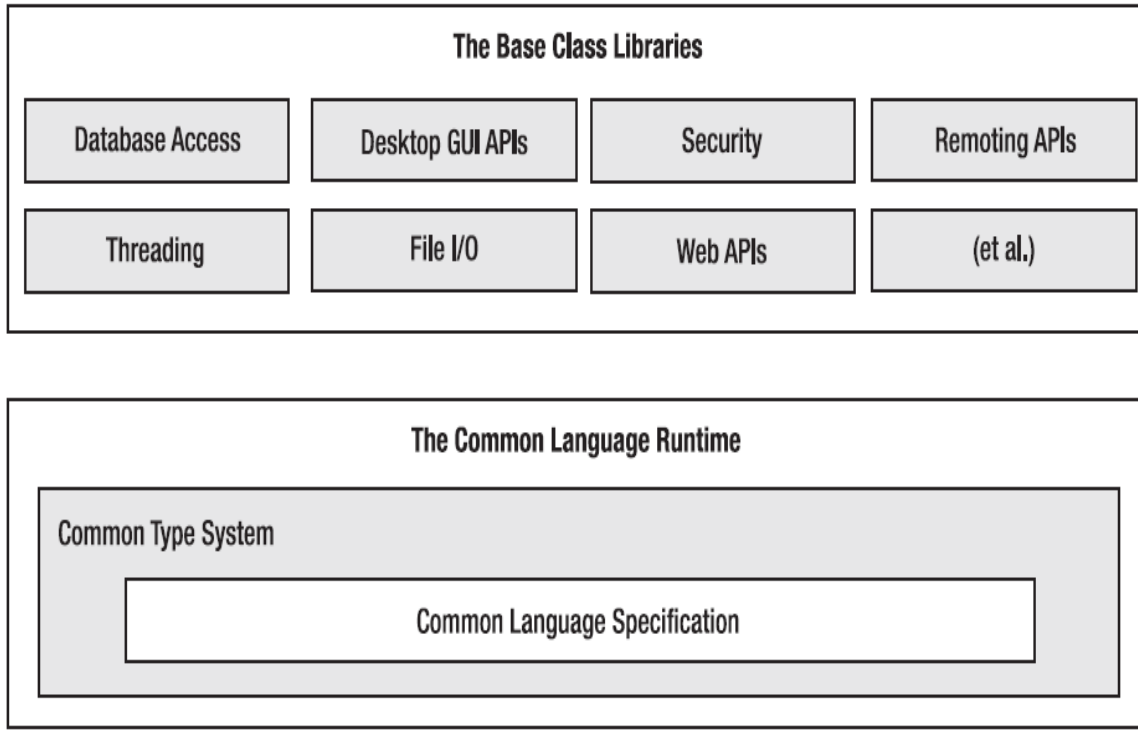


## C# Programming With .NET (06CS/IS761)

Chapter wise questions and Answers appeared in previous years:

