Rules of Interface
++++++++++++++++++
case 1:
 1. Whenever we are implementing an interface, we need to give body for all the abstract methods present inside the
    interface.If we fail to give body for all the methods present inside the interface, then we need to mark the class as "abstract".
 2. For an interface instantiation(creation of object) is not possible.
 3. For an interface, creation of reference is possible.
 4. Through interface we achieve :: TruePolymorphism(Overriding).

JDK1.7 Version : interface
++++++++++++++++++++++++++
```
//Pure-Abstract-class : SRS[Software Requirment Specification]
interface ICalculator
{
      //By Default methods are :: public abstract
       void add(int a,int b);
       void sub(int a,int b);
       void mul(int a,int b);
       void div(int a,int b);
}

//Implemented class : concrete class
class CalculatorImpl implements ICalculator
{
      @Override
      public void add(int a,int b)
      {
            int sum = a+b;
            System.out.println("The sum is :: "+sum);
      }

      @Override
      public void sub(int a,int b)
      {
            int diff = a-b;
            System.out.println("The sub is :: "+diff);
      }

      @Override
      public void mul(int a,int b)
      {
            int res = a*b;
            System.out.println("The res is :: "+res);
      }

      @Override
      public void div(int a,int b)
      {
            int quotient = a/b;
            System.out.println("The Quotient is :: "+quotient);
      }
}

public class Test
{
      public static void main(String[] args)
      {
```

```
            //Creating a reference of interface
            ICalculator calc;

            calc = new CalculatorImpl();

            //Calling the method based on runtime object
            calc.add(10,20);
            calc.sub(20,10);
            calc.mul(10,20);
            calc.div(5,2);
      }
}
```

```
Output
The sum is :: 30
The sub is :: 10
The res is :: 200
The Quotient is :: 2
```

```
Given
1. public class KungFu {
2.    public static void main(String[] args) {
3.          Integer x = 400;
4.          Integer y = x;
5.          x++;
6.          StringBuilder sb1 = new StringBuilder("123");
7.          StringBuilder sb2 = sb1;
8.          sb1.append("5");
9.          System.out.println((x == y) + " " + (sb1 == sb2));
10.   }
11.}
```

```
What is the result?
A. true true
B. false true
C. true false
D. false false
E. Compilation fails.
F. An exception is thrown at runtime.

Answer: B
```

```
Case2: Whenever we are implementing an interface method compulsory, it should be
declared as public otherwise we will get "CompileTime Error".

//Pure-Abstract-class : SRS
interface ICalculator
{
      //By Default methods are :: public abstract
       void add(int a,int b);
       void sub(int a,int b);
}

//Implemented class : concrete class
class CalculatorImpl implements ICalculator
{
```

```
	@Override
	void add(int a,int b)
	{
		int sum = a+b;
		System.out.println("The sum is :: "+sum);
	}

	@Override
	void sub(int a,int b)
	{
		int diff = a-b;
		System.out.println("The sub is :: "+diff);
	}
}

public class Test
{
	public static void main(String[] args)
	{
		//Creating a reference of interface
		ICalculator calc;

		calc = new CalculatorImpl();

		//Calling the method based on runtime object
		calc.add(10,20);
		calc.sub(20,10);
	}
}

Output
	CE: attempting to assign weaker access privileges; was public


case3:
 => Relationship b/w interface to class is always "implements".
 => Relationship b/w interface to interface is always "extends".
 => If we implemented the interface which has extended from one more interface,
then as a programmer the implementation class should give
    body for all the abstract methods present in the interface, if not we need to
mark the class as "abstract", otherwise the code would result
    in "CompileTime Error".

eg#1.
//Pure-Abstract-class : SRS
interface ICalculator1
{
	//By Default methods are :: public abstract
	 void add(int a,int b);
	 void sub(int a,int b);
}

//Pure-Abstract-class : SRS
interface ICalculator2 extends ICalculator1
{
	//By Default methods are :: public abstract
	void mul(int a,int b);
	void div(int a,int b);
}
```

```
//Implemented class : concrete class
class CalculatorImpl implements ICalculator2
{
	@Override
	public void add(int a,int b)
	{
		int sum = a+b;
		System.out.println("The sum is :: "+sum);
	}

	@Override
	public void sub(int a,int b)
	{
		int diff = a-b;
		System.out.println("The sub is :: "+diff);
	}

	@Override
	public void mul(int a,int b){
		int res = a*b;
		System.out.println("The res is :: "+res);
	}

	@Override
	public void div(int a,int b){
		int quotient = a/b;
		System.out.println("The quotient is :: "+quotient);
	}
}

public class Test
{
	public static void main(String[] args)
	{
		//Creating a reference of interface
		ICalculator2 calc;

		calc = new CalculatorImpl();

		//Calling the method based on runtime object
		calc.add(10,20);
		calc.sub(20,10);
		calc.mul(10,20);
		calc.div(5,2);
	}
}
Output
The sum is :: 30
The sub is :: 10
The res is :: 200
The quotient is :: 2


case4:
 At a time one class can extend from how many classes?
 Answer. One because java doesn't support multiple inheritance through class to
avoid "Ambiguity problem".
```

