**Java final keyword**

final is a keyword or modifier which can be used on variables , methods, classes.



Final Keyword is used to stop user accessing the java variable or class or method.

We can use final keyword in following three manner –

1. Making Final Variable
2. Making Final Method
3. Making Final Class

The **final keyword** behaves very differently when applied to variables, methods and classes. Even though its functionality (purpose) is different in the three contexts, but one thing is common; it does not allow the programmer to do certain actions.

1. A final variable cannot be reassigned.
2. A final method of the super class cannot be overridden by subclass
3. A final class cannot be extended by any other class. Final class does not fit for inheritance.

## Way 1 : Final Variable

"**const**" keyword of C/C++ is not supported by Java and in its place, Java uses "**final**" keyword. That is, a **final variable cannot be reassigned**. Once assigned, cannot be modified later.

If variables are declared using final keyword then the value of that variable cannot be changed at any times which are called as constants. We can apply final keyword for instance variables, static variables, local variables…..

*public class Book{*

*public static void main(String args[]){*

*int cost = 400; //* ***non-final, can be modified later***

*System.out.println("Original Book cost Rs." + cost);*

*cost = 500;*

*System.out.println("Modified Book cost Rs." + cost);*

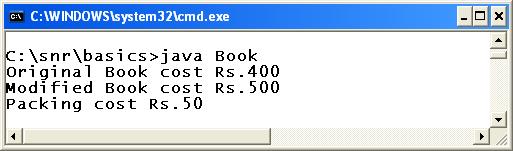
*final int packing = 50;*  ***// final, cannot be modified later***

*System.out.println("Packing cost Rs." + packing);*

*// packing = 100;* ***// raises error***

*}*

*}*

[](http://way2java.com/wp-content/uploads/2010/11/third30.jpg)

In the above code, the variable **cost**is declared as non-final and **packing**as final. The cost variable is reassigned to 500 but packing cannot be (if the comments are removed in the code, it is compilation error). After comfortable with final variables let us go to final methods.

**If we make the variable final then the value of that variable cannot be changed once assigned.**

**Consider the following example –**

*package com.c4learn.inheritance;*

*public class ShapeClass {*

*final* ***int*** *SHAPE\_SIDES = 3;*

***void*** *setShape() {*

*SHAPE\_SIDES = 4;*

*}*

*public* ***static******void******main****(****String****[] args) {*

*ShapeClass c1 = new ShapeClass();*

*c1.setShape();*

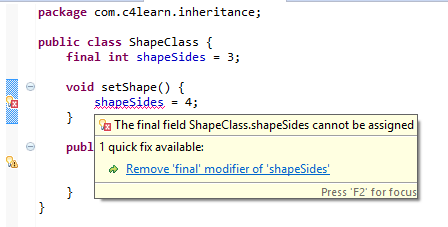
*}*

*}*

### Output :

Compile Time Error

We cannot modify the value of the final variable, we will get following error message –



**Way 2 : Final Method**

If any method is declared then the definition of the method we can change by overriding the method in child classes.

But if we want to declare the method whose definition can not be changed by overriding the method in child classes then the particular method must be declared as final methods.

A final method is a method which cannot be override the method in child classes we can also call this final methods as must not overridable methods.

The super class declares those methods as **final**. That is, **final methods of the super class cannot be overridden**.

**Example**

*class Bird{*

*public final void eat()* ***{ // final method, cannot be overridden***

*System.out.println("Birds eat");*

*}*

*public void fly(){*  ***// non-final method, can be overridden***

*System.out.println("Birds fly");*

*}*

*}*

*public class Peacock extends Bird{*

*public void fly() {*

*System.out.println("Peacock flies small distances");*

*}*

*public static void main(String args[]){*

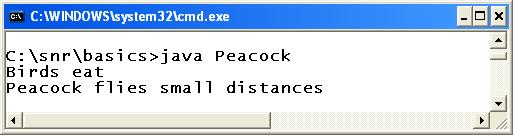
*Peacock p1 = new Peacock();*

*p1.eat(); //* ***calling super class final method***

*p1.fly(); //* ***calling its own overridden method***

*}*

*}*



In Bird's class, **eat()** method is declared as final and **fly()**as non-final. The **eat()**method cannot be overridden by the subclass **Peacock**and regarding **fly()**method, subclass is at liberty to override ([method overriding](http://way2java.com/oops-concepts/method-overriding/)). In the above code, it is overridden. Infact, the subclass can call super class fly() method also with **super.fly()** (not used in the above code).

