Difference b/w extends vs implements

case1:
extends :: One class can extends only class at a time
```
   eg:: class One{}
           class Two extends One{}
```

implements:: One class can implements any no of interface at a time.
```
   eg:: interface One{
              void methodOne();
           }
          interface Two{
             void methodTwo();
          }
       class Demo implements One,Two{
            public void methodOne(){}
            public void methodTwo(){}
       }
```

case2:
A class can extend a class and can implements any no of interfaces simultaneously.
```
     eg: interface One{
              public void methodOne();
           }
          class Two{
           public void methodTwo(){}
          }
          class Three extends Two implements One{
          @Override
            public void methodOne(){
                ....
            }

            @Override
            public void methodTwo(){
            ..
            }
         }
```

case3:
An interface can extend any no of interfaces at at time.
```
        eg:: interface One{
                     public void methodOne();
                }
                interface Two{
                     public void methodTwo();
                }
                interface Three extends One,Two{
                   public void methodOne();
               public void methodTwo();
                   public void methodThree();
                }
```

Answer the following
==========
Which of the following is true?
  a. A class can extend any no of class at a time.

  b. An interface can extend only one interface at at time.
  c. A class can implement only one interface at at a time.
  d. A class can extend a class and can implement an interface but not both simultaneously.
  e. An interface can implements any no of Interfaces at a time.
  f. None of the above

answer: f

Consider the expression X extends Y which of the possiblity of X and Y expression is true?
   1. Both x and y should be classes.
   2. Both x and y should be interfaces.
   3. Both x and y can be classes or can be interfaces.
   4. No restriction.

Equation : X extends Y == > true
            class extends class => true
            interface extends interface => true

Answer: 3

Q>
Predict X,Y,Z
  a. X extends Y,Z?
  b. X extends Y implements Z?
  c. X implements Y,Z?
  d. X implements Y extends Z?

a. X extends Y,Z?
      X => interface
      Y => interface
      Z => interface

b. X extends Y implements Z?
      X => class
      Y => class
      Z => interface

c. X implements Y,Z?
      X => class
      Y => interface
      Z => interface

d. X implements Y extends Z
      combination is illegal

Interface Methods
==========
 Every method present inside the interface is public.
 Every method present inside the inteface is abstract.
   How many declaration are valid?
       a. void methodOne();
       b. public void methodOne();
       c. abstract void methodOne();
       d. public abstract void methodOne();
answer: All are valid

public   => To make the method available for every implementation class.

abstract => Implementation class is responsible for providing the implementation.

```
eg: jdbc api (java.sql.*)
         |
         |
   implementation should be given by
      a. mysql
      b. oracle
      c. postgresql
```

Since the methods present inside the interface is
   => public,abstract methods illegal combination of modifiers are
       static,private,protected,strictfp,synchronized,native,final.

Interface variables

  => Inside the interface we can define variables.
  => Inside the interface variables define requirement level constants.
  => Every variable present inside the interface is by default public static final.

```
eg:: interface ISample{
            int x=10;
        }
     public :: To make it available for implementation class Object.
     static  :: To access it without using implementation class Name.
     final    :: Implementation class can access the value without any
modification.
```

variable declaration inside interface
  a. int x=10;
  b. public int x=10;
  c. static int x=10;
  d. final int x=10;
  e. public static int x=10;
  f. public final int x=10;
  g. static final int x=10;
  h. public static final int x=10;

Answer: All are legal

since the variable defined in interface is public static final,we cannot use
modifiers like private,protected,transient,volatile.
since the variable is static and final,compulsorily it should be initialized at the
time of declaration otherwise it would result in
compile time error.

```
eg:: interace IRemote{ int x;}// compile time error.
```

=> interface variables can be accessed from implementation class,but cannot modify
if we try to modify
     it would result in compile time error.

```
eg::  interface Remote{
         int VOLUME = 100;
      }
      class Lg implements Remote{
         public static void main(String... args){
            VOLUME=0;//CE:: cannot assign a value to final variable VOLUME
```

```
            System.out.println("value of volume is ::"+VOLUME);
           }
      }

eg:: interface Remote{
         int VOLUME = 100;
      }
      class Lg implements Remote{
         public static void main(String... args){
            int VOLUME=0;//local variable
            System.out.println("value of volume is ::"+VOLUME);//0
           }
      }
```

