ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 7 — January 21, 2013

Patrick Lam *version 1*

Real-time Systems

In a *real-time system*, the software needs to respond to an external event in a fixed amount of time. Note that this fixed amount of time is not necessarily small; it just has to be fixed, and may potentially be fixed and large. Many embedded systems must satisfy real-time constraints.

Example.

In upper-year courses, you'll see both embedded systems and real-time systems in more detail.

Variable Names

Choosing good variable names takes taste, which you can develop.

- If a variable has a short lifetime, like a loop variable, then a short name is good. It's totally fine to write `for (int i = 0; i < 5; i++)`.
- If a variable is visible to a whole class, you want a more descriptive name. In Assignment 2, you may choose to put the `Runnable` at class level. That's fine. But then it should have a more descriptive name, like `beeperRunnable`, since any method in the class could refer to it, and the reader of that method should be able to know what the variable does.

Integrated Devlopment Environments

IDEs combine a number of tools in a single environment to improve programmer productivity. These tools include an editor, a compiler, and a debugger. The IDE concept has been around for over 25 years.

We'll use a modern IDE in this class, Eclipse. Eclipse is free software: you can download the code and modify it yourself. Also, it was initially developed by IBM in Ottawa.

Beyond the three core tools, IDEs can also contain support for collaboration (revision control systems, which we'll discuss); documentation and modelling, e.g. UML diagrams; and extensible programmer tools like autocomplete and refactoring. I'll talk a bit about autocomplete today.

What are some examples of IDEs that you know about?

Templates. IDEs often allow you to start your project from a template, which is easier than starting from scratch. You've used the Android Application Project template. Some other useful templates:

- Android Test Project (you'll use these);
- Java Project
- Java Class
- Java Interface

Project Development Workflow. I expect that you now have experience with using Eclipse, both in Lab 1 and Assignment 2. I'll just recap the steps here.

0. Figure out what you'll need to do.
1. Start a new project from a template or, more realistically, check it out from a version control repository.
2. Make the edits that you need.
3. Test your edits by running the application.
4. Debug your edits.
5. Commit your files to the version control repository.

Content Assist. Even if you type really fast, Eclipse can help you enter code more quickly, with the Content Assist helper. Start typing a name and hit Ctrl-Space. Eclipse will give you a choice of names to choose from, and you can choose the correct one. For further reading, see:

`http://wiki.eclipse.org/FAQ_How_can_Content_Assist_make_me_the_fastest_coder_ever%3F`

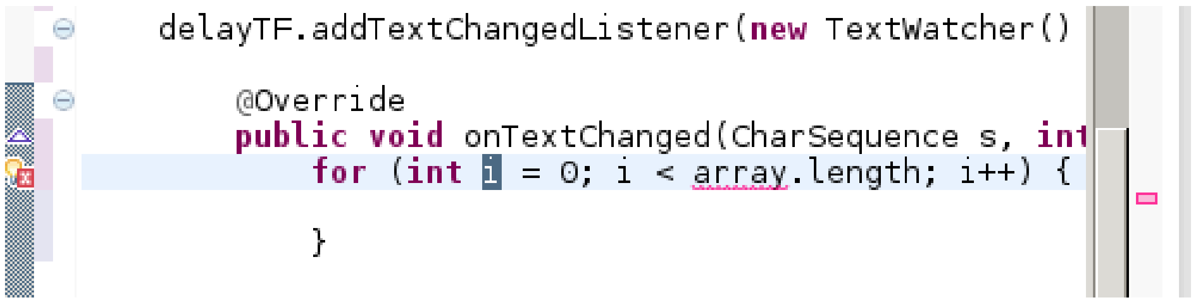
Content Assist can also help you with filling in templates. You can type "for" into Eclipse, hit Ctrl-Space, and choose the type of for loop you're trying to write. Choosing "for — iterate over array" inserts the template:

```
for (int i = 0; i < array.length; i++) {

}
```

and puts your cursor at the "i" so that you can choose your own index variable. Of course, you still have to figure out what your code needs to do.

Quick Fix. The other really useful Eclipse feature is Quick Fix. When you get a red squiggle underneath your code, there's an error. If the squiggle's corresponding editor marker bar (on the left) contains a lightbulb, you can Quick Fix it.



```
    delayTF.addTextChangedListener(new TextWatcher()

        @Override
        public void onTextChanged(CharSequence s, int
            for (int i = 0; i < array.length; i++) {

            }
```

Put the cursor at or near the squiggle, hit Ctrl-1, and Eclipse will propose some fixes. They may be correct (or not).

You can read more about Quick Fixes in the Eclipse help:

Help >Help Contents >Java Development User Guide >Concepts >Quick Fix

In-Class Demo. I'll show you Eclipse usage in class.

About Bugs

You may have noticed that sometimes software doesn't quite do what you want it to do. We call this a *bug*. We'll discuss some content from [Zel09] (highly recommended), in particular ways to remove bugs from programs.

Three things have to happen before you can observe a bug:

1. A programmer puts a defect in the code—some code doesn't do what it's supposed to do.
2. This defect sets some program state (e.g. a variable) to an incorrect, or "infected" value.
3. The infected value has to propagate to program output to cause an observable failure.

Here is a high-level overview of the debugging process.

- Identify the bug and steps to reproduce it.
- Figure out the cause of the bug.
- Correct the bug.

Next, we'll talk about a general strategy for debugging, and explain some techniques (tactics) to help diagnose and fix problems. Eclipse can help, but the real action is inside your head.

References

[Zel09] Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging, Second Edition.* Morgan Kaufmann Publishers, 2009.