeping the instance variables private or protected."

.NET Framework



Common Language Runtime (CLR) features

- **Automatic memory management**: - The CLR provides the Garbage Collection feature for managing the life time of object. This relives a programmer from memory management task.
- **Standard Type System**: - The CLR Implement a formal Specification called the Common Type System (CTS). CTS is important part of rules that ensures that objects written in different language can interact with each other.
- **Language interoperability**: - It is the ability of an application to interact with another application written in a different programming language. Language interoperability helps maximum code reuse. The CLR provides support for language interoperability by specifying and enforcing CTS and by providing metadata.
- **Platform Independence**: - The Compiler compiles code language, which is CPU-independent. This means that the code can be executed from any platform that supports the .Net CLR.

- **Security Management**: - In .net platform, Security is achieved through the code access Security (CAS) model. In
  the model, CLR enforces the restriction an managed code through the object called "permissions". The CLR allows the code to perform only that task for which it has permissions. In other words, the CAS model specifies what the code can access instead of specifies who can access resources.
- **Type Safety**: - This feature ensures that object is always accessed in compatible ways. Therefore the CLR will prohibit a code from assign a 10-byte value to an object that occupies 8 bytes.

.Net Framework Class Library (FCL)

- The .Net Framework class library (FCL) provides the core functionality of .Net Framework architecture. The .Net Framework Class Library (FCL) includes a huge collection of reusable classes, interfaces, and value types that expedite and optimize the development process and provide access to system functionality.
  The .Net Framework class library (FCL) organized in a hierarchical tree structure and it is divided into Namespaces. Namespaces is a logical grouping of types for the purpose of identification. Framework class library (FCL) provides the consistent base types that are used across all .NET enabled languages. The Classes are accessed by namespaces, which reside within Assemblies. The System Namespace is the root for types in the .NET Framework. The .Net Framework class library (FCL) classes are managed classes that provide access to System Services . The .Net Framework class library (FCL) classes are object oriented and easy to use in program developments. Moreover, third-party components can integrate with the classes in the .NET Framework.

Common Type System

- Common Type System (CTS) describes a set of types that can be used in different .Net languages in common . That is , the Common Type System (CTS) ensure that objects written in different .Net languages can interact with each other. For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level .
  These types can be Value Types or Reference Types . The Value Types are passed by values and stored in the stack. The Reference Types are passed by references and stored in the heap. Common Type System (CTS) provides base set of Data Types which is responsible for cross language integration. The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS) .Most of the members defined by types in the .NET Framework Class Library (FCL) are Common Language Specification (CLS) compliant Types.

Common Language Specification

- Common Language Specification (CLS) is a set of basic language features that .Net Languages needed to develop Applications and Services, which are compatible with the .Net Framework. When there is a situation to communicate Objects written in different .Net Complaint languages, those objects must expose the features that are common to all the languages. Common Language Specification (CLS) ensures complete interoperability among applications, regardless of the language used to create the application.

