1. What is type safety in .NET?

Type safety in .NET has been introduced to prevent the objects of one type from peeking into the memory assigned for the other object. Type safety helps the compiler and CLR in .NET to execute the code in the memory space defined for the objects and instances. It further helps to build the robust and error free applications which are definitely error free at run time.

```
private void ChangeValue(out object par)
{
   par = new String('x', 10);
}


SomeOtherType obj = new SomeOtherType();
ChangeValue(out obj); //compile time error
```

If .NET would have allowed this code to execute, type safety could have easily been compromised here resulting in some unpredictable results and hence in turn reducing the credibility of the code. You can read more about

2. What is the base class from which all the classes are derived from?

System.Object is the base class from which all the reference type as well value type are derived from. In C# we cannot create a class which doesn't derive from these two classes.

3. What are the functions which System.Object class have.

Ans. Following are the functions which are present in the System.Object class

· **Equals**(Object) – Determines whether the specified object is equal to the current object.
· **Equals**(Object, Object) – Determines whether the specified object instances are considered equal.
· **Finalize** – Allows an object to try to free resources and perform other clean up operations before it is reclaimed by garbage collection.

· **GetHashCode** – Serves as the default hash function.
· **GetType** – Gets the Type of the current instance.
· **MemberwiseClone** – Creates a shallow copy of the current Object.
· **ReferenceEquals** – Determines whether the specified Object instances are the same instance.
· **ToString** – Returns a string that represents the current object.

4. Why do we override Equals() method of System.Object.

The default implementation of the Equals look like following.

```
public class Object
{
   public virtual Boolean Equals(Object obj)
   {
      //If both the references point to the same location in memory. They must have same value
      if(this == obj)
         return true;

      //if the object does not have same references
      return false;
   }
}
```

From the above code snippet we can check that the default implementation only checks the references of the current object and the obj parameter.

But suppose if we want to override this default implementation in that case we can override this function and develop our own equality compare logic as shown below.

```
public class Person
{
   public string Name{get; set;}
   public override Boolean Equals(Object obj)
   {
      Person p = obj as Person; ;
      //If both the references point to the same location in memory. They must have same value
      if(p != null && !string.IsNullOrWhiteSpace(p.Name))
      {
         return (p.Name.Equals(this.Name));
      }
      //if the object does not have same references
      return false;
   }
}
```

5. What are the functions which we can override in base class derived from System.object?

Following are the methods which we can override in the user defined type
Equals – Supports comparisons between objects.
Finalize – Performs cleanup operations before an object is automatically reclaimed.
GetHashCode – Generates a number corresponding to the value of the object to support the use of a hash table.
ToString – Manufactures a human-readable text string that describes an instance of the class

6. What is difference between compile time polymorphism and run time polymorphism

Compile time polymorphism or the static polymorphism is the type of polymorphism which helps to identify the called methos at the compile time. We use different signature of the same method in the defining class as shown below

```
public class MyClass
{
  public int Add(int a, int b)
  {
     return a + b;
  }

  public int Add(int a, int b, int c)
  {
     return a + b + c;
  }
}
```

Run time polymorphism – Run time polymorphism or dynamic polymorphism is the type of polymorphism which helps us to determine at the run time which function should be called. Example of the same is as below .

```
public class MyClass
  {
     public virtual int Calculation(int a, int b)
```

```
    {
        return a + b;
    }
}

public class MyDerivedClass : MyClass
{
    public override int Calculation(int a, int b)
    {
        return a * b;
    }
}
```

This type of polymorphism can be achieved by using the abstract or virtual keyword with the functions in the base class, which is turn need to be overridden in the derived class.

7. Does C# supports multiple inheritance.

C# does not support multiple inheritance with classes. It means that we cannot have a class derived from multiple classes in C#.

9. How is multiple inheritance achieved in C#?

Multiple inheritance in C# can be achieved using interface as shown in the following code snippet. .

```
interface IA
{

}

interface IB
{

}

public class MyClass:IA, IB
{

}
```

10. What are interfaces in C# and how they are different from abstract class.

Ans . This is one of the common interview questions.An interface contains only signatures or declarations for a number of related functions that a class or a struct can implement. As we know that multiple inheritance is not

supported in C#, it can be achieved using the interfaces as shown in the previous question. Moreover interfaces only contains the method declarations and not the implementation. Apart from methods interfaces can also contain Property and events.

```
public abstract class Abstract
  {
    public void GeneralizedFunction()
    {
      //do something genaralized which derived classes adhere to
    }

    public abstract int AbsrtactMethod();
  }

  public class ConcreteClass:Abstract
  {
    public override int AbsrtactMethod()
    {
      return 1;
    }
  }
```

11. What are static classes.

- Ans Static classes are the classes which cannot be instantiated.
- These classes cannot be derived from and the method parameters cannot be of type static class.
- One class is shared across the whole application.

12. When are static classes initialized.

Ans. The static classes are loaded in the memory as soon as any of the data member or the function member is called for the first time.

13. What is difference between static class and singleton class.

Static class

Cannot be initialized
Cannot be the base class
Cannot be the parameter to a function
All the member variables and functions should be static
Can have only static constructor.

Singleton Class

Can have only one instance per app domain
Can be passed as parameter to a function.
Singleton class usually have a single private constructor

14. Is it possible to have a static constructor in class. If yes why do we need to have a static constructor.

Static constructors are used to initialize the static member variables of the class. Hence a static constructor executed once per type instead of once per instance. A type can define only one static constructor and it must be parameterless. The runtime automatically invokes a static constructor just prior to type being used. Two things trigger this-

- Instantiating the type
- Accessing a static member in the type.

```
class TestClass
{
  static Test(){ Console.WriteLine("Type Initialized"); }
}
```

15. What are delegates.

Delegates are reference types which are used to contain the function pointers. Delegates are type safe and they adhere to compile type safety. public delegate void MyDelegate(int number);

16. What is a multicast delegate.

Each delegate type is derived from Multicast delegate, which would help to have a invocation list for each and every delegate.

17. How can we achieve asynchronous programming using delegates.

We can call a method asynchronously using the delegate by using the BeginInvoke() function of the delegate.

18. Are delegates type safe.

Yes delegates are type safe and at compile time itself we can get to know the parameter type and return type of a delegate.

19. What is the difference between Convert.ToSting() and Object.ToSting() instance method of a class.

Convert.ToString() can handle the NULL values but ToString() cannot handle null, it will throw object reference null exception and we can override Object.ToString() to provide custom implementation for the class.

20. What are is and as keywords used for?

The **is** keyword is used to check the casting at runtime.

```
if(var is Employee)
{
    // if variable is of type Employee then work on it
}
```

As keyword is used to cast the variable of one type to other type(base type).

```
emp = var as Employee;
if(emp != null)
{
//use the emp variable
   emp = var as Employee;
}
```