At a time one class can implement how many interfaces?
Answer: Yes possible, so we can say mulitple inheritance is supported in java through "interfaces" and "Ambiguity problem " won't occur because
Compiler will keep the method signature in the implementation class only if it is not available.
As noticed in the below example ICalculator1 and ICalculator2 both have void add(int a,int b) method,but compiler will keep only
one method void add(int a,int b) in the implementation class through which "Ambiguity problem" will not occur in interfaces.

At a time can one class implement an interface and extends a class?
Answer: yes,but first we need to have extends and followed by implements.

eg#1.

```
//Pure-Abstract-class : SRS
interface ICalculator1
{
	//By Default methods are :: public abstract
	 void add(int a,int b);
	 void sub(int a,int b);
}

//Pure-Abstract-class : SRS
interface ICalculator2
{
	//By Default methods are :: public abstract
	void mul(int a,int b);
	void div(int a,int b);
	void add(int a,int b);
}

//Implemented class : concrete class
class CalculatorImpl implements ICalculator1,ICalculator2
{
	@Override
	public void add(int a,int b)
	{
		int sum = a+b;
		System.out.println("The sum is :: "+sum);
	}

	@Override
	public void sub(int a,int b)
	{
		int diff = a-b;
		System.out.println("The sub is :: "+diff);
	}

	@Override
	public void mul(int a,int b){
		int res = a*b;
		System.out.println("The res is :: "+res);
	}

	@Override
	public void div(int a,int b){
		int quotient = a/b;
		System.out.println("The quotient is :: "+quotient);
	}
```

```
}

public class Test
{
	public static void main(String[] args)
	{
		//Creating a reference of interface
		CalculatorImpl calc;

		calc = new CalculatorImpl();

		//Calling the method based on runtime object
		calc.add(10,20);
		calc.sub(20,10);
		calc.mul(10,20);
		calc.div(5,2);
	}
}

Output
The sum is :: 30
The sub is :: 10
The res is :: 200
The quotient is :: 2

To promote loose coupling we follow the rule of
	interface ->abstract class -> class

//Pure-Abstract-class : SRS
interface ICalculator1
{
	//By Default methods are :: public abstract
	 void add(int a,int b);
	 void sub(int a,int b);
}

//Pure-Abstract-class : SRS
interface ICalculator2
{
	//By Default methods are :: public abstract
	void mul(int a,int b);
	void div(int a,int b);
	void add(int a,int b);
}

abstract class Calculator implements ICalculator1,ICalculator2
{

}

//Implemented class : concrete class
class CalculatorImpl extends Calculator
{
	@Override
	public void add(int a,int b)
	{
		int sum = a+b;
		System.out.println("The sum is :: "+sum);
	}
```

```
	@Override
	public void sub(int a,int b)
	{
		int diff = a-b;
		System.out.println("The sub is :: "+diff);
	}

	@Override
	public void mul(int a,int b){
		int res = a*b;
		System.out.println("The res is :: "+res);
	}

	@Override
	public void div(int a,int b){
		int quotient = a/b;
		System.out.println("The quotient is :: "+quotient);
	}
}

public class Test
{
	public static void main(String[] args)
	{
		//Creating a reference of interface
		Calculator calc;

		calc = new CalculatorImpl();

		//Calling the method based on runtime object
		calc.add(10,20);
		calc.sub(20,10);
		calc.mul(10,20);
		calc.div(5,2);
	}
}
Output
The sum is :: 30
The sub is :: 10
The res is :: 200
The quotient is :: 2

eg#2.
interface ICalculator
{
	public void add(int a,int b);
}

class CalculatorDemo
{
	public void sub(int a,int b)
	{
		System.out.println("The sub is :: "+(a-b));
	}
}
class CalculatorImpl extends CalculatorDemo implements ICalculator
{
	@Override
```

```
	public void add(int a,int b){
		System.out.println("The sum is :: "+(a+b));
	}
}

public class Test
{
	public static void main(String[] args)
	{
			CalculatorImpl calc;
			calc = new CalculatorImpl();
			calc.add(10,20);
			calc.sub(10,3);
	}
}
Output
The sum is :: 30
The sub is :: 7

eg#3.
interface ICalculator
{
	public void add(int a,int b);
}

class CalculatorDemo
{
	public void sub(int a,int b)
	{
		System.out.println("The sub is :: "+(a-b));
	}
}
abstract class Calculator extends CalculatorDemo implements ICalculator
{

}

class CalculatorImpl extends Calculator
{
	@Override
	public void add(int a,int b){
		System.out.println("The sum is :: "+(a+b));
	}
}

public class Test
{
	public static void main(String[] args)
	{
			Calculator calc;

			calc = new CalculatorImpl();

			calc.add(10,20);
			calc.sub(10,3);
	}
}

Output
```

```
The sum is :: 30
The sub is :: 7

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.
Answer: 3

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


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


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

  d. X implements Y extends Z?
      invalid case.

Interface variables
++++++++++++++++++
  => Inside the interface we can define variables.
  => Inside the interface variables is to 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 valid

Note:
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 IRemote
{
      //public static final
      int MIN_VOLUME = 0;
      int MAX_VOLUME = 100;
}

public class Test implements IRemote
{
      public static void main(String[] args)
      {     int MIN_VOLUME = -5;
            System.out.println(MIN_VOLUME);
            System.out.println(IRemote.MIN_VOLUME);
            System.out.println(Test.MIN_VOLUME);
      }
}
Output
-5
0
0

eg#2.
interface IRemote
{
      //public static final
      int MIN_VOLUME = 0;
      int MAX_VOLUME = 100;
}

public class Test implements IRemote
{
      public static void main(String[] args)
      {     MIN_VOLUME = -5;
            System.out.println(MIN_VOLUME);
            System.out.println(IRemote.MIN_VOLUME);
            System.out.println(Test.MIN_VOLUME);
      }
}
Output
CE: final variable value can't be modified.


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#1.