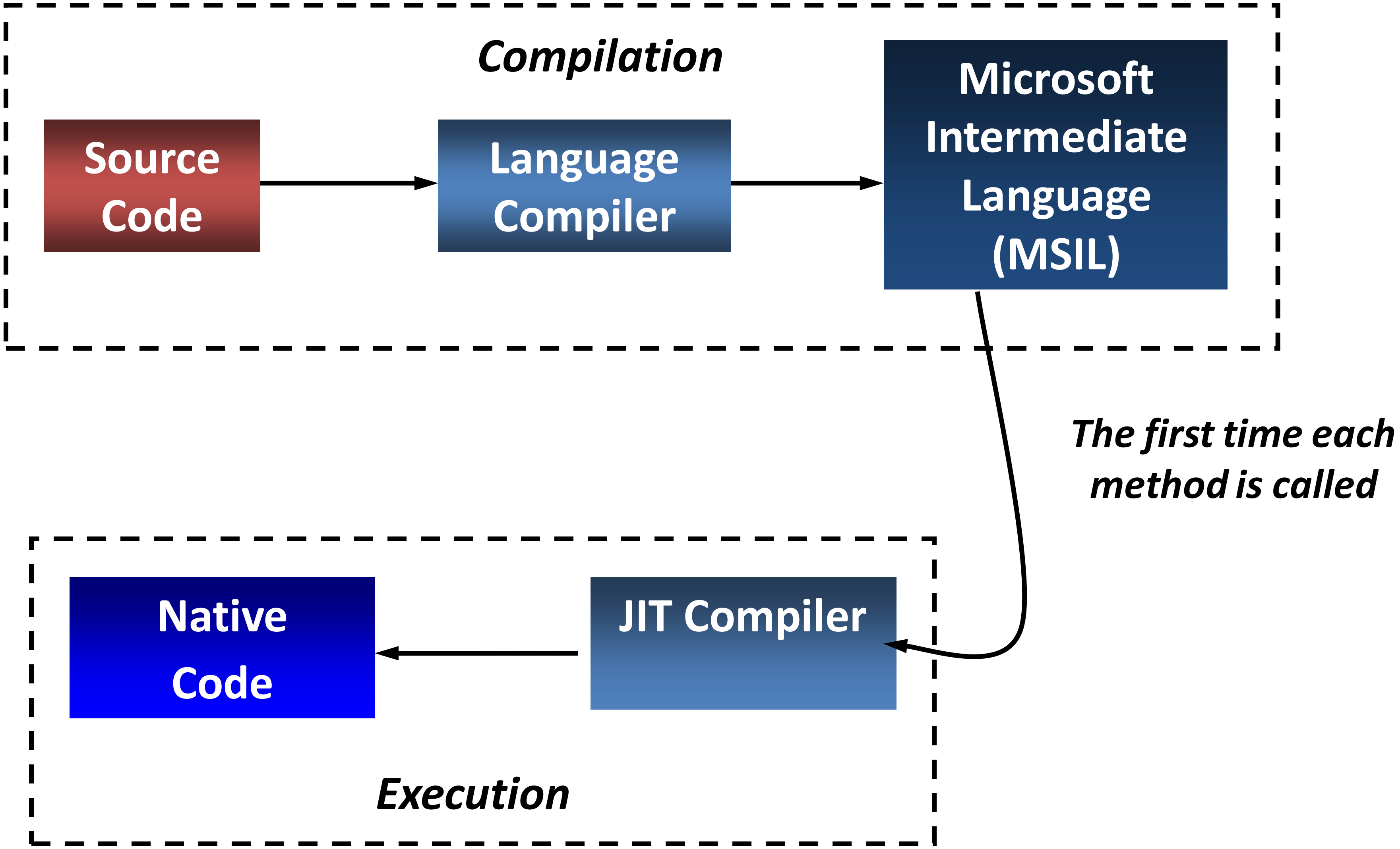
- Common Language Specification (CLS) defines a subset of Common Type System (CTS). Common Type System (CTS) describes a set of types that can use different .Net languages have in common, which ensure that objects written in different languages can interact with each other

