

Caveat:

This is mostly a set of notes about what we didn't cover, why we didn't cover it, and whether you need to look at it more closely yourself.

1 Chapter 5

I didn't go over the "patterns" section explicitly (pages 257 ff.). But you should read it a couple of times.

2 Chapter 6

I am in fact a stickler for use of the **this** in Java. I think this is a major advancement in programming languages, and I happen to think that all references to variables should be done in such a way that the variable's provenance is as clear as possible. One problem that occurs in reading code is trying to find the variables, and clear use of the **this** will distinguish instance variables for a class (used inside a class) from temporary variables for a method.

I am also a stickler for the prolific use of the **get** and **set** constructs. This is because I have all too often seen instances in which code segments originally written in one module get ripped out to be placed in another

code module. If one uses the **get** and **set** even inside the class in which the instance variables are declared, then this can be done without a code rewrite.

I did not go over (except for a brief mention) the use of the **final** for constants. I will, however, be using this a fair bit, and you should learn this.

In my own code, I tend to use this kind of constant in two ways. Most of my constants tend to accumulate in a separate “header” file, so in my code these constants would be put into same class into which I put the methods to open **Scanner** and **PrintWriter** instances. (I called this class **Utilities** in one of my examples, but in my own real code I usually would call this **DAB** with my initials so as to minimize the need to worry about overriding or extending some other class that happened to also be called **Utilities**.)

The other point at which I tend to use constants is in a routine that formats output for display or printing. Often there is a place for constants for sizing things properly.

I did not go over (except for a brief mention) the use of a true *class* variable, which is something you get with the **static** keyword. At the moment, except for **static** methods like **ScannerOpen**, I can’t see that we will be using this much in this class. You should therefore be aware of this, but perhaps not concentrate on mastering its use at this point.

Chapter 6 also mentions debuggers. Eclipse itself is a reasonably good debugging environment.

I also didn't explicitly go over the patterns mentioned at the end of Chapter 6.

3 Chapter 7

With reference to the debuggers from Chapter 6, and with reference to the **Tester** stuff: This is something that Becker wrote for his book. Although it is a neat feature, it is not something that is portable either to other Java environments or to other projects you'll be working on, unless you carry over this part of Becker's library explicitly.

My opinion is that you should be learning things that are generally applicable and portable, and for my money, getting into good programming and debugging habits using Eclipse will be much more valuable and useful to you than learning Becker's **Tester**. Eclipse has about a 40% market share for programming environments, so it is not something one can call "obscure." (Visual Studio has about an equal share.) Life is short: you should be keeping your options open rather than boxing yourself into a corner by using a tool you will have to exert yourself to use everywhere.

I have not covered the use of multiple **main** methods for debugging. I can see the use for such things, but I won't be emphasizing this in this course. Using a **main** method in this way allows one to test the code inside the class. It does not, however, allow one to test the code inside the class in exactly the same way in

which the code will be used when invoked from another class. If I were writing code, I would probably (unless directed otherwise by project or company requirements) write a testing method inside the class (but not call it **main**) and then invoke that method from the usual invoking method. That way, the test is performed in context of the larger program, rather than as a stand-alone.

I didn't cover the **NumberFormat** stuff. Except for something that prints money efficiently, I think the **printf** is maybe just as good if not better.

I didn't cover the use of **toString**, but this is something you are likely to see later in this course, so read about it.

Similarly, I didn't cover the use of enumerated data types, but I will be using them in example programs, so you should read about them.

Similarly, I didn't explicitly cover the **Math** class, but I have used the **random** method in the past, and we will be using these in the future. Put a bookmark in your brain that will enable you to find these methods later.