1. **What is object-oriented programming (OOP)?**

OOP is a technique to develop logical modules, such as classes that contain properties, methods, fields, and events. An object is created in the program to represent a class. Therefore, an object encapsulates all the features, such as data and behavior that are associated to a class. OOP allows developers to develop modular programs and assemble them as software. Objects are used to access data and behaviors of different software modules, such as classes, namespaces, and sharable assemblies. .NET Framework supports only OOP languages, such as Visual Basic .NET, Visual C#, and Visual C++.

1. **What is a class?**

A class describes all the attributes of objects, as well as the methods that implement the behavior of member objects. It is a comprehensive data type, which represents a blue print of objects. It is a template of object.   
  
A class can be defined as the primary building block of OOP. It also serves as a template that describes the properties, state, and behaviors common to a particular group of objects.  
  
A class contains data and behavior of an entity. For example, the aircraft class can contain data, such as model number, category, and color and behavior, such as duration of flight, speed, and number of passengers. A class inherits the data members and behaviors of other classes by extending from them.

1. **What is an object?**

They are instance of classes. It is a basic unit of a system. An object is an entity that has attributes, behavior, and identity. Attributes and behavior of an object are defined by the class definition.

1. **What is the relationship between a class and an object?**

A class acts as a blue-print that defines the properties, states, and behaviors that are common to a number of objects. An object is an instance of the class. For example, you have a class called Vehicle and Car is the object of that class. You can create any number of objects for the class namedVehicle, such as Van, Truck, and Auto.  
  
The new operator is used to create an object of a class. When an object of a class is instantiated, the system allocates memory for every data member that is present in the class.

1. **Explain the basic features of OOPs.**

The following are the four basic features of OOP:

* + **Abstraction** - Refers to the process of exposing only the relevant and essential data to the users without showing unnecessary information.
  + **Polymorphism** - Allows you to use an entity in multiple forms.
  + **Encapsulation** - Prevents the data from unwanted access by binding of code and data in a single unit called object.
  + **Inheritance** - Promotes the reusability of code and eliminates the use of redundant code. It is the property through which a child class obtains all the features defined in its parent class. When a class inherits the common properties of another class, the class inheriting the properties is called a derived class and the class that allows inheritance of its common properties is called a base class.

1. **What is the difference between arrays and collection?**

**Array**:

* + You need to specify the size of an array at the time of its declaration. It cannot be resized dynamically.
  + The members of an array should be of the same data type.

**Collection**:

* + The size of a collection can be adjusted dynamically, as per the user's requirement. It does not have fixed size.
  + Collection can have elements of different types.

1. **What are collections and generics?**

A collection can be defined as a group of related items that can be referred to as a single unit. The System.Collections namespace provides you with many classes and interfaces. Some of them are - ArrayList, List, Stack, ICollection, IEnumerable, and IDictionary. Generics provide the type-safety to your class at the compile time. While creating a data structure, you never need to specify the data type at the time of declaration. TheSystem.Collections.Generic namespace contains all the generic collections.

1. **How can you prevent your class to be inherited further?**

You can prevent a class from being inherited further by defining it with the sealed keyword.

1. **What is the index value of the first element in an array?**

In an array, the index value of the first element is 0 (zero).

1. **Can you specify the accessibility modifier for methods inside the interface?**

All the methods inside an interface are always public, by default. You cannot specify any other access modifier for them.

1. **Is it possible for a class to inherit the constructor of its base class?**

No, a class cannot inherit the constructor of its base class.

1. **How is method overriding different from method overloading?**

Overriding involves the creation of two or more methods with the same name and same signature in different classes (one of them should be parent class and other should be child).   
  
Overloading is a concept of using a method at different places with same name and different signatures within the same class.

1. **What is the difference between a class and a structure?**

**Class**:

* 1. A class is a reference type.
  2. While instantiating a class, CLR allocates memory for its instance in heap.
  3. Classes support inheritance.
  4. Variables of a class can be assigned as null.
  5. Class can contain constructor/destructor.

**Structure**:

* 1. A structure is a value type.
  2. In structure, memory is allocated on stack.
  3. Structures do not support inheritance.
  4. Structure members cannot have null values.
  5. Structure does not require constructor/destructor and members can be initialiazed automatically.