Microsoft Intermediate Language

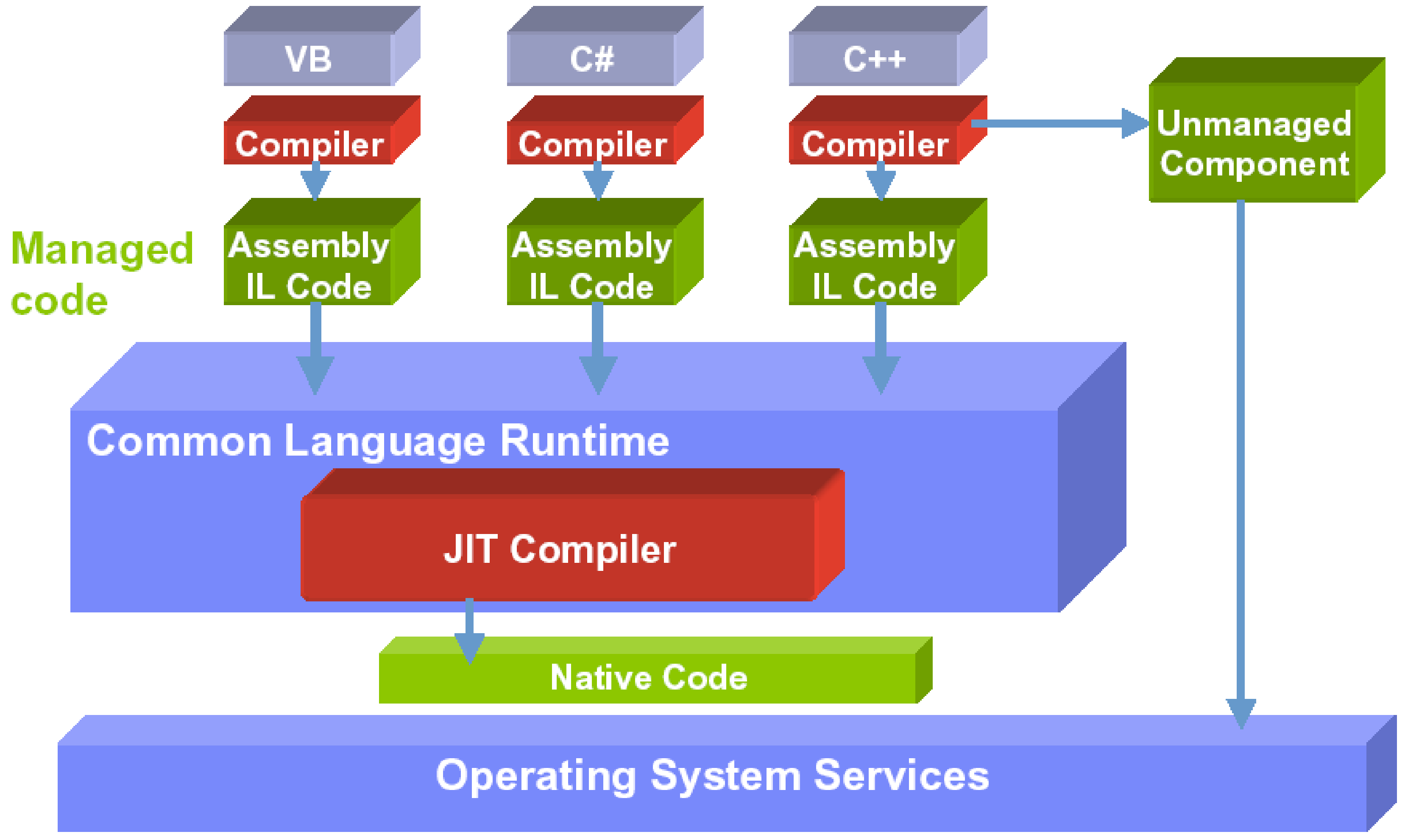
- MSIL stands for Microsoft Intermediate Language. We can call it as Intermediate Language (IL) or Common Intermediate Language (CIL). During the compile time , the compiler convert the source code into Microsoft Intermediate Language (MSIL) .Microsoft Intermediate Language (MSIL) is a CPU-independent set of instructions that can be efficiently converted to the native code. During the runtime the Common Language Runtime (CLR)'s Just In Time (JIT) compiler converts the Microsoft Intermediate Language (MSIL) code into native code to the Operating System.
  When a compiler produces Microsoft Intermediate Language (MSIL), it also produces Metadata. The Microsoft Intermediate Language (MSIL) and Metadata are contained in a portable executable (PE) file . Microsoft Intermediate Language (MSIL) includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations
  Just In Time Compiler

