```
Pillars of oops
+++++++++++++++
1. Datahiding (using private access modifier)
2. Abstraction(using abstract class and interfaces)
3. Encapsulation = Datahiding + Abstraction
4. Polymorphism(Overloading, Overriding)


interfaces in java
++++++++++++++++++
 1. In java interfaces has been introduced to provide "SRS".
    SRS -> Software Requirement Specification.

 2. In java interfaces would act like a contract b/w the client and the service
provider.
    In java interfaces also represents the mechanism to put rules from the client
end to the service provider
      eg: JDBC API   (rules given by SUNMS) for Database vendors like MySQL,
Oracle,Sybase .....
          Servlet API(rules given by SUNMS) for Server vendors like Tomcat,
Glassfish, JBoss,......

 3. In java interfaces always represents 100%abstraction.
      -> Inside java interfaces we can write only abstract methods,we can't write
concrete methods


Note: Inside interface the methods we write always indicates "public and abstract".

eg#1.
interface ICalculator
{
      //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);
}

=> interfaces should be implemented by a class.

=> Which ever class is implementing an interface that class should compulsorily
give the body for all the abstract methods
   present in the interface.

=> if the implementation class fails to give the body for atleast one abstract
methods then that class would become abstract class.

eg#2.
interface ICalculator
{
      //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);
}
abstract class CalculatorImpl implements ICalculator
{
}


eg#3.
interface ICalculator
{
      //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);
}
class CalculatorImpl implements ICalculator
{
      public void add(int a,int b){
            System.out.println("The sum is :: "+(a+b));
      }
      public void sub(int a,int b){
            System.out.println("The diff is :: "+(a-b));
      }
      public void mul(int a,int b){
            System.out.println("The mul is :: "+(a*b));
      }
      public void div(int a,int b){
            System.out.println("The div is :: "+(a/b));
      }
}
```

Q> Can we create an object for abstract class?
Ans. No

Q> Can we create a reference for an abstract class?
Ans. yes

Q> Can we create an object for an interface(100% abstraction)?
Ans. No

Q> Can we create a reference for an interface?
Ans. yes

```
eg#4
interface ICalculator
{
      //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);
}
class CalculatorImpl implements ICalculator
{
      public void add(int a,int b){
            System.out.println("The sum is :: "+(a+b));
      }
      public void sub(int a,int b){
            System.out.println("The diff is :: "+(a-b));
      }
```

```
	public void mul(int a,int b){
		System.out.println("The mul is :: "+(a*b));
	}
	public void div(int a,int b){
		System.out.println("The div is :: "+(a/b));
	}
}

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

		//loose coupling : Polymorphism
		ICalculator calculator = new CalculatorImpl();
		calculator.add(10,20);
		calculator.sub(100,20);
		calculator.mul(10,20);
		calculator.div(100,20);
	}
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
The sum is  :: 30
The diff is :: 80
The mul is  :: 200
The div is  :: 5
```

extends vs implements
++++++++++++++++++++++
 a. A class can extends only from one class at a time.

eg#1.

```
class One{
	public void methodOne(){}
}
class Two extends One{
	public void methodTow(){}
}
```

 b. A class can implements any no of interfaces at a time.

eg#1.

```
interface ICalculator
{
	//public abstract
	void add(int a,int b);
	void sub(int a,int b);

}
interface IAdvancedCalculator
{
	//public abstract
	void mul(int a,int b);
	void div(int a,int b);
}

class CalculatorImpl implements ICalculator,IAdvancedCalculator
```

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

      //IAdvancedCalculator
      public void mul(int a,int b){
            System.out.println("The mul is :: "+(a*b));
      }
      public void div(int a,int b){
            System.out.println("The div is :: "+(a/b));
      }
}

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

            //loose coupling : Polymorphism
            ICalculator calculator = new CalculatorImpl();
            calculator.add(10,20);
            calculator.sub(100,20);

            //loose coupling : Polymorphism
            IAdvancedCalculator advCalculator =new CalculatorImpl();
            advCalculator.mul(10,20);
            advCalculator.div(100,20);
      }
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
The sum is :: 30
The diff is :: 80
The mul is :: 200
The div is :: 5
```

3. A class can extends a class and can implement any no of interfaces simultaneously.

eg#1.