1. **What are similarities between a class and a structure.**

Structures and classes are the two most important data structures that are used by programmers to build modular programs by using OOP languages, such as Visual Basic .NET, and Visual C#. The following are some of the similarities between a class and a structure:

* 1. Access specifiers, such as public, private, and protected, are identically used in structures and classes to restrict the access of their data and methods outside their body.
  2. The access level for class members and struct members, including nested classes and structs, is private by default. Private nested types are not accessible from outside the containing type.
  3. Both can have constructors, methods, properties, fields, constants, enumerations, events, and event handlers.
  4. Both structures and classes can implement interfaces to use multiple-inheritance in code.
  5. Both structures and classes can have constructors with parameter.
  6. Both structures and classes can have delegates and events.

1. **What is a multicast delegate?**

Each delegate object holds reference to a single method. However, it is possible for a delegate object to hold references of and invoke multiple methods. Such delegate objects are called multicast delegates or combinable delegates.

1. **Can you declare an overridden method to be static if the original method is not static?**

No. Two virtual methods must have the same signature.

1. **Why is the virtual keyword used in code?**

The virtual keyword is used while defining a class to specify that the methods and the properties of that class can be overridden in derived classes.

1. **Can you allow a class to be inherited, but prevent a method from being overridden in C#?**

Yes. Just declare the class public and make the method sealed.

1. **Define enumeration?**

Enumeration is defined as a value type that consists of a set of named values. These values are constants and are called enumerators. An enumeration type is declared using the enum keyword. Each enumerator in an enumeration is associated with an underlying type that is set, by default, on the enumerator. The following is an example that creates an enumeration to store different varieties of fruits:  
  
enum Fruits {Mango, Apple, orange, Guava};   
  
In the preceding example, an enumeration Fruits is created, where number 0 is associated with Mango, number 1 with Apple, number 2 with Orange, and number 3 with Guava. You can access the enumerators of an enumeration by these values.

1. **In which namespace, all .NET collection classes are contained?**

The System.Collections namespace contains all the collection classes.

1. **Is it a good practice to handle exceptions in code?**

Yes, you must handle exceptions in code so that you can deal with any unexpected situations that occur when a program is running. For example, dividing a number by zero or passing a string value to a variable that holds an integer value would result in an exception.

1. **Explain the concept of constructor?**

Constructor is a special method of a class, which is called automatically when the instance of a class is created. It is created with the same name as the class and initializes all class members, whenever you access the class. The main features of a constructor are as follows:

* + Constructors do not have any return type.
  + Constructors can be overloaded.
  + It is not mandatory to declare a constructor; it is invoked automatically by .NET Framework.

1. **Can you inherit private members of a class?**

No, you cannot inherit private members of a class because private members are accessible only to that class and not outside that class.

1. **Does .NET support multiple inheritance?**

.NET does not support multiple inheritance directly because in .NET, a class cannot inherit from more than one class. .NET supports multiple inheritance through interfaces.

1. **How has exception handling changed in .NET Framework 4.0?**

In .NET 4.0, a new namespace, System.Runtime.ExceptionServices, has been introduced which contains the following classes for handling exceptions in a better and advanced manner:

* + HandleProcessCorruptedStateExceptionsAttribute Class - Enables managed code to handle the corrupted state exceptions that occur in an operating system. These exceptions cannot be caught by specifying the try...catch block. To handle such exceptions, you can apply this attribute to the method that is assigned to handle these exceptions.
  + FirstChanceExceptionEventArgs Class - Generates an event whenever a managed exception first occurs in your code, before the common language runtime begins searching for event handlers.

1. **What is a delegate?**

A delegate is similar to a class that is used for storing the reference to a method and invoking that method at runtime, as required. A delegate can hold the reference of only those methods whose signatures are same as that of the delegate. Some of the examples of delegates are type-safe functions, pointers, or callbacks.

1. **What is the syntax to inherit from a class in C#?**

When a class is derived from another class, then the members of the base class become the members of the derived class. The access modifier used while accessing members of the base class specifies the access status of the base class members inside the derived class.  
  
The syntax to inherit a class from another class in C# is as follows:   
class MyNewClass : MyBaseclass

1. **State the features of an interface.**

An interface is a template that contains only the signature of methods. The signature of a method consists of the numbers of parameters, the type of parameter (value, reference, or output), and the order of parameters. An interface has no implementation on its own because it contains only the definition of methods without any method body. An interface is defined using the interface keyword. Moreover, you cannot instantiate an interface. The various features of an interface are as follows:

* + An interface is used to implement multiple inheritance in code. This feature of an interface is quite different from that of abstract classes because a class cannot derive the features of more than one class but can easily implement multiple interfaces.
  + It defines a specific set of methods and their arguments.
  + Variables in interface must be declared as public, static, and final while methods must bepublic and abstract.
  + A class implementing an interface must implement all of its methods.
  + An interface can derive from more than one interface.

