sealed keyword in java ?
----
* It is new feature introduced from JDK 17V.
* In JDK 15V, It was the integral part of java. [Preview Version]

It is an improvement over final keyword.

By using sealed keyword we can declare classes and interfaces as sealed.

It is one kind of restriction that describes which classes and interfaces can extend or implement from Sealed class Or interface.

It is similar to final keyword with less restriction because here we can permit the classes to extend from the original Sealed class.

The class which is inheriting from the sealed class must be final, sealed or non-sealed.

The sealed class must have atleast one sub class.

We can create object for Sealed class.

It provides the following modifiers :

1) sealed : Can be extended only through permitted class.

2) non-sealed : Can be extended by any sub class, if a user wants to give permission to its sub classes.

3) permits : We can provide permission to the sub classes, which are inheriting through Sealed class OR sealed interface

4) final : we can declare permitted sub class as final so, it cannot be extended further.

Example :

```
sealed class Payment permits UPI, CreditCard, DebitCard
{
}

non-sealed class UPI extends Payment
{}

final class CreditCard extends Payment
{}

final class DebitCard extends Payment
{}
```

//Programs :

```
package com.ravi.sealed_demo;

sealed class Bird permits Parrot, Peacock
{
    public void fly()
    {
        System.out.println("Generic Bird is flying");
    }
}
non-sealed class Parrot extends Bird
{
    @Override
    public void fly()
    {
        System.out.println("Parrot Bird is flying");
    }
}
final class Peacock extends Bird
{
    @Override
    public void fly()
    {
        System.out.println("Peacock Bird is flying");
    }
}
public class SealedDemo1
{
    public static void main(String[] args)
    {
        Bird b = null;

        b = new Parrot(); b.fly();
        b = new Peacock(); b.fly();

    }

}
```

```
package com.ravi.sealed_demo;

sealed class OnlineClass permits Laptop, Mobile
{
    public void attendJavaOnline()
    {
        System.out.println("Sunday online class at 9: 30AM");
    }
}
final class Laptop extends OnlineClass
{
    @Override
    public void attendJavaOnline()
    {
        System.out.println("Attending online class through Laptop");
    }
}
final class Mobile extends OnlineClass
{
    @Override
    public void attendJavaOnline()
    {
        System.out.println("Attending online class through Mobile");
    }
}
public class SealedDemo2 {

    public static void main(String[] args)
    {
      OnlineClass onlineClass = null;

      onlineClass = new Laptop(); onlineClass.attendJavaOnline();
      onlineClass = new Mobile(); onlineClass.attendJavaOnline();

    }

}
```

---

2) To decalre a method as a final (Overriding is not possibble):

Whenever we declare a method as a final then we can't override that method in the sub class otherwise there will be a compilation error.

We should declare a method as a final if the body of the method i.e the implementation of the method is very important and we don't want to override or change the super class method body by sub class method body then we should declare the super class method as final method.

//Programs :

```
class A
{
    protected int a = 10;
    protected int b = 20;

   public final  void calculate()
    {
        int sum = a+b;
        System.out.println("Sum is :"+sum);
    }
}
class B extends A
{
    @Override
    public void calculate() //error
    {
        int mul = a*b;
        System.out.println("Mul is :"+mul);
    }
}
public class FinalMethodEx
{
    public static void main(String [] args)
    {
        A a1 = new B();
        a1.calculate();
    }
}
```

```
class Alpha
{
    private final void accept()
    {
        System.out.println("Alpha class accept method");
    }
}
class Beta extends Alpha
{
    public void accept()
    {
        System.out.println("Beta class accept method");
    }
}
public class FinalMethodEx1
{
    public static void main(String [] args)
    {
        new Beta().accept();
    }
}
```

Note : In the above program we don't have CE because private method is not visible to the sub class so in sub class It is method Re-declaration.

---

3) To declare a field/variable as a final (Re-assignment is not possible)

In older langugaes like C and C++ we use "const" keyword to declare a constant variable but in java, const is a reserved word for future use so instead of const we should use "final" keyword.

If we declare a variable/field as a final then we can't perform re-assignment (i.e nothing but re-initialization) of that variable.

//Program

```
class A
{
    final int A = 10;

    public void setData()
    {
        A = 10;
        System.out.println(A);
    }
}
class FinalVarEx
{
    public static void main(String[] args)
    {
        A a1 = new A();
        a1.setData();
    }
}
```

error: cannot assign a value to final variable A

```
class Student
{
    private final int rollNumber;
    private final String studentName;

    public Student(final int rollNumber, final String studentName)
    {
        this. rollNumber = rollNumber;
        this.studentName = studentName;
    }

    @Override
    public String toString()
    {
        return "Roll is :"+this.rollNumber+" Name is : "+this.studentName;
    }
}
class FinalVarEx1
{
    public static void main(String[] args)
    {
        final Student s1 = new Student(111,"Scott");
        System.out.println(s1);

        //s1 = new Student(222,"Alen");  [Invalid]
        System.out.println(s1);
    }
}
```

o/p: Roll is :111 Name is : Scott

Serious Polymorphism :

Abstraction [Hiding the complexity]
--
* Showing the essential details (Required by the user OR User interface) without showing the background details is called abstraction.

* In order to acheive abstraction we can use the following two concepts of java :

1) Abstract class and abstract method [We can achieve **0 to 100%** abstraction]
2) interface [**100% abstration**]

Abstract class and abstract method :
----
a) General OR Concrete OR Instance Method :

```
public void accept()
{

}
```

b) Abstract method :
----
public **abstract** void show();

i) Must have abstract keyword
ii) Should not contain any method body
iii) There must be a terminator at the end

Working with abstract method and abstract class :

```
abstract class Vehicle
{
  public abstract void run();

}
```

```
class Bike  extends Vehicle
{
   public void  run()
   {
   }
}
```

```
class Car  extends Vehicle
{
   public void  run()
   {
   }
}
```

```
class Bus  extends Vehicle
{
    public void  run()
    {
    }
}
```

Points :
----
1) Whenever **action is common but implementations are different** then we should use abstract method.

2) If a class contains **at-least 1 method as an abstract method** then compulsory we should declare that class as an abstract class otherwise CE

3) We cannot create an object for abstract class.

4) All the abstract methods declared in the super class **must be overridden in all the sub classes otherwise sub class will also become as an abstract class hence object will not be created.**