**Example:**

*class Parent{*

*final void show(){*

*System.out.println("Parent Class show() method");*

*}*

*}*

*class Child extends Parent{*

***/\****

***//final method can not be ovveriden***

***void show(){***

***System.out.println("Child Class show() method");***

***}\*/***

*}*

*class FinalMethodDemo{*

*public static void main(String args[]){*

*Child c = new Child();*

*c.show();*

*}*

*}*

We cannot Override the final method as we cannot change the method once they are declared final. Consider the following example –

*package com.c4learn.inheritance;*

*public class ShapeClass {*

***int*** *SHAPE\_SIDES = 3;*

*final* ***void*** *setShape() {*

*SHAPE\_SIDES = 4;*

*}*

*public* ***static******void******main****(****String****[] args) {*

*ShapeClass c1 = new ShapeClass();*

*c1.setShape();*

*}*

*}*

*class Circle extends ShapeClass {*

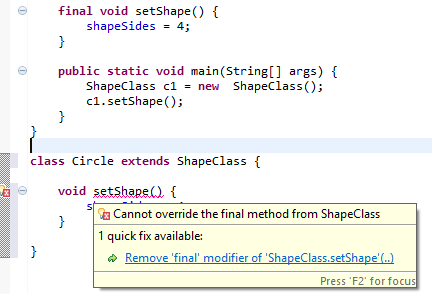
***void*** *setShape() { // Error Here*

*SHAPE\_SIDES = 4;*

*}*

*}*

In this example we have declared method as final so we cannot override the final method, otherwise we will get compile time error –



**Note:**

1. We can not override a final method but we can call and access it

2. Final methods can be overloaded

**Way 3 : Final Class**

If any class is declared any body can extend the particular class but we don't want to allow the others to extend our class then we should declare our class as final.

A final class is a class which can not be extend by any other class by declaring final class we can restrict others to use only Has-A relation because here Is-A relation is not possible

We can consider all the methods of final class as final methods because which can not be override, But all the variables of final class are not final.

In inheritance, a class can be extended by another class; but it is at the mercy of the super class. If the super class does not like any one to inherit, simply it declares its class as "**final**". A final class cannot be extended by any other class. **A final class does not fit for inheritance**.

**Example:**

*final class Spacecraft{ // class is final*

*public void fly(){*

*System.out.println("Spacecraft flies");*

*}*

*}*

*public class Aeroplane{*

*public void fuel() {*

*System.out.println("Aeroplane uses Hi-Octane petrol");*

*}*

*public static void main(String args[]){*

*Aeroplane a1 = new Aeroplane();*

*a1.fuel();*

*Spacecraft s1 = new Spacecraft();*

*s1.fly();*  ***// calling final class method with composition***

*}*

*}*



**Spacecraft**class is declared as final and for this reason, **Aeroplane**cannot extend it. But Aeroplane class can make use of final class methods by "[**composition**](http://way2java.com/oops-concepts/composition-has-a-relationship/)".

**Example:**

*final class Parent {*

*int a=10;*

*int b=20;*

*void show(){*

*System.out.println("a="+a);*

*System.out.println("b="+b);*

*}*

*}*

***/\****

***//we can not hinherit from final class Parent***

***class Child extends Parent{***

***}***

***\*/***

*class FinalClassDemo{*

*public static void main(String args[]){*

*Parent p = new Parent();*

*p.show();*

*p.a=100;*

*p.b=200;*

*p.show();*

*}*

*}*

We cannot inherit the final class.

*package com.c4learn.inheritance;*

*public final class ShapeClass {*

***int*** *SHAPE\_SIDES = 3;*

*final* ***void*** *setShape() {*

*SHAPE\_SIDES = 4;*

*}*

*public* ***static******void******main****(****String****[] args) {*

*ShapeClass c1 = new ShapeClass();*

*c1.setShape();*

*}*

*}*

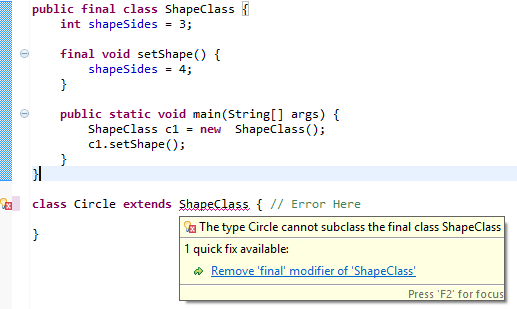
*class Circle extends ShapeClass { // Error Here*

*}*

### Output :

*Compile Time Error*

In this example, we have declared class as final so we cannot inherit the final class



**Note:**

1. A final class can not be inherited by any other class but a final class can extend any other class and also override the methods if needed.

