*** What is Method Hiding in Java ?
OR
Can we Override static method ?
OR
Can we override main method [main is static methdo before java 25]

In order to work with Method Hiding we have different Cases :

Case 1 :

Any public static method of super class is by default available to sub class so from sub class we can call super class static method as shown in the program below :

```
class SuperClass
{
    public static void accept()
    {
        IO.println("Static method of super class");
    }
}
class SubClass extends SuperClass
{
}

class MethodHidingDemo1
{
    public static void main(String[] args)
    {
        SubClass.accept();
        SubClass sub = new SubClass();
        sub.accept(); //Not Recommended becaue we should call static
                      //Method with the help of class name
    }
}
```

Case 2 :

We can't override a static method with non static method because static method belongs to class and non static method belongs to object, If we try to override static method with non static method then it will generate an error i.e overridden method is static as shown below.

```
class Alpha
{
    public static void m1() //SM
    {
    }
}
class Beta extends Alpha
{
    public void m1() //NSM
    {
    }
}
public class MethodHidingDemo2
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Case 3 :

We can't override any non static method with static method, If we try then it will generate an error, Overriding method is static.

```
class Alpha
{
    public void m1() //NSM
    {
    }
}
class Beta extends Alpha
{
    public static void m1() //SM
    {
    }
}
public class MethodHidingDemo3
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

So, the conclusion is we cannot override static with non static method as well as non-static with static method because static method belongs to class and non-static method belongs to object.

Case 4 :

The following program explains that Method Hiding is only possible with static method of super and sub class.

```
class Super
{
    public static void m1()
    {
        System.out.println("m1  static method of super class");
    }
}
class Sub extends Super
{
    public static int m1()
    {
        System.out.println("m1 static method of Sub class");
        return 0;
    }
}

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

    }
}
```

From the above program it is clear that :
Method Hiding belons to static Method
Method Overriding belons to non-static Method

Case 5 :

We can't override static method because it belong to class but not object, If we write static method in the sub class with same signature and compatable return type then It is Method Hiding but not Method Overriding here compiler will search the method of super class and JVM will also execute the method of super class because method is not overridden.[Single copy and belongs to class area, shareble by all the objects]

Note :- 1) We can't apply @Override annotation on static methods.

2) Static methods can't be overridden so behavior is same for all the Objects hence it is STATIC Polymorphism.

```
package com.ravi.method_hiding;

class Animal
{
    public static int numberOfEyes = 2;

    public int numberOfEars = 2;

    public static void sleep()
    {
        IO.println("Animal is sleeping");
    }
}
class Dog extends Animal
{

    public static void sleep()
    {
        IO.println("Dog is sleeping");
    }
}
class Horse extends Animal
{

    public static void sleep()
    {
        IO.println("Horse is sleeping");
    }
}
public class MethodHiding
{
    public static void main(String[] args)
    {
        Animal animal = new Horse();
        IO.println("Number of eyes :"+animal.numberOfEyes);
        IO.println("Number of ears :"+animal.numberOfEars);
        animal.sleep();
    }

}
```

Mantra :

Using upcasting, whenever we call static field, non-static field and static method always these are executed by using current reference.

Co-Variant return type :

As we know while working method overriding, the method signature must be same as well as return type must be compaitable, If return type is not compaitable we will get compilation error as shown in the program.

```
class Super
{
    public void m1()
    {
    }
}
class Sub extends Super
{
    @Override
    public int m1()
    {
        return 0;
    }
}
public class CoVariant
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Note : error, return type int is not compaitable with void.

But from JDK 1.5V onwards we can change the return type of the method in only one case that the return type of both the METHODS(SUPER AND SUB CLASS METHODS) MUST BE IN INHERITANCE RELATIONSHIP (IS-A relationship so it is compatible) called Co-Variant as shown in the program below.

Note :- Co-variant will not work with primitive data type, it will work only with reference type.

***Co-variant represents only one direction that means sub class method return type object we can assign super class method return object i.e in one direction. (A a = new B(); The Object of B we can assign to A)

```
package com.ravi.co_variant;

class Animal
{}

class Dog extends Animal
{}

class Alpha
{
    public Animal show()
    {
        IO.println("Alpha class show method");
        return null;
    }
}
class Beta extends Alpha
{
    @Override
    public Dog show()
    {
        IO.println("Beta class show method");
        return null;
    }
}

public class CoVariantDemo
{
    public static void main(String[] args)
    {
        Alpha a = new Beta();
        a.show();
    }

}
```

If I am able to assign Dog class (sub class method return type) to super class Animal class then It is called Co-Variant

```
package com.ravi.co_variant;


class RBI
{
    public Object loan()
    {
        System.out.println("Bank should provide loan");
        return this;
    }
}
class SBI extends RBI
{
    public RBI loan()
    {
        System.out.println("Providing loan @ 9.2% ROI");
        return null;
    }
}

public class CoVariantDemo1
{
    public static void main(String[] args)
    {
        RBI r = new SBI();
        r.loan();

    }

}
```

POLYMORHISM