500
JAVA
TIPS
Let's make things easier
http://JavaFAQ.nu
500 JAVA TIPS

500 Java Tips

Hello dear friend!

I am very glad that you are here!

"500 Java Tips" is my collection of good questions and answers from my site, numerous Java forums and newsletters.

This is DEMO VERSION!

Full version is here

Plus Present for you:

- 3 Great Java Tools see here

P.S. It is possible that this info is obsolete and I have more presents for you (CNET.com does not let frequent updates for free...)
So, just check and probably you get even more!

Please visit my site at: http://JavaFAQ.nu!

Receive our newsletter with new tips! More than 11,000 subscribers (by 10 January 2003) can not be wrong! They read our tips every week!

To subscribe to the "Java FAQ Daily Tips" weekly edition newsletter please send e-mail with "subscribe" word in the header and the body (write just subscribe without ""!!!) to:

javafaqtips-request@javafaq.nu

or on the web:

http://javafaq.nu/mailman/listinfo/javafaqtips_javafaq.nu

Contents at a Glance

Excuse me for possible mistakes! English is not native language for me.
I will be glad if you send me your corrections of my mistakes!

All questions please mailto:info@javafaq.nu For advertisers

500 Java Tips

Java FAQ.nu

?!! javaFAQ.nu JavaQ&A

This Page - example of a few tips from "500 Java Tips"
and just a tiny part of REALLY BIG E-Book - 389 pages!

A4 format!

Q: Why we can not declare constructor as final?

Answer: The keyword final when dealing with methods means the method cannot be overridden. Because constructors are never inherited and so will never have the oportunity to be overridden, final would have no meaning to a constructor.

Q: Why can not I mix AWT and Swing?

Recently, I have been hearing a lot of people from various newsgroups and website saying, java swing and awt can't be in the same application. They will not work together and they might produce unexpected results. At the same time, i don't hear people saying "why" you shouldn't use swing and awt together. Could someone out there shed some light for me. Is their any logical reason why we shouldn't mix swing and awt in the same application/applet. If there is a problem mixing swing and awt... what are the results, what can happen? I design using IBM's Visual Age for Java 3.0, and I mix swing and awt in the same application/applet, it works fine when testing in the IDE (I haven't tested it outside of the IDE yet). If you have tested application/applets outside of the IDE, please let me know what happened?

Answer: There are findamental incompatibilities in the way they draw themselves.
AWT java classes are not "pure" Java classes, they use underlaying C/C++ native code (dependable on operation system) that can cause different appearence in different OSs.
Swing is pure Java implementation and has no native code at all. Swing applications look the same.
> If there is a problem mixing swing and awt... what are the results,
> what can happen?

Some objects drawn on top of others are not properly occluded. This is most obvious with drop down menus, which have a tendency to stay visible even after you have selected a menu item. Another problem is that if you use AWT components on a JTabbedPane they will not disappear when you switch tabs. There are many similar issues.

Q: Again about difference between AWT and Swing

I have a question: What are the architectural differences between Swing and AWT??

Answer: by Odd Vinje
There are no big architectural differences, the class hierarchy is almost the same. The reason is that Swing is built upon AWT.

The most significant difference is how the components are drawn to the screen. AWT is so called heavyweight components and have their own viewport which sends the output to the screen. Swing is ligthweight components and does not write itself to the screen, but redirect it to the component it builds on. Heavyweight components also have their own z-ordering. This is the reason why you can't combine AWT and Swing in the same container. If you do, AWT will always be drawn on top of the Swing components.

You can combine AWT and Swing, just don't do it in the same container (e.g. panel, groupbox, etc.) and don't put a heavyweight component inside a lightweight.

Another difference is that Swing is pure Java, and therefore platform independent. Swing looks identically on all platforms, while AWT looks different on different platforms.

Q: I've got a Java application that I need to install on my future customer's computers but this is the first program that I've ever tried to sell to anybody and I want to keep my costs down.

Is there something out there that installs the JRE, if it needs to, and then my program ?

I know of InstallShield, InstallAnywhere and the like but I'm hoping to find something a lot cheaper since I'm just a small time operator (who may not sell even one of his programs).