2. We can create an object for a final class and we can access the members of final class.

**Some Important Points : Final Keyword**

1. Final Keyword can be applied only on Method, Class and Variable

| **Final Entity** | **Description** |
| --- | --- |
| **Final Value** | Final Value cannot be modified |
| **Final Method** | Final Method cannot be overriden |
| **Final Class** | Final Class cannot be inherited |

1. Final Variable must be initialized at the time of declaration or inside constructor
2. Re-Assigning Value is not allowed for final member variable
3. Local Final Variable should be initialized during the time of declaration
4. Final and Finally both are different. [Finally keyword](http://www.c4learn.com/java/java-finally-block/) is used on Exception handling in Java

**1. Final Vs Non Final Method, Which is more efficient ?**

### A: Final

1. Final methods execute more efficiently than non final method.
2. Compiler knows at compile time that a call to a final method won’t be overridden by some other method

**Let’s See, How ?**

Consider the final method,

*package com.c4learn.inheritance;*

*public class ShapeClass {*

*final* ***void*** *setShape() {*

*System.out.println("I am Inherited");;*

*}*

*public* ***static******void*** *main(String[] args) {*

*Circle c1 = new Circle();*

*c1.setShape();*

*}*

*}*

*class Circle extends ShapeClass {*

*}*

Whenever we create an object of the child class then compiler will check whether method written inside the parent class is overriden or not at run time.

*final* ***void*** *setShape() {*

*System.out.println("I am Inherited");;*

*}*

In this case compiler already knows that, method is declared as final so it does not waste time in checking whether method is final or not.

**2. Can we inherit Final Method ?**

### A: Yes

In case of final method, Method is inherited but cannot be overriden so always method from parent class will be executed.

**Consider the following example –**

*package com.c4learn.inheritance;*

*public class ShapeClass {*

*final* ***void*** *setShape() {*

*System.out.println("I am Inherited");;*

*}*

*public* ***static******void******main****(****String****[] args) {*

*Circle c1 = new Circle();*

*c1.setShape();*

*}*

*}*

*class Circle extends ShapeClass {*

*}*

**Output :**

I am Inherited

We can inherit the final method but cannot override the final method inside the child class.

**3. Can Java have blank final Value ?**

### A: No

*package com.c4learn.inheritance;*

*public class BlankFinalValue {*

*final* ***int*** *myValue;*

*public BlankFinalValue(){*

***this****.myValue = 3;*

*}*

*public* ***static******void*** *main(String[] args) {*

*BlankFinalValue s1 = new BlankFinalValue();*

*}*

*}*

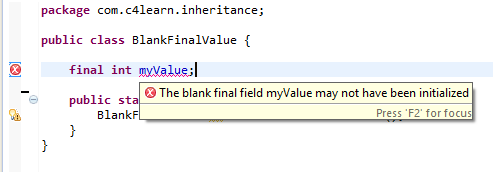
### Output :

Compiler Successfully

**Explanation :**

1. We cannot have final Value which is uninitialized.
2. You will get compile time error if you did not provide initialization.
3. In this case we have to provide initialization in the constructor. (As shown in above program)

If final variable is not initialized in the constructor then it will throw **compile time error as below** –



**If You have Multiple Constructor then –**

We need to initialize final variable in each of the constructor like below program –

*package com.c4learn.inheritance;*

*public class BlankFinalValue {*

*final* ***int*** *myValue;*

*public BlankFinalValue(){*

***this****.myValue = 3;*

*}*

*public BlankFinalValue(****int*** *num){*

***this****.myValue = num;*

*}*

*public* ***static******void*** *main(String[] args) {*

*BlankFinalValue s1 = new BlankFinalValue();*

*}*

*}*

If you failed to initialize final variable in any of the constructor then you will get compile time error –

*The blank final field myValue may not have been initialized*

**Member variable**:

If a member variable of a class is marked as final, then it has to be initialized while it is declared or initialized in the constructor. Otherwise it causes a compilation error. Here the initializations in class A and B are correct, where as for class C, it is a problem since the variable j is neither initialized when declared nor initialized in the constructor.