1. **Can you use the 'throws' clause to raise an exception?**

No, the throws clause cannot be used to raise an exception. The throw statement signals the occurrence of an exception during the execution of a program. When the program encounters a throwstatement, the method terminates and returns the error to the calling method.

1. **Define an array.**

An array is defined as a homogeneous collection of elements, stored at contiguous memory locations, which can be referred by the same variable name. All the elements of an array variable can be accessed by index values. An Index value specifies the position of a particular element in an array variable.

1. **What are methods?**

Methods are the building blocks of a class, in which they are linked together to share and process data to produce the result. In other words, a method is a block of code that contains a series of statements and represents the behavior of a class. While declaring a method you need to specify the access specifier, the return value, the name of the method, and the method parameters. All these combined together is called the signature of the method.

1. **What is a namespace?**

Namespace is considered as a container that contains functionally related group of classes and other types.

1. **Do events have return type?**

No, events do not have return type.

1. **What is the function of the Try-Catch-Finally block?**

The try block encloses those statements that can cause exception and the catch block handles the exception, if it occurs. Catch block contains the statements that have to be executed, when an exception occurs. The finally block always executes, irrespective of the fact whether or not an exception has occurred. The finally block is generally used to perform the cleanup process. If any exception occurs in the try block, the program control directly transfers to its corresponding catch block and later to thefinally block. If no exception occurs inside the try block, then the program control transfers directly to the finally block.

1. **How can you prevent a class from overriding in C# and Visual Basic?**

You can prevent a class from overriding in C# by using the sealed keyword; whereas, theNotInheritable keyword is used to prevent a class from overriding in Visual Basic.

1. **What are abstract classes? What are the distinct characteristics of an abstract class?**

An abstract class is a class that cannot be instantiated and is always used as a base class.  
The following are the characteristics of an abstract class:

* + You cannot instantiate an abstract class directly. This implies that you cannot create an object of the abstract class; it must be inherited.
  + You can have abstract as well as non-abstract members in an abstract class.
  + You must declare at least one abstract method in the abstract class.
  + An abstract class is always public.
  + An abstract class is declared using the abstract keyword.

The basic purpose of an abstract class is to provide a common definition of the base class that multiple derived classes can share.

1. **Give a brief description of properties in C# and the advantages that are obtained by using them in programs.**

In C#, a property is a way to expose an internal data element of a class in a simple and intuitive manner. In other words, it is a simple extension of data fields. You can create a property by defining an externally available name and then writing the set and get property accessors. The get property accessor is used to return the property value. The set property accessor is used to assign a new value to the property.

1. **Explain different types of inheritance.**

Inheritance in OOP is of four types:

* + **Single inheritance** - Contains one base class and one derived class
  + **Hierarchical inheritance** - Contains one base class and multiple derived classes of the same base class
  + **Multilevel inheritance** - Contains a class derived from a derived class
  + **Multiple inheritance** - Contains several base classes and a derived class

All .NET languages supports single, hierarchical, and multilevel inheritance. They do not support multiple inheritance because in these languages, a derived class cannot have more than one base class. However, you can implement multiple inheritance in.NET through interfaces.

1. **You have defined a destructor in a class that you have developed by using the C# programming language, but the destructor never executed. Why did the destructor not execute?**

The runtime environment automatically invokes the destructor of a class to release the resources that are occupied by variables and methods of an object. However, in C#, programmers cannot control the timing for invoking destructors, as Garbage Collector is only responsible for releasing the resources used by an object. Garbage Collector automatically gets information about unreferenced objects from .NET's runtime environment and then invokes the Finalize() method.  
  
Although, it is not preferable to force Garbage Collector to perform garbage collection and retrieve all inaccessible memory, programmers can use the Collect() method of the Garbage Collector class to forcefully execute Garbage Collector.

1. **What is a hashtable?**

Hashtable is a data structure that implements the IDictionary interface. It is used to store multiple items and each of these items is associated with a unique string key. Each item can be accessed using the key associated with it. In short, hashtable is an object holding the key-value pairs.