Answer:
1. ZipCentral is a free Windows app that will create a zip file, or a self-extracting executable. It has the ability to execute a BAT file after unzipping your files. You can download a copy from

http://zipcentral.iscool.net/

2. Sun's Java Web Start (the price is right - free):

http://java.sun.com/products/javawebstart/index.html

JavaTM Web Start -- a new application-deployment technology -- gives you the power to launch full-featured applications with a single click from your Web browser. You can now download and launch applications, such as a complete spreadsheet program or an Internet chat client, without going through complicated installation procedures

3. Zerog's Now! http://www.zerog.com/downloads_01.html

There are no restrictions on the use of free products:
InstallAnywhere Now! and PowerUpdate Now!. Now! 4.01 is free, installs a JRE

Q: Does anyone know how to make a file NOT "read-only"???

Does anyone know how to make a file NOT "read-only"?? I know how to make a File read-only, but I do not know how to do the opposite. I have consulted the JavaDoc for File, FileDescriptor, FileSystem and have not found a way of doing this. Any help would be greatly appreciated.

To set Read-Only:

```
File file = new File( "c:/testFile.txt" );
file.setReadOnly();
```

Answer: You can't from Java. See:
http://developer.java.sun.com/developer/bugParade/bugs/4167472.html

I find Sun's argument rather weak however.

You can get around the problem by calling Runtime.exec() with a command like "chmod" specific to your operating system, in order to modify the file mode.

Q: Is it possible to change delays that affect appearing, keeping and disappearing of tooltip?

Answer: It was difficult to find the answer but finally I found in "Swing" book that is free to you on our site. The ToolTipManager is a service class that maintains a shared instance registered with AppContext. We can access
the ToolTipManager directly by calling its static sharedInstance() method:

```
ToolTipManager toolTipManager = ToolTipManager.sharedInstance();
```

Internally this class uses three non-repeating Timers with delay times defaulting to 750, 500, and 4000. ToolTipManager uses these Timer's in coordination with mouse listeners to determine if and when to display a JToolTip with a component's specified tooltip text. When the mouse enters a components bounds ToolTipManager
will detect this and wait 750ms until displaying a JToolTip for that component. This is referred to as the initial delay time.
A JToolTip will stay visible for 4000ms or until we move the mouse outside of that component's bounds, whichever comes first. This is referred to as the dismiss delay time. The 500ms Timer represents the reshow delay time which specifies how soon the JToolTip we have just seen will appear again when this component is re-entered.
Each of these delay times can be set using ToolTipManager's setDismissDelay(), setInitialDelay(), and setReshowDelay() methods.

ToolTipManager is a very nice service to have implemented for us, but it does have significant limitations. When we construct our polygonal buttons we will find that it is not robust enough to support non-rectangular components.

Q: I am wondering if JDK supports to open a file in the exclusive mode?

Answer: No, file locking is not supported in current VMs. You can implement lockfiles, however, using the File.createNewFile() method. Since JDK 1.3:
"Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. The check for the ' existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file. This method, in combination with the deleteOnExit() method, can therefore serve as the basis for a simple but reliable cooperative file-locking protocol."

Q: Why we can reengineer Java byte code (.class file) back to Java source code? But why binary .exe file we are unable to do it? What is the significant difference?

Answer: AFAIK, Java byte code goes back to _some_ source, not to the original source. So, reverse engineering is limited.

> But why binary .exe file we are unable to do it? What is the significant difference?

(a) There is more than one way to do something using C++.
(b) There are highly optimizing C++ compilers.
(c) You won't get the original source anyway. See (a).

Imagine that your C++ code contains inline functions. The compiler is free do place the body of it replacing a call, or instantiate the function and provide a real call to it. First, there is no way to know when it is going to do one or the other. Second, with some code inlined, how to decide what _was_ an inline function and what wasn't?

So, the answer is: the distances between the levels of abstraction between byte code and Java source and between machine code and C++ source are significantly different. The bigger the distance, the less possible it is to recreate the source code.

Victor Bazarov

in java bytecode all the original variable names are kept. in an exe file smaller symbols are used.

Some .exe decompilers are good enough to convince a jury that the "original" source code was reconstituted. See the Microsoft v. Stac case for an example.

