**stack**

In computer science, a **stack** is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed.

**Inheritance**

In object-oriented programming, **inheritance** is when an object or class is based on another object (prototypal **inheritance**) or class (class-based **inheritance**), using the same implementation (inheriting from an object or class) specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior).  It is a mechanism for [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a [hierarchy](https://en.wikipedia.org/wiki/Hierarchy).

Visibility of inherited members

|  |  |  |  |
| --- | --- | --- | --- |
| [**Base class**](https://en.wikipedia.org/wiki/Base_class)**visibility** | **Derived class visibility** | | |
|  | **Public derivation** | **Private derivation** | **Protected derivation** |
| * Private → * Protected → * Public → | * Not inherited * Protected * Public | * Not inherited * Private * Private | * Not inherited * Protected * Protected |

<http://www.codeproject.com/Articles/1825/The-Common-Language-Runtime-CLR-and-Java-Runtime-E>

**What is a Java virtual machine?**

Before I discuss the JVM in details, let me clarify a few related terms.

* **Java Development Kit (JDK)**: This includes ALL the basic Java framework packages, a compiler (javac), JRE, a JVM, debugger etc. in short all you need to develop, debug, compile and run our Java program.
* **Java Runtime Environment (JRE)**: This is a subset of the JDK. It does not include a debugger, compiler, and framework classes. This includes the bare minimum that a computer needs in order to run a *.class* file.
* **Java Virtual Machine (JVM)**: JVM is a part of JRE. The *.class* file is passed over to JVM which then runs the program. The JRE ensures that the code does not violate any of the security restrictions. Remember that the byte-code (.class file) is not directly run on the host machine; it needs to be converted to the host machine's language. This conversion is done by the JVM. While converting the JVM ensures the security and may also optimize the code. There are many commercial JVMs available in the market - different JVMs have different capabilities, and varying degree of performance. In order to produce efficient, code with minimum delay a JVM needs to have great amount of intelligence built into it. Which would also make the JVM larger in size. Remember that for a Java program to run, the JVM must be loaded in the memory, and it is obvious that a large sized JVM would need much more computer resources than a compact one. So there has to be a fine balance between the size of a JVM and its capabilities. This is why a Java program is always 30-70% slower than equivalent C++ program.

The initial JVMs were extremely slow and were resource hungry - thus explaining the constant churning of your hard-disk when you ran a Java program. In recent years lot of efficient JVMs have surfaced. These JVMs use different compilation techniques to produce efficient machine code in as less a time as possible. One such technique is called Just-In-Time (JIT) compilation. This technique has also been used in .NET.

Just-in-time (JIT) compilers promise to improve the performance of Java applications. Rather than letting the JVM run byte code, a JIT compiler translates code into the host machine's native language. Thus, applications gain the performance enhancement of compiled code while maintaining Java's portability.

Given the concept of the JVM, it is obvious that any programming language that compiles into Java byte code can use the JVM for running the program. We are all aware of how Java code (.java) is converted into byte code (.class) which is then run by the JVM on the host machine. What if we make a compiler of C++, that converts a C++ source file (.c or .cpp) into a java-byte code file (.class) rather than into an .obj file. Theoretically it is possible, whether it is practical or not is a different issue all together. In fact there have been many languages that have compilers which produce java byte code that can then be run by the JVM. A detail of such languages can be found in [[9](http://www.codeproject.com/Articles/1825/The-Common-Language-Runtime-CLR-and-Java-Runtime-E#_9)]. This article belittles Microsoft's claim that the CLR is the only platform to support the language antagonism. JVM can also (and in fact already is) be used by different languages.

## What is the CLR?

The CLR is therefore an environment in which we can run our .NET applications that have been compiled to IL.

The programmer must first write the source code and then compile it. Windows programmers have always compiled their programs directly into machine code - but with .NET things have changed. The language compiler would compile the program into an intermediate language "MSIL" or simply "IL" (much like Java Byte code). The IL is fed to the CLR then CLR would use the IL compiler to convert the IL to the host machine code.

**JIT**

The **JIT** compiler is an important element of CLR, which loads MSIL on target machines for execution. The MSIL is stored in .**NET** assemblies after the developer has compiled the code written in any .**NET**-compliant programming language, such as Visual Basic and C#.

**MSIL (Microsoft Intermediat Lanugage) vs Java Byte Code**

<http://stackoverflow.com/questions/95163/differences-between-msil-and-java-bytecode>

They are essentially doing the same thing, MSIL is Microsoft's version of Java bytecode.

The main differences internally are:

1. Bytecode was developed for both compilation and interpretation, while MSIL was developed explicitly for JIT compilation
2. MSIL was developed to support multiple languages (C# and VB.NET, etc.) versus Bytecode being written for just Java, resulting in Bytecode being more similar to Java syntactically than IL is to any specific .NET language
3. MSIL has more explicit delineation between value and reference types

There are other language differences but most of them are not expressed at the byte code level, for example if memory serves Java's non-static inner classes (which do not exist in .NET) are not a bytecode feature, the compiler generates an additional argument to the inner class's constructor and passes the outer object. The same is true for .NET lambda expressions.

**language interoperability:**

When you compile your C# program, it is compiled into IL (Intermediate Language). That compiled program can then be used by any other .NET language, for example a Visual Basic program could reference yours, create an instance of your Math class and use it.

Visual Basic doesn't care (or know) that your Math class was written in C#, all it sees is a .NET class. That's language interoperability. If VB.NET could only use classes written in VB.NET, and C# could only use classes written C#, those languages wouldn't be interoperable.

Language Interoperability is the ability of code to interact with code that is written using a different programming language. Language Interoperability can help maximize code reuse and, therefore, improve the efficiency of the development process.  
  
Microsoft intermediate language (MSIL) is the CPU-independent instruction set into which .NET Framework programs are compiled. It contains instructions for loading, storing, initializing, and calling methods on objects.   
  
Combined with metadata and the [Common Type System](http://www.codeguru.com/forum/showthread.php?t=368669), MSIL allows for true cross-language integration. Prior to execution, MSIL is converted to machine code. It is not interpreted. 

* Who benefits from Language Interoperability?  
    
  Developers use a wide variety of tools and technologies, each of which might support different features and types, Language compilers and tools that target the common language runtime benefit from the runtime's built-in support for language interoperability.
* How is Language Interoperability possible?  
    
  The common language runtime provides the necessary foundation for language interoperability by specifying and enforcing a[Common Type System](http://www.codeguru.com/forum/showthread.php?t=368669) and by providing metadata. Because all languages targeting the runtime follow the [Common Type System](http://www.codeguru.com/forum/showthread.php?t=368669)rules for defining and using types, the usage of types is consistent across languages. Metadata enables language interoperability by defining a uniform mechanism for storing and retrieving information about types. Compilers store type information as metadata, and the common language runtime uses this information to provide services during execution; the runtime can manage the execution of multilanguage applications because all type information is stored and retrieved in the same way, regardless of the language the code was written in.