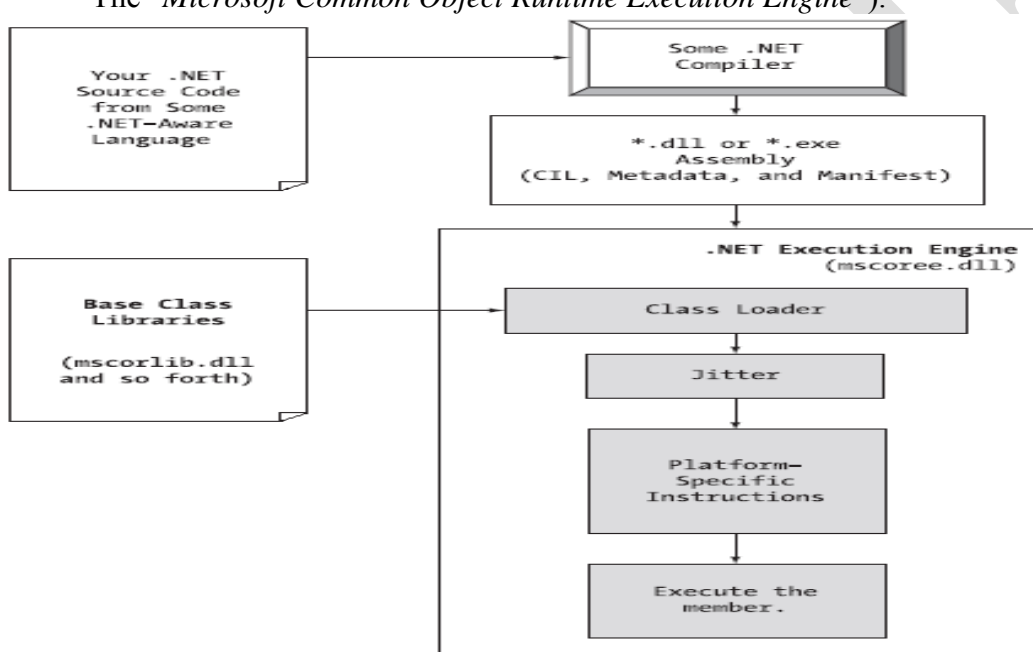
| UNIT I: | Philosophy of the .NET   | Markes &<br>Year<br>Appeared   |
|---------|--|--|
| 1       | <p>What are the building blocks of .NET platform? Give the relationship between .NET runtime layer and the base class library.</p> <p style="text-align: center;">Or</p> <p>What are the building blocks of .NET framework? Explain their relationship with a neat block diagram. Explain the CTS, in detail.</p> <p style="text-align: center;">Or</p> <p>Explain with neat diagram, the relation between .NET runtime layer and the base class library.</p> <p style="text-align: center;">Or</p> <p>Explain the features and Building blocks of .NET framework.</p>   | <p>June- 12<br/>(08)</p> <p>DEC- 10<br/>(10)</p> <p>Dec-<br/>09/Jan-<br/>10<br/>(08)</p> |
| Ans:    | <p>The three building blocks of .NET framework are: <b><u>CLR, CTS, and CLS.</u></b><br/>.NET is a runtime environment and a comprehensive base class library.</p> <ul style="list-style-type: none"> <li>• <b>Common Language Runtime (CLR):</b> <ul style="list-style-type: none"> <li>• The runtime layer is properly referred to as the <i>common language runtime (CLR)</i>.</li> <li>• The primary role of the CLR is to: Locate, Load, Manage .NET types on your behalf.</li> <li>• The CLR also takes care of a number of low-level details such as: <ul style="list-style-type: none"> <li>• Memory management;</li> <li>• Creating application domains,</li> <li>• Threads, and object context boundaries, Performing various security checks.</li> </ul> </li> </ul> </li> <li>• <b>Common Type Systems (CTS):</b> <ul style="list-style-type: none"> <li>• The CTS specification fully describes all possible <b>DATA TYPES</b> and programming constructs supported by the .NET runtime.</li> <li>• It specifies how these entities can interact with each other.</li> <li>• Details how they are represented in the .NET metadata format.</li> </ul> </li> <li>• <b>Common Language Specification (CLS):</b> <ul style="list-style-type: none"> <li>• It is a related specification that defines a subset of common types and programming constructs that all .NET programming languages can agree on.</li> <li>• Thus, if you build .NET types that only expose CLS-compliant features, you can rest assured that all .NET-aware languages can consume them.</li> <li>• Conversely, if you make use of a data type or programming construct that is outside of the bounds of the CLS, you cannot guarantee that every .NET programming language can interact with your .NET code library.</li> </ul> </li> <li>• The .NET platform provides a base class library that is available to all .NET programming languages.</li> <li>• This base class library encapsulate various primitives such as threads, file input/output (I/O), graphical rendering, and interaction with various external hardware devices.</li> </ul> | <p>June-<br/>July<br/>-<br/>11(10M)</p>  |



- It also provides support for a number of services required by most real-world applications.
  - For example, the base class libraries define types that facilitate database access,
  - Manipulation of XML documents,
  - Programmatic security,
  - The construction of web-enabled (as well as traditional desktop and console-based) front ends.

