

Practical 8: Debugging

Task 1

Please download the NetBeans project called **program_8** from your practicals folder on the KLE.

This is similar to an example that you had accompanying one of your recent lectures.

The program simulates the repeated rolling of two dice until the outcome is a double-six in which case execution stops. After each roll (trial) of the dice the trial number and the outcome is reported (printed) on the console window.

Study the program and run it a few times. Make sure you understand it before you proceed to the next task. If you are having any problems please ask for help from a demonstrator.

Task 2

Part of the Apache NetBeans development environment (and most computer language development environments) is a facility called the *debugger*. The debugger can be used to pause the program execution at programmer-defined points (called *breakpoints*) in order to allow the inspection of the program's **state** at those points. For example, you can pause execution at a specific point to inspect the value of a certain variable at that point.

You can add several breakpoints in your code. When you subsequently run your program (in debug mode) as soon as the first breakpoint is reached the debugger will pause and wait for your next action.

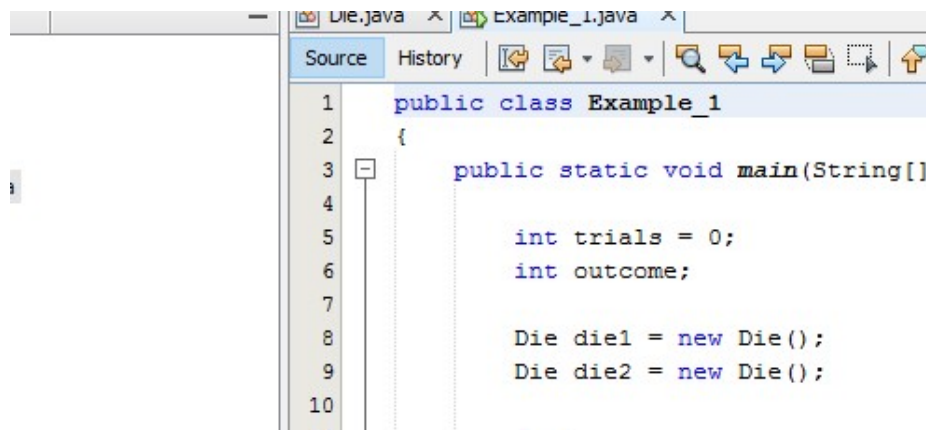
You can do a lot of things while your program is paused. The most common are:

- to inspect the values of variables or the state of objects in your program,
- to step to the next instruction (i.e. execute the next instruction and pause again) or
- to continue until the next breakpoint or to the end of the program if there is no other breakpoint.

Let's use the supplied example program with the Apache NetBeans debugger.

Insert a breakpoint at line 13 of the `main` method in class `Example_1`. The quick way to do this is by pointing and clicking at its line number on the left. The corresponding line of code will be highlighted and a square will replace the line number:

School of Computing and Mathematics
CSC-10024
Practical 8 (Week 5)



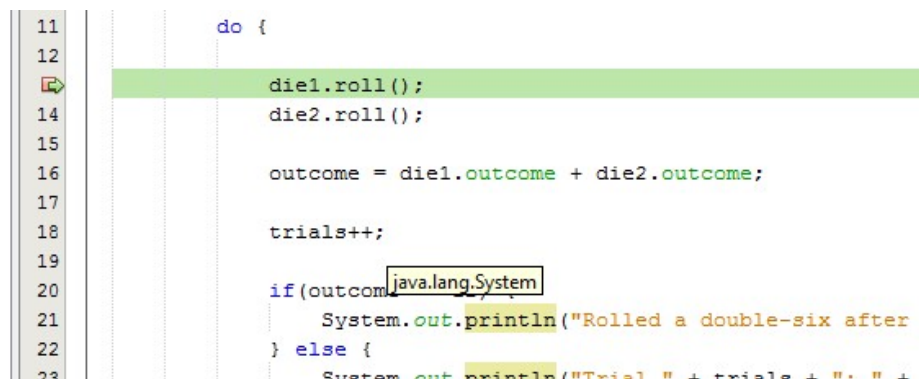
```
1 public class Example_1
2 {
3     public static void main(String[] args)
4     {
5         int trials = 0;
6         int outcome;
7
8         Die die1 = new Die();
9         Die die2 = new Die();
10    }
```

Now run the program with the debugger. To do this you need to select “Debug File” instead of “Run File” when you right-click on the class name.

The program will start running but will pause at line 13. Note that at that point line 13 will not have been executed yet.

While the program is paused, hover over variable names with the mouse pointer in order to inspect their value.