**Is .Net plateform independent?**

.**NET** is **Platform** Dependent because once the code is written it is complied into Microsoft Intermediate Language(MSIL) code which is **independent** of **platform** but it is half compiled code,then Common Language Runtime(CLR) convert it into device specific code i.e it is**platform** dependent.(MSIL is send to JIT through CLR

**What is assembly in .net? Types of Assembly**

Microsoft **.Net Assembly** is a logical unit of code, that contains code which the[Common Language Runtime](http://vb.net-informations.com/framework/common_language_runtime.htm) (CLR) executes. It is the smallest unit of deployment of a .net application and it can be a **.dll**or an **exe** . Assembly is really a collection of types and resource information that are built to work together and form a logical unit of functionality. It include both executable application files that you can run directly from Windows without the need for any other programs (.exe files), and libraries (.dll files) for use by other applications.

**DLL:**

Since DLLs are essentially the same as EXEs, the choice of which to produce as part of the linking process is for clarity, since it is possible to export functions and data from either.

It is not possible to directly execute a DLL, since it requires an EXE for the operating system to load it through an entry point, hence the existence of utilities like RUNDLL.EXE or RUNDLL32.EXE which provide the entry point and minimal framework for DLLs that contain enough functionality to execute without much support.

DLLs provide a mechanism for shared code and data, allowing a developer of shared code/data to upgrade functionality without requiring applications to be re-linked or re-compiled. From the application development point of view Windows and OS/2 can be thought of as a collection of DLLs that are upgraded, allowing applications for one version of the OS to work in a later one, provided that the OS vendor has ensured that the interfaces and functionality are compatible.

A *dynamic-link library* (DLL) is a module that contains functions and data that can be used by another module (application or DLL).

A DLL can define two kinds of functions: exported and internal. The exported functions are intended to be called by other modules, as well as from within the DLL where they are defined. Internal functions are typically intended to be called only from within the DLL where they are defined. Although a DLL can export data, its data is generally used only by its functions. However, there is nothing to prevent another module from reading or writing that address.

DLLs provide a way to modularize applications so that their functionality can be updated and reused more easily. DLLs also help reduce memory overhead when several applications use the same functionality at the same time, because although each application receives its own copy of the DLL data, the applications share the DLL code.

**Namespace vs dll**  
A class library is a library of classes and in some cases also other types of files (resources) that are compiled into an assembly that can be referenced from other applications. Please refer to the following page for more information: <http://msdn.microsoft.com/en-us/library/gg145045(v=vs.110).aspx>.

A namespace is used within a code source file to declare a scope that contains a set of related objects. Please refer to the following page for more information about namespaces in C#:<http://msdn.microsoft.com/en-us/library/z2kcy19k.aspx>

**Static class….**

## Static Members

The members of a class can be declared using the storage class modifier [static](https://www.mql5.com/en/docs/basis/variables/static). These data members are shared by all instances of this class and are stored in one place. Non-static data members are created for each class object variable.

The inability to declare static members of a class would have led to the need to declare these data on the [the global level](https://www.mql5.com/en/docs/basis/variables/global) of the program. It would break the relationship between the data and their class, and is not consistent with the basic paradigm of the OOP - joining data and methods for handling them in a class. The static member allows class data that are not specific to a particular instance to exist in the class scope.

Since a static class member does not depend on the particular instance, the reference to it is as follows:

|  |
| --- |
| class\_name::variable |

where *class\_name* is the name of the class, and *variable* is the name of the class member.

As you see, to access the static member of a class, [context resolution operator ::](https://www.mql5.com/en/docs/basis/operations/other#context_allow) is used. When you access a static member within class methods, the context operator is optional.

Static member of a class has to be explicitly initialized with desired value. For this it must be declared and initialized in global scope. The sequence of static members initialization will correspond to the sequence of their declaration in global scope.

<https://www.mql5.com/en/docs/basis/oop/staticmembers>

**Access Modifiers (Access Specifiers)** describes as the scope of accessibility of an Object and its members. All C# types and type members have an accessibility level . We can control the scope of the member object of a class using access specifiers. We are using access modifiers for providing security of our applications. When we specify the accessibility of a type or member we have to declare it by using any of the access modifiers provided by [CSharp](http://csharp.net-informations.com/) language.

C# provide five access specifiers , they are as follows :

**public, private , protected , internal and protected internal .**

**public :**

public is the most common access specifier in C# . It can be access from anywhere, that means there is no restriction on accessibility. The scope of the accessibility is inside class as well as outside. The type or member can be accessed by any other code in the same assembly or another assembly that references it.

**private :**

The scope of the accessibility is limited only inside the classes or struct in which they are declared. The private members cannot be accessed outside the class and it is the least permissive access level.

**protected :**

The scope of accessibility is limited within the class or struct and the class derived (Inherited )from this class.

**internal :**

The internal access modifiers can access within the program that contain its declarations and also access within the same assembly level but not from another assembly.

**protected internal :**

Protected internal is the same access levels of both protected and internal. It can access anywhere in the same assembly and in the same class also the classes inherited from the same class .

<http://www.completecsharptutorial.com/basic/access-specifiers.php>

Type Capacity

Int16 -- (-32,768 to +32,767)

Int32 -- (-2,147,483,648 to +2,147,483,647)

Int64 -- (-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)

The only real difference here is the size. All of the int types here are signed integer values which have varying sizes

* Int16: 2 bytes
* Int32 and int: 4 bytes
* Int64 : 8 bytes

**Difference b/w int & int32:**

**int**

It is a primitive data type defined in C#.

It is mapped to Int32 of FCL type.

It is a value type and represent System.Int32 struct.

It is signed and takes 32 bits.

It has minimum -2147483648 and maximum +2147483647 capacity.

**Int16**

It is a FCL type.

In C#, *short* is mapped to Int16.

It is a value type and represent System.Int16 struct.

It is signed and takes 16 bits.

It has minimum -32768 and maximum +32767 capacity.

**Int32**

It is a FCL type.

In C#, *int* is mapped to Int32.

It is a value type and represent System.Int32 struct.

It is signed and takes 32 bits.

It has minimum -2147483648 and maximum +2147483647 capacity.

**Int64**

It is a FCL type.

In C#, *long* is mapped to Int64.

It is a value type and represent System.Int64 struct.

It is signed and takes 64 bits.

It has minimum –9,223,372,036,854,775,808 and maximum 9,223,372,036,854,775,807 capacity.