```
interface ICalculator
{
      //public abstract
      void add(int a,int b);
      void sub(int a,int b);

}
class CalculatorAdvanced
{
      public void mul(int a,int b){
            System.out.println("The mul is :: "+(a*b));
      }
      public void div(int a,int b){
```

```
		System.out.println("The div is :: "+(a/b));
	}
}

//inheritance
//implementation
class CalculatorImpl  extends CalculatorAdvanced implements ICalculator
{
	//ICalculator
	public void add(int a,int b){
		System.out.println("The sum is :: "+(a+b));
	}
	public void sub(int a,int b){
		System.out.println("The diff is :: "+(a-b));
	}

}

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

		//loose coupling : Polymorphism
		CalculatorImpl calculator = new CalculatorImpl();
		calculator.add(10,20);
		calculator.sub(100,20);
		calculator.mul(10,20);
		calculator.div(100,20);
	}
}
```

```
Output
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
The sum is :: 30
The diff is :: 80
The mul is :: 200
The div is :: 5
```

4. An interface can extend any no of interfaces

eg#1.

```
interface ICalculator
{
	//public abstract
	void add(int a,int b);
	void sub(int a,int b);

}
interface IAdvancedCalculator extends ICalculator
{
	//public abstract
	void mul(int a,int b);
	void div(int a,int b);
}

class CalculatorImpl  implements IAdvancedCalculator
{
	//ICalculator
	public void add(int a,int b){
		System.out.println("The sum is :: "+(a+b));
	}
	public void sub(int a,int b){
		System.out.println("The diff is :: "+(a-b));
	}
	public void mul(int a,int b){
		System.out.println("The mul is :: "+(a*b));
	}
	public void div(int a,int b){
		System.out.println("The div is :: "+(a/b));
	}
}
public class Test {
	public static void main(String[] args) {

		//loose coupling : Polymorphism
		IAdvancedCalculator calculator = new CalculatorImpl();
		calculator.add(10,20);
		calculator.sub(100,20);
		calculator.mul(10,20);
		calculator.div(100,20);
	}
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java
D:\Decode Java1.0Batch>java Test
The sum is :: 30
The diff is :: 80
The mul is :: 200
The div is :: 5
```

Can we write a variable inside an interface?
Ans. yes

eg#1.

```
interface IRemote
{
	//public static final
	int MIN_VOLUME = 0;
	int MAX_VOLUME = 100;
}
```

Can we write a constructor inside an interface?
Ans. No

Can we write static block inside an interface?
Ans. No

Can we write instance block inside an interface?
Ans. No

eg#1.

```
interface IDemo
{
	//public static final
```

```
      int x =10;
}
public class Test implements IDemo{
      public static void main(String[] args) {
            int  x = 100;//local variable
            System.out.println(x);
            System.out.println(IDemo.x);
            System.out.println(Test.x);
      }
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java
D:\Decode Java1.0Batch>java Test
100
10
10
```

Note:
 Inside an interface we can write
      a. variables
      b. methods(public abstract)

 We can also write an interface without variable and methods, this type of interface is called "MarkerInterface".

```
      eg: interface IDemo{

          }
```

 Benift of maker interface is the implementation class would get some extra advantage like
      1. If a class implements Serializable then the object can be sent over the network.
      2. If a class implements Cloneable then object can be cloned(duplicated).

Inbuilt MarkerInterfaces
++++++++++++++++++++++++

```
1. public interface java.lang.Cloneable {

   }

2. public interface java.io.Serializable {

   }
```