*class A  
{  
    final int i = 10;  
}  
class B{  
    final int k;  
    B(int k){  
        this.k = k;  
    }  
}  
class C{  
    final int j; // Causes a compilation error  
    C()  
    {  
    }  
}*

**Class or static variable**:

If a static variable of a class is marked as final, then it has to be initialized while it is declared or initialized in the static block. Otherwise it causes a compilation error. Here the initializations in class M and N are correct, where as for class O, it is a problem since the variable j is neither initialized when declared nor initialized in the static block.

*class M  
{  
    static final int i = 10;  
}  
class N{  
    static final int k;  
    static{  
        k = 4 \* 20;  
    }  
}  
class O{  
    static final int j; // Causes a compilation error  
}*

**Method**:

If a method of a class is marked as final, then it cannot be overridden in any of its sub-classes. It causes a compilation error, if we try to override a final method.

*class X{  
    final void print(){  
        System.out.println("This method can not be overridden.");  
    }  
}  
class Y extends X{  
    void print() // Causes a compilation error, can not override a final method  
    {  
    }      
}*

**Class:**

If a class itself is marked as final, then it can not be inherited (or extended). It causes a compilation error, if we try to extend a final class. Here class P is marked as final, hence it cannot be inherited.

**When you use final modifier for class, it means**

* Class is fully implemented and there should not be any sub-class of this class.
* Modifier final is totally opposite of abstract modifier. Hence for a class, final and abstract modifier cannot be used together.

*final class P  
{  
}  
class Q extends P{  
    // Causes a compilation error, can not inherit a final class  
}*

**Singleton design pattern**

A design pattern is something which a common solution for a common problem. In programming, although there are different domains or fields like banking, communications, travel etc., there are usually problems or situations which are common across all these domains. Some of these most common problems are identified as patterns and one such simple pattern is Singletondesign pattern.

Singleton design pattern, as the name suggests is a problem in which we want only one object to be created. And the same object should be used across the application. This can be achieved by making the only constructor of the class as private and using the static method to create that only object and return it no matter how many times it is called.

*class Singleton{  
    private static Singleton onlyObject;  
    public static Singleton getInstance(){  
        if(onlyObject == null)        {  
            onlyObject = new Singleton();  
        }  
        return onlyObject;  
    }  
    private Singleton()  
    {  
    }  
    public void doSomeAction()  
    {  
    }  
}  
class A{  
    void method1(){  
        Singleton s = Singleton.getInstance();  
        s.doSomeAction();  
    }  
}  
class B{  
    void method2(){  
        Singleton.getInstance().doSomeAction();  
    }  
}*

Here we have called the Singleton.getInstance() method in both class A and B. We do not know which method - method1()or method2() is called first. Irrespective of which method gets called, only one object is created and it is shared between these two methods. The same object will be used, no matter how many times method1 or method2 gets executed. 

Please note that we have only discussed the simplest from of Singleton and there are other complexities which are not included here.

# [10 Points Every Java Programmer Should Know About final keyword in java](http://javaconceptoftheday.com/final-keyword-in-java/)

A **final keyword in java** can be used with a class, with a variable and with a method. final keyword restricts the further modification. When you use final keyword with an entity (class or variable or method), it gets the meaning that entity is complete and can not be modified further.

In this post, we will discuss some about 10 important points about final keyword which every java programmer should know. Let’s start with some simple basic things about final keyword in java.

