Improving in 2110 through better study habits

David Gries

Introduction

This note contains two items:

1. Suggestions for improving study habits. These are based on my own experience. To me they make sense.
2. Definitions, principles, algorithms, etc. that you should master for CS2110. Some of them are Java based, some are about correctness of programs, others are about algorithms and data structures that you are learning in CS2110.

Study habits

1. **Study on a regular, frequent, basis** —every other day, perhaps. Not for a long time —1/2 to 1 hours. Just as you practice every day when learning a musical instrument, you need steady, constant practice to gain fluency with Java, Eclipse, algorithm development, etc.
2. **Don't just read and listen, DO**. Just reading and watching videos is not an effective way to learn; to absorb the material, you have to work with the material yourself.

Example. Here's how to learn how the new-expression is evaluated.

On a blank piece of paper, write a new-expression, e.g. new C(5) , and then write down how to evaluate it. There are 3 steps. When done, compare what you wrote to how we said it is evaluated and note any differences. For example, you might have said for the second step, "Store the value 5 in some field", while our second step was "Execute the constructor call C(5)". Think about the difference between them. We don't see the constructor, so we don't really know whether it stores a value in a field. All we know is that the constructor call has to be executed.

Next day, do exactly the same thing. You will do better, and you may even get it right. If not, look at our answer, note the differences, and do it again the next night. After spending 5-10 minutes on 2-3 evenings, you *know* that material.

Also try evaluating a new-expression yourself. Write class C with an int field and a constructor, then evaluate "new C(5)" yourself: Draw the new object, execute the constructor call, and then write above the new-expression what its value is.

Example. We said that you should be able to *develop* (not memorize the code for) several algorithms for searching/sorting. Practice developing them! Consider selection sort (we do this with formulas instead of array diagrams because the latter are too hard to write here:

Precondition Q: We don't know anything about the contents of array b

Postcondition R: b[0..b.length-1] is sorted

Generalize to a loop invariant:

Inv P1: $0 \le k \le b.length$ and b[0..k-1] is sorted.

But you know that this is the invariant for insertion sort, and there is ONE more piece of the invariant you have to put in:

Inv P2: $b[0..k-1] \le b[k..]$

Invariant P2 is the *one* point you have to memorize. Now, develop the loop using the four loopy questions.

Our goal in asking you to learn to develop these searching/sorting methods in this way is two-fold. (1) You then really "know" these algorithms, not because you memorized code but because you can develop them. Every self-respecting programmer should know these basic searching/sorting algorithms. (2) You gain a skill that you can later use to effectively and efficiently develop your own algorithms/loops.

3. **Don't just ask for the answer to a question**, **first look for the answer yourself.**

One sees questions on the Piazza like this. "What is the difference between static and abstract?" "What List method can we use to prepend something to a List? The answer to the first question can be found immediately by looking in JavaHyperText. For the second question, look at the API documentation for class List!

We don't mind answering questions on the Piazza; that's what the Piazza is for. At the same time, we won't be available to answer your questions after this course. You have to get in the habit of looking for answers yourself —on JavaHyperText, in the Java API documentation, and anywhere on the internet.

4. **Use Eclipse frequently to try things out**.

Did you develop selection sort? See whether it is correct by putting it into Eclipse and testing it. Have a doubt about not being able to create an instance of an abstract class? Try it.

For this purpose, have a project in Eclipse, just for playing around and trying things. It should include one class where you can place methods and a JUnit testing class, where you can write test cases. You can add a class or interface quickly if you need to try something out and delete it when you have finished with it.

Java definitions and principles

We list some points that you should know about Java.

1. **Execution/evaluation** of the following constructs. Be able to write them down. What you write does not have to be word-for-word the same as ours, but it should be correct, precise, brief, without a lot of noise and clutter. Look in JavaHyperText for them.
    a. Assignment statement
    b. If-statement, if-else statement
    c. New-expression
    d. Loops (for, while, foreach)
    e. Method call
    f. Try-statement, throw statement
2. **Compiletime, runtime, syntax, semantics**.
3. **How we draw objects.** Type "Object, class" into the JavaHyperText Filter Field to see how we draw objects, and about the *superest* class of them all, Object. Whenever you have a problem that deals with several objects, drawing them often provides understanding. So, practice drawing them.
4. **Constructors**. Type constructors into the JavaHyperText Filter Field and look at the pdf file on the last line. It summarizes all the important points about constructors. Learn them all.
5. **Referencing fields and methods**. Look these up in JavaHyperText:
    a. compile-time reference rule
    b. inside-out rule,
    c. bottom-up/overriding rule
    d. this evaluates to the pointer to the object in which it appears. Use of **this**. to overcome shadowing.
    e. super use **super.** to call an overridden method.
6. **Abstract class, method.** Know the purpose of making a class abstract. Same for making a method abstract. Know the syntax for each.

Correctness of programs, developing loops

Algorithms you should be able to develop