Abstract class :

abstract method..

```
public abstract void makeNoise();


abstract class Animal {

}
```

---partial abstraction.

```
Animal a=new Animal(); //CE

Animal a= new Dog();

Animal a= new Cat();

a.makeNoise();

Dog d =new Dog();

@Autowire
Animal  a;
```

Interface:
=======

--it is a full unimplemented structure in java 100 %

--till Java 1.7 interface use to contains only abstract method and final variable.

--from Jdk 1.8 onwards we can place method with body also inside an interface (default method or static method).

*****Note: With the help of an interface we achive loose coupling in Java.**

--inside an interface if we place any method wituout body, that method will be public and abstract wheter we mention it or not.

X.java:

```
package com.masai;

public interface X {

	public abstract void fun1();

	 void fun2();

}

class A{

}
```

```
                         default constructor
A.java ----------->compile ----------------> A.class

X.java -------------> compiler --------------> X.class
```

--constructor concept is not applicable with an interface.

--As a class is extended by another class , an inteface need to be implemented by another class.

rule:

--if a class implements an interface , then that class has to override all the abstract method defined inside that interface otherwise we need to mark that implemented class as an abstract class.

example;

X.java:

```
package com.masai;
```

```
public interface X {

     void fun1();

     void fun2();

}
```

XImpl.java:

```
package com.masai;

public class XImpl implements X{

	@Override
	public void fun1() {

		System.out.println("inside fun1 of XImpl");
	}

	@Override
	public void fun2() {
		System.out.println("inside fun2 of XImpl");
	}


	//specific method
	public void fun3() {
		System.out.println("inside fun3 of XImpl");
	}


}
```

Note: we can not create object of an interface. but we can define a reference variable for an interface.

```
X x1 = new X(); //CE

X x1 = ?   // 2 possible value
```

X x1 = new XImpl(); // any implemented class object

X x1= null;

Note: we can define variable of any 3 valid structure like (concrete class, abstract class or an interface)
but the object should be created only for the concrete class.

X x1= new XImpl(); // here also super class ref and sub class object rule is applicable .

Demo.java:

```
package com.masai;

public class Demo {


	public static void main(String[] args) {

		X x1= new XImpl();

		x1.fun1();
		x1.fun2();

		//interface ref downcasted to implemented class obj.
		XImpl xx= (XImpl)x1;

		xx.fun3();

		XImpl x2 = new XImpl();
		x2.fun1();
		x2.fun2();
		x2.fun3();

	}

}
```

--inside an interface, in addition to an abstract method, we can have variables also.

--if we define any variable inside an interface , it will be by default "public static final" wheter we mention it or not.

--that variable must be initialized at time of declaration.

--variable defined inside an interface can be accessed by the implemented class object also.

example:

X.java:

```
package com.masai;

public interface X {

            int i=100;

            void fun1();

            void fun2();

}
```

XImpl.java:
----

as previous

Demo.java:

```
package com.masai;

public class Demo {


           public static void main(String[] args) {

                      X x1= new XImpl();
```

```
		x1.fun1();
		x1.fun2();



		//interface ref downcasted to implemented class obj.
		XImpl xx= (XImpl)x1;

		xx.fun3();

		System.out.println(xx.i);
		System.out.println(x1.i);
		System.out.println(X.i);

	}
}
```

Interface as a method parameter:

--if a method is defined to take an interface, then we can call that method by supplying any of its implemented class obj or null.

Another.java:

```
package com.masai;

public class Another implements X{

	@Override
	public void fun1() {
		System.out.println("inside fun1 of Another");

	}

	@Override
	public void fun2() {
		System.out.println("inside fun2 of Another");
	}
}
```

Demo.java:

```
package com.masai;

public class Demo {


                public void funDemo(X x1) {

                        if(x1 != null) {
                        System.out.println("inside funDemo of Demo");

                        x1.fun1();
                        x1.fun2();

                        }
                        else
                                System.out.println("null is not allowed..");
                }


        public static void main(String[] args) {

                Demo d1= new Demo();

                //XImpl obj = new XImpl();
                //d1.funDemo(obj);

                //d1.funDemo(new XImpl());
                //d1.funDemo(new Another());

                d1.funDemo(null);

        }

}
```

*****Note: with the help of an interface also we achieve IS-A relationship.**

--implemented class object Is-A type of an interface.

interface as a method return type:
===========================

--if a method having return type as an interface then that method should return any of the implementation class obj of that interface or null value.

Demo.java:

```
package com.masai;

public class Demo {


		public X funDemo() {

			System.out.println("inside funDemo of Demo");

			return new XImpl();

			//return new Another();
			//return null;

		}


	public static void main(String[] args) {

		Demo d1= new Demo();

//		X x1= d1.funDemo();
//
//		if(x1 != null) {
//			x1.fun1();
//			x1.fun2();
//
//		}else
//                          System.out.println("returning null value");


                    Object obj= d1.funDemo();

                    //first level downcasting
                     X x1= (X)obj;

                     x1.fun1();
                     x1.fun2();

                     //second level of downcasting
                     XImpl xx= (XImpl)x1;

                     xx.fun3();

                    //directly downcasting obj to XImpl object

                     XImpl xx2= (XImpl)obj;
                    xx2.fun1();
                    xx2.fun2();
                    xx2.fun3();

          }

}
```

Class is a blueprint for an object, where as Interface is also like blueprint of the class.