**final class in java :**

We can’t create a subclass to the class or we can’t extend a class or we can’t modify a class which is declared as **final**.

|  |
| --- |
| *final class FinalClass{*  *//some statements*  *}*  *class SubClass extends FinalClass{*  *//compile time error*  *//Can't create sub class to the final class*  *}* |

**final method in java :**

We can’t override a method or we can’t modify a method in the sub class which is declared as **final** in the super class.

|  |
| --- |
| *class SuperClass{*  *final void methodOne(){*  *//some statements*  *}*  *}*  *class SubClass extends SuperClass{*  *@Override*  *void methodOne(){*  ***//Compile time error***  ***//can not override fin****al method*  *}*  *}* |

**final variable in java :**

The value of a final variable cannot be changed in the whole execution once it got initialized.

|  |
| --- |
| *class AnyClass{*  *final int i = 10;*  *void methodOne()   {*  *i = 20;* ***//compile time error***  ***//final field cannot be re-assigned***  *}*  *}* |

**10 Points Every Java Programmer Should Know About final Keyword In Java :**

**1)** Any class or any method can be either **abstract or final** but not both. abstract and final are totally opposite. Because, abstract class or abstract method must be implemented or modified in the sub classes but final does not allow this. This creates an ambiguity.

|  |
| --- |
| ***//The following class gives compile time error***  *final abstract class AnyClass*  *{*  ***//Any class cannot be final and abstract***    *final abstract void methodOne();*  ***//method cannot be final and abstract at a time***  *}* |

**2)** final method can be overloaded and that overloaded method can be overridden in the sub class.

|  |
| --- |
| *class SuperClass{*  *final void methodOne(){*  *//final method*  *}*  *void methodOne(int i){*  *//final method can be overloaded*  *}*  *}*  *class SubClass extends SuperClass{*  *@Override*  *void methodOne(int i){*  *//Overloaded method can be overridden*  *}*  *}* |

|  |
| --- |
| *class AnyClassOne{*  *final int i = 10;*  *void methodOne()   {*  *i++;*  ***//above statement gives Compile time error.***  ***//value of final variable can not be changed***  *int j = i;****//final variable can be used to initialize other variables.***  *System.out.println(i);  //final variable can be used*  *}*  *}* |

**3)** final variable can not be re-initialized but final variable can be used to initialize other variables.

**4)** When an array reference variable is declared as final, only variable itself is final but not the array elements.

|  |
| --- |
| *public class UseOfFinalKeyword{*  *public static void main(String[] args){*  *final int X[] = new int[10];     //****final array variable***  *X[2] = 10;*  *X[2] = 20;****//Array element can be re-assigned***    *X = new int[30];* ***//compile time error***  ***//can't re-assign new array object to final array variable***  *}*  *}* |

**5)** When a reference variable is declared as final, you can’t re-assign a new object to it once it is referring to an object. But, you can change the state of an object to which final reference variable is referring.

|  |
| --- |
| *class A{*  *int i = 10;*  *}*  *public class UseOfFinalKeyword{*  *public static void main(String[] args){*  *final A a = new A();* ***//final reference variable***  *a.i = 50;*  ***//you can change the state of an object to which final reference variable is pointing***  *a = new A();* ***//compile time error***  ***//you can't re-assign a new object to final reference variable***  *}*  *}* |

***6)****Static variables, non-static variables and local variables all can be final. once the final variables are initialized, even you can’t re-assign the same value.*

|  |
| --- |
| *class A{*  *static final int i = 10;   //final static variable*  *final int j = 20;          //final non-static variable*  *void methodOne(final int k)  {*  *//k is final local variable*  *k = 20;   //compile time error*  *}*  *}*  *public class UseOfFinalKeyword{*  *public static void main(String[] args){*  *A a = new ();*  *a.i = 10;     //Compile time error*  *a.j = 20;     //even you can't assign same value to final variables*  *a.methodOne(20);*  *}*  *}* |

**7)** If the global variables are not initialized explicitly, they get default value at the time of object creation. But final global variables don’t get default value and they must be explicitly initialized at the time of object creation. Uninitialized final field is called **Blank Final Field**.

|  |
| --- |
| *class A{*  *int i;   //Non-final global variable, no need to initialize them*  *final int j;         //Blank Final Field*  *A()   {*  *j=20;*  ***//final global variable must get a value at the time of object creation.***  *}*  *}*  *public class UseOfFinalKeyword{*  *public static void main(String[] args){*  *A a = new A();*  *}*  *}* |