Java is easier, but C is still possible.

Q: Is it possible to know if a file changed?

Answer: The are some techniques:

1. Trust the last change date the OS maintains. The file could have had an Oscar Wilde pulled on it, insert a comma and take it out again so the last change date records a change, but it is not really changed.

2. Check the file size. Most chances will trigger a file size change.

3. Compute a digest, e.g. a 64 bit Adlerian checksum. If it is the same, chances are nothing really

changed. The odds are quite astronomically small you could have a change and not detect it, but it is theoretically possible.

4. Compare the old and new files with .equals() after reading them into RAM, possibly in chunks.

I talk peripherally about these issues in the automatic file updater student project and the delta creator project in the student projects section of the java glossary.

Q: Is it possible to stop an object from being created during construction?

For example if an error occurs inside the constructor (e.g. the parameters pass in were invalid) and I wanted to stop an object being created would it be possible to return null rather than a reference to a new object. (I know the term return is technically correct in this case but you know what I mean). Basically, is it possible to cancel object creation?

Answer: Yes, have the constructor throw an exception. Formally, an object _will_ be created (since the constructor is a method invoked after the actual method creation), but nothing useful will be returned to the program, and the dead object will be later reclaimed by Garbage Collector.

But the clean way is as another reply suggests, that you leave calls to the constructor to a static factory method which can check the parameters and return null when needed.

Note that a constructor - or any method in general - throwing an exception will not "return null", but will leave the "assign target" as it was.
Tor Iver Wilhelmsen

Q: Is it true that ThreadDeath exception was "secret" and accidentally exposed to public?

Answer: Yes it is true. The original implementation was not intended to show it to us. But due to some mistakes of SUN programmers it got out and now it is a part of API.
Warning: "An application should catch instances of this class only if it must clean up after being terminated asynchronously. If ThreadDeath is caught by a method, it is important that it be rethrown so that the thread actually dies." from API reference for java.lang.ThreadDeath.

Q: My friend claim that garbage collectors do not collect int value since they are not created with new() method..
Is he right?

Answer: Programmers know about the importance of initialization, but often forget the importance of cleanup. After all, who needs to clean up an int? But with libraries, simply “letting go” of an object once you’re done with it is not always safe. Of course, Java has the garbage collector to reclaim the memory of objects that are no longer used. Now consider a very unusual case. Suppose your object allocates “special” memory without using new. The garbage collector knows only how to release memory allocated with new, so it won’t know how to release the object’s “special” memory. To handle this case, Java provides a method called finalize( ) that you can define for your class. Here’s how it’s supposed to work. When the garbage collector is ready to release the storage used for your object, it will first call finalize( ), and only on the next garbage-collection pass will it reclaim the object’s memory. So if you choose to use finalize( ), it gives you the ability to perform some important cleanup at the time of garbage collection.

source: http://www.javafaq.nu/java/book/Chapter04.shtml#Heading169

Q: Could you please tell the advantages and disadvantages of having Garbage Collecting in Java?

Answer:
1. Although the programmer still allocates data structures, they are never explicitly freed. Instead, they are "garbage collected" when no live references to them are detected. This avoids the problem of having a live pointer to a dead object.
So, GC "keeps" eye on amount of memory allocated by program and tries to free memory of unreferenced objects in "good" (when your program does not consume much CPU) time.
You do not need to do free() operation like in C++. GC does it for you. Most of memory leaks happen due to bugs in Java itself rather than bad programming (happens also :-))
In a large application, a good garbage collector is more efficient than malloc/free
2. GC makes heap defragmentation (merges the small pieces of free memory into one big piece) that increases performance on the fly. It is difficult to do such thing easy in most of programs written on C++.
3. Your time! If you have fast enough CPU and good GC you will save a lot of time. Manual tuning of a millions pieces of code like malloc/free will take so much time that increases the cost of project dramatically!
Let say like this if you have slow CPU and small program then C++ with malloc/free is more efficient. If you have big one - rely on GC!
4. Security issue: Java programmers cannot crash the JVM by incorrectly freeing memory

Main disadvantage is that GC adds overhead that can affect performance. In some real time it is critically important that no GC-ing will run in definite periods of time..
--
AP

Here you see one example of article with code examples.

