final keyword in java ?

final keyword in java :

It is used to provide some kind of restriction in our program.
We can use final keyword in ways 3 ways in java.

1) To declare a class as a final. (Inheritance is not possible)
2) To declare a method as a final (Overriding is not possible)
3) To declare a variable (Field) as a final (Re-assignment is not possible)

1) To declare a class as a final :

Whenever we declare a class as a final class then we cann't extend or inherit that class otherwise we will get a compilation error.

We should declare a class as a final if the composition of the class (logic of the class) is very important and we don't want to share the feature of the class to some other developer to modify the original behavior of the existing class, In that situation we should declare a class as a final.

Declaring a class as a final does not mean that the variables and methods declared inside the class will also become as a final, only the class behavior is final that means we can modify the variables value as well as we can create the object for the final classes.

Note :- In java String and All wrapper classes are declared as final class.

```
final class A
{
     private int x = 100;

     public void setData()
     {
          x = 120;
          System.out.println(x);
     }
}
class B extends A
{
}
public class FinalClassEx
{
     public static void main(String[] args)
     {
          B b1 = new B();
          b1.setData();
     }
}
```

Note : class A is final so we can't inherit hence we will get compilation error.

```
final class Test
{
     private int data = 100;

     public Test(int data)
     {
          this.data = data;
          System.out.println("Data value is :"+data);
     }
}
public class FinalClassEx1
{
     public static void main(String[] args)
     {
          Test t1 = new Test(200);

     }
}
```

Note : for final class we can create object as well as we can modify the data.

```
class Sample
{
     private Sample() //constructor is private
     {
       System.out.println("Private Constructor");
     }
}
class Sub extends Sample   //error
{
}

public class FinalClassEx2
{
     public static void main(String[] args)
     {

     }
}
```

Sealed class in Java :

It is a new feature introduced from java 15v (preview version) and become the integral part of java from 17V(LTS).

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.

```java
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();

    }

}
```

---

2) To declare a method as a final (Overriding is not possible)

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.

```
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 : private methods are not available to the sub class so the above program will compile & execute

3) To declare a variable/Field as a final : [Cannot perform re-assignment]

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.

In java It is always a better practise to declare a final variable by uppercase letter according to the naming convention.

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

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

     }
}


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);
     }
}
```