Interface Naming Conflicts
==========

Case 1::
If 2 interfaces contain a method with same signature and same return type in the implementation class only
one method implementation is enough.

```
eg::
   interface Left{ public void methodOne();}
   interface Right{public void methodOne();}
   class Test implements Left,Right{
      @Override
        public void methodOne(){
            ...
        }
   }
```

Case2:
If 2 interfaces contain a method with same name but different arguments in the implementation class we have to provide
implementation for both methods and these methods acts as a Overload methods.

```
eg::
   interface Left{ public void methodOne();}
   interface Right{public void methodOne(int i);}
   class Test implements Left,Right{
        @Override
        public void methodOne(){
             ...
        }

      @Override
        public void methodOne(int i){
            ...
        }
   }
```

case3::
If two interfaces contains a method with same signature but different return types then it is not possible to implement
both interface simultaneously.

```
eg:: interface Left { public void methodOne(); }
        interface Right{ public int methodOne(); }
       class Test implements Left,Right{
```

```
        @Override
        public void methodOne(){
             ...
        }

      @Override
        public int methodOne(){
            ...
        }
    }
```

Note:

Q> Can a java class implements 2 interfaces simultaneously?

yes possible, except if two interfaces contains a method with same signature but different return types.

Variable naming conflicts::

Two variables can contain a variable with same name and there may be a chance variable naming conflicts but we can resolve variable naming conflicts by using interface names.

example1:

```
  interface Left{ int x=888;}
  interface Right{ int x=999;}
  public class Test implements Left,Right{
      public static void main(String... args){
            System.out.println(Left.x);
            System.out.println(Right.x);
      }
  }
```

Note:

inside interface the methods are by default "public and abstract".

inside interface the variables are by default " public static and final".

We can also write an interface without any variable or abstract methods.

```
interface Serializable{

}
class SampleImpl implements Serializable{

}

interface Cloneable{

}
class SampleImple implements Cloneable{

}
```

MarkerInterface

==========

=> If an interface does not contain any methods and by implementing that interface if our Object will get some ability such type of interface are called "Marker Interface"/"Tag Interface"/"Ability Interface".

=> example

Serializable,Cloneable,SingleThreadModel.

example1
   By implementing Serializable interface we can send that object across the network and we can save state of an
   object into the file.

example2
   By implementing SingleThreadModel interfaace servlet can process only one client request at a time so that we can
   get "Thread Safety".

example3
   By implementing Cloneable Interface our object is in a position to provide exactly duplicate cloned object.

Without having any methods in marker interface how objects will get ability?
  Ans.JVM is responsible to provide requried ability.

Why JVM is providing the required ability to Marker Interfaces?
 Ans. To reduce the complexity of the programming.

Can we create our own marker interface?
  Yes, it is possible but we need to customize JVM.(hard for beginner)

==========
Adapter class(It is a design pattern allowed to solve the problem of direct implementation of interface methods)
==========
It is a simple java class that implements an interface only with empty implememtation for every method.
If we implement an interface compulsorily we should give the body for all the methods whether it
is required or not. This approach increases the length of the code and reduces readability.

```
eg:: interface X{
      void m1();
        void m2();
        void m3();
      void m4();
        void m5();
     }
    class Test implements X{
      public void m3(){
          System.out.println("I am from m3()");
        }
        public void m2(){}
        public void m3(){}
      public void m4(){}
        public void m5(){}
    }
```

In the above approach, even though we want only m3(), still we need to give body for all the abstract methods, which increase the
length of the code, to reduce this we need to use "Adapater class".
Instead of implementing the interface directly we opt for "Adapter class".
Adapter class are such classes which implements the interface and gives dummy

implementation for all the abstract methods of interface.
So if we extends Adapter classes then we can easily give body only for those methods which are interested in giving the body.

```
eg::
interface X{
      void m1();
        void m2();
        void m3();
      void m4();
        void m5();
}
abstract class AdapaterX implements X{
      public void m1(){}
        public void m2(){}
        public void m3(){}
      public void m4(){}
        public void m5(){}
}
class TestApp extends AdapterX{
      public void m3(){
            System.out.println("I am from m3()");
      }
}
```

```
eg:
  Servlet(I)
      | implements
GenericServlet(abstract class)
      | extends
HttpServlet(abstract class)
      | extends
 MyServlet(class)
```