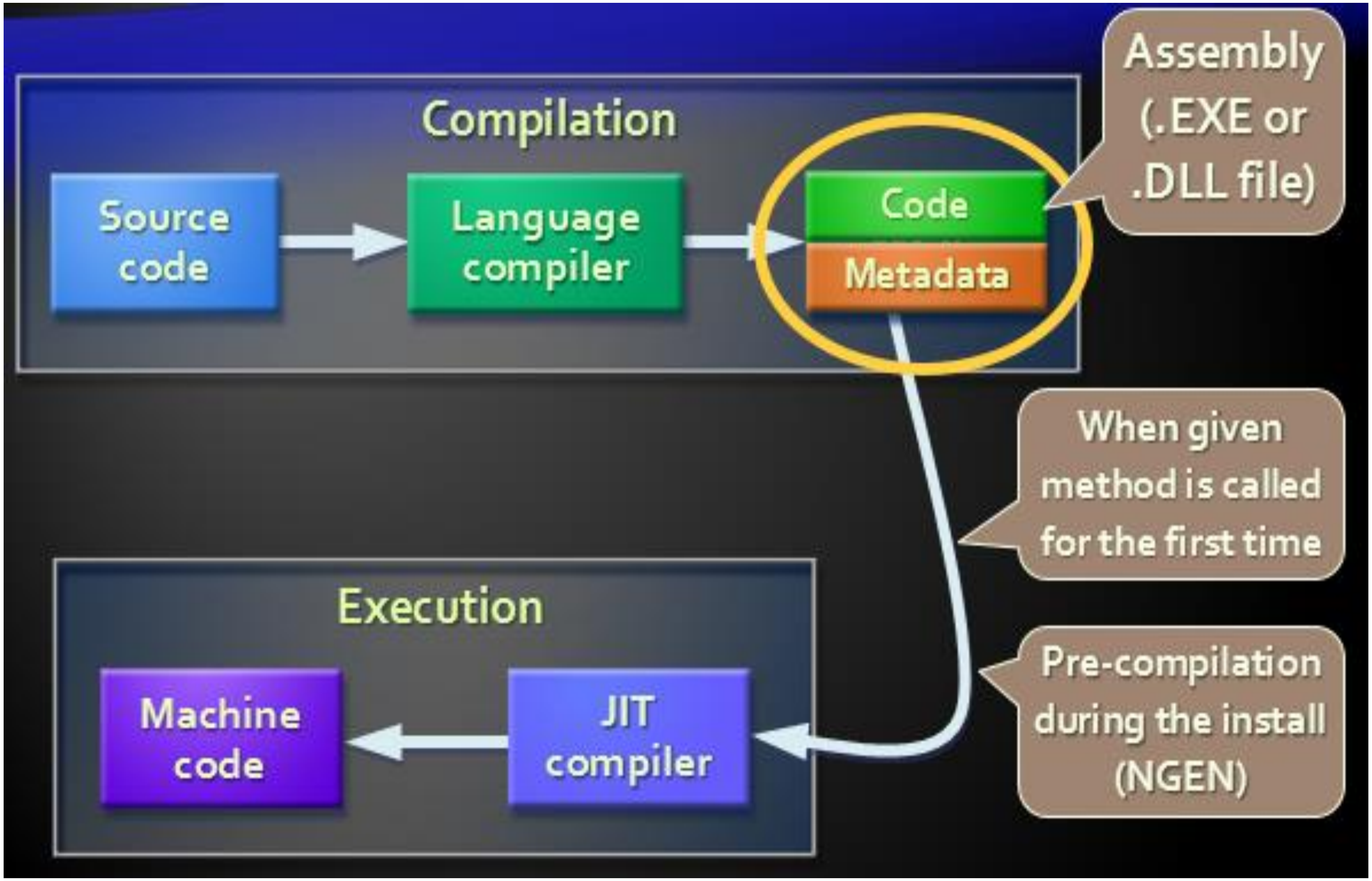
- The .Net languages , which is conforms to the Common Language Specification (CLS), uses its corresponding runtime to run the application on different Operating Systems . During the code execution time, the Managed Code compiled only when it is needed, that is it converts the appropriate instructions to the native code for execution just before when each function is called. This process is called Just In Time (JIT) compilation, also known as Dynamic Translation . With the help of Just In Time Compiler (JIT) the Common Language Runtime (CLR) doing these tasks.
  The Common Language Runtime (CLR) provides various Just In Time compilers (JIT) and each works on a different architecture depending on Operating System. That is why the same Microsoft Intermediate Language (MSIL) can be executed on different Operating Systems without rewrite the source code. Just In Time (JIT) compilation preserves memory and save time during application initialization. Just In Time (JIT) compilation is used to run at high speed, after an initial phase of slow interpretation. Just In Time Compiler (JIT) code generally offers far better performance than interpreters.