From a high level, you can visualize the relationship between the CLR, CTS, CLS, and the base class library, as shown in above Figure.

|      |   |   |
|------|---|---|
| 2    | <p>What is the role of .NET type Metadata? Give an example.<br/>Or<br/>What is the role of .NET type Metadata? Give an example.</p>   | June- 12<br>(04)<br>Dec-<br>09/Jan-<br>10<br>(04) |
| Ans: | <ul style="list-style-type: none"> <li>• Metadata is an information which hold the information of other data types.</li> <li>• A .NET assembly contains full, complete, and accurate metadata, which describes each and every type living within the binary:                     <ul style="list-style-type: none"> <li>➤ Class,</li> <li>➤ Structure,</li> <li>➤ Enumeration, and so forth) defined in the binary, as well as the members of each type like:                             <ul style="list-style-type: none"> <li>➤ Properties,</li> <li>➤ Methods,</li> <li>➤ Events, and so on).</li> </ul> </li> </ul> </li> <li>• For example, if you have a class named SportsCar, the type metadata describes details such as SportsCar’s base class, which interfaces are implemented by SportsCar (if any), as well as a full description of each member supported by the SportsCar type.</li> <li>• It is always the job of the compiler (not the programmer) to emit the latest and greatest type metadata.</li> <li>• Finally, in addition to CIL and type metadata, assemblies themselves are also described using metadata, which is officially termed a <i>manifest</i>.</li> <li>• <i>The manifest contains information about the current</i> version of the assembly, culture information (used for localizing string and image resources),</li> <li>• A list of all externally referenced assemblies that are required for proper execution of</li> </ul> |   |

|                      |   |                          |
|----------------------|---|--------------------------|
|                      | the given program or application.   |                          |
| <p>3</p> <p>Ans:</p> | <p>Explain the CLR. Illustrate the workflow that takes place between the source code, given .NET compiler and the .NET execution engine.</p> <ul style="list-style-type: none"> <li>• <b>Common Language Runtime (CLR):</b> <ul style="list-style-type: none"> <li>• The runtime layer is properly referred to as the <i>common language runtime CLR</i>.</li> <li>• <i>The primary role of the CLR</i> is to:                             <ul style="list-style-type: none"> <li>• Locate, Load, Manage .NET types on your behalf.</li> <li>• The CLR also takes care of a number of low-level details such as:                                     <ul style="list-style-type: none"> <li>• Memory management;</li> <li>• Creating application domains,</li> <li>• Threads, and object context boundaries,</li> <li>• Performing various security checks.</li> </ul> </li> </ul> </li> <li>• Programmatically speaking, the term <i>runtime can be understood as</i> a collection of external services that are required to execute a given compiled unit of code.</li> <li>• Other popular languages also have a corresponding runtime.</li> <li>• The .NET runtime provides a single well-defined runtime layer that is shared by <i>all languages and platforms that are .NET aware</i>.</li> <li>• The crux of the CLR is physically represented by a library named mscorlib.dll (a.k.a. The “<i>Microsoft Common Object Runtime Execution Engine</i>”).</li> </ul>  <ul style="list-style-type: none"> <li>• When an assembly is referenced for use, mscorlib.dll is loaded automatically, which in turn loads the required assembly into memory.</li> <li>• The runtime engine is responsible for a number of tasks:             <ul style="list-style-type: none"> <li>• First and foremost, it is the entity in charge of resolving the location of an assembly,</li> <li>• Finding the requested type within the binary by reading the contained metadata.</li> <li>• The CLR then lays out the type in memory, compiles the associated CIL into platform-specific instructions, performs any necessary security checks, and then executes the code in question.</li> </ul> </li> <li>• In addition to loading your custom assemblies and creating your custom types, the CLR will also interact with the types contained within the .NET base class libraries when required.</li> </ul> </li></ul> | <p>June- 12<br/>(08)</p> |
| <p>4</p> <p>Ans:</p> | <p>Explain the complexities found in the technologies prior to .NET. Briefly describe. Explain how .NET attempts to simplify the same.</p> <p>The technologies found prior to .NET are:</p> <ul style="list-style-type: none"> <li>• <b>Life as a C/WIN32 API Programmer</b></li> </ul>   | <p>Dec- 11<br/>(10)</p>  |

- **Life as a C++/ MFC Programmer**
- **Life as a Visual Basic 6.0 Programmer**
- **Life as a Java/ J2EE Programmer**
- **Life as A COM Programmer**
- **Life as a Windows DNA Programmer**

#### **Life as a C/WIN32 API Programmer**

- In 95's, Developing software's for Windows Operating Systems involved using C-Programming Language in conjunction with Windows API (Application Programming Interface).
- But building applications using raw API is a complex approach, as C is a very terse(in brief/ unfriendly) lang.
- And C developers are forced to work with:
  - Manual memory management
  - Ugly pointer arithmetic
  - Ugly syntactical constructs.

