Should I avoid using Java Label Statements?

Asked 16 years, 3 months ago Modified 3 years, 3 months ago

Viewed 35k times

69

Today I had a coworker suggest I refactor my code to use a label statement to control flow through 2 nested for loops I had created. I've never used them before because personally I think they decrease the readability of a program. I am willing to change my mind about using them if the argument is solid enough however. What are people's opinions on label statements?

java loops

Share

Improve this question

Follow

edited Jan 30, 2014 at 7:19

Developer Marius Žilėnas

6,099 ● 2 ● 45 ● 75

asked Sep 5, 2008 at 18:32

Aaron

23.8k ● 10 ● 50 ● 48

8 I like how open-minded you are :) *"I am willing to change my mind about using them"* – luigi7up Feb 6, 2013 at 9:48

(As you can put loops into methods, this is entirely equivalent to single/multiple/early exit from methods.) – greybeard Aug 14, 2016 at 7:50

12 Answers

Sorted by: Highest score (default)

49

Many algorithms are expressed more easily if you can jump across two loops (or a loop containing a switch statement). Don't feel bad about it. On the other hand, it may indicate an overly complex solution. So stand back and look at the problem.

Some people prefer a "single entry, single exit" approach to all loops. That is to say avoiding break (and continue) and early return for loops altogether. This may result in some duplicate code.

What I would strongly avoid doing is introducing auxilary variables. Hiding control-flow within state adds to confusion.

Splitting labeled loops into two methods may well be difficult. Exceptions are probably too heavyweight. Try a single entry, single exit approach.

Share Improve this answer Follow

answered Sep 5, 2008 at 19:03

Tom Hawtin - tackline
147k ● 30 ● 221 ● 312

11 Would have upvoted this except for the concluding "try a single entry, single exit approach" remark. – L. Cornelius Dol

May 22, 2009 at 19:32

1 I think it's worth trying on for size once in a while. You always have ^Z. – Tom Hawtin - tackline May 23, 2009 at 1:13

3 "Hiding control-flow within state adds to confusion" is good advice in general – j_v_wow_d Mar 29, 2016 at 15:54

36

Labels are like goto's: Use them sparingly, and only when they make your code faster **and** more importantly, more understandable,

> e.g., If you are in big loops six levels deep and you encounter a condition that makes the rest of the loop pointless to complete, there's no sense in having 6 extra trap doors in your condition statements to exit out the loop early.

Labels (and goto's) aren't evil, it's just that sometimes people use them in bad ways. Most of the time we are actually trying to write our code so it is understandable for you and the next programmer who comes along. Making it uber-fast is a secondary concern (be wary of premature optimization).

When Labels (and goto's) are misused they make the code less readable, which causes grief for you and the next developer. The compiler doesn't care.

Share Improve this answer Follow

edited Dec 8, 2015 at 18:16

answered Sep 5, 2008 at 19:03

BIBD
15.4k ● 26 ● 87 ● 141

18 Yes, +1: GOTOs don't kill applications - programmers kill applications. – L. Cornelius Dol May 22, 2009 at 16:57

5 Historically, programmers have used GOTOs to kill applications. The GOTO received a bad name for a reason. – Michael Easter Jan 5, 2010 at 22:59

It all comes down to Dijkstra's letter "A Case Against the Goto Statement" / "Go To Statement Considered Harmful" which argues against it because of the tendency towards making a program into spaghetti code vs. what you would find in something more structured. In reality you are using goto's all the time (being it a break, continue, try/catch, etc.). Almost always it throws the execution forward. If you aren't disciplined in only goto'ing forward, you can increase the complexity of a function by an order of magnitude. – BIBD Aug 21, 2017 at 15:10

Goto'ing backward might be easy enough for you to understand today, but the next dev or you in 3 years will want to kill the SOB who saved 5 minutes of work, and left them with 3 hours are parsing the flow of the function. – BIBD Aug 21, 2017 at 15:12

27

There are few occasions when you need labels and they can be confusing because they are rarely used. However if you need to use one then use one.