Compiling and executing managed code


Compilation
Source Code
Language Compiler
Microsoft Intermediate Language (MSIL)
The first time each method is called
JIT Compiler
Native Code
Execution

Common Language Runtime

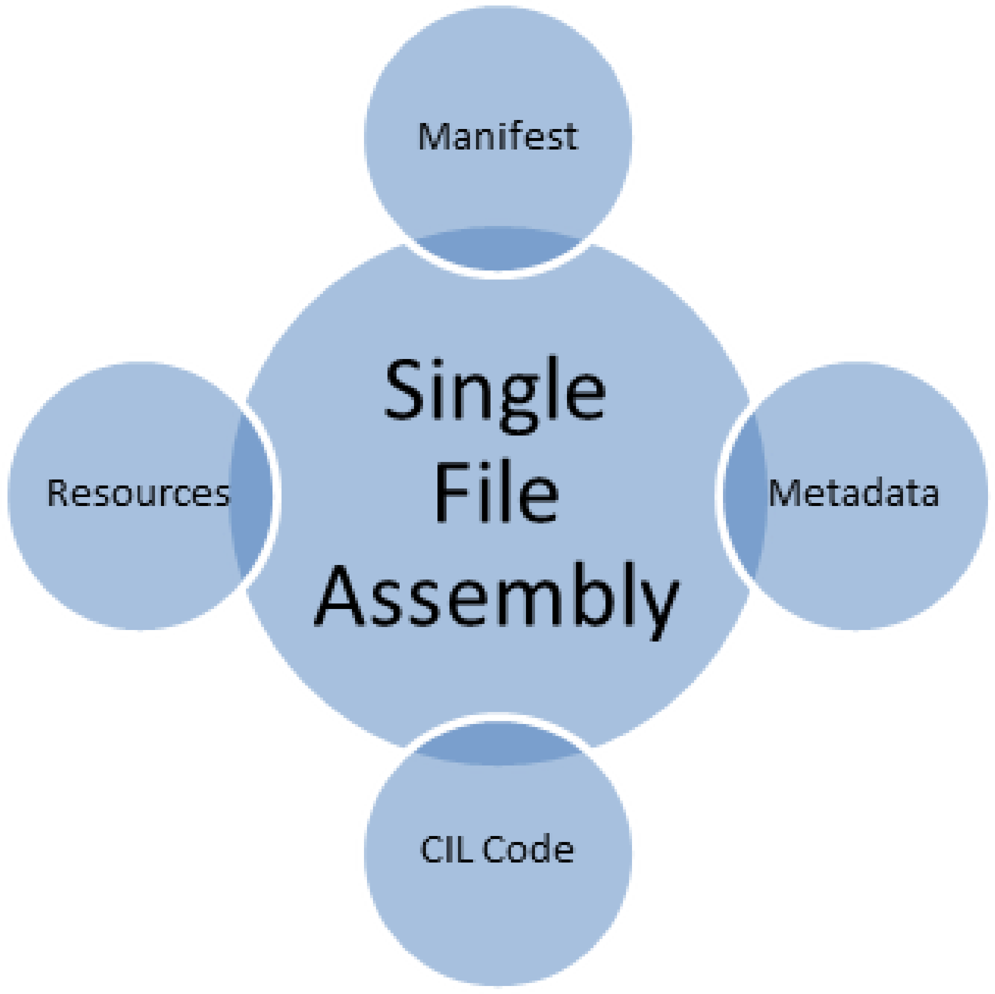



Compilation
Source code
Language compiler
Code
Metadata
Assembly (.EXE or .DLL file)
When given method is called for the first time
Pre-compilation during the install (NGEN)
Execution
Machine code
JIT compiler