**Difference b/w string & String:**

|  |  |
| --- | --- |
|  | [string](https://msdn.microsoft.com/en-us/library/362314fe.aspx) is an alias in C# for [System.String](https://msdn.microsoft.com/en-us/library/system.string.aspx). So technically, there is no difference. It's like [int](http://stackoverflow.com/questions/62503/c-int-or-int32-should-i-care)*[vs.](http://stackoverflow.com/questions/62503/c-int-or-int32-should-i-care)*[System.Int32](http://stackoverflow.com/questions/62503/c-int-or-int32-should-i-care).  As far as guidelines, I think it's generally recommended to use string any time you're referring to an object.  e.g.  string place = "world";  Likewise, I think it's generally recommended to use String if you need to refer specifically to the class.  e.g.  string greet = String.Format("Hello {0}!", place); |

## Difference b/w Constants & readonly:

## Constants

* Constants are static by default
* They must have a value at compilation-time (you can have e.g. 3.14 \* 2, but cannot call methods)
* Could be declared within functions
* Are copied into every assembly that uses them (every assembly gets a local copy of values)
* Can be used in attributes

## Readonly instance fields

* Must have set value, by the time constructor exits
* Are evaluated when instance is created

## Static readonly fields

* Are evaluated when code execution hits class reference (when new instance is created or a static method is executed)
* Must have an evaluated value by the time the static constructor is done
* It's not recommended to put ThreadStaticAttribute on these (static constructors will be executed in one thread only and will set the value for its thread; all other threads will have this value uninitialized)

 Consider a class defined in AssemblyA.

* public class Const\_V\_Readonly
* {
* public const int I\_CONST\_VALUE = 2;
* public readonly int I\_RO\_VALUE;
* public Const\_V\_Readonly()
* {
* I\_RO\_VALUE = 3;
* }
* }

AssemblyB references AssemblyA and uses these values in code. When this is compiled,

* in the case of the const value, it is like a find-replace, the value 2 is 'baked into' the AssemblyB's IL. This means that if tomorrow I'll update I\_CONST\_VALUE to 20 in the future. *AssemblyB would still have 2 till I recompile it*.
* in the case of the readonly value, it is like a ref to a memory location. The value is not baked into AssemblyB's IL. This means that if the memory location is updated, AssemblyB gets the new value without recompilation. So if I\_RO\_VALUE is updated to 30, you only need to build AssemblyA. All clients do not need to be recompiled.

public class Const\_V\_Readonly

{

public const int I\_CONST\_VALUE = 2;

public readonly char[] I\_RO\_VALUE = new Char[]{'a', 'b', 'c'};

public UpdateReadonly()

{

I\_RO\_VALUE[0] = 'V'; //perfectly legal and will update the value

I\_RO\_VALUE = new char[]{'V'}; //will cause compiler error

}

}

There is a small gotcha with readonly. A readonly field can be set multiple times within the constructor(s). Even if the value is set in two different chained constructors it is still allowed.

public class Sample {

private readonly string ro;

public Sample() {

ro = "set";

}

public Sample(string value) : this() {

ro = value; // this works even though it was set in the no-arg ctor

}

}

**const:** Can't be changed anywhere.

**readonly:** This value can only be changed in the constructor. Can't be changed in normal functions.

<http://www.exceptionnotfound.net/const-vs-static-vs-readonly-in-c-sharp-applications/>

<http://www.c-sharpcorner.com/UploadFile/2072a9/constant-vs-readonly-vs-static-keywords-in-C-Sharp/>

<https://abdullin.com/post/const-vs-readonly-vs-static-readonly-in-c-net/>

**ADO.NET** is a set of computer software components that programmers can use to access data and data services from the database. It is a part of the [base class library](https://en.wikipedia.org/wiki/Base_Class_Library) that is included with the [Microsoft .NET Framework](https://en.wikipedia.org/wiki/.NET_Framework). It is commonly used by programmers to access and modify data stored in [relational database systems](https://en.wikipedia.org/wiki/Relational_DBMS), though it can also access data in non-relational sources. ADO.NET is sometimes considered an evolution of [ActiveX Data Objects](https://en.wikipedia.org/wiki/ActiveX_Data_Objects) (ADO) technology, but was changed so extensively that it can be considered an entirely new product.

<https://en.wikipedia.org/wiki/ADO.NET>

**SQLCOMMAND, SQLCONNECTION,SQL DATA READER:**

using (SqlConnection cn = new SqlConnection("myConnectionstring"))

{

using (SqlCommand cm = new SqlCommand("myQuery", cn))

{

// maybe add sql parameters

using (SqlDataReader reader = cm.ExecuteReader())

{

// read values from reader object

return myReadValues;

}

}

}

var cn = new SqlConnection("myConnectionstring");

var cm = new SqlCommand("myQuery", cn);

var reader = cm.ExecuteReader(CommandBehavior.CloseConnection);

return reader;

<http://stackoverflow.com/questions/16985876/sqlconnection-sqlcommand-sqldatareader-idisposable>

<http://stackoverflow.com/questions/15005974/sqldatareader-and-sqlcommand>

<http://www.dotnetperls.com/sqlclient>

there are three objects when you fetch a resultset. Connection and command - two objects - would be enough if you'd do INSERT, UPDATE or DELETE against the db e.g anything that doesn't return a resultset. There are multitude of objexts because they've been thought to represent the concepts, when working with database.

- SqlConnection represents the connection to SQL Server database,   
- SqlCommand represents the action which "commands" the database, that is query something or do something with the data - basically it represents t-SQL statement / stored procedure  
- SqlDataReader is read-only, forward-only resultset created by the SqlCommand e.g based on what ther database was instructed to return - which can then be iterated or bound to a databound controls whatever

There are also a few more relevant objects than these.

- SqlParameter represents a parameter for SqlCommand's action e.g basically a parameter for parameterized SQL query or stored procedure.   
- SqlDataAdapter is an adapter object creating disconnected resultset based on SqlCommand - just like SqlDataReader is a resultset - but the disconnected resultset in ADO.NET is DataTable or DataSet (multiple DataTables in one set).

What comes to the code

**SqlConnection con = new SqlConnection(\_connectionstring);**

Declares SqlConnection variable and assigns it a new SqlConnection instance with given connection string. The connection string is set to ConnectionString property of SqlConnection and parsed at once, for revealing immediately possible parsing errors. Some members of SqlConnection are read-only and are based on the connection string, so their values are updated at this time. Nothing else is done at this point.

**SqlCommand cmd = new SqlCommand("'select, ID, Name, Price.....",con);**

Note that setting some of these in ctor vs via properties doesn't make any difference, it's just matter of usage convenience. Declares SqlCommand variable and assigns it a new SqlCommand instance with given ctor arguments. Command text / SQl statement is set directly to CommandText property, and SqlConnection is set to Connection proeprty. At this point nothing else is done, they are just referenced via internal members of SqlCommand.

**con.Open();**

Fetches an open database connection object from connection pool if one exists (there by default exists one pool per unique connection string to improve performance and keep the physical connection use in control). If not, it creates a new physical database connection to the SQL Server database. Also some stuff related to connection string can basically be checked only at this point for example success of the login to SQL server.