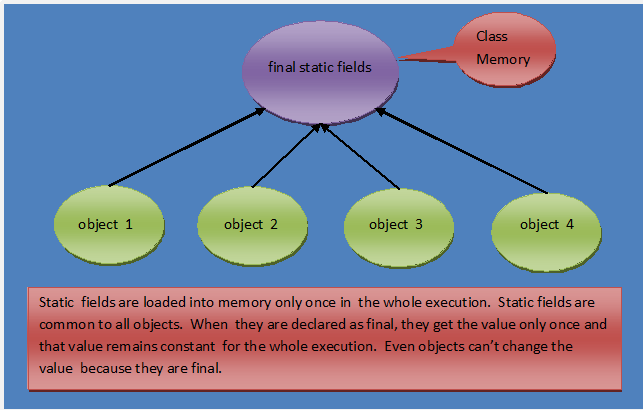
**8)** final non-static global variable must be initialized at the time of declaration or in all constructors or in any one of IIBs – Instance Initialization Blocks.

|  |
| --- |
| *class A*  *{*  *final int i;  //Final non-static global variable may be initialized here  OR*    *//may be initialized in any one of IIB's,*  *// because while object creation, all IIBs are called.  OR*    *{*  *i = 30;*  *}*    *{*  *//i = 40;*  *}*    *//must be initialized in all constructors.*  *//because while object creation, only one constructor is called*    *A()*  *{*  *//i=20;*  *}*    *A(int j)*  *{*  *// i=j;*  *}*    *A(int j, int k)*  *{*  *// i = 50;*  *}*  *}* |

**9)** final static global variable must be initialized at the time of declaration or in any one of SIBs – Static Initialization Blocks. (final static global variable can’t be initialized in constructors)

|  |
| --- |
| *class A{*  *static final int i;* ***//final static global variable may be initialized here OR***  ***//may be initialized in any one of SIBs.***  *static{*  *i = 30;*  *}*  *static{*  *//****i = 40;***  *}*  */****/final static global variable can not be initialized in constructors***  *A(){*  */****/i=20;***  *}*  *A(int j){*  ***//i=j****;*  *}*  *A(int j, int k){*  ***//i = 50;***  *}*  *}* |

**10)** The global variable which is declared as final and static remains unchanged for the whole execution. Because, Static members are stored in the class memory and they are loaded only once in the whole execution. They are common to all objects of the class. If you declare static variables as final, any of the objects can’t change their value as it is final. Therefore, variables declared as final and static are sometimes referred to as **Constants**. All fields of interfaces are referred as constants, because they are final and static by default.



**Java Interview Questions And Answers On final Keyword :**

**1) What is the use of final keyword in java?**

final keyword in java is used to make any class or a method or a field as unchangeable. You can’t extend a final class, you can’t override a final method and you can’t change the value of a final field. final keyword is used to achieve high level of security while coding.

**2) What is the blank final field?**

Uninitialized final field is called blank final field.

**3) Can we change the state of an object to which a final reference variable is pointing?**

Yes, we can change the state of an object to which a final reference variable is pointing, but we can’t re-assign a new object to this final reference variable.

**4) What is the main difference between abstract methods and final methods?**

Abstract methods must be overridden in the sub classes and final methods are not at all eligible for overriding.

**5) What is the use of final class?**

A final class is very useful when you want a high level of security in your application. If you don’t want inheritance of a particular class, due to security reasons, then you can declare that class as a final.

**6) Can we change the value of an interface field? If not, why?**

No, we can’t change the value of an interface field. Because interface fields, by default, are final and static. They remain constant for whole execution of a program.

**7) Where all we can initialize a final non-static global variable if it is not initialized at the time of declaration?**

In all constructors or in any one of instance initialization blocks.

**8) What are final class, final method and final variable?**

final class —> can not be extended.

final method —> can not be overridden in the sub class.

final variable —> can not change it’s value once it is initialized.

(Click [here](http://javaconceptoftheday.com/final-keyword-in-java/) for more info on final class, final method and final variable)

