PSet 3 Hints

Don't Panic!

You will probably feel overwhelmed by this problem set the first time you look at it. There's a lot of new stuff: the Java bytecode specification, BCEL, Dot, etc. You may have never used a third party library (eg, BCEL) before. Don't panic. This is a cookie-cutter pset: we have given you template code and a template report. Just fill in the blanks.

You may feel tempted to print out the VM spec and start reading it like a novel. Resist this temptation: it will fill your head with many details that are not directly relevant to this problem set. An API or a specification is a reference document: use it like a dictionary or an encyclopedia, to look up the answer to a specific question you have.

The fastest way to complete this pset is to focus on the blanks you need to fill in. Read the pset and the solution template and the hints carefully. Focus your efforts. It's actually fairly straight-forward if you don't let yourself feel overwhelmed by the detail. Don't panic.

This problem set is different from many other problem sets you may have done. It is often the case that all of the material accompanying a pset is highly relevant to solving that pset. That is not the case with this pset, because that is not a typical experience of working with a new (to you) technology to solve a problem. There is a lot of material accompanying this pset, and only some of it is directly relevant to solving the pset. For example, the VM spec chapters on bytecode verifiers are not particularly relevant to this pset. One of the challenges of this pset is for you to be able to quickly discern what you need to read and what you can safely ignore. This is an important intellectual skill that will serve you well throughout your higher education and working life.

Software

There are sometimes problems with running Dot on the MIT server Solaris

Dot is supposed to work on Solaris, and the MIT server is supposed to be one nice homogeneous environment, but our experience with this pset so far is that Dot may have some trouble on the MIT server Solaris. It seems to work on fine on the MIT server Linux.

Eclipse doesn't automatically ping the file system

If you do something that modifies the file system of your project (eg, run build.xml), then you will need to tell Eclipse to refresh its view of the file system before you see those changes: right-click/refresh on the project.

Do not run multiple instances of Eclipse on the same workspace

This will result in the two instances clobbering each other's metadata, and you may have to delete your workspace/.metadata directory and re-configure your workspace.

Object Models

There are two kinds of edges, only one of them has multiplicity markings.

Problems 3 and 4

1. The way to think about this question is: "I've read the code in src/ps3/ObjectModel.java, and I can see it opens up a .class file for me. I wonder how I can tell if that class file contains a class or an interface? There must be a bit in there some there that records this information, and it's likely the BCEL API makes this bit available to me."
2. If you do not find the answer to this question quickly, move on and come back to it later. The answer will probably become evident as you progress through the problem set.
3. This is intellectually the most important exercise, but the rest of the pset does not depend on it that much.
4. Discuss with colleagues, look in the language or the vm spec, see what the compiler does.

Problem 5

This table has two purposes: to help you get a handle on what your tool needs to do, and to help you realize that some stuff is hard to do mechanically. A number of the questions in this pset are of the form: "there are some things a tool could do: how hard are they? make your tool do the easy ones." "Hard" basically means "reading and understanding code in a method body", and "easy" means "not having to do hard stuff".

Problem 6

Look at src/ps3/ObjectModel.java. Run build.xml. View doc/report.html in a web browser. You already have a working object model extractor. It's just that the models it extracts aren't very good. You need to make them better. Look at the source code for the input programs. Draw object models for them (manually). Look at the current output of the tool. See how it differs from what you drew by hand. Try to improve the output of the tool. It's an iterative and incremental process.

Problem 6.4

Problem 6.4 asks you to do some stuff with method bodies. Is that hard? No. What's the difference? Part (d) is asking you to do a *heuristic* analysis: something that's probably good enough in most cases, but may sometimes be wrong. Admitting the possibility of an incorrect output makes what may be, in general, impossibly hard, into something that may be tractable. Looking at method bodies isn't all that difficult, but trying to mechanically deduce anything from them with absolute certainty is. The point, for the user of a tool such as the one you're writing, is to consider this question: is it more useful to have a tool that tells more information, but may sometimes be incorrect, or a tool that is always correct but doesn't say much?