BTW: this compiles and runs.

```
class MyFirstJavaProg {
        public static void main(String args[]) {
           http://www.javacoffeebreak.com/java101/java
           System.out.println("Hello World!");
        }
}
```

Share Improve this answer

Follow

edited Jul 15, 2011 at 12:51

Community Bot
1 ● 1

answered Jan 26, 2009 at 21:47

Peter Lawrey
533k ● 82 ● 767 ● 1.1k

11 Yeah, and a good syntax highlighter should make it clear why. – L. Cornelius Dol May 22, 2009 at 17:02

24 This is a terrible interview question, unless it's for a company whose codebase is utterly, obfuscated kludge. You will never see this in the wild, and if you do, a quick web search will show you what it means. There are more important things you can be testing prospective developers on, rather than silly little riddles. – jr. Oct 21, 2013 at 5:53

1 @studro code highlighting and formatting will show you what is about ;) The point is, you will always find things in other peoples code which isn't the way you would do things or what you would expect and how do you handle that. It's how you answer the question, rather than the answer you give which matters. Interviews should be about a quiz for what you know but what would you be like to work with and how you handle diverse ways of doing things. – Peter Lawrey Oct 21, 2013 at 12:04

1 It's most definitely obscure; it's great to say "it's how you handle it", but it's a gimmie for a poor developer who just

luckily happens to know about it, and an excluding factor for a good developer who hasn't encountered it in the wild (because it doesn't occur that often at all). "I'd Google 'labels in Java'" probably doesn't cut it in an interview. – jr. Oct 21, 2013 at 13:01

8

I'm curious to hear what your alternative to labels is. I think this is pretty much going to boil down to the argument of "return as early as possible" vs. "use a variable to hold the return value, and only return at the end."

Labels are pretty standard when you have nested loops. The only way they really decrease readability is when another developer has never seen them before and doesn't understand what they mean.

Share Improve this answer Follow

answered Sep 5, 2008 at 18:38

Tim Frey

9,931 ● 9 ● 45 ● 61

1 A recursive approach. Or you could conditionally break, if(found) break; and have that at the end of all your loops. You might also be able to split the inner loops into methods, and return instead of break. You should also question the benefits of breaking early, could you gain more performance from parallelization or a different algorithm. On a few occasions, I even realized the nesting was unnecessary, the loops could simply be refactored to run one after the other. – Didier A. Aug 17, 2015 at 17:19

6

I have use a Java labeled loop for an implementation of a Sieve method to find prime numbers (done for one of the project Euler math problems) which made it 10x faster compared to nested loops. Eg if(certain condition) go back to outer loop.

```
private static void testByFactoring() {
    primes: for (int ctr = 0; ctr < m_toFactor.length;
        int toTest = m_toFactor[ctr];
        for (int ctr2 = 0; ctr2 < m_divisors.length; c
            // max (int) Math.sqrt(m_numberToTest) + 1
            if (toTest != m_divisors[ctr2]
                        && toTest % m_divisors[ctr2] =
                continue primes;
            }
        } // end of the divisor loop
    } // end of primes loop
} // method
```

I asked a C++ programmer how bad labeled loops are, he said he would use them sparingly, but they can occasionally come in handy. For example, if you have 3 nested loops and for certain conditions you want to go back to the outermost loop.

So they have their uses, it depends on the problem you were trying to solve.

Share Improve this answer Follow

edited Apr 26, 2017 at 12:54

Daniel Alder
5,355 ● 2 ● 49 ● 57

answered Apr 20, 2010 at 15:41

Selenia
286 ● 2 ● 3

5

I've never seen labels used "in the wild" in Java code. If you really want to break across nested loops, see if you can refactor your method so that an early return statement does what you want.

Technically, I guess there's not much difference between an early return and a label. Practically, though, almost every Java developer has seen an early return and knows what it does. I'd guess many developers would at least be surprised by a label, and probably be confused.

I was taught the single entry / single exit orthodoxy in school, but I've since come to appreciate early return statements and breaking out of loops as a way to simplify code and make it clearer.

