Java

Debugging

FSR Informatik

October 11, 2016

Overview

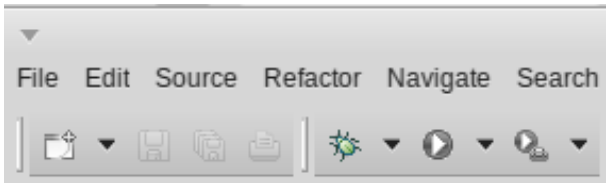
A debugger helps the programmer to trace errors in their code.

Normally a program executes every line of code and stops afterwards. With a debugger you can watch every state of the execution.
The inspection of a specific state can help to find errors.

Debug Perspective

Eclipse offers multiple perspectives for multiple tasks.

- The *Java perspective* is standard while programming.
- The *Debug perspective* can be used for debugging.



The bug icon starts the debugger. You can also press F11.

You can change the perspective via: Window > Open Perspective > ... or press CTRL + F8.

Breakpoint

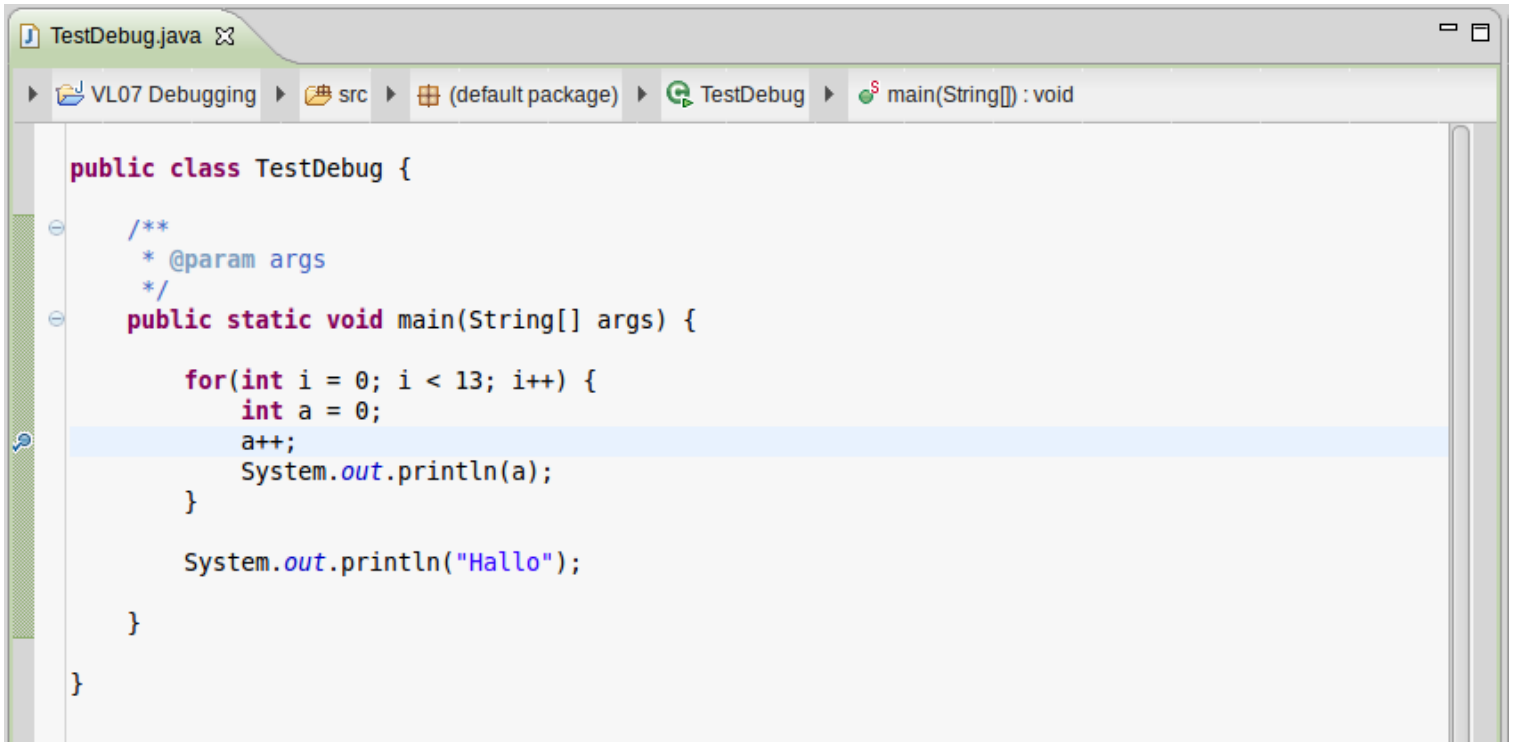
A breakpoint marks a specific line in the code.