**SqlDataReader reader  = cmd.ExecuteReader();**

Declares SqlDataReader variable and assigns it instance of SqlDataReader instantiated by SqlCommand. SqlCommand does this by sending the T-SQl command to the database by using the given connection (which uses SQL Server .NEt Data provider to do the job), and builds SqlDataReader. The protocol used is specific to SQl Server (TDS - Tabular Data Stream)

 What it does as a whole, is very very complicated in terms of what happens in methods at code level. If you want you can inspect it by using Reflector tool, or viewing the .NET Framework 3.5's source code or with VS2008 you can step into .NET Framework 3.5's source while debugging (<http://weblogs.asp.net/scottgu/archive/2007/10/03/releasing-the-source-code-for-the-net-framework-libraries.aspx>).

**Partial class:**

There are several situations when splitting a class definition is desirable:

* When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time.
* When working with automatically generated source, code can be added to the class without having to recreate the source file. Visual Studio uses this approach when it creates Windows Forms, Web service wrapper code, and so on. You can create code that uses these classes without having to modify the file created by Visual Studio.
* To split a class definition, use the [partial](https://msdn.microsoft.com/en-us/library/wbx7zzdd.aspx) keyword modifier, as shown here:

**C#**

public partial class Employee

{

public void DoWork()

{

}

}

public partial class Employee

{

public void GoToLunch()

{

}

}

The **partial** keyword indicates that other parts of the class, struct, or interface can be defined in the namespace. All the parts must use the **partial**keyword. All the parts must be available at compile time to form the final type. All the parts must have the same accessibility, such as **public**,**private**, and so on.

If any part is declared abstract, then the whole type is considered abstract. If any part is declared sealed, then the whole type is considered sealed. If any part declares a base type, then the whole type inherits that class.

All the parts that specify a base class must agree, but parts that omit a base class still inherit the base type. Parts can specify different base interfaces, and the final type implements all the interfaces listed by all the partial declarations. Any class, struct, or interface members declared in a partial definition are available to all the other parts. The final type is the combination of all the parts at compile time.

<https://msdn.microsoft.com/en-us/library/wa80x488.aspx>

**Reflection**

Reflection provides objects (of type [Type](https://msdn.microsoft.com/en-us/library/system.type.aspx)) that describe assemblies, modules and types. You can use reflection to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties. If you are using attributes in your code, reflection enables you to access them. For more information, see [Extending Metadata Using Attributes](https://msdn.microsoft.com/en-us/library/5x6cd29c.aspx).

Here's a simple example of reflection using the static method **GetType** - inherited by all types from the **Object** base class - to obtain the type of a variable:

C#

[**VB**](https://msdn.microsoft.com/en-us/library/ms173183.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

// Using GetType to obtain type information:

int i = 42;

System.Type type = i.GetType();

System.Console.WriteLine(type);

The output is:

System.Int32

The following example uses reflection to obtain the full name of the loaded assembly.

C#

[**VB**](https://msdn.microsoft.com/en-us/library/ms173183.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

// Using Reflection to get information from an Assembly:

System.Reflection.Assembly info = typeof(System.Int32).Assembly;

System.Console.WriteLine(info);

The output is:

mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

|  |
| --- |
| **System_CAPS_noteNote** |
| The C# keywords **protected** and **internal** have no meaning in IL and are not used in the reflection APIs. The corresponding terms in IL are Familyand Assembly. To identify an **internal** method using reflection, use the [IsAssembly](https://msdn.microsoft.com/en-us/library/system.reflection.methodbase.isassembly.aspx) property. To identify a **protected internal** method, use the[IsFamilyOrAssembly](https://msdn.microsoft.com/en-us/library/system.reflection.methodbase.isfamilyorassembly.aspx). |

## [Reflection Overview](javascript:void(0))

Reflection is useful in the following situations:

* When you have to access attributes in your program's metadata. For more information, see [Retrieving Information Stored in Attributes](https://msdn.microsoft.com/en-us/library/71s1zwct.aspx).
* For examining and instantiating types in an assembly.
* For building new types at runtime. Use classes in [System.Reflection.Emit](https://msdn.microsoft.com/en-us/library/system.reflection.emit.aspx).
* For performing late binding, accessing methods on types created at run time. See the topic [Dynamically Loading and Using Types](https://msdn.microsoft.com/en-us/library/k3a58006.aspx).

**Reflection** objects are used for obtaining type information at runtime. The classes that give access to the metadata of a running program are in the**System.Reflection** namespace.

The **System.Reflection** namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application.

## Applications of Reflection

Reflection has the following applications:

* It allows view attribute information at runtime.
* It allows examining various types in an assembly and instantiate these types.
* It allows late binding to methods and properties
* It allows creating new types at runtime and then performs some tasks using those types.

## Viewing Metadata

We have mentioned in the preceding chapter that using reflection you can view the attribute information.

The **MemberInfo** object of the **System.Reflection** class needs to be initialized for discovering the attributes associated with a class. To do this, you define an object of the target class, as:

System.Reflection.MemberInfo info = typeof(MyClass);

<http://www.tutorialspoint.com/csharp/csharp_reflection.htm>

<http://www.codeproject.com/Articles/17269/Reflection-in-C-Tutorial>

**Garbage collection** (GC)

In computer science, **garbage collection** (GC) is a form of automatic memory management. The **garbage collector**, or just **collector**, attempts to reclaim **garbage**, or memory occupied by objects that are no longer in use by the program.

The .NET Framework's garbage collector manages the allocation and release of memory for your application. Each time you create a new object, the common language runtime allocates memory for the object from the managed heap. As long as address space is available in the managed heap, the runtime continues to allocate space for new objects. However, memory is not infinite. Eventually the garbage collector must perform a collection in order to free some memory. The garbage collector's optimizing engine determines the best time to perform a collection, based upon the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory.

In the common language runtime (CLR), the garbage collector serves as an automatic memory manager. It provides the following benefits:

* Enables you to develop your application without having to free memory.
* Allocates objects on the managed heap efficiently.
* Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Managed objects automatically get clean content to start with, so their constructors do not have to initialize every data field.
* Provides memory safety by making sure that an object cannot use the content of another object.

**GC.Collect Method ()**

<https://msdn.microsoft.com/en-us/library/xe0c2357(v=vs.110).aspx>

**Difference between generic and collections:**

Briefly, the basic difference between generic and non-generic collections:  
  
- Non-Generic collections - These are the collections that can hold elements of different data types. It holds all elements as object type.  
So it includes overhead of type conversions.  
  
- Generic collections - These are the collections that can hold data of same type and we can decide what type of data that collections can hold.  
  
Some advantages of generic collections - Type Safe, Secure, reduced overhead of type conversions.

One thing you need to understand, and which I haven't read in the existing answers yet, is that Collections and Generics have NOTHING to do with each other. They simply complement each other, but Collections can exist without Generics and Generics can exist without Collections.  
A collection is simply a collection of whatever. Object, strings, integers, Persons, all of the above...