#### **Life as a C++/ MFC Programmer**

- C++ is thought to be an Object- oriented layer on top of C (Partially object oriented programming language).
- Pillars of OOPs:
  - Encapsulation:
  - Inheritance:
  - Polymorphism:
- Microsoft Fundamental Classes (MFC): set of existing C++ classes to facilitate the construction of WIN APIs.
- Regardless of this helpfulness, C++ programming remains as difficult and error-prone experience.

#### **Life as a Visual Basic 6.0 Programmer**

- Many programmers had shifted away from world of C++ based frameworks to kinder, gentler lang VB 6.0
- VB was popular due to its ability to built complex UI, code libraries (COM servers), data access logic with minimal fuss and logic.
- Since VB is not a fully OOP but Object Aware Language:
  - VB 6.0 does not support "Is a relationship" between types ( I.e: Classical Inheritance).
  - Does not support parameterized class construction.
  - Does not support building multithreaded applications.

#### **Life as a Java/ J2EE Programmer**

- JAVA is a complete OOP lang.
- JAVA is a platform independent Language.
- JAVA cleanup all unsavory syntactical aspects of C++.
- JAVA as a platform provides programmers with large set of predefined 'Packages', containing various classes and interface definitions.
- Using the above merits JAVA programmers are able to build "100% pure JAVA" applications complete with database connectivity, messaging support, web- enabled front ends and rich user interfaces.
- Problems: You must use JAVA front- to- back during the development cycle.
- In effect java offers little hope for language independence.
- Pure java is not appropriate for many graphical/ numerically intensive applications.

#### **Life as A COM Programmer:**

- COM is an architecture that says in effect "*If you build your classes in accordance with rules of COM, you end up with a block of reusable binary code*".
- COM can be accessed in language independent manner, I. e: C++ programmers can build COM classes which can be used by VB.
- BUT, there is no way to derive a new COM type using an existing COM type (No

support for Classical inheritance).

- Use “has –a relationship” to reuse COM types.
- COM is a location- transparent nature, i. e: Using constructors like application identifiers (AppIDs), stubs, proxies, and COM runtime environment to avoid the need to work with raw sockets, RPC calls, and other low level details.

#### Life as a Windows DNA Programmer:

- Microsoft has been adding more internet- aware features into its family of operating systems.
- Building a complete web application using classic Windows DNA (**Distributed iNternet Architecture**).
- Complexity is due to a fact that Windows DNA requires the use of technologies and languages (ASP, HTML, XML, Java Script, VB Script, COM+ as well as data access APIs like ADO).
- Since these items are completely unrelated from a syntactic point of view.
- The result is a highly confused mishmash of technologies.
- And perhaps more important, each language and/ or technology has its own type system:
  - An “int” in javaScript is not the same as an “int” in C, which is different from an “Integer” in VB proper.

#### Features of .NET:

- The .NET Framework is a *Radical and Brute- Force* approach to making our lives easier.
- The .NET Framework is a completely new model for building systems on the Windows family of operating systems, as well as on numerous non-Microsoft operating systems such as Mac OS X and various Unix/Linux distributions.
- To set the stage, here is a quick rundown of some core features provided courtesy of .NET:
  - *Comprehensive interoperability with existing code,*
  - *Complete and total language integration,*
  - *A common runtime engine shared by all .NET-aware languages,*
  - *A comprehensive base class library,*
  - *No more COM plumbing,*
  - *A truly simplified deployment model.*

5

Explain the formal definitions of all CTS types.

Or

List and explain the intrinsic CTS data types and .NET namespaces in C#.

