**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.

**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 final method**

**}**

**}**

**Final Variable in Java:**

The value of a final variable can not 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 can not be re-assigned**

**}**

**}**

Summary:

|  |  |
| --- | --- |
| **Keyword** | **Key Points** |
| **Final** | 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. |
| 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. |
| The value of a final variable can not be changed in the whole execution once it got initialized. |
| 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. |
| final method can be overloaded and that overloaded method can be overridden in the sub class. |
| final variable can not be re-initialized but final variable can be used to initialize other variables. |
| When an array reference variable is declared as final, only variable itself is final but not the array elements. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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) |
| 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. |

**10 Key Points about Final Keyword**

**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 can not be final and abstract**

**final abstract void methodOne();**

**//method can not 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**

**}}**

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

**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**

**}**

**}**

**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.

