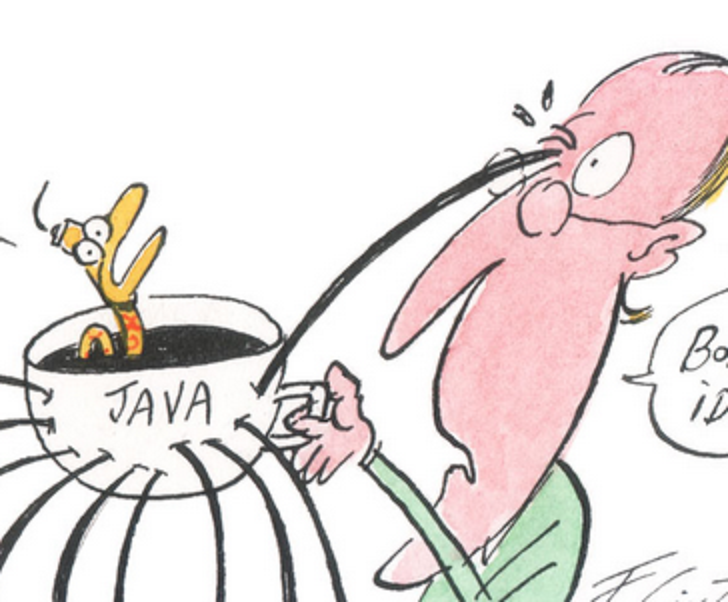
[**What is Effectively Final variable of Java 8**](https://javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html)

Apart from the big three, [Lambda expression](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html), [Stream API](http://java67.blogspot.sg/2014/04/java-8-stream-api-examples-filter-map.html), and [new Date and Time API](http://javarevisited.blogspot.sg/2015/03/20-examples-of-date-and-time-api-from-Java8.html), Java 8 has also introduced a new concept called "effectively final" variable. A non-final local variable or method parameter whose value is never changed after initialization is known as effectively final. It's very useful in the context of the lambda expression. If you remember, prior to Java 8, we cannot use a non-final local variable in an anonymous class. If you have to access a local variable ( a method parameter or a variable decreed in the method itself) in the Anonymous class, you have to make it final. This restriction is relaxed a bit with lambda coming up. Java designers have taken the need to make local variable final if it's not changed once initialized.

**Why we cannot use a non-final local variable in an anonymous class?**

The difference is between local (method) variables vs class member variables. A member variable exists during the lifetime of the enclosing object, so it can be referenced by the inner class instance. A local variable, however, exists only during the method invocation, and is handled differently by the compiler, in that an implicit copy of it is generated as the member of the inner class. Without declaring the local variable final, one could change it, leading to subtle errors due to the inner class still referring to the original value of that variable.  
  
This is really a good proactive step from them, they must have anticipated frequent use of lambda expressions as compared to the minimal use of Anonymous class and realized the pain to declare a local variable final every time in order to access it inside the lambda.  
  
The rule has not changed, any local variable still has to be effectively final to be used inside lambda expression or anonymous inner class, it's just that you don't need to use the [final keyword](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) anymore, saving a few keystrokes.  
  
If you don't follow this then you will get the following error: *"local variables referenced from a lambda expression must be final or effectively final"*. Let's see some examples of effectively final variables with lambdas and anonymous class and how it makes the life of Java developer easy.

**How to use Effective final variable in Java 8**

[](https://2.bp.blogspot.com/-7-_uZoptA4o/VRlmEeeGZRI/AAAAAAAACrE/V5nv_pMgjS8/s1600/Java+8+lambdas+effectively+final+variable.png)

The effective final variable is nothing but a gift from Java designer to us, Java developers. There is hardly any difference between a [final local variable](http://java67.blogspot.sg/2012/09/what-is-blank-final-variable-in-java.html) and an effectively final variable, once declared you cannot change the values of both of them.  
  
If you do, the compiler will raise an error. Here is an example of using an effectively final variable in Java 8, if you run the same program with Java source 1.7, your program will not compile and give error "local variable nonFinal is accessed from within inner class; needs to be declared final" because nonFinal is a local variable but not final.  
  
BTW, if you run the same program in Java 8 it will run fine and produce the output in the form of the print statement we have given.

Now in Java 8, if you try to reassign a new value to our non-final local variable, the compiler will again complain about the error "local variables referenced from an inner class must be final or effectively final". This line is commented in our example, you can try by yourself by uncommenting it.

package test;

/\*\*

\* Java Program to demonstrate effectively final variable in Java 8.

\*

\* @author Javin Paul

\*/

public class EffectiveFinalDemo{

public static void main(String args[]) {

String nonFinal = "I am non final local variable";

Runnable r = new Runnable() {

@Override

public void run() {

System.out.println("Using non-final local variable

inside anonymous class in Java");

System.out.println("Value of non-final variable is : "

+ nonFinal);

// compile time error - must be final or effective final

// nonFinal = "Can I change non-final variable

// inside anonymous class";

}

};

Thread t = new Thread(r);

t.start();

}

}

If you run this program it will print following line in Java 8:

Using non-final local variable inside anonymous class in Java

Value of non-final variable is : I am non final local variable

Now let's see how to use effectively final variable in lambda expression in Java 8:

public class EffectiveFinalWithLambda{

public static void main(String args[]) {

String effectiveFinal = "I am non final local variable";

Runnable r = () -> {

System.out.println("Value of effectively variable is : "

+ effectiveFinal);

};

Thread t = new Thread(r);

t.start();

}

}

You can see that we can access "effectiveFinal" variable without any issue in the lambda expression. Now just try to assign this variable a new value before lambda, just to make it non-final and see what happens, you will receive an error "local variables referenced from a lambda expression must be final or effectively final".  
  
  
That's all about **what is the effectively final variable of Java 8** and how to use it in your Java program. Now it becomes one more thing to note about final variables in Java. Remember till Java 7, you cannot use a non-final local variable inside an [anonymous class](http://java67.blogspot.sg/2012/10/nested-class-java-static-vs-non-static-inner.html), but from Java 8 you can.  
  
The concept of Effective final variables in only available from Java 8 onward. A variable is considered an effective final if it is not modified after initialization in the local block.  
  
This means you can now use the local variable without the final keyword inside an anonymous class or lambda expression, provided they must be effectively final. In short, you can save some keystroke while declaring local final variables indented to be used by the anonymous class.  
  
There is no real difference between a final local variable and effectively final variable of Java 8, once initialized you cannot change the value of both of them.

Read more: <https://javarevisited.blogspot.com/2015/03/what-is-effectively-final-variable-of.html#ixzz6ui8UnAm3>