Ans:

In the world of .NET, **TYPE** is simply a general term used to refer to a member from the set :

- Class,
- Interface,
- Structure,
- Enumeration,
- Delegate.
- When you build solutions using a .NET-aware language, you will most likely interact with many of these types.
  - For example, your assembly may define a single class that implements some number of interfaces.
  - Perhaps one of the interface methods takes an enumeration type as an input parameter and returns a structure to the caller.
- Recall that the CTS is a formal specification that documents how types must be defined in order to be hosted by the CLR.

#### CTS (Class Types):

- Every .NET-aware language supports, at the very least, the notion of a *class type*, which is the cornerstone of object-oriented programming (OOP).

DEC- 11  
(10)  
Dec-  
09/Jan-  
10  
(08)

|   |   |   |
|---|---|---|
|   | <ul style="list-style-type: none"> <li>• A class may be composed of any number of members like: <ul style="list-style-type: none"> <li>• Properties,</li> <li>• Methods,</li> <li>• Events</li> <li>• Data points (fields).In C#, classes are declared using the class keyword: <pre>// A C# class type. class Calc {     public int Add(int x, int y)     { return x + y; } }</pre> </li> </ul> </li> </ul> <p><b>CTS (Interface Types):</b></p> <ul style="list-style-type: none"> <li>• Interfaces are nothing more than a named collection of abstract member definitions, which may be supported (i.e., implemented) by a given class or structure.</li> <li>• C# interface types are defined using the interface keyword,</li> <li>• On their own, interfaces are of little use.</li> <li>• However, when a class or structure implements a given interface in its unique way, you are able to request access to the supplied functionality using an interface reference in a polymorphic manner. <pre>// A C# interface type is usually // declared as public, to allow types in other // assemblies to implement their behavior. public interface IDraw {     void Draw(); }</pre> </li> </ul>  |   |
| 6 | <p>Briefly explain how the history of .NET. Explain the building components of .NET and their responsibilities.</p> <p>Ans: Please refer to Question no 1 and 4 for Ans's.</p>  | May-<br>June- 10<br>(06)                        |
| 7 | <p>Explain Jitter. Along with its benefits. Explain how CLR host an application on .NET platforms. Give the block diagram.</p> <p>Ans: <u><b>Jitter</b></u></p> <ul style="list-style-type: none"> <li>• The entity that compiles CIL code into meaningful CPU instructions is termed a <i>just-in-time (JIT) compiler</i>, which sometimes goes by the friendly name of <u><b>Jitter</b></u>.</li> <li>• The .NET runtime environment leverages a JIT compiler for each CPU targeting the runtime, each optimized for the underlying platform.</li> <li>• For example, if you are building a .NET application that is to be deployed to a handheld device (such as a Pocket PC), the corresponding Jitter is well equipped to run within a low-memory environment.</li> </ul> <p>For Ans of CLR hosting an application on .NET platform please refer to Question no 3.</p>   | May-<br>June- 10<br>(06)                        |
| 8 | <p>What is an assembly? Explain each component of an assembly. Differentiate between single file and multi file assembly.</p> <p style="text-align: center;">Or</p> <p>What is .NET assembly? What does it contain? Explain each of them.</p> <p>Ans: <u><b>.NET Assembly:</b></u></p> <ul style="list-style-type: none"> <li>• When a *.dll or an *.exe has been created using a .NET-aware compiler, the resulting module is bundled into an <i>assembly</i>.</li> <li>• An assembly contains CIL code, which is conceptually similar to Java bytecode in that it is not compiled to platform-specific instructions until absolutely necessary.</li> <li>• Typically, “absolutely necessary” is the point at which a block of CIL instructions (such as a method implementation) is referenced for use by the .NET runtime.</li> <li>• Assemblies also contain <i>metadata that describes in vivid detail</i> the characteristics of every “type” living within the binary. <ul style="list-style-type: none"> <li>• For example, if you have a class named SportsCar, the type metadata describes details such as SportsCar’s base class, which interfaces are implemented by SportsCar (if any), as well as a full description of each member supported by the</li> </ul> </li> </ul> | May-<br>June- 10<br>(08)<br><br>Dec- 10<br>(10) |

