can you please explain covariant return type, i have problem to understand what its actually return.

```
class Parent{
      //returns Object
      public Object getObject(){
            return new Object();
      }
}
class Child extends Parent{

     //returns String
    @Override
    public String getObject(){
            return new String("sachin");
    }

}
```

Covariant Type
==========

```
Object
    |
    |IS-A
    |
String
```

how to take input from user of HAS-A variables

```
    class University{
         //HAS-A
        private  Department dept;
    }
   class Department{
        private Integer deptId;
        private String depName;

       //setXXXX and getXXXX methods
   }
```

sir concrete means in short ??

concrete -> It refers to completeness

abstract -> It refers to incompleteness

Output of the code
==========

```
package Practice;
class Demo1 extends Demo {

      //It is a specialized method
      public void display() {
            System.out.println("train");
      }
}
class Demo3 extends Demo{

         //It is a specialized method
       public void display() {
             System.out.println("In Demo3");
       }
```

```
 }
public class Demo{ //parent class

	private void display() {//parent method
		System.out.println("trainee");
	}
	public static void main(String[] args) {
		Demo d = new Demo1();
		d.display();//trainee
	}
}
```

Note: Since the method is not overriding,JVM will not bind the method call, rather Compiler will bind the call based on the reference type.4
If it is overriding, then JVM will bind the method call based on the run time object, where compiler would just perform type checking.

If we have two default methods in two interfaces, lets assume both the methods have same name.
Child class implements both the interfaces. we can only override one method in child class.
Then how are we beating ambiquity in this case??Please try this in editor.

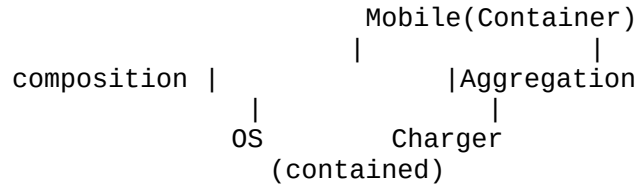
```
interface Left{
	default void m1(){
		System.out.println("Default method from Left");
	}
}
inteface Right{
	default void m1(){
		System.out.println("Default method from Right");
	}
}
class Test implement Left,Right{}//CompileTime Error.

class Test implement Left,Right{
	@Override
	public void m1(){
		//syntax to call interface specific methods
		InterfaceName.super.methodName()

		Left.super.m1();//Default method from Left
		Right.super.m1();//Default method from Right
		System.out.println("Implementation from Test Class");
	}
	p.s.v.m(String[] args){
		Test t1=new Test();
		t1.m1();//Implementation from Test Class
	}
}
```

Dependent object Not ready means what?? means please explain term not ready a

```
  Employee(Target Object)
	|
	|
	|
  Account(Dependent Object)
```

In composition how does target object gets deleted automatically? how dependency

object becomes null? just confused on this.
Composition => Container and Contained Object



Note: Interface static method will not reach to its implementation class,so it is possible to access through child
reference it should be accessed only using interface name.

```
interface X{

   //utility Method
   public static void foo(){
      System.out.println("foo");
   }
}

class Y implements X{
      //static method will not participate in inheritance, so overiding here is not possible
      public static void foo(){
            System.out.println("hello from Y");
      }
}

public class Z
{
   public static void main(String[] args)
   {
      X.foo(); // it is valid
      Y.foo(); //CE

      Y y =new Y();
      y.foo(); //Hello from Y(compiler will bind the method call)

      X x =new Y();//Compiler will bind the call based on reference type
      x.foo();//foo

   }
}
```

Overriding => Compiler duty is to just use the reference and check whether the methods are
available in the respective class or not,and jvm duty is to bind the method call
based on the run time object.

Overloading/Method hiding => Compiler duty is to check the reference and bind the method call based
on the method signature, jvm duty is to just the execute the method which
is binded by the compiler.

```
Compile Time
============
      Compiler will perform TypeChecking
            a. variable type checking
                        byte a = 127;//valid
                        byte b= 300;//CE
            b. reference type checking
                        class Parent{
                                    public void m1(){}
                        }  class Child extends Parent{
                                    public void m2(){}
                        }
                        Parent p=new Child();
                        p.m1();//valid
                        p.m2();//invalid



RunTime
=======
      It creates the Object and perform the desired operation by communicating with
JVM
      class Parent{
            public void m1(){}
      }  class Child extends Parent{
                        @Override
                        public void m1(){}
      }

            Parent p=new Parent();
            p.m1();//parent class m1() executed

            Child c=new Child();
            c.m1();//child class m1() executed


            Parent p=new Child();
            p.m1();//child class m1() executed


abstract class => does it have a constructor? yes
                     Constructor chaining possible ? yes

interface        => does it have a constructor? no
                    Constructor chaining possible? no

Customer/Software Requirement specification
   interface  Remote{
            int minVolume =0;
            int maxVolume =100;

            public String changeChannels();
   }

            1                                                                    M
JAVA  =====SRS(OracleTeam)==================> Database
                                                                        a. MySQL
                                                                        b. Oracle
                                                                        c.
```

PostgreSQL

d. Sybase

infinite for loop syntax
==========
```
for(;;)
   System.out.println("Hello");//infinite hello
```

Association
========
```
                M                                   1
   Employee---------------------------------- Department
```

Ananomyous innner class---------- > interface implemenation

Lambda Expression ----------------> FunctionalInterface implementation

```
interface IFirstSample{
      //few requirements
}

interface ISecondSample extends IFirstSample{
      //few requirements

      //special requirements
}
```

JDBC API
========
```
interface Statement{
      //abstract methods
}
interface PreparedStatement extends Statement{
      //abstract methods
}
interface CallableStatement extends PreparedStatement{
      //abstract methods
}
```