.Net Framework Introduction - Assignment

1. *Demonstrate the process of conversion of Source code into the native machine code in .Net framework with the help of a flowchart.*

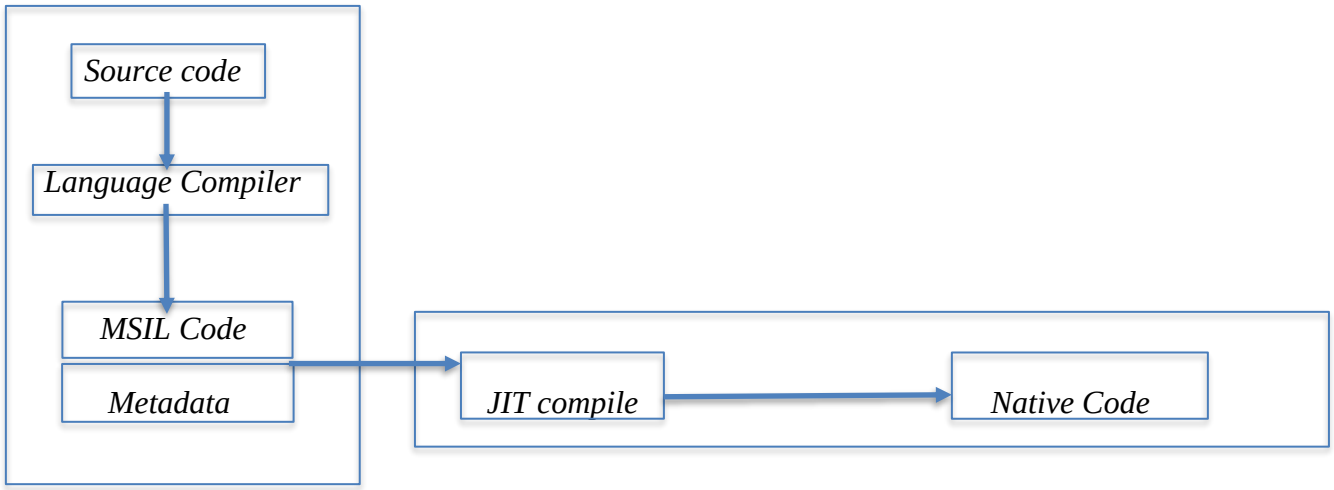

Fig 1: Flowchart showing conversion of source code into native machine code in .NET Framework

CLR is the basic and Virtual Machine component of the .NET Framework**.** It is the run-time environment 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. 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.

- 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 known 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.
- Now CLR comes into existence. 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.

2. *Explain CTS and how the .net framework implements CTS.*

.NET implementation is *language agnostic* which doesn't just mean that a programmer can write their code in any language that can be compiled to IL but that also means that they need to be able to interact with code written in other languages that are able to be used on a .NET implementation.

In order to do this transparently, there has to be a common way to describe all supported types. This is what the Common Type System (CTS) is in charge of doing. CTS is responsible for the following:

- Provide an object-oriented model to support implementing various languages on a .NET implementation.
- Define a set of rules that all languages must follow when it comes to working with types.
- Provide a library that contains the basic primitive types that are used in application development (such as, Boolean, Byte, Char etc.)
- Establish a framework for cross-language execution.

CTS defines two main kinds of types that should be supported: reference and value types. Their names point to their definitions. It also defines all other properties of the types, such as access modifiers, what are valid type members, how inheritance and overloading works and so on.

All types in .NET are either value types or reference types.

Value types are data types whose objects are represented by the object's actual value. If an instance of a value type is assigned to a variable, that variable is given a fresh copy of the value.

Reference types are data types whose objects are represented by a reference (similar to a pointer) to the object's actual value. If a reference type is assigned to a variable, that variable references (points to) the original value. No copy is made.

The common type system in .NET supports the following five categories of types:

- Classes
- Structures
- Enumerations
- Interfaces
- Delegates

3. *Name at least 3 runtime services provided by CLR and explain their role in .net framework.*

Common Language Runtime (CLR) manages the execution of .NET programs. The just-in-time compiler converts the compiled code into machine instructions. This is what the computer executes.

The services provided by CLR include memory management, exception handling, type safety.

1. Memory Management

   The Garbage Collector (GC) is the part of the .NET Framework that allocates and releases memory for your .NET applications. The Common Language Runtime (CLR) manages allocation and deallocation of a managed object in memory. C# programmers never do this directly, there is no delete keyword in the C# language. It relies on the garbage collector.

   The .NET objects are allocated to a region of memory termed the managed heap. They will be automatically destroyed by the garbage collector. Heap allocation only occurs when you are creating instances of classes. It eliminates the need for the programmer to manually delete objects that are no longer required for program execution. This reuse of memory helps reduce the amount of total memory that a program needs to run. Objects are allocated in the heap continuously, one after another. It is a very fast process, since it is just adding a value to a pointer.

