Debugging and Eclipse Features

The goal of this recitation is to teach you some of Eclipse's useful tools and shortcuts. We will begin by examining some basic features, and we will end with an exercise using the Eclipse debugger.

Eclipse Features

Eclipse has some extremely useful tools that can save you time and generate common Java boilerplate code.

1. **Code completion**

   The following is a short (certainly not exhaustive) list of useful shortcuts that you can use for code completion and error-fixing:

   ```
   Ctrl + 2 L (Cmd + 2 L) -- Assign to local variable
   Ctrl + 2 F (Cmd + 2 F) -- Assign to field
   Ctrl + 1 -- Open quick fix menu
   Ctrl + Space -- Content assist
   Code completion -- Tab then enter to accept suggestion
   ```

   To practice these shortcuts, open the `edu.cmu.cs.cs214.rec04.Container` class and follow these instructions:

   (a) With your cursor in the class body, type `Ctrl + Space`. Then press `Enter`. You've created the default constructor for the class!

   (b) Suppose you want to add a field called `_foo` and a parameter called `foo` that initializes `_foo`. In the constructor type `_foo = foo;`. Notice that there's an error. Go to `_foo`, use `Ctrl + 1`, and choose `Create field _foo`. Do the same for `foo` except make it a parameter.

   (c) Use the code completion tool to avoid typing long method names. For example, to use `System.currentTimeMillis` you can just type `System.cTM` and use the code completion tool to generate the full method name. It also works with camelCase.

   (d) CamelCase code completion works with class names as well. To instantiate a new `SortedIntLinkedList` you can just type `SILL` and then use the `Ctrl + Space` shortcut again.

2. **Renaming classes and variables**

   Manually renaming a variable can be tedious if the variable is pervasive in the code, and using automated tools such as find-and-replace can have unintended and potentially disastrous results. Refactoring is an automated tool to rename classes, methods, and variables without unintended consequences. Follow these instructions inside the `edu.cmu.cs.cs214.rec04.SortedLinkedListTest` class:

   (a) Mike Ehrmantraut called: No more half measures. With your cursor over the `halfMeasure` variable, use the `Alt + Shift + R (Cmd + Alt + R)` shortcut to rename that field to `fullMeasure`.

   (b) Select the `SortedLinkedListTest` class in the package finder and use the same `Alt + Shift + R (Cmd + Alt + R)` shortcut to rename the class to `SortedIntLinkedListTest`.

3. **Extracting interfaces**

   If you want to use a class as the basis for defining an interface, you can use Eclipse's extract interface tool to automatically define an interface based on that class's methods. Suppose you wrote `edu.cmu.cs.cs214.rec04.SortedIntLinkedList` and you decide you also want to provide an unsorted implementation, with both classes implementing the same interface. With the file open, use `Ctrl + Alt + T (Cmd + Option + T)` to open the refactoring menu and choose the "Extract Interface" option. (Which methods are good candidates to extract into an interface?)

4. **Implementing interfaces**

   If you are implementing an interface, Eclipse can automatically generate all the methods for you. With the `edu.cmu.cs.cs214.rec04.ExampleQueue` class, implement the `java.util.Queue` class. Use the `Ctrl + 1` shortcut to fix the errors (and implement all unimplemented methods).

5. **Extending abstract classes**

   Eclipse can similarly generate default methods when you extend an abstract class. With `edu.cmu.cs.cs214.rec04.AbstractListExample`, extend the `java.util.AbstractList` class. Use the `Ctrl + 1` shortcut and fix the errors that appear.

Using the Eclipse Debugger

Many IDEs have a built-in debugger. A debugger is a tool that lets you pause and inspect the properties of a program as it is executing so that you can better understand and debug program behavior. You can observe the statements being executed as well as see data values.

The `edu.cmu.cs.cs214.rec04.SortedIntLinkedList` class contains several bugs. Use the Eclipse Debugger (click on Run → Debug) to identify and fix any errors. On a piece of paper, please describe the errors you found and (briefly) how you fixed them.

You should master at least four main debugger tools: setting breakpoints, stepping into methods, stepping over methods, and inspecting variables.

1. **Setting breakpoints**

   To set a breakpoint, go to the line of code and then press Run → Toggle Breakpoint. You will see a blue dot which indicates that a breakpoint has been set. The program execution will stop at the breakpoint.

   Press the `F8` button to go to the next breakpoint.

2. **Setting conditional breakpoints**

   To set a conditional breakpoint, go to the "Breakpoint" view in the debug perspective. Right-click the breakpoint you want to modify and go to `Breakpoint properties...` From here you can set conditions for which the breakpoint stops execution of your program.

3. **Step into methods**

   This feature advances the execution of the program by one line of code, pausing execution and allowing you to inspect the program state inside a method call if a function call is made. A shortcut to do this in the Eclipse debugger is to press the `F5` button.

4. **Step over methods**

   This feature also advances the execution of the program by one line of code, but runs any method calls to their completion so that you can continue inspecting the state of the currently-running method. A shortcut to do this is to press the `F6` button.

5. **Inspecting variables**

   You can inspect the state of all current variables in the debugger by looking at the top right corner, under the "Variables" section.

6. **Evaluating expressions**

   You can evaluate expressions with the current program state using the "Expressions" view. For example, if you set a breakpoint in the middle of a method and want to determine the return value of another method (maybe `size()`) at that point in time, you can use the "Expressions" view to do this.