Generic: ["In the simplest definition, generic programming is a style of computer programming in which algorithms are written in terms of to-be-specified-later types that are then instantiated when needed for specific types provided as parameters."](http://en.wikipedia.org/wiki/Generic_programming)

basically a 'Generic' is a method to make something work with ANY type.

<https://msdn.microsoft.com/en-us/library/ms172181(v=vs.110).aspx>

**Non-Generic                        Similar Generic Type**  
ArrayList              List<T>  
Hashtable              Dictionary<TKey,TValue>  
SortedList             SortedList<TKey,TValue>  
Queue                  Queue<T>  
Stack                  Stack<T>  
IEnumerable            IEnumerable<T>  
ICollection            N/A (use IEnumerable<T> anything that extends it)  
N/A                    ICollection<T>   
IList                  IList<T>  
CollectionBase         Collection<T>  
ReadOnlyCollectionBase ReadOnlyCollection<T>  
DictionaryBase         N/A (just implement IDictionary<TKey,TValue>  
N/A                    SortedDictionary<TKey,TValue>  
N/A                    KeyedCollection<TKey,TItem>  
N/A                    LinkedList<T>

<https://blogs.msdn.microsoft.com/kcwalina/2005/09/23/system-collections-vs-system-collection-generic-and-system-collections-objectmodel/>

<https://msdn.microsoft.com/en-us/library/b5bx6xee.aspx>

<http://www.codeproject.com/Questions/396402/what-is-difference-between-generics-and-collection>

<http://www.dotnet-tricks.com/Tutorial/csharp/U08E301212-Difference-between-Generics-and-Collections-with-example.html>

**Interface in c#**

An interface is defined as a syntactical contract that all the classes inheriting the interface should follow. The interface defines the **'what'** part of the syntactical contract and the deriving classes define the **'how'** part of the syntactical contract.

Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

**abstract class vs interface**

Abstract classes to some extent serve the same purpose, however, they are mostly used when only few methods are to be declared by the base class and the deriving class implements the functionalities.

## Declaring Interfaces

Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default. Following is an example of an interface declaration:

public interface ITransactions

{

// interface members

void showTransaction();

double getAmount();

}

<http://www.tutorialspoint.com/csharp/csharp_interfaces.htm>

<https://msdn.microsoft.com/en-us/library/87d83y5b.aspx>

**Interface in oop**

An interface is a description of the actions that an object can do... for example when you flip a light switch, the light goes on, you don't care how, just that it does. In Object Oriented Programming, an Interface is a description of all functions that an object must have in order to be an "X". Again, as an example, anything that "ACTS LIKE" a light, should have a turn\_on() method and a turn\_off() method. The **purpose** of interfaces is to allow **the computer** to **enforce** these properties and to know that an object of TYPE T (whatever the interface is ) must have functions called X,Y,Z, etc.

An interface is a programming structure/syntax that allows the computer to enforce certain properties on an object (class). For example, say we have a car class and a scooter class and a truck class. Each of these three classes should have a start\_engine() action. How the "engine is started" for each vehicle is left to each particular class, but the fact that **they must** have a start\_engine action is the domain of the interface.

## The syntax of an Interface

An interface has a very simple syntax that looks very much like a class definition... public interface XYZZY. Inside the {} of the interface is a list of functions that must be found in any object that purports to "follow" the interface.

Interfaces are placed in their own files which have the same name as the interface (are Capitalized) and end with the familiar language extension (e.g., ".as"). The following interface would be placed in a "Vehicle.as" file.

Here is an example of the Vehicle interface referred to above (only a partial definition).

<http://www.cs.utah.edu/~germain/PPS/Topics/interfaces.html>

<http://www.hitthebits.com/2012/11/why-use-interfaces.html>

**Difference Between Abstract Class and Interface in Java**

|  |  |
| --- | --- |
| **abstract Classes** | **Interfaces** |
| 1 | abstract class can extend only one class or one abstract class at a time | interface can extend any number of interfaces at a time |
| 2 | abstract  class  can extend from a class or from an abstract class | interface can extend only from an interface |
| 3 | abstract  class  can  have  both  abstract and concrete methods | interface can  have only abstract methods |
| 4 | A class can extend only one abstract class | A class can implement any number of interfaces |
| 5 | In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract | In an interface keyword ‘abstract’ is optional to declare a method as an abstract |
| 6 | abstract  class can have  protected , public and public abstract methods | Interface can have only public abstract methods i.e. by default |
| 7 | abstract class can have  static, final  or static final  variable with any access specifier | interface  can  have only static final (constant) variable i.e. by default |

.

<http://beginnersbook.com/2013/05/abstract-class-vs-interface-in-java/>

<http://www.javaworld.com/article/2077421/learn-java/abstract-classes-vs-interfaces.html>

<http://www.programmerinterview.com/index.php/java-questions/interface-vs-abstract-class/>

<http://www.codeproject.com/Articles/11155/Abstract-Class-versus-Interface>

**IN C#**

* If you anticipate creating multiple versions of your component, create an abstract class. Abstract classes provide a simple and easy way to version your components. By updating the base class, all inheriting classes are automatically updated with the change. Interfaces, on the other hand, cannot be changed once created. If a new version of an interface is required, you must create a whole new interface.
* If the functionality you are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing common functionality to unrelated classes.
* If you are designing small, concise bits of functionality, use interfaces. If you are designing large functional units, use an abstract class.
* If you want to provide common, implemented functionality among all implementations of your component, use an abstract class. Abstract classes allow you to partially implement your class, whereas interfaces contain no implementation for any members.

<https://msdn.microsoft.com/en-us/library/scsyfw1d(v=vs.71).aspx>

**IN OOP:**

Interfaces

An interface is a **contract**: the guy writing the interface says, "*hey, I accept things looking that way*", and the guy using the interface says "*Ok, the class I write looks that way*".

**An interface is an empty shell**, there are only the signatures of the methods, which implies that the methods do not have a body. The interface can't do anything. It's just a pattern.

E.G (pseudo code):

// I say all motor vehicles should look like this:

interface MotorVehicle

{

void run();

int getFuel();

}

// my team mate complies and writes vehicle looking that way

class Car implements MotorVehicle

{

int fuel;

void run()

{

print("Wrroooooooom");

}

int getFuel()

{

return this.fuel;

}

}

Implementing an interface consumes very little CPU, because it's not a class, just a bunch of names, and therefore there is no expensive look-up to do. It's great when it matters such as in embedded devices.

**Abstract classes**

Abstract classes, unlike interfaces, are classes. They are more expensive to use because there is a look-up to do when you inherit from them.

Abstract classes look a lot like interfaces, but they have something more : you can define a behavior for them. It's more about a guy saying, "these classes should look like that, and they have that in common, so fill in the blanks!".

e.g:

// I say all motor vehicles should look like this :

abstract class MotorVehicle

{

int fuel;

// they ALL have fuel, so why not let others implement this?