   The process of releasing memory is called garbage collection. It releases only objects that are no longer being used in the application. A root is a storage location containing a reference to an object on the managed heap. The runtime will check objects on the managed heap to determine whether they are still reachable (in other words, rooted) by the application. The CLR builds an object graph, that represents each reachable object on the heap. Object graphs are used to document all reachable objects.

2. Exception Handling

   An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

In the .Net Framework, exceptions are represented by classes. The exception classes in .Net Framework are mainly directly or indirectly derived from the System.Exception class. Some of the exception classes derived from the System.Exception class are
the System.ApplicationException and System.SystemException classes.

The System.ApplicationException class supports exceptions generated by application programs. So, the exceptions defined by the programmers should derive from this class.
The System.SystemException class is the base class for all predefined system exception.

.Net provides a structured solution to the exception handling problems in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

These error handling blocks are implemented using the Try, Catch and Finally keywords.

3. Type Safety

Type safety in .NET has been introduced to prevent the objects of one type from peeking into the memory assigned for the other object. Writing safe code also means to prevent data loss during conversion of one type to another.

Type-safe code accesses only the memory locations it is authorized to access. For example, type-safe code cannot directly read values from another object's private fields or code areas.
It accesses types only in well-defined, allowable ways, thereby preventing overrun security breaches.
Type safety helps isolate objects from each other and therefore helps protect them from inadvertent or malicious corruption.
It also provides assurance that security restrictions on code can be reliably enforced.
Type-safe code accesses only the memory locations it is authorized to access. (type-safety specifically refers to memory type safety.) For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways.
Type safety means that the compiler will validate types while compiling, and throw an error if you try to assign the wrong type to a variable.

4. *What are the differences between Library vs DLL vs .Exe? Explain.*

The term EXE is a shortened version of the word executable as it identifies the file as a program. On the other hand, DLL stands for Dynamic Link Library, which commonly contains functions and procedures that can be used by other programs.

In the base application package, you would find at least a single EXE file that may or may not be accompanied with one or more DLL files. An EXE file contains the entry point or the part in the code where the operating system is supposed to begin the execution of the application. DLL files do not have this entry point and cannot be executed on their own.

The most major advantage of DLL files is in its reusability. A DLL file can be used in other applications as long as the coder knows the names and parameters of the functions and procedures in the DLL file. Because of this capability, DLL files are ideal for distributing device drivers. The DLL would facilitate the communication between the hardware and the application that wishes to use it. The application would not need to know the intricacies of accessing the hardware just as long as it is capable of calling the functions on the DLL.

Launching an EXE would mean creating a process for it to run on and a memory space. This is necessary in order for the program to run properly. Since a DLL is not launched by itself and is called by another application, it does not have its own memory space and process. It simply shares the process and memory space of the application that is calling it. Because of this, a DLL might have limited access to resources as it might be taken up by the application itself or by other DLLs.

Summarising:
1.EXE is an extension used for executable files while DLL is the extension for a dynamic link library.

2.An EXE file can be run independently while a DLL is used by other applications.
3.An EXE file defines an entry point while a DLL does not.
4.A DLL file can be reused by other applications while an EXE cannot.
5.A DLL would share the same process and memory space of the calling application while an EXE creates its separate process and memory space.

5. *How does CLR in .net ensure security and type safety? Explain.*

The common language runtime and the .NET provide many useful classes and services that enable developers to easily write secure code and enable system administrators to customize the permissions granted to code so that it can access protected resources. In addition, the runtime and the .NET provide useful classes and services that facilitate the use of cryptography and role-based security.

Type safety and security

Type-safe code accesses only the memory locations it is authorized to access. (For this discussion, type safety specifically refers to memory type safety and should not be confused with type safety in a broader respect.) For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways.

During just-in-time (JIT) compilation, an optional verification process examines the metadata and Microsoft intermediate language (MSIL) of a method to be JIT-compiled into native machine code to verify that they are type safe. This process is skipped if the code has permission to bypass verification.

Although verification of type safety is not mandatory to run managed code, type safety plays a crucial role in assembly isolation and security enforcement. When code is type safe, the common language runtime can completely isolate assemblies from each other. This isolation helps ensure that assemblies cannot adversely affect each other and it increases application reliability. Type-safe components can execute safely in the same process even if they are trusted at different levels. When code is not type safe, unwanted side effects can occur. For example, the runtime cannot prevent managed code from calling into native (unmanaged) code and performing malicious operations. When code is type safe, the runtime's security enforcement mechanism ensures that it does not access native code unless it has permission to do so.