Determining Memory Usage in Java

Author: Dr. Heinz M. Kabutz

(C)opyright Maximum Solutions, South Africa

Welcome to the 29th issue of "The Java(tm) Specialists' Newsletter". I could start off with a witty comment about how the newsletter is going to hit the **big three** at the next issue, but I might step on the toes of my old friend (haha) John Green who is turning 30 today - happy birthday! At least I'm not *that* old yet :-) By the time you read the next newsletter, or maybe this newsletter, I will probably be father the second time round.

This week I am showing you one of my most dear trade secrets. Please be very careful who you show this newsletter to, only send it to friends and people on your local JUG. If this gets into the wrong hands, project troubleshooters like me will be out of a job.

One of the fun parts in Java is guessing how much memory is being used by your object. We are conditioned to ignore memory altogether when programming in Java and that can easily land us in trouble. Java does not have a construct like C/C++ that tells us how much space an object is taking, at least until this newsletter...

Warning: The results in this newsletter were derived experimentally rather than looking at the innards or the JVM. Please try out the experiments if you are running on a non-WinNT machine and tell me if you get different results.

Memory Usage in Java

In Java, memory is allocated in various places such as the stack, heap, etc. In this newsletter I'm only going to look at objects which are stored on the heap. Please don't take me to task for not mentioning the others, they might appear in a future newsletter.

Say I have a class Foo, how much memory will one instance of that class take? The amount of memory can be determined by looking at the data members of the class and all the superclasses' data members. The algorithm I use works as follows:

1. The class takes up at least 8 bytes. So, if you say **`new`** `Object();` you will allocate 8 bytes on the heap.
2. Each data member takes up 4 bytes, except for long and double which take up 8 bytes. Even if the data member is a byte, it will still take up 4 bytes! In addition, the amount of memory used is increased in 8 byte blocks. So, if you have a class that contains one byte it will take up 8 bytes for the class and 8 bytes for the data, totalling 16 bytes (groan!).
3. Arrays are a bit more clever, at least smaller primitives get packed. I'll deal with these later.

In order to be able to test many different types of objects, I have written a MemoryTestBench class that takes an ObjectFactory which is able to create the type of object that you want to test. The MemoryTestBench can either tell you how many bytes are used by that object or it can print out a nicely formatted result for you. You get the most accurate results if you make sure that supplementary memory is already allocated when you start counting. I therefore construct the object, call the methods for finding the memory, and then set the handle to null again. The garbage collector is then called many times, which should free up all unused memory. The memory is then counted, the object created, garbage collected, and the memory counted again. The difference is the amount of memory used by your object, voila!

```
public class MemoryTestBench {
  public long calculateMemoryUsage(ObjectFactory factory) {
    Object handle = factory.makeObject();
    long mem0 = Runtime.getRuntime().totalMemory() -
      Runtime.getRuntime().freeMemory();
    long mem1 = Runtime.getRuntime().totalMemory() -
      Runtime.getRuntime().freeMemory();
    handle = null;
    System.gc(); System.gc(); System.gc(); System.gc();
    System.gc(); System.gc(); System.gc(); System.gc();
    System.gc(); System.gc(); System.gc(); System.gc();
    System.gc(); System.gc(); System.gc(); System.gc();
    mem0 = Runtime.getRuntime().totalMemory() -
      Runtime.getRuntime().freeMemory();
    handle = factory.makeObject();
    System.gc(); System.gc(); System.gc(); System.gc();
    System.gc(); System.gc(); System.gc(); System.gc();
    System.gc(); System.gc(); System.gc(); System.gc();
    System.gc(); System.gc(); System.gc(); System.gc();
    mem1 = Runtime.getRuntime().totalMemory() -
      Runtime.getRuntime().freeMemory();
    return mem1 - mem0;
  }
  public void showMemoryUsage(ObjectFactory factory) {
    long mem = calculateMemoryUsage(factory);
    System.out.println(
      factory.getClass().getName() + " produced " +
      factory.makeObject().getClass().getName() +
      " which took " + mem + " bytes");
  }
}
```

The ObjectFactory interface looks like this:

```
  public interface ObjectFactory {
    public Object makeObject();
  }
```