// let's make it for everybody

int getFuel()

{

return this.fuel;

}

// that can be very different, force them to provide their

// implementation

abstract void run();

}

// my team mate complies and writes vehicle looking that way

class Car extends MotorVehicle

{

void run()

{

print("Wrroooooooom");

}

}

<http://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class>

**Virtual keyword, override keyword**

### **Virtual Keyword**

Virtual keyword is used for generating a virtual path for its derived classes on implementing method overriding. Virtual keyword is used within a set with override keyword. It is used as:

Hide   Copy Code

*// Base Class*

class A

{

public virtual void show()

{

Console.WriteLine("Hello: Base Class!");

Console.ReadLine();

}

}

### **Override Keyword**

Override keyword is used in the derived class of the base class in order to override the base class method.Override keyword is used with virtual keyword, as:

*// Base Class*

class A

{

public virtual void show()

{

Console.WriteLine("Hello: Base Class!");

Console.ReadLine();

}

}

*// Derived Class*

class B : A

{

public override void show()

{

Console.WriteLine("Hello: Derived Class!");

Console.ReadLine();

}

}

### **New Keyword**

New keyword is also used in polymorphism concept, but in the case of method overloading So what does overloading means, in simple words we can say procedure of hiding your base class through your derived class.

It is implemented as:

Hide   Copy Code

class A

{

public void show()

{

Console.WriteLine("Hello: Base Class!");

Console.ReadLine();

}

}

class B : A

{

public new void show()

{

Console.WriteLine("Hello: Derived Class!");

Console.ReadLine();

}

}

<http://www.codeproject.com/Articles/816448/Virtual-vs-Override-vs-New-Keyword-in-Csharp>

<https://msdn.microsoft.com/en-us/library/ms173153.aspx>

<https://msdn.microsoft.com/en-us/library/9fkccyh4.aspx>

<https://msdn.microsoft.com/en-us/library/ebca9ah3.aspx>

<http://en.cppreference.com/w/cpp/language/override>

<http://stackoverflow.com/questions/159978/c-sharp-keyword-usage-virtualoverride-vs-new>

# **Sealed Class:**

 Sealed class is used to define the inheritance level of a class.

 The sealed modifier is used to prevent derivation from a class. An error occurs if a sealed class is specified as the base class of another class.

**Some points to remember:**     
  
1.  A class, which restricts inheritance for security reason is declared, sealed class.  
2.  Sealed class is the last class in the hierarchy.  
3.  Sealed class can be a derived class but can't be a base class.  
4.  A sealed class cannot also be an abstract class. Because abstract class has to provide functionality and here we are   
     restricting it to inherit.

<http://www.c-sharpcorner.com/UploadFile/puranindia/what-are-sealed-classes-and-sealed-methods/>

<http://www.c-sharpcorner.com/UploadFile/mahesh/SealedClasses11142005063733AM/SealedClasses.aspx>

<http://stackoverflow.com/questions/16217313/static-vs-sealed-class-difference>

<https://www.techopedia.com/definition/25637/sealed-class-c>

<http://www.functionx.com/csharp/classes/sealed.htm>

**ASP.NET** is an [open-source](https://en.wikipedia.org/wiki/Open_source) [server-side](https://en.wikipedia.org/wiki/Server-side_scripting) [web application framework](https://en.wikipedia.org/wiki/Web_application_framework) designed for [web development](https://en.wikipedia.org/wiki/Web_development) to produce [dynamic web pages](https://en.wikipedia.org/wiki/Dynamic_web_page). It was developed by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) to allow [programmers](https://en.wikipedia.org/wiki/Programmer) to build dynamic [web sites](https://en.wikipedia.org/wiki/Web_site), [web applications](https://en.wikipedia.org/wiki/Web_application) and [web services](https://en.wikipedia.org/wiki/Web_service).

It was first released in January 2002 with version 1.0 of the [.NET Framework](https://en.wikipedia.org/wiki/.NET_Framework), and is the successor to Microsoft's [Active Server Pages](https://en.wikipedia.org/wiki/Active_Server_Pages) (ASP) technology. ASP.NET is built on the [Common Language Runtime](https://en.wikipedia.org/wiki/Common_Language_Runtime) (CLR), allowing programmers to write ASP.NET code using any supported[.NET language](https://en.wikipedia.org/wiki/List_of_CLI_languages). The ASP.NET [SOAP](https://en.wikipedia.org/wiki/SOAP) extension framework allows ASP.NET components to process SOAP messages.

ASP.NET is in the process of being re-implemented as a modern and modular [web framework](https://en.wikipedia.org/wiki/Web_framework), together with other frameworks like [Entity Framework](https://en.wikipedia.org/wiki/Entity_Framework). The new framework will make use of the new open-source [.NET Compiler Platform](https://en.wikipedia.org/wiki/.NET_Compiler_Platform) (code-name "Roslyn") and be [cross platform](https://en.wikipedia.org/wiki/Cross_platform).[ASP.NET MVC](https://en.wikipedia.org/wiki/ASP.NET_MVC), ASP.NET Web API, and ASP.NET Web Pages (a platform using only [Razor](https://en.wikipedia.org/wiki/ASP.NET_Razor) pages) will merge into a unified MVC 6.[[4]](https://en.wikipedia.org/wiki/ASP.NET#cite_note-asp.net-4) The project is called [ASP.NET vNext](https://en.wikipedia.org/wiki/ASP.NET_vNext).

**Page Life cycle:**

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. It is important for you to understand the page life cycle so that you can write code at the appropriate life-cycle stage for the effect you intend.

If you develop custom controls, you must be familiar with the page life cycle in order to correctly initialize controls, populate control properties with view-state data, and run control behavior code. The life cycle of a control is based on the page life cycle, and the page raises many of the events that you need to handle in a custom control.

In general terms, the page goes through the stages outlined in the following table. In addition to the page life-cycle stages, there are application stages that occur before and after a request but are not specific to a page. For more information, see [Introduction to the ASP.NET Application Life Cycle](http://go.microsoft.com/fwlink/?LinkId=133108) and [ASP.NET Application Life Cycle Overview for IIS 7.0](https://msdn.microsoft.com/en-us/library/bb470252.aspx).

