final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many contexts.

Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable (It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Output: Compile Time Error

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike{
  final void run(){System.out.println("running");}
```

```
}

class Honda extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda honda= new Honda();
  honda.run();
  }
}
```

Output: Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Output: Compile Time Error

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
class Honda2 extends Bike{
  public static void main(String args[]){
```

```
        new Honda2().run();
      }
    }
```

Output: running...

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

Q) Can we initialize blank final variable?

Yes, but only in constructor. For example:

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Benefits of final keyword in Java

Here are few benefits or advantage of using final keyword in Java:

1. Final keyword improves performance. Not just JVM can cache **final variable** but also application can cache frequently use final variables.

2. Final variables are safe to share in multi-threading environment without additional synchronization overhead.

3. **Final keyword** allows JVM to optimized method, variable or class.

Java Garbage Collection

In java, garbage means unreferenced objects. Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects. To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

1) By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By annonymous object:

```
new Employee();
```

finalize() method

finalize method in java is a special method much like main method in java. finalize() is called before Garbage collector reclaim the Object, its last chance for any object to perform cleanup activity i.e. releasing any system resources held, closing connection if open etc. The intent is for finalize() to release system resources such as open files or open sockets before getting collected.

Main issue with finalize method in java is its not guaranteed by JLS that it will be called by Garbage collector or exactly when it will be called, for example an object may wait indefinitely after becoming eligible for garbage collection and before its finalize() method gets called. similarly even after finalize gets called its not guaranteed it will be immediately collected. finalize method is called by garbage collection thread before collecting object and if not intended to be called like normal method. finalize method is called by garbage collection thread before collecting object and if not intended to be called like normal method.

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

public static void gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Simple Example of garbage collection in java

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

Output:

```
object is garbage collected
object is garbage collected
```

Java Nested Classes

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

The Java programming language allows you to define a class within another class. Such a class is called a *nested class* and is illustrated here:

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

Terminology: Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are called *static nested classes*. Non-static nested classes are called *inner classes*.

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private. Static nested classes do not have access to other members of the enclosing class. As a member of the OuterClass, a nested class can be declared private, public, protected, or *package private*. (Recall that outer classes can only be declared public or *package private*.)

Why Use Nested Classes?

Compelling reasons for using nested classes include the following:

- **It is a way of logically grouping classes that are only used in one place**: If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.
- **It increases encapsulation**: Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

- **It can lead to more readable and maintainable code**: Nesting small classes within top-level classes places the code closer to where it is used.

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are,

Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

Code Optimization: It requires less code to write.

EXTRA

Nested Class
A class within another class is known as Nested class. The scope of the nested is bounded by the scope of its enclosing class.

Static Nested Class
A satic nested class is the one that has **static** modifier applied. Because it is static it cannot refer to non-static members of its enclosing class directly. Because of this restriction static nested class is seldom used.

Non-static Nested class
Non-static Nested class is most important type of nested class. It is also known as **Inner** class. It has access to all variables and methods of **Outer** class and may refer to them directly. But the reverse is not true, that is, **Outer** class cannot directly access members of **Inner** class.

One more important thing to notice about an **Inner** class is that it can be created only within the scope of **Outer** class. Java compiler generates an error if any code outside **Outer** class attempts to instantiate **Inner** class.

Example of Inner class

```
class Outer
{
 public void display()
 {
  Inner in=new Inner();
  in.show();
 }

 class Inner
 {
  public void show()
  {
   System.out.println("Inside inner");
  }
 }
}

class Test
{
 public static void main(String[] args)
 {
  Outer ot=new Outer();
  ot.display();
 }
}
```

Example of Inner class inside a method

```
class Outer
{
 int count;
 public void display()
 {
  for(int i=0;i<5;i++)
  {
   class Inner       //Inner class defined inside for loop
   {
    public void show()
    {
     System.out.println("Inside inner "+(count++));
    }
   }
   Inner in=new Inner();
   in.show();
  }
 }
}

class Test
{
 public static void main(String[] args)
 {
  Outer ot=new Outer();
  ot.display();
 }
}
```

Example of Inner class instantiated outside Outer class

```
class Outer
{
 int count;
 public void display()
 {
  Inner in=new Inner();
  in.show();
 }

 class Inner
 {
  public void show()
  {
   System.out.println("Inside inner "+(++count));
  }
 }
}

class Test
{
 public static void main(String[] args)
 {
  Outer ot=new Outer();
  Outer.Inner in= ot.new Inner();
  in.show();
 }
}
```