1. **Can users define their own exceptions in code?**

Yes, customized exceptions can be defined in code by deriving from the System.Exception class.

1. **Is it possible to execute two catch blocks?**

You are allowed to include more than one catch block in your program; however, it is not possible to execute them in one go. Whenever, an exception occurs in your program, the correct catch block is executed and the control goes to the finally block.

1. **What do you mean by data encapsulation?**

Data encapsulation is a concept of binding data and code in single unit called object and hiding all the implementation details of a class from the user. It prevents unauthorized access of data and restricts the user to use the necessary data only.

1. **What is the difference between procedural and object-oriented programming?**

Procedural programming is based upon the modular approach in which the larger programs are broken into procedures. Each procedure is a set of instructions that are executed one after another. On the other hand, OOP is based upon objects. An object consists of various elements, such as methods and variables.  
  
Access modifiers are not used in procedural programming, which implies that the entire data can be accessed freely anywhere in the program. In OOP, you can specify the scope of a particular data by using access modifiers - public, private, internal, protected, and protected internal.

1. **Explain the concept of destructor?**

A destructor is a special method for a class and is invoked automatically when an object is finally destroyed. The name of the destructor is also same as that of the class but is followed by a prefix tilde (~).  
  
A destructor is used to free the dynamic allocated memory and release the resources. You can, however, implement a custom method that allows you to control object destruction by calling the destructor.  
  
The main features of a destructor are as follows:

* + Destructors do not have any return type
  + Similar to constructors, destructors are also always public
  + Destructors cannot be overloaded.

1. **Can you declare a private class in a namespace?**

The classes in a namespace are internal, by default. However, you can explicitly declare them aspublic only and not as private, protected, or protected internal. The nested classes can be declared as private, protected, or protected internal.

1. **A structure in C# can implement one or more interfaces. Is it true or false?**

Yes, it is true. Like classes, in C#, structures can implement one or more interfaces.

1. **What is a static constructor?**

Static constructors are introduced with C# to initialize the static data of a class. CLR calls the static constructor before the first instance is created.  
  
The static constructor has the following features:

* + No access specifier is required to define it.
  + You cannot pass parameters in static constructor.
  + A class can have only one static constructor.
  + It can access only static members of the class.
  + It is invoked only once, when the program execution begins.

1. **What are the different ways a method can be overloaded?**

The different ways to overload a method are given as follows:

* + By changing the number of parameters used
  + By changing the order of parameters
  + By using different data types for the parameters

1. **Differentiate between an abstract class and an interface.**

**Abstract Class**:

* + A class can extend only one abstract class
  + The members of abstract class can be private as well as protected.
  + Abstract classes should have subclasses
  + Any class can extend an abstract class.
  + Methods in abstract class can be abstract as well as concrete.
  + There can be a constructor for abstract class.
  + The class extending the abstract class may or may not implement any of its method.
  + An abstract class can implement methods.

**Interface**

* + A class can implement several interfaces
  + An interface can only have public members.
  + Interfaces must have implementations by classes
  + Only an interface can extend another interface.
  + All methods in an interface should be abstract
  + Interface does not have constructor.
  + All methods of interface need to be implemented by a class implementing that interface.
  + Interfaces cannot contain body of any of its method.

1. **What are queues and stacks?**

Stacks refer to a list in which all items are accessed and processed on the Last-In-First-Out (LIFO) basis. In a stack, elements are inserted (push operation) and deleted (pop operation) from the same end called**top**.   
  
Queues refer to a list in which insertion and deletion of an item is done on the First-In-First-Out (FIFO) basis. The items in a queue are inserted from the one end, called the **rear** end, and are deleted from the other end, called the **front** end of the queue.

1. **Define an event.**

Whenever an action takes place in a class, that class provides a notification to other classes or objects that are assigned to perform particular tasks. These notifications are called events. For example, when a button is clicked, the class generates an event called Click. An event can be declared with the help of the event keyword.

1. **What are structures?**

Structure is a heterogeneous collection of elements referenced by the same name. A structure is declared using the struct keyword. The following is an example that creates a structure to store an employee's information:

struct emp

{

fixed int empID[15];

fixed char name[30];

fixed char addr[50];

fixed char dept[15];

fixed char desig[15];

}

The preceding example defines a structure emp and the members of this structure specify the information of an employee.

1. **When do you really need to create an abstract class?**

We define abstract classes when we define a template that needs to be followed by all the derived classes.