Basic Objects

Let's start with the easiest case, a BasicObjectFactory that simply returns a new instance of Object.

```
  public class BasicObjectFactory implements ObjectFactory {
    public Object makeObject() {
      return new Object();
    }
  }
```

When we run this, we get the following output:

```
  BasicObjectFactory produced java.lang.Object which took 8 bytes
```

Bytes

I suggested earlier that bytes are *not* packed in Java and that memory usage is increased in 8 byte blocks. I have written the ByteFactory and the ThreeByteFactory to demonstrate this:

```
  public class ByteFactory implements ObjectFactory {
    public Object makeObject() {
      return new Byte((byte)33);
    }
  }

  public class ThreeByteFactory implements ObjectFactory {
    private static class ThreeBytes {
      byte b0, b1, b2;
    }
    public Object makeObject() {
      return new ThreeBytes();
    }
  }
```

When we run these, we get the following output:

```
  ByteFactory produced java.lang.Byte which took 16 bytes
```

```
ThreeByteFactory produced ThreeByteFactory$ThreeBytes which took 24 bytes
```

This is great (not). When I first started using Java I used to spend hours deciding whether a variable should be an int or short or a byte in order to minimize the memory footprint. **I was wasting my time.** As I said earlier, I don't know if this is only a problem under NT or if it's the same on all platforms. Knowing Java's dream of being equally inefficient on all platforms, I suspect that it would be the same.

Booleans

Let's carry on and look at a smaller unit of information, the boolean. Now a boolean is simply a bit, true or false, yes or no, zero or one. If I have a class that contains 64 booleans, guess how much memory it will take? 8 for the class, and 4 for each of the boolean data members, i.e. 264 bytes!!! Since a boolean is essentially the same as a bit, we could have stored the same information in one long. If you don't believe me, have a look at the following class:

```
public class SixtyFourBooleanFactory implements ObjectFactory {
  private static class SixtyFourBooleans {
    boolean a0, a1, a2, a3, a4, a5, a6, a7;
    boolean b0, b1, b2, b3, b4, b5, b6, b7;
    boolean c0, c1, c2, c3, c4, c5, c6, c7;
    boolean d0, d1, d2, d3, d4, d5, d6, d7;
    boolean e0, e1, e2, e3, e4, e5, e6, e7;
    boolean f0, f1, f2, f3, f4, f5, f6, f7;
    boolean g0, g1, g2, g3, g4, g5, g6, g7;
    boolean h0, h1, h2, h3, h4, h5, h6, h7;
  }
  public Object makeObject() {
    return new SixtyFourBooleans();
  }
}
```

When we run this, we get the following output:

```
SixtyFourBooleanFactory produced SixtyFourBooleanFactory$SixtyFourBooleans
  which took 264 bytes
```

Admittedly, the example was a little bit contrived, as you would seldom have that many booleans in one class, but I hope you get the idea.

Sun must have realised this problem so they made constants in java.lang.Boolean for TRUE and FALSE that both contain instances of java.lang.Boolean. I think that the constructor for Boolean should have been private to stop people from creating 16 byte objects that are completely unnecessary.

Arrays of Boolean Objects

A Boolean Array takes up 16 bytes plus 4 bytes per position with a minimum of 8 bytes at a time. In addition to that, we obviously have to count the actualy space taken by Boolean objects.

```
public class BooleanArrayFactory implements ObjectFactory {
  public Object makeObject() {
    Boolean[] objs = new Boolean[1000];
    for (int i=0; i<objs.length; i++)
      objs[i] = new Boolean(true);
    return objs;
    }
  }
```

Try guess how many bytes would be taken up by a Boolean array of size 1000 with Boolean objects stuck in there. Ok, I'll help you: 16 + 4*1000 (for the pointers) + 16*1000 (for the actual Boolean objects) = 20016. Run the code and see if I'm right ;-) If we, instead of making a new Boolean object each time, use the Flyweights provided in Boolean, we'll get to 16 + 4*1000 = 4016 bytes used.