Managed Code

- Managed Code in Microsoft .Net Framework, is the code that has executed by the Common Language Runtime (CLR) environment. On the other hand Unmanaged Code is directly executed by the computer's CPU. Data types, error-handling mechanisms, creation and destruction rules, and design guidelines vary between managed and unmanaged object models.
  The benefits of Managed Code include programmers convenience and enhanced security . Managed code is designed to be more reliable and robust than unmanaged code , examples are Garbage Collection , Type Safety etc. The Managed Code running in a Common Language Runtime (CLR) cannot be accessed outside the runtime environment as well as cannot call directly from outside the runtime environment. This makes the programs more isolated and at the same time computers are more secure . Unmanaged Code can bypass the .NET Framework and make direct calls to the Operating System. Calling unmanaged code presents a major security risk.

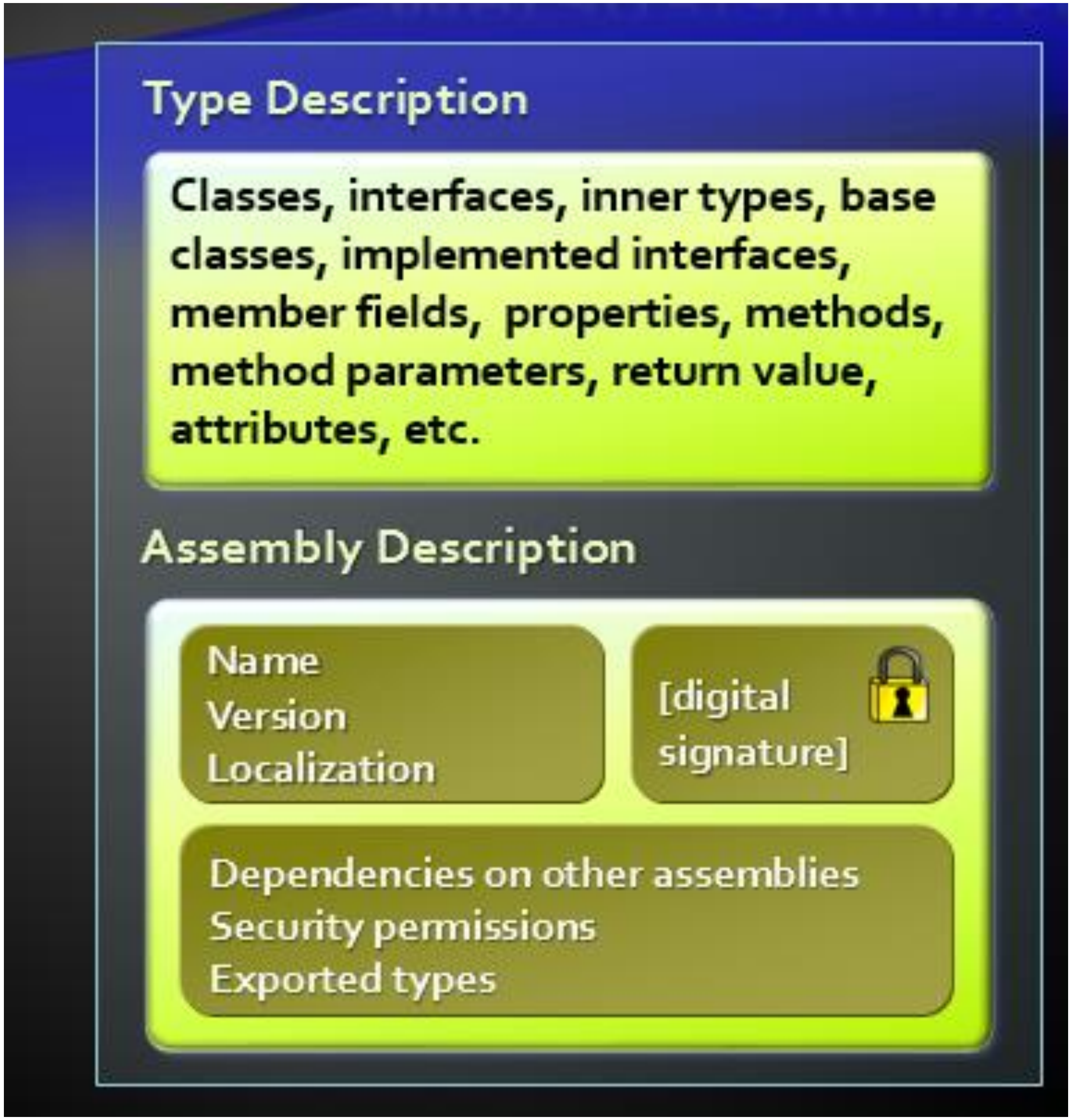
.Net Assembly



- Microsoft .Net Assembly is a logical unit of code, it contains code that the Common Language Runtime (CLR) executes. Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality. During the compile time Metadata is created, with Microsoft Intermediate Language (MSIL), and stored in a file called a Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. Manifest contains information about itself. This information is called Assembly Manifest, it contains information about the members, types, references and all the other data that the runtime needs for execution.

.Net Metadata

- Metadata in .Net is binary information which describes the characteristics of a resource . This information include Description of the Assembly , Data Types and members with their declarations and implementations, references to other types and members , Security permissions etc. A module's metadata contains everything that needed to interact with another module.
  During the compile time Metadata created with Microsoft Intermediate Language (MSIL) and stored in a file called a Manifest . Both Metadata and Microsoft Intermediate Language (MSIL) together wrapped in a Portable Executable (PE) file. During the runtime of a program Just In Time (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code. When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on. Moreover Metadata eliminating the need for Interface Definition Language (IDL) files, header files, or any external method of component reference.



Type Description

Classes, interfaces, inner types, base classes, implemented interfaces, member fields, properties, methods, method parameters, return value, attributes, etc.

Assembly Description

Name
Version
Localization

[digital signature]

Dependencies on other assemblies
Security permissions
