Inhetritance:
----

extends

IS-A relationship

A extends B

method overriding

--constructor : Object class

Object class belongs to java.lang

Super class ref and child class obj.

--to a parent class variable we can assign its any child class object.

A a1 = new A();

A a1 = new B(); // when B is a child of A class.

A a1= null;

LgOldTv.java:

```
package com.masai;

public class LgOldTv {


	public void start() {
		System.out.println("Tv Starting");
	}

	public void increaseVolume() {
		System.out.println("volume increased..");
	}
	public void changeChannel() {

		System.out.println("Channel changed in Old Way...");
	}
}
```

LgSmartTv.java:

```
package com.masai;

public class LgSmartTv extends LgOldTv {

	@Override
	public void changeChannel() {

		System.out.println("Channel Changed in smart way..");
	}


	public void playGame() {

		System.out.println("Game Starts...");
	}

}
```

Demo.java:

```
package com.masai;

public class Demo {


	public static void main(String[] args) {

//		LgOldTv remote = new LgOldTv();
//
//		remote.start();
//		remote.increaseVolume();
//		remote.changeChannel(); // old way
//

//                  LgSmartTv remote=new LgSmartTv();
//
//                  remote.start();
//                  remote.increaseVolume();
//                  remote.changeChannel();//smart way
//                  remote.playGame();

                    //upcasting
                    LgOldTv remote = new LgSmartTv();

                    remote.start();
                    remote.increaseVolume();
                    remote.changeChannel();// smart way

                    //LgSmartTv nRemote=new LgSmartTv();

                    //object downcasting
                    LgSmartTv sRemote= (LgSmartTv)remote;
                    sRemote.playGame();

          }
}
```

exmaple2:

Demo.java:

```
package com.masai;

public class Demo {

          public void fun1(Object obj) {
                    System.out.println("inside fun1 of Demo");
          }

          public static void main(String[] args) {

                    Demo d1 = new Demo();
```

```
		d1.fun1(new A());
		d1.fun1(new B());
		d1.fun1(null);


	}
}
```

example3:

```
package com.masai;

public class Demo {

	public Object fun1(int x) {

		System.out.println("inside fun1 of Demo");

		if(x > 10)
			return new A();
		else
			return new B();
	}

	public static void main(String[] args) {

		Demo d1 = new Demo();

		//A a1= d1.fun1(20); //CE

		Object obj= d1.fun1(20);

		A a1= (A)obj;

		a1.funA();

	}

}
```

example4:

```
package com.masai;

public class Demo {

	public Object fun1(int x) {

		System.out.println("inside fun1 of Demo");

		if(x > 10)
			return new A();
		else
			return new B();
	}

	public static void main(String[] args) {

		Demo d1 = new Demo();


		Object obj= d1.fun1(50);

		if(obj instanceof A) {
			A a1= (A)obj;
			a1.funA();
		}else {
			B b1= (B)obj;
			b1.funB();
		}


	}


}
```

example:

```
		//Object obj= d1.fun1(50);
		//A a1= (A)obj;
```

```
                    A a1= (A)d1.fun1(10);
```

toString() method:
================

--this method belongs to Object class.

Note: Object class methods are also called as universal method, we can call these methods on any class objects.

public String toString();

--if we call this method on any object then it will convert that object address into the String and return that String.

--the above functionality is written inside the toString() method of the Object class.

--toString() method internally called from the println() method.

--println--- PrintStream

```
println(primitives){
//print the content
}

println(){
//extra line
}

println(String s){
print the content of String
}

println(Object obj){

 String s= obj.toString();

println(s);
}
```

**Note: if we print any class object reference then it should print the address of that object
in the form of String , but if it is not printing the address, and it prints somting else (some message, some content) then, the meaning is, that class has overriden the toString() method from the object class.

--toString() is called by the concept of dynamic polymorphism.

--which method will be called decided at runtime. it is also known as late binding.

early binding vs late binding:

----

binding : connecting the method body with method call is known as binding.

--if it is decided at compile time then it is known as early binding(method overloading or static polymorphism).

--if it is decided by the jvm at runtime then it is known as late binding (method overriding or dynamic polymorphism).

Student.java:

```
package com.masai;

public class Student {

	private int roll;
	private String name;
	private int marks;


	public Student() {

	}


	public Student(int roll, String name, int marks) {
		super();
		this.roll = roll;
		this.name = name;
		this.marks = marks;
	}



	public int getRoll() {
		return roll;
	}
	public void setRoll(int roll) {
		this.roll = roll;
	}


	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

	public int getMarks() {
		return marks;
	}

	public void setMarks(int marks) {
		this.marks = marks;
	}


	@Override
	public String toString() {
		return "Student [roll=" + roll + ", name=" + name + ", marks=" + marks + "]";
	}

}
```

final keyword:
===========

--in java every keyword has some specific functionality, but there is a keyword whose functionality differs depending upon the situation where it is used i.e final keyword.

1. if we define a variable as final, then we can not change its value elsewhere in program.

Note: if we define any final variable as a instance variable then it should be initialized at the same place or from the contructor.

ex:

Demo.java:

```
public class Demo {


	final int x=10;

	final int y;


	Demo(int y){
		this.y=y;
	}

	public static void main(String[] args) {

		Demo d1= new Demo(20);

	}

}
```

2. if we define a method as final then we can not override that method inside the child class.

3.if we define a class as final then we can not extends/inherit that class.

String class is a final class.

Access modifiers:
==============

--it specifies the accessebility of a class or its members outside that class or package.

we have 4 types of access modifiers:

1.public : accessebiltiy is global (other package or other classes also)

2.default: it is similar to public as long as inside the same package, but outside the package it works as private

3.protected: it is simillar to default, but outside the package we can access it by using inheritance.

4.private : the accessebility is restricted to that class only.

Note: A normal class/outer-class can only be either default or public but that class members (variables, methods, constructors, inner classes) can be private ,public, default, protected.

Note: we can not apply access modifiers to the local variables.

--default constructor given by java compiler is always public.

--if we make constructor of a class a default then we can not create its object or extends this class outside that package.

--if we make constructor of a class a private then we can not create its object or extends this class outide that class even in the same package also.

example:

A.java:

```
package com.masai;

public  class A {


	public static A getAObject() {
		A a1= new A();

		return a1;
	}

	private A() {

	}


	void funA() {

		System.out.println("inside funA of A");
	}


}
```

Demo.java:

```
package com.masai;

import java.util.Calendar;

public class Demo {


	public static void main(String[] args) {

		//static factory method
		A a1= A.getAObject();

		a1.funA();

		Calendar cal = Calendar.getInstance();

	}
```