While debugging the excecution of the program will stop before every marked line. At this point you see every current variable and object with their current values.

The debugger steps through the code breakpoint by breakpoint.

Toggle Breakpoints

With Shift + Ctrl + B you can add or remove a breakpoint to the current line.



Breakpoint at the line: `a++;`

Stepping through the Code - 1

If your code contains breakpoints Eclipse will jump to the *Debug Perspective* automatically when you start the debugger.

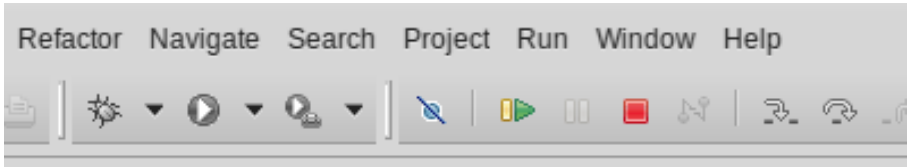
```
1   public class TestDebug {
2
3       public static void main(String[] args) {
4
5           for(int i = 0; i < 13; i++) {
6               int a = 0;
7               a++;
8               System.out.println(a);
9           }
10      }
11  }
12
```

Assume breakpoints in line 6, 7 and 8. Start the debugger.

Stepping through the Code - 2

The table with all current variables is in the upper right corner. The view is called *Variables*.

You can step through the code with F8 or using the green arrow symbol. The red square or Ctrl + F2 will end the debugging.



While stepping through the code variables appear and change their values. Current changes of the values will be marked yellow.

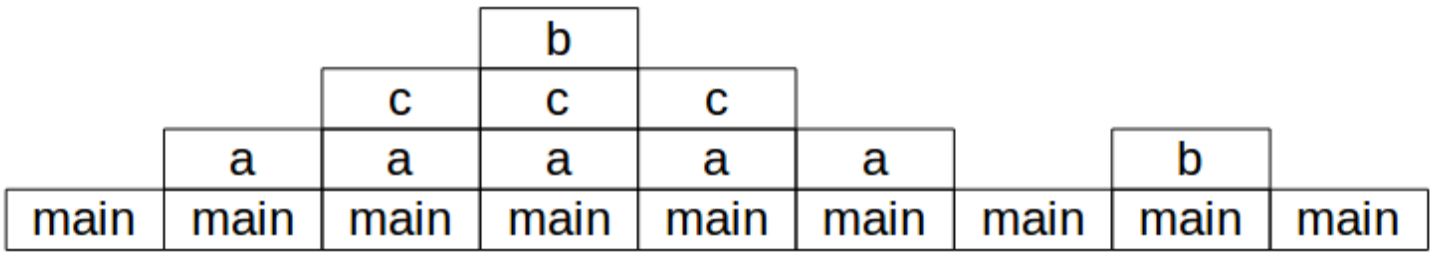
Stack

The stack trace shows all methods a program called at a certain point.

Every method call increases the stack.
Every return and end of a void-method decreases the stack.

Development over Time

```
1       public static void a() {
2           c();
3       }
4       public static void b() {}
5       public static void c() {
6           b();
7       }
8       public static void main (String[] args) {
9           a();
10          b();
11      }
12
```



Printed Stack Trace

```
3       public static void a() {
4           c();
5       }
6       public static void b() {
7           throw new RuntimeException();
8       }
9       public static void c() {
10          b();
11      }
12      public static void main (String[] args) {
13          a();
14          b();
15      }
16
```

```
Exception in thread "main" java.lang.RuntimeException
at TestStack.b(TestStack.java:8)
at TestStack.c(TestStack.java:12)
at TestStack.a(TestStack.java:4)
at TestStack.main(TestStack.java:16)
```

Debug Perspective

The view *Debug* shows the current stack.
It is located in the upper left corner of the debug perspective.