UCL DEPARTMENT OF COMPUTER SCIENCE

Wolfgang Emmerich, Dr rer-nat, CEng, MIEE
Professor of Distributed Computing
Director of Research

GS04: Tools and Environments

Lab Session 4: Finding defects with debuggers

The aim of this lab session is for you to gain experience in using the debugging functionality provided by modern interactive development environments. We will use the Debugger that is bundled into Eclipse JDT.

Setup

Create a new Eclipse plug-in project using the model solution of Lab Session 3. This is available from the course web site at http://www.cs.ucl.ac.uk/staff/w.emmerich/GS04-0708/

Breakpoints

Set a number of breakpoints in classes `BeanAnalyzer` and `SampleHandler` and lauch the static analysis plug-in with the Eclipse debugger. Note how you regain control when the main Eclipse thread is suspended as soon as a breakpoint is reached. Observe how you can resume the suspended thread and continue to the next break point.

Watchpoints are special breakpoints for variables. They suspend execution whenever a variable is modified. Set a watchpoint on field result. Then resume your execution and note how you are taken from one statement that modifies result to the next.

Stepping through a program

Sometimes it is necessary to go through a program step by step and observe what effect the program's instruction have on the state of the application. Set a breakpoint in class `SampleHandler` and step into the constructor of `BeanAnalyzer` when a new analyzer instance is created. Step over other methods of the Eclipse API to ensure you follow the program execution through class `BeanAnalyzer`. Then step through the constructor and when you are satisfied step up again to `SampleHandler`.

Variables

Set a method breakpoint in BeanAnalyzer.java at line 95 that suspends execution of the `visit(FieldDeclaration)` method of the anonymous Java visitor. Now resume execution a number of times and review how the HashMap is filled with tuples that correspond to all field declarations of the class that is being analyzed. Then work out where you need to set a breakpoint so that you can review how `methods` is filled. Set that breakpoint and repeat the execution.

Understanding the Call Stack

Use the breakpoints that you have set above and now review the call stack in the debug view that has lead to Line 95 being reached. Note that our anonymous visitor is actually called by the Eclipse JDT class. Try to double click on an item in the call stack and note how you can use that to navigate to different source lines.

Displaying Expressions

Clear all breakpoints you have set so far and set a new breakpoint in line 97 of class BeanAnalyzer and debug the application. Now select the entire string expression that is used as an actual parameter to the error report variable and display the value of that expression. Now disable the breakpoint in line 97 and set a new one in Line 117. Then select the expression on the right hand side of the assignment to toCaml and Watch the expression. Then resume the execution. Note how you can see the value changing in the Expression view.

University College London   Gower Street   London   WC1E 6BT
Tel: +44 (0)20 7679 4413   (Direct Dial)   Fax: +44 (0)20 7387 1397
www.cs.ucl.ac.uk/staff/W.Emmerich
w.emmerich@cs.ucl.ac.uk