You can also see all variables and their values if you bring the “Variables” panel in focus (by clicking on it) at the bottom of your NetBeans window:

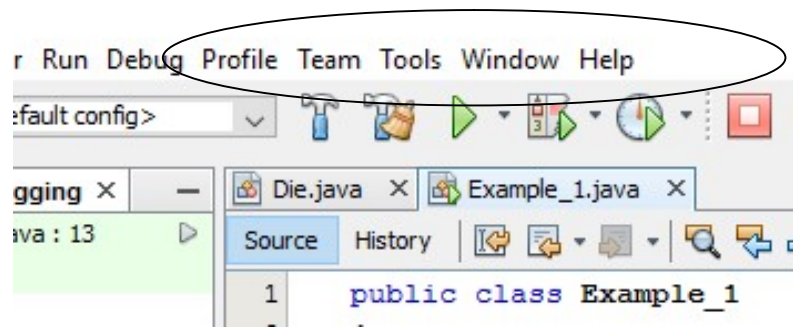


```
11 do {
12
13     die1.roll();
14     die2.roll();
15
16     outcome = die1.outcome + die2.outcome;
17
18     trials++;
19
20     if (outcome == 12) {
21         System.out.println("Rolled a double-six after " + trials + " trials");
22     } else {
23         System.out.println("Trial " + trials + " failed");
24     }
25 }
```

Don’t forget to bring the “Output” panel (i.e. the console) back into focus if you want to see the output of your program again.

While in debug mode, the debug toolbar (circled below) will be visible at the top of your NetBeans window:

School of Computing and Mathematics
CSC-10024
Practical 8 (Week 5)



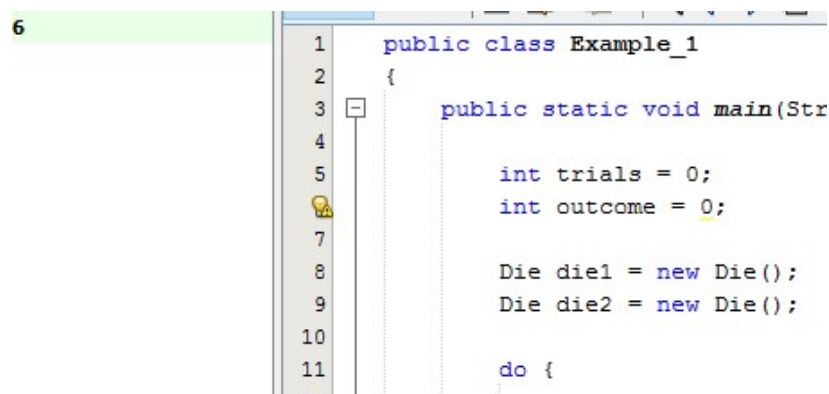
This provides a shortcut for the most common actions that you need while debugging your code. Hover over each button in the toolbar to find out their label/function.

Click a few times (2-3) on *Continue* and provided that your program hasn't ended¹ you should still be in debug mode and your program will be suspended on line 13. What happens each time you click *Continue* is that execution carries on until the next breakpoint is met. In this case there is only the one breakpoint at line 13 and it is met repeatedly because of the (do-while) loop in the program.

Again, hover over variables with the mouse to investigate their value. Are the values what you expect them to be?

Now instead of clicking *Continue* click on *Step Over*. When you do this, the current line (13) is executed and execution pauses on the next line. You can now reinvestigate the state of your program to see the effect of the line you just stepped over (i.e. 13).

You can carry on stepping over one line at a time.



The line you pause on will be highlighted as green and it will have a right-pointing arrow instead of its number as shown above.

As you step through your program observe how the `if` statement is negotiated. Another useful function of the debugger is that it reveals the course your program takes, i.e. which instructions are met and which are omitted.

¹ Your program will end if you roll a double-six. If you have, then re-start it in debug mode.

If you want to stop debugging your program you need to click the toolbar button labelled *Finish Debugger Session*.

Task 3

Debug your program again but this time insert more than one breakpoint in order to understand better how the *Continue* button hops from one breakpoint to the next.

Notice that you can toggle (switch) breakpoints on and off during debug time.

You can also create a more temporary breakpoint by placing your editing cursor at any point in your code and then clicking *Run to Cursor* to run your program up to that point.

Task 4

Notice that up to now, lines 13 and 14 were executed in one go when you were stepping over them during debug mode. These lines however are *method calls* and represent several lines of code defined in the `Die` class.

If you want the debugger to go through and pause at each line of the method being called then you need to click the *Step Into* button when you are on the method call line (such as line 13 or 14) instead of the *Step Over* button.

You can go as deep as you like with the *Step Into* button, even into code written in Java classes such as the `Math` class. You can experience this for example if you step into the method call to `Math.random()` in line 7 of the `Die` class.

If you are going through the statements of a particular method and you want to proceed quickly to the point where the method completes execution then you need to click the *Step Out* button on the debugger toolbar.

Try experiencing the above.

The debugger is an extremely useful utility that can save you hours of frustrating error tracking time. Spending time to familiarise yourself with it now will benefit you immensely when you come to create larger and more complex programs.