Primitives get packed in arrays, so if you have an array of bytes they will each take up one byte (wow!). The memory usage of course still goes up in 8 byte blocks.

```
  public class PrimitiveByteArrayFactory implements ObjectFactory {
    public Object makeObject() {
      return new byte[1000];
    }
  }
```

When we run this, we get the following output:

```
  PrimitiveByteArrayFactory produced [B which took 1016 bytes
```

java.lang.String

Strings actually fare quite well since they can be "internalised" meaning that only one instance of the same String is kept. If you, however, construct your String dynamically, it will not be interned and will take up a bit of memory. Inside String we find:

```
  // ...
  private char value[];
  private int offset;
  private int count;
  private int hash = 0;
  // ...
```

Say we want to find out how much "Hello World!" would take. We start adding up 8 (for the String class) + 16 (for the char[]) + 12 * 2 (for the characters) + 4 (value) + 4 (offset) + 4 (count) + 4 (hash) = 64 bytes. It's quite difficult to measure this, as we have to make sure the String is not internalized by the JVM. I used the StringBuffer to get this right:

```
  public class StringFactory implements ObjectFactory {
    public Object makeObject() {
      StringBuffer buf = new StringBuffer(12);
      buf.append("Hello ");
      buf.append("World!");
      return buf.toString();
    }
  }
```

When we run this, we get, as expected, the following output:

```
  StringFactory produced java.lang.String which took 64 bytes
```

java.util.Vector

Now we get to the real challenge: How much does a java.util.Vector use in memory? It's easy to say, now that we have a MemoryTestBench, but it's not so easy to explain. We start by looking inside the java.util.Vector class. Inside we find the following:

```
// ...
protected Object elementData[];
protected int elementCount;
// ...
```

Using the knowledge we already have, we decide that the amount of memory used will be 8 (for the class) + 4 (for the pointer to elementData) + 4 (for elementCount). The elementData array will take 16 (for the elementData class and the length) plus 4 * elementData.length. We then follow the hierarchy up and discover the variable `int modCount` in the superclass `java.util.AbstractList`, which will take up the minimum 8 bytes. For a Vector of size 10, we will therefore take up: 8 + 4 + 4 + 16 + 4*10 + 8 = 80 bytes, or simply 40 + 4*10 = 80 bytes, which agrees with our experiment:

```
public class VectorFactory implements ObjectFactory {
  public Object makeObject() {
    return new java.util.Vector(10);
  }
}
```

When we run this, we get the following output:

```
VectorFactory produced java.util.Vector which took 80 bytes
```

So, what happens when we create a JTable with a DefaultTableModel with 100x100 cells? The DefaultTableModel keeps a Vector of Vectors so this will take 40 + 4*100 + (40 + 4*100) * 100 = 440 + 44000 = 44440 bytes just for the empty table. If we put an Integer in each cell, we will end up with another 100*100*16 = 160'000 bytes used up.

java.util.LinkedList

What's better, a java.util.LinkedList or a java.util.ArrayList? Experienced followers of these newsletters will of course say: "Neither, the CircularArrayList is better" ;-). Let's see what happens when we put 10000 objects into an ArrayList (which uses the same amount of memory as the Vector) vs. a LinkedList. Remember that each Object takes up 8 bytes, so we will subtract 80000 bytes from each answer to get comparable values:

```
import java.util.*;
public class FullArrayListFactory implements ObjectFactory {
  public Object makeObject() {
    ArrayList result = new ArrayList(10000);
    for (int i=0; i<10000; i++) {
      result.add(new Object());
    }
    return result;
  }
}

import java.util.*;
public class FullLinkedListFactory implements ObjectFactory {
  public Object makeObject() {
    LinkedList result = new LinkedList();
    for (int i=0; i<10000; i++) {
      result.add(new Object());
    }
    return result;
  }
}
```

When we run this, we get the following output:

```
FullArrayListFactory produced java.util.ArrayList which took 120040 bytes
FullLinkedListFactory produced java.util.LinkedList which took 320048 bytes
```

When we subtract 80000 bytes from each, we find that the ArrayList takes up 40040 bytes (as expected) and the LinkedList uses 240048 bytes. How many of us consider issues like this when we code?

We have come to the end of yet another newsletter. I am trying to put newsletters together that will be worthwhile to send out, so as a result they will not always appear every week, unless I feel particularly inspired.

Until the next issue...

Heinz