Some parts of the life cycle occur only when a page is processed as a postback. For postbacks, the page life cycle is the same during a partial-page postback (as when you use an [UpdatePanel](https://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control) as it is during a full-page postback.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Page request | The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page. |
| Start | In the start stage, page properties such as [Request](https://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) and [Response](https://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) are set. At this stage, the page also determines whether the request is a postback or a new request and sets the [IsPostBack](https://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property. The page also sets the [UICulture](https://msdn.microsoft.com/en-us/library/system.web.ui.page.uiculture.aspx) property. |
| Initialization | During page initialization, controls on the page are available and each control's [UniqueID](https://msdn.microsoft.com/en-us/library/system.web.ui.control.uniqueid.aspx) property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state. |
| Load | During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state. |
| Postback event handling | If the request is a postback, control event handlers are called. After that, the [Validate](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basevalidator.validate.aspx) method of all validator controls is called, which sets the [IsValid](https://msdn.microsoft.com/en-us/library/system.web.ui.ivalidator.isvalid.aspx) property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.) |
| Rendering | Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the [Render](https://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method for each control, providing a text writer that writes its output to the [OutputStream](https://msdn.microsoft.com/en-us/library/system.web.httpresponse.outputstream.aspx) object of the page's [Response](https://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) property. |
| Unload | The [Unload](https://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as [Response](https://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) and [Request](https://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) are unloaded and cleanup is performed. |

<https://msdn.microsoft.com/en-us/library/ms178472.aspx>

ASP.NET life cycle specifies, how:

* ASP.NET processes pages to produce dynamic output
* The application and its pages are instantiated and processed
* ASP.NET compiles the pages dynamically

The ASP.NET life cycle could be divided into two groups:

* Application Life Cycle
* Page Life Cycle

## ASP.NET Application Life Cycle

The application life cycle has the following stages:

* User makes a request for accessing application resource, a page. Browser sends this request to the web server.
* A unified pipeline receives the first request and the following events take place:
  + An object of the class ApplicationManager is created.
  + An object of the class HostingEnvironment is created to provide information regarding the resources.
  + Top level items in the application are compiled.
* Response objects are created. The application objects such as HttpContext, HttpRequest and HttpResponse are created and initialized.
* An instance of the HttpApplication object is created and assigned to the request.
* The request is processed by the HttpApplication class. Different events are raised by this class for processing the request.

## ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

The Page class creates a hierarchical tree of all the controls on the page. All the components on the page, except the directives, are part of this control tree. You can see the control tree by adding trace= "true" to the page directive. We will cover page directives and tracing under 'directives' and 'event handling'.

The page life cycle phases are:

* Initialization
* Instantiation of the controls on the page
* Restoration and maintenance of the state
* Execution of the event handler codes
* Page rendering

Understanding the page cycle helps in writing codes for making some specific thing happen at any stage of the page life cycle. It also helps in writing custom controls and initializing them at right time, populate their properties with view-state data and run control behavior code.

Following are the different stages of an ASP.NET page:

* **Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.
* **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
* **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
* **Page load** - At this stage, control properties are set using the view state and control state values.
* **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
* **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
* **Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
* **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

## ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

* **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page\_PreInit handler.
* **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page\_Init handler.
* **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
* **LoadViewState** - LoadViewState event allows loading view state information into the controls.
* **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
* **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page\_PreLoad handler.
* **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page\_Load handler.
* **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page\_LoadComplete handler
* **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
* **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
* **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page\_Render handler.
* **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page\_UnLoad handler.

<http://www.tutorialspoint.com/asp.net/asp.net_life_cycle.htm>

<http://www.codeproject.com/Tips/444310/ASP-NET-Page-Life-Cycle-Events>

<http://www.codeproject.com/Articles/667308/ASP-NET-Page-Life-Cycle-Events>

<http://ajax.asp.net/ajax/documentation/live/overview/ajaxclientevents.aspx>

**Exception Handling:**

An exception (or exceptional event) is a problem that arises during the **execution** of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.

An exception can occur for many different reasons, below given are some scenarios where exception occurs.

* A user has entered invalid data.
* A file that needs to be opened cannot be found.
* A network connection has been lost in the middle of communications or the JVM has run out of memory.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these we have three categories of Exceptions you need to understand them to know how exception handling works in Java,

* **Checked exceptions:** A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of (handle) these exceptions.

For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then an*FileNotFoundException*occurs, and compiler prompts the programmer to handle the exception.

import java.io.File;

import java.io.FileReader;

public class FilenotFound\_Demo {

public static void main(String args[]){

File file=new File("E://file.txt");

FileReader fr = new FileReader(file);

}

}

If you try to compile the above program you will get exceptions as shown below.

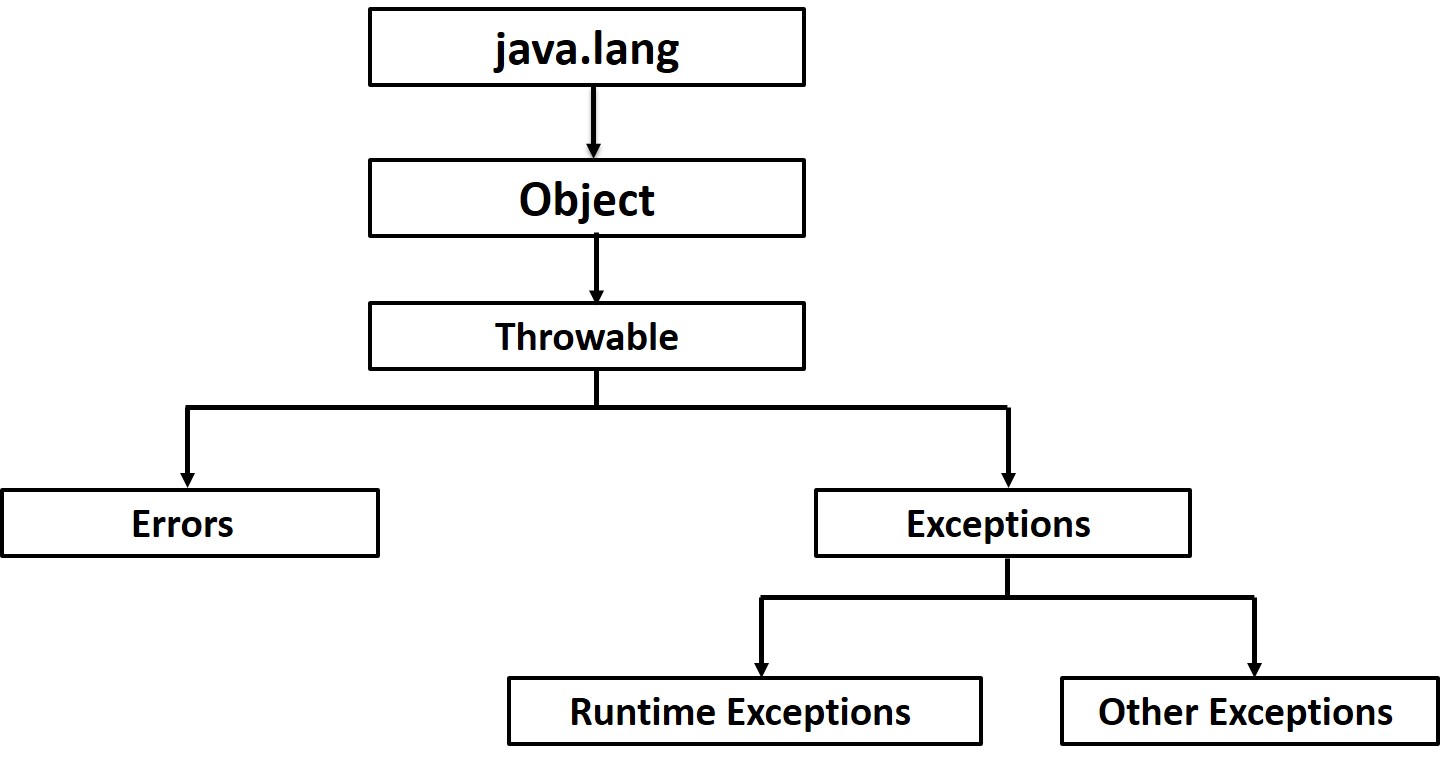
C:\>javac FilenotFound\_Demo.java

FilenotFound\_Demo.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be thrown

FileReader fr = new FileReader(file);

^

1 error



**Q1) What is an Exception?**

Ans) The exception is said to be thrown whenever an exceptional event occurs in java which signals that something is not correct with the code written and may give unexpected result. An exceptional event is a occurrence of condition which alters the normal program flow. Exceptional handler is the code that does something about the exception.

