A Simple Introduction to Eclipse Debugging Features (Java)

Toby Qin

General Idea

The time spent on debugging is usually way more than that on writing the program. Sometimes when people are not familiar with the debugging features, usually the printing statement would be used as the "tool" to look at what the program is doing. Although it is true that this technique is sometimes efficient, it is not when the program is complex. Therefore, we need to utilize the debugger.

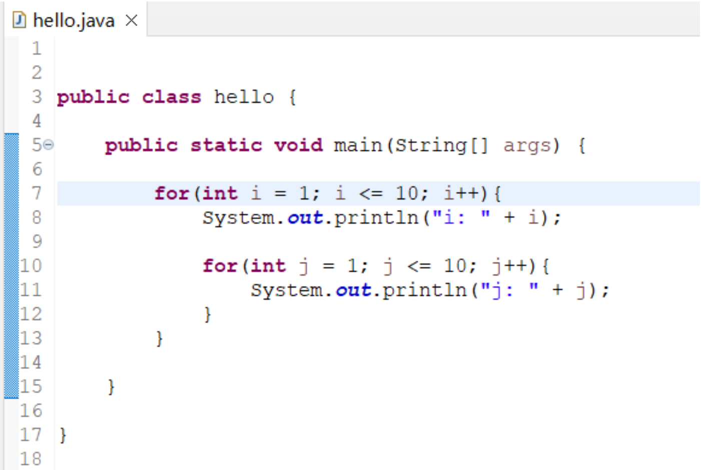
Debug Perspective

To open the debug perspective, simply click the of , which is located on the right top corner of the Eclipse window.

If the Eclipse doesn't have this button, clock and click the debug tab. Then the button will be added.

Debugger: How to Start it

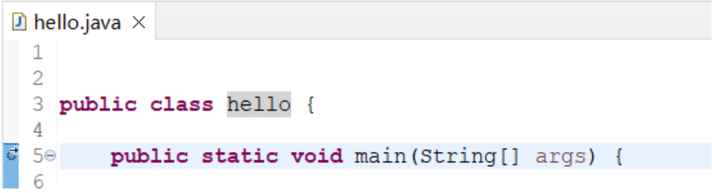
To demo the debugger, here is a simple program that has a outer loop and inner loop, and they all print out the value of integer i and j.



hello.java

```
 1
 2
 3 public class hello {
 4
 5⊖    public static void main(String[] args) {
 6
 7        for(int i = 1; i <= 10; i++){
 8            System.out.println("i: " + i);
 9
10            for(int j = 1; j <= 10; j++){
11                System.out.println("j: " + j);
12            }
13        }
14
15    }
16
17 }
18
```

Firstly, let's try to set a breakpoint at the beginning of the main at line 5. So, double click the margin to the left of the line numbers at line 5. We will see this:

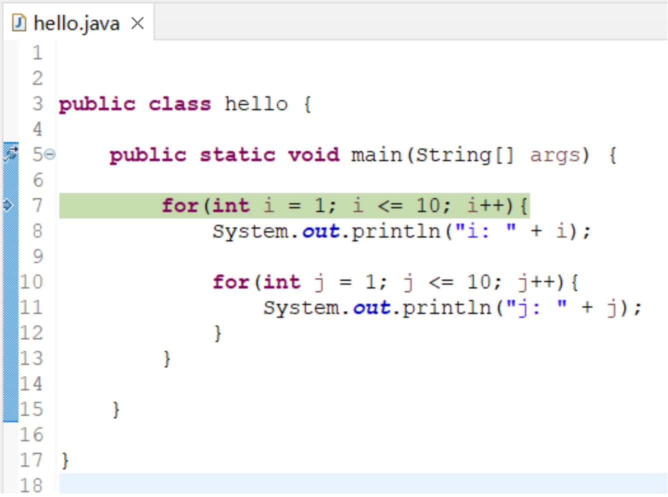


hello.java

```
 1
 2
 3 public class hello {
 4
 5⊖    public static void main(String[] args) {
 6
```

Then, from the click the debug button of , which is on the top of the window.

So, the window would appear like this:

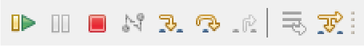


```
 1
 2
 3 public class hello {
 4
 5     public static void main(String[] args) {
 6
 7         for(int i = 1; i <= 10; i++){
 8             System.out.println("i: " + i);
 9
10             for(int j = 1; j <= 10; j++){
11                 System.out.println("j: " + j);
12             }
13         }
14
15     }
16
17 }
18
```

In this case, the line of code marked in green means the line that has not been run. Therefore, it means it has been run when line 8 is marked as green.

Debugger: Tools

Now we may see that in the debugging perspective, there are these buttons on the top of the window



From left to right:

Resume: continue running the program to the end unless a breakpoint stops the program.

Terminate: terminate the whole debugging process.

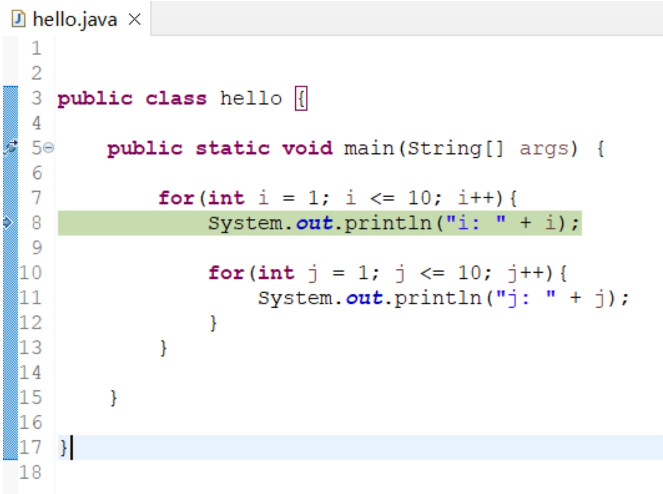
Step Over: run the current line marked in green and go to the next line (but not yet to run).

Step Into: goes to the insider of the code (such as the body of a function, or a pre-written feature, etc.)

Step Return: the opposite of Step Into, goes to the outside of the function.

Debugger: Continue Debugging