|            |   |                           |
|------------|---|---------------------------|
|            | <p>SportsCar type.</p> <ul style="list-style-type: none"> <li>Finally, in addition to CIL and type metadata, assemblies themselves are also described using metadata, which is officially termed a <i>manifest</i>.</li> <li><i>The manifest contains information about the current version of the assembly, culture information (used for localizing string and image resources),</i></li> <li>A list of all externally referenced assemblies that are required for proper execution as shown in figure below.</li> </ul> <p><b>Single File and Multi file Assemblies</b></p> <ul style="list-style-type: none"> <li><b>Single-file assemblies</b> contain all the necessary CIL, metadata, and associated manifest in an autonomous, single, well-defined package.</li> <li>In a great number of cases, there is a simple one-to-one correspondence between a .NET assembly and the binary file (*.dll or *.exe).</li> <li>Thus, if you are building a .NET *.dll, it is safe to consider that the binary and the assembly are one and the same. Likewise, if you are building an executable desktop application, the *.exe can simply be referred to as the assembly itself.</li> <li><b>Multi-file assemblies</b> are composed of numerous .NET binaries, each of which is termed a <i>module</i>.</li> <li>When building a multifile assembly, one of these modules (termed the <i>primary module</i>) must contain the assembly manifest (and possibly CIL instructions and metadata for various types).</li> <li>The other related modules contain a module-level manifest, CIL, and type metadata.</li> <li>As you might suspect, the primary module documents the set of required secondary modules within the assembly manifest.</li> </ul> |                           |
| 9<br>Ans:  | <p>Write a note on .NET namespaces.</p> <ul style="list-style-type: none"> <li>The hierarchical representation of code into namespaces is a logical function only.</li> <li>Namespaces can be used to organize code in any way the programmer desires.</li> <li>Namespaces avoid naming conflicts between identifiers. Namespace directives are C# language elements that allow a program to identify namespaces use in a program.</li> <li>They allow the namespace members to be used without specifying fully qualified name. C# has two namespace directives: using and alias.</li> </ul> <p>e. g: using system;<br/>namespace hello<br/>{<br/>class program<br/>{<br/>Static void main(string[] args)<br/>{<br/>Console.WriteLine("Helllo from System.");<br/>}<br/>}<br/>}</p>  | June-<br>July -11<br>(04) |
| 10<br>Ans: | <p>Explain the role of Common Intermediate Language (CIL).</p> <p><b>Common Intermediate Language (CIL)</b></p> <ul style="list-style-type: none"> <li><b>CIL</b> is a language that sits above any particular platform-specific instruction set. <ul style="list-style-type: none"> <li>I. e: Regardless of which .NET aware language you choose (C#, VB, Eiffel .NET), the associated compiler emits the CIL instructions.</li> <li>For e.g: The following C# code models a trivial calculator. Don't concern yourself with the exact syntax for now, but do notice the format of the Add() method in the Calc class:</li> </ul> </li> <li>Once you compile this code file using the C# compiler (csc.exe), you end up with a single-file *.exe assembly that contains <b>a manifest, CIL instructions, and metadata describing each aspect of the Calc and Program classes.</b></li> <li>If you open this assembly using ildasm.exe (examined a little later in this chapter), you</li> </ul>  | June-<br>July -11<br>(06) |

would find that the Add() method is represented using CIL such as the following:

- ```
.method public hidebysig instance int32 Add(int32 x, int32 y) cil managed {  
    // Code size 9 (0x9)  
    .maxstack 2  
    .locals init (int32 V_0)  
    IL_0000: nop  
    IL_0001: ldarg.1  
    IL_0002: ldarg.2  
    IL_0003: add  
    IL_0004: stloc.0  
    IL_0005: br.s IL_0007  
    IL_0007: ldloc.0  
    IL_0008: ret  
} // end of method Calc::Add
```
- **Benefits of CIL**
  - You might be wondering exactly what is gained by compiling source code into CIL rather than directly to a specific instruction set.
  - One benefit is language integration.
  - Each .NET-aware compiler produces nearly identical CIL instructions.
  - Therefore, all languages are able to interact within a well-defined binary arena.