**9) Where all we can initialize a final static global variable if it is not initialized at the time of declaration?**

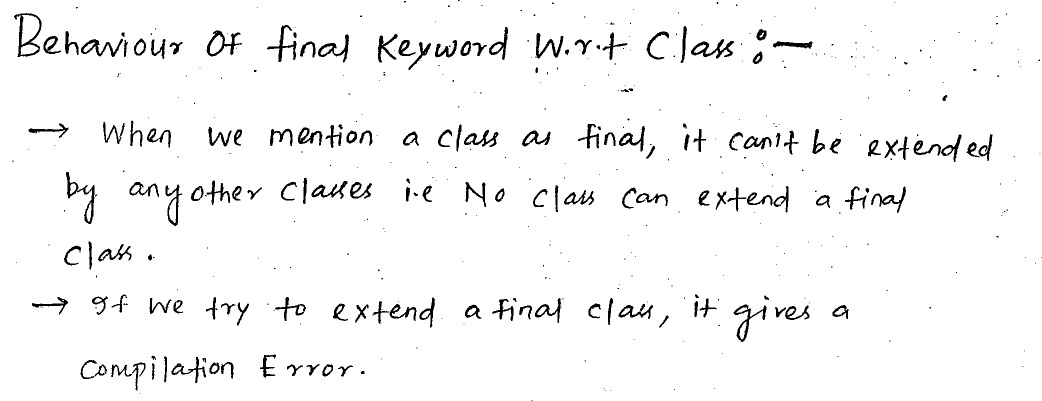
In any one of static initialization blocks.

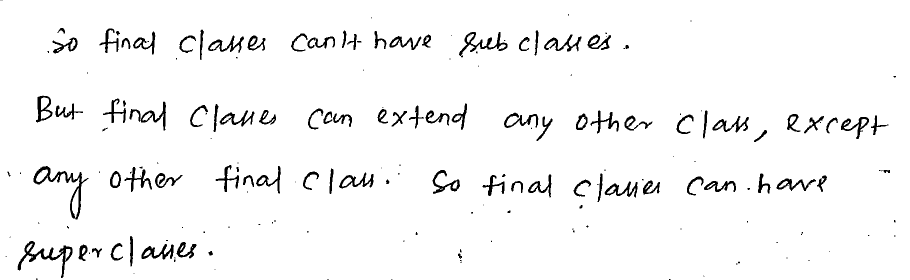
**10) Can we use non-final local variables inside a local inner class?**

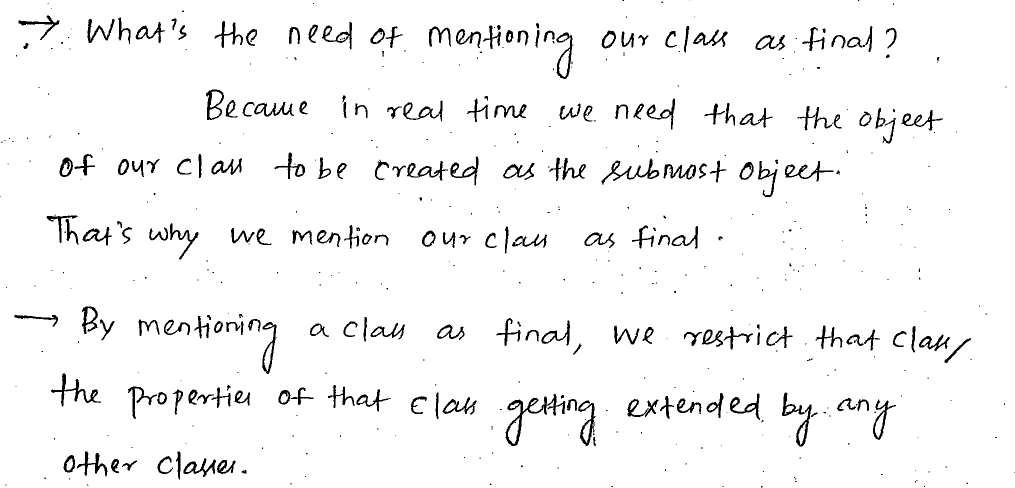
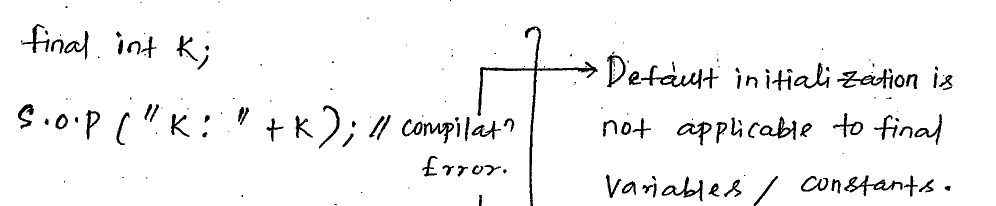
No. Only final local variables can be used inside a local inner class.

**11) Can we declare constructors as final?**

No, constructors cannot be final.





**Example:**