Exported types

Difference between Metadata and Manifest

- Manifest Maintains the information about the assemblies like version, name locale and an optional strong name that uniquely identifying the assembly. This manifest information is used by the CLR. The manifest also contains the security demands to verify this assembly. It also contains the names and hashes of all the files that make up the assembly. The .NET assembly manifest contains a cryptographic hash of different modules in the assembly. And when the assembly is loaded, the CLR recalculates the hash of the modules at hand, and compares it with the embeded hash. If the hash generated at runtime is different from that found in the manifest, .NET refuses to load the assembly and throws an exception.

- Metadata means Data about the data.metadata yields the types available in that assembly, viz. classes, interfaces, enums, structs, etc., and their containing namespaces, the name of each type, its visibility/scope, its base class, the interfaces it implemented, its methods and their scope, and each method’s parameters, type’s properties, and so on. The assembly metada is generated by the high-level compilers automatically from the source files. The compiler embeds the metadata in the target output file, a dll, an .exe or a .net module in the case of multi-module assembly.

Types of assemblies:

- a. Private Assemblies: are accessible by a single application. They reside within the application folder and are unique by name. They can be directly used by copying and pasting them to the bin folder.
- b. Shared Assemblies: are shared between multiple applications to ensure reusability. They are placed in GAC.
- c. Satellite Assemblies: are assemblies that are used to deploy language and culture specific resources for an application. In an application, a separate product ID is assigned to each language and a satellite assembly is installed in a language specific sub-directory.

Garbage Collection

- The .Net Framework provides a new mechanism for releasing unreferenced objects from the memory (that is we no longer needed that objects in the program) ,this process is called Garbage Collection (GC). When a program creates an Object, the Object takes up the memory. Later when the program has no more references to that Object, the Object's memory becomes unreachable, but it is not immediately freed. The Garbage Collection checks to see if there are any

DLR (Dynamic Language Runtime)

- One of the powerful features of Microsoft .NET Framework 4.0 is the Dynamic Types.
- Dynamic Types allow us to write code in such a way that we can go around compile time checking. Note that bypass does not mean that we can remove the compile time errors, it means if the operation is not valid, then we cannot detect the error at compile time. This error will appear only in run time.