**Q2) Exceptions are defined in which java package?**

Ans) All the exceptions are subclasses of java.lang.Exception

**Q3) How are the exceptions handled in java?**

Ans) When an exception occurs the execution of the program is transferred to an appropriate exception handler. The **try-catch-finally** block is used to handle the exception.

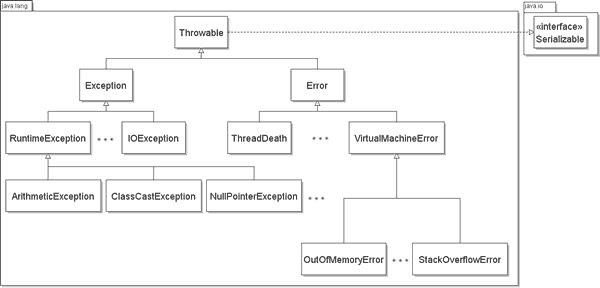
The code in which the exception may occur is enclosed in a try block, also called as a guarded region.

The **catch clause** matches a specific exception to a block of code which handles that exception.

And the clean up code which needs to be executed no matter the exception occurs or not is put inside the **finally block**

**Q4) Explain the exception hierarchy in java.**

Ans) The hierarchy is as follows:



Throwable is a parent class of all Exception classes. There are two types of Exceptions:**Checked exceptions** and **UncheckedExceptions or RunTimeExceptions**. Both type of exceptions extends Exception class.

**Q5) What is Runtime Exception or unchecked exception?**

Ans) Runtime exceptions represent problems that are the result of a programming problem. Such problems include arithmetic exceptions, such as dividing by zero; pointer exceptions: such as trying to access an object through a null reference; and indexing exceptions: such as attempting to access an array element through an index that is too large or too small.

Runtime exceptions need not be explicitly caught in try catch block as it can occur anywhere in a program, and in a typical one they can be very numerous. Having to add runtime exceptions in every method declaration would reduce a program's clarity. Thus, the compiler does not require that you catch or specify runtime exceptions (although you can). The solution to rectify is to correct the programming logic where the exception has occurred or provide a check.

**Q6) What is checked exception?**

Ans) Checked exception are the exceptions which forces the programmer to catch them explicitly in try-catch block. It is a subClass of Exception. Example: IOException.

**[Q7) What is difference between Error and Exception?](http://java-questions.com/Exceptions-interview-questions.html" \l "difference-error-exception)**

Ans) An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you can not repair them at runtime.Though error can be caught in catch block but the execution of application will come to a halt and is not recoverable.

While exceptions are conditions that occur because of bad input or human error etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.)

**Q8) What is difference between ClassNotFoundException and NoClassDefFoundError?**

Ans) A ClassNotFoundException is thrown when the reported class is not found by the ClassLoader in the CLASSPATH. It could also mean that the class in question is trying to be loaded from another class which was loaded in a parent classloader and hence the class from the child classloader is not visible.

Consider if NoClassDefFoundError occurs which is something like

java.lang.NoClassDefFoundError

src/com/TestClass

does not mean that the TestClass class is not in the CLASSPATH. It means that the class TestClass was found by the ClassLoader however when trying to load the class, it ran into an error reading the class definition. This typically happens when the class in question has static blocks or members which use a Class that's not found by the ClassLoader. So to find the culprit, view the source of the class in question (TestClass in this case) and look for code using static blocks or static members.

**Q9) What is throw keyword?**

Ans) Throw keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application.The exception thrown should be subclass of Throwable.

public void parent(){

try{

child();

}catch(MyCustomException e){ }

}

public void child{

String iAmMandatory=null;

if(iAmMandatory == null){

throw (new MyCustomException("Throwing exception using throw keyword");

}

}

**Q10) What is use of throws keyword?**

Ans) If the function is not capable of handling the exception then it can ask the calling method to handle it by simply putting the **throws clause** at the function declaration.

public void parent(){

try{

child();

}catch(MyCustomException e){ }

}

public void child throws MyCustomException{

//put some logic so that the exception occurs.

}

**Q11) What are the possible combination to write try, catch finally block?**

Ans)

1 try{

//lines of code that may throw an exception

}catch(Exception e){

//lines of code to handle the exception thrown in try block

}finally{

//the clean code which is executed always no matter the exception occurs or not.

}

2 try{}finally{}

3 try{

}catch(Exception e){

//lines of code to handle the exception thrown in try block

}

The catch blocks must always follow the try block. If there are more than one catch blocks they all must follow each other without any block in between. The finally block must follow the catch block if one is present or if the catch block is absent the finally block must follow the try block.

**Q12) How to create custom Exception?**

Ans) To create you own exception extend the Exception class or any of its subclasses.

* class New1Exception extends Exception { } // this will create Checked Exception
* class NewException extends IOExcpetion { } // this will create Checked exception
* class NewException extends NullPonterExcpetion { } // this will create UnChecked exception

**Q13) When to make a custom checked Exception or custom unchecked Exception?**

Ans) If an application can reasonably be expected to recover from an exception, make it a checked exception. If an application do want to do anything to recover from the exception, make it an unchecked exception. For e.g in client server model, if server is not able to talk with DB or some IO operation went wrong, its ok to throw the unchecked exception so that conatiner can handle it and throw appropriate error response.

**Q14)What is StackOverflowError?**

Ans) The StackOverFlowError is an Error Object thorwn by the Runtime System when it Encounters that your application/code has ran out of the memory. It may occur in case of recursive methods or a large amount of data is fetched from the server and stored in some object. This error is generated by JVM.

**Q15) Why did the designers decide to force a method to specify all uncaught checked exceptions that can be thrown within its scope?**

Ans) Any Exception that can be thrown by a method is part of the method's public programming interface. Those who call a method must know about the exceptions that a method can throw so that they can decide what to do about them. These exceptions are as much a part of that method's programming interface as its parameters and return value.

<http://www.journaldev.com/2167/java-exception-interview-questions-and-answers>

<http://www.tutorialspoint.com/java/java_exceptions.htm>