Share Improve this answer Follow

answered Sep 5, 2008 at 21:17
Don Kirkby
56.5k ● 27 ● 219 ● 301

5

I'd argue in favour of them in some locations, I found them particularly useful in this example:

```
nextItem: for(CartItem item : user.getCart()) {

  nextCondition : for(PurchaseCondition cond : item.ge
     if(!cond.check())
        continue nextItem;
```

```
      else
         continue nextCondition;

   }
   purchasedItems.add(item);
}
```

Share Improve this answer

Follow

answered Jan 26, 2009 at 12:26

Ceilingfish
5,445 ● 4 ● 47 ● 72

8 Well, the continue nextCondition is superfluous an just noise.
– L. Cornelius Dol May 22, 2009 at 17:00

5

I think with the new for-each loop, the label can be really clear.

For example:

```
sentence: for(Sentence sentence: paragraph) {
  for(String word: sentence) {
    // do something
    if(isDone()) {
      continue sentence;
    }
  }
}
```

I think that looks really clear by having your label the same as your variable in the new for-each. In fact, maybe Java should be evil and add implicit labels for-each variables heh

Share Improve this answer Follow

edited Aug 13, 2014 at 21:02

answered Sep 24, 2008 at 22:44

Pyrolistical
28k ● 21 ● 84 ● 109

1 Well, the `continue sentence;` does *not* continue processing the sentence the words of (which?) were processed by the inner loop: it continues with the _next sentence_/same paragraph. – greybeard Jan 15, 2016 at 19:36

3

I never use labels in my code. I prefer to create a guard and initialize it to *null* or other unusual value. This guard is often a result object. I haven't seen any of my coworkers using labels, nor found any in our repository. It really depends on your style of coding. In my opinion using labels would decrease the readability as it's not a common construct and usually it's not used in Java.

Share Improve this answer Follow

answered Sep 5, 2008 at 18:43

Bartosz Bierkowski
2,792 ● 1 ● 19 ● 18

1

Yes, you should avoid using label unless there's a specific reason to use them (the example of it simplifying implementation of an algorithm is pertinent). In such a case I would advise adding sufficient comments or other documentation to explain the reasoning behind it so that someone doesn't come along later and mangle it out of some notion of "improving the code" or "getting rid of code smell" or some other potentially BS excuse.

I would equate this sort of question with deciding when one should or shouldn't use the ternary if. The chief rationale being that it can impede readability and unless the programmer is very careful to name things in a reasonable way then use of conventions such as labels might make things a lot worse. Suppose the example using 'nextCondition' and 'nextItem' had used 'loop1' and 'loop2' for his label names.

Personally labels are one of those features that don't make a lot of sense to me, outside of Assembly or BASIC and other similarly limited languages. Java has plenty of more conventional/regular loop and control constructs.

Share Improve this answer Follow

answered Apr 10, 2016 at 3:44

Jeremy Harton

68 ● 5

1

I found labels to be sometimes useful in tests, to separate the usual setup, excercise and verify phases and group related statements. For example, using the BDD terminology:

```
@Test
public void should_Clear_Cached_Element() throws Excep
    given: {
        elementStream = defaultStream();
        elementStream.readElement();
```

```
            Assume.assumeNotNull(elementStream.lastRead())
        }
        when:
            elementStream.clearLast();
        then:
            assertThat(elementStream.lastRead()).isEmpty()
    }
```

Your formatting choices may vary but the core idea is that labels, in this case, provide a noticeable distinction between the logical sections comprising your test, better than comments can. I think the Spock library just builds on this very feature to declare its test phases.

Share Improve this answer Follow

answered Nov 11, 2016 at 13:48

polaretto
799 ● 10 ● 11

0

Personally whenever I need to use nested loops with the innermost one having to break out of all the parent loops, I just write everything in a method with a return statement when my condition is met, it's far more readable and logical.

Example Using method:

```
    private static boolean exists(int[][] array, int sea
      for (int[] nums : array) {
        for (int num : nums) {
          if (num == searchFor) {
            return true;
          }
        }
      }
    return false;
  }
```

Example Using label (less readable imo):

```
boolean exists = false;
existenceLoop:
for (int[] nums : array) {
  for (int num : nums) {
    if (num == searchFor) {
      exists = true;
      break existenceLoop;
    }
  }
}

return exists;
```

Share Improve this answer

Follow

answered Sep 19, 2021 at 14:34

Yassir Khaldi

1,742 ● 2 ● 21 ● 31