```
interface IRight
{
	public void methodOne();
}
interface ILeft
{
	public void methodOne();
}

public class Test implements ILeft,IRight
{
	@Override
	public void methodOne()
	{
		System.out.println("Impl for MethodOne...");
	}
	public static void main(String[] args)
	{
		Test t =new Test();
		t.methodOne();
	}
}
```

Output

```
Impl for 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#1.

```
interface IRight
{
	public void methodOne();
}
interface ILeft
{
	public void methodOne(int i);
}

public class Test implements ILeft,IRight
{
	@Override
	public void methodOne()
	{
		System.out.println("Impl for MethodOne...");
	}

	@Override
	public void methodOne(int i)
	{
		System.out.println("Impl for MethodOne with One argument");
	}
```

```
	public static void main(String[] args)
	{
		Test t =new Test();
		t.methodOne();
		t.methodOne(10);
	}
}
```

Output

```
Impl for MethodOne...
Impl for MethodOne with One argument
```

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#1.

```
interface IRight
{
	public void methodOne();
}
interface ILeft
{
	public int methodOne();
}

public class Test implements ILeft,IRight
{
	@Override
	public void methodOne()
	{
		System.out.println("Impl for MethodOne...");
	}

	@Override
	public int methodOne()
	{
		System.out.println("Impl for MethodOne with One argument");
	}
	public static void main(String[] args)
	{
		Test t =new Test();

		//Overloading
		t.methodOne();
		t.methodOne();
	}
}
```

Output

```
CE: ambigous method call.
```

Can a java class implement any no of interfaces simultaneously?
Answer.yes, 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.

eg#1.

```
//SRS :: methods   -> public abstract
//SRS :: variables -> public static final
interface IRight
{
      int x = 888;
}
interface ILeft
{
      int x = 999;
}

public class Test implements ILeft,IRight
{
      public static void main(String[] args)
      {
            //System.out.println(x);
            //System.out.println(Test.x);
            System.out.println(IRight.x);
            System.out.println(ILeft.x);
      }
}
```

Output
888
999

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,RandomAccess.

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 cusomtize JVM.

==========
Adapter class
==========

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:: interface Servlet{....}
```

```
     abstract class GenericServlet  implements Servlet{}
     abstract class HttpServlet extends GenericServlet{}
     class MyServlet extends HttpServlet{}
```

Note:: Adapter class and Marker interface are big utilites to programmer to simplify programming.

Interview Questions
++++++++++++++++++
Q>What is the difference b/w abstract class and interface?
Q>Every method present inside the interface is abstract,but in abstract class also we can take only abstract methods also then what is the need
   of interface concept?
Q> Why abstract class can contains constructor and interface doesn't contains constructor?
Q> When to go for interface and when to go for abstract class?

Tommo class :: 8.00PM to 10.30PM