Contents

[Use of Debugging 1](#_Toc504837457)

[Garbage Collector Fundamentals 2](#_Toc504837458)

[Java Interfaces – Soda Machine and a Monkey 2](#_Toc504837459)

# Use of Debugging

In the weeks to come you’ll encounter issues with your application that requires a fair amount of ‘debugging’.

All IDEs have built-in debuggers and the use of them is**imperative to our success as a software developer!**

In my past candidate interviews (and I've heard from others as well), candidates are required to write an application and then demonstrate use of debugger.

**Why the emphasis on using the debugger?**

Sure, one can walk up to another developer in the real-world and ask how to fix an issue. However, 'debugging' is considered a fundamental skill that all developers are expected to know, especially given the vast amount of tutorials on them. In my decades of software development and observation, over-reliance on others at work can quickly lead to negative issues and something we want to avoid at all costs. I've seen this so much that I feel compelled to share it.

Eclipse has an excellent built-in help for using its debugger.

1. To access it, go to Help -> Search and enter Debugger
2. For purpose of our course, we are interested in
   1. Local debugging
   2. Adding Breakpoints (all breakpoint related tasks)
   3. [Stepping through the execution of a program](http://127.0.0.1:53423/help/ntopic/org.eclipse.jdt.doc.user/tasks/task-stepping.htm)

Some sample tutorials:

Eclipse :

<https://www.youtube.com/watch?v=9gAjIQc4bPU>

# Garbage Collector Fundamentals

Granted this is not part of the course material, however the GC (Garbage Collector) is a vital function of the JVM and something not too many have a good understanding of. Understanding this while we are learning Java will enable us to build better performing applications. Attached document goes over the fundamentals of GC.

In the weeks to come, we'll be exploring the use of some of the JDK bin utilities (that you reviewed in Week1) in relation to garbage collectors.

One key takeaway from this read should be that every time you use the 'new' keyword in Java, you are adding to the performance of your application and the GC has to work hard to clean up. So nothing is free, so be very aware of when you create objects, that they serve a precise purpose. In following week's topic of Exceptions, you'll see the initial desire to create Exceptions from different layers of your application - they come with a cost!

Again, this is not part of the course material and I’m providing this to expand your understanding so your inclination and interest is what that will get you there!

# Java Interfaces – Soda Machine and a Monkey

I can’t take credit for this anecdote, but came across this explanation by the author’s professor:

Does anyone here know how a soda machine works? No? Let me tell you.

You drop in your change, and inside the machine is a little monkey who counts all your change to make sure you put in enough money. When you press the button for your soda, a little light comes on telling the monkey which button you pressed, and if you entered the right amount of change, he grabs your choice and throws it into the little hole for you to grab your soda.

This is the concept of encapsulation. We hide the implementation of the soda machine. Unless it's got one of those fancy, clear windows to let you see the inside, you honestly have no idea how it really works. All you know is that you put in some cash, you press a button, and if you put in enough, you get your drink.

To add to that, you know how to use a soda machine's **interface**, so therefore as long as the machine's **interface** follows the usual soda machine interface, you can use it. This is called the interface contract. The machine can be bringing the drinks from Antarctica on a conveyor belt for all you care, as long as you get your drink, its cold, and you get change back.

Polymorphism is the idea that when you use the soda machine interface, it could be doing different things. This is why encapsulation and polymorphism are closely related. In polymorphism, all you know is that you're using a SodaMachine implementation, which can be changed, and as a result, different things can be done behind the scenes. This leads to the driving concept of polymorphism, which is the ability of one object, the SodaMachine, to actually act as both a MonkeySodaMachine and a ConveyorSodaMachine depending on the machine actually behind the interface.