Hotel.java:

```
package com.masai;

public interface Hotel {

          public void chickenBiryani();

          public void masalaDosa();


}
```

TajHotel.java:

```
package com.masai;

public class TajHotel implements Hotel{

	@Override
	public void chickenBiryani() {
		System.out.println("ChickenBiryani from TajHotel");

	}

	@Override
	public void masalaDosa() {
		System.out.println("Masala Dosa from TajHotel");
	}

	//specific method of TajHotel class
	public void paneerMasalaDosa() {

		System.out.println("paneer masala dosa from Taj Hotel");
	}

}
```

RoadSideHotel.java:

```
package com.masai;

public class RoadSideHotel implements Hotel{

	@Override
	public void chickenBiryani() {
		System.out.println("ChickenBiryani from RoadSide Hotel");

	}

	@Override
	public void masalaDosa() {
		System.out.println("ChickenBiryani from RoadSide Hotel");
	}

}
```

Demo.java:

```
package com.masai;

public class Demo {


	public Hotel provideFood(int amount) {

		Hotel hotel= null;


		if(amount > 500)
			hotel = new TajHotel();
		else if(amount > 200 && amount <=500)
			hotel = new RoadSideHotel();


		return hotel;
	}


	public static void main(String[] args) {


		Demo d1= new Demo();

		Hotel h= d1.provideFood(800);

		if(h != null) {
			h.chickenBiryani();
			h.masalaDosa();

			if(h instanceof TajHotel) {
```

```
                                        TajHotel taj= (TajHotel)h;
                                        taj.paneerMasalaDosa();
                                }

                        }
                        else
                                System.out.println("Amount should be greater than 200");


                }

        }
```

--interface can not extends another class and can not implement any interface also.
****--but one interface can extends more than one interface simulteniously(multiple inheritance)

```
Intr1.java(I)

funA();
funB();

Intr2.java(I)

funC()
funD()


interface Intr3 extends Intr1,Intr2{

funX();
funY();

}


class Demo implements Intr3{

//it has to override all the methods of Intr1, Intr2 and Intr3 interfaces, otherwise
Demo class has to be marked as abstract class.

}
```

```
Demo d1 = new Demo();

d1.funA();
d1.funB();
d1.funC();
d1.funD();
d1.funX();
d1.funY();


Intr1 i1 = new Demo();

i1.funA();
i1.funB();

Intr2 i2= new Demo();
i2.funC();
i2.funD();


Intr3 i3 = new Demo();
i3.funA();
i3.funB();
i3.funC();
i3.funD();
i3.funX();
i3.funY();
```

--from java 1.8 onwards some new feature introduced in interface.

1.default method

2.static method

--both method should have a body.

1.default method:

-- we can define a default method with the body inside an interface.

--this default method need not override inside the implementation classes.

--if we want , we can override this default method inside any implementation classes.

--these default method are bydefault inherited inside the implementation classes.

--we can call these default method from any implementation class object.

Hotel.java:

```
-------------
package com.masai;

public interface Hotel  {

	public void chickenBiryani();

	public void masalaDosa();

	public default void iceCream() {

		System.out.println("iceCream from Hotel");
	}


}
```

TajHotel.java:

```
-----------------

package com.masai;

public class TajHotel implements Hotel{

	@Override
	public void chickenBiryani() {
		System.out.println("ChickenBiryani from TajHotel");

	}

	@Override
	public void masalaDosa() {
        System.out.println("Masala Dosa from TajHotel");
    }

    //specific method of TajHotel class
    public void paneerMasalaDosa() {

        System.out.println("paneer masala dosa from Taj Hotel");
    }


    @Override
    public void iceCream() {
        System.out.println("Ice cream from TajHotel");
    }

}
```

RoadSideHotel.java:
----

from previous example

Demo.java:

```
package com.masai;

public class Demo {


    public Hotel provideFood(int amount) {

        Hotel hotel= null;


        if(amount > 500)
            hotel = new TajHotel();
        else if(amount > 200 && amount <=500)
            hotel = new RoadSideHotel();


        return hotel;
    }
```

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


		Demo d1= new Demo();

		Hotel h= d1.provideFood(800);

		if(h != null) {
			h.chickenBiryani();
			h.masalaDosa();
			h.iceCream();


			if(h instanceof TajHotel) {

				TajHotel taj= (TajHotel)h;
				taj.paneerMasalaDosa();
			}

		}
		else
			System.out.println("Amount should be greater than 200");


	}

}
```

2.static method:

--we can define a static method also inside an interface from java 1.8

--this static method must have body.

--static method of an inteface will not be inherited inside the implementation class object.
--so we can not call this static method of an interface by using implementation class object.

****Note: we can call the static method of an interface only by using Interface name.**
we can not call static method of an interface even by using interface variable also.

Note:- we can define same static method as static or non-static method inside the implementation class also.
which is already defined statically inside the interface. (this concept is called as method hinding)

Hotel.java:

```
package com.masai;

public interface Hotel  {

	public void chickenBiryani();

	public void masalaDosa();

	public default void iceCream() {

		System.out.println("iceCream from Hotel");
	}

	public static void drinkingWater() {

		System.out.println("drinking water from Hotel");
	}



}
```

Demo.java:

```
package com.masai;

public class Demo {


	public Hotel provideFood(int amount) {
        Hotel hotel= null;


        if(amount > 500)
            hotel = new TajHotel();
        else if(amount > 200 && amount <=500)
            hotel = new RoadSideHotel();


        return hotel;
    }


    public static void main(String[] args) {


        Demo d1= new Demo();

        Hotel h= d1.provideFood(800);

        if(h != null) {
            h.chickenBiryani();
            h.masalaDosa();
            h.iceCream();
            Hotel.drinkingWater();
            //h.drinkingWater();  //CE

            if(h instanceof TajHotel) {

                TajHotel taj= (TajHotel)h;
                taj.paneerMasalaDosa();
            }

        }
        else
            System.out.println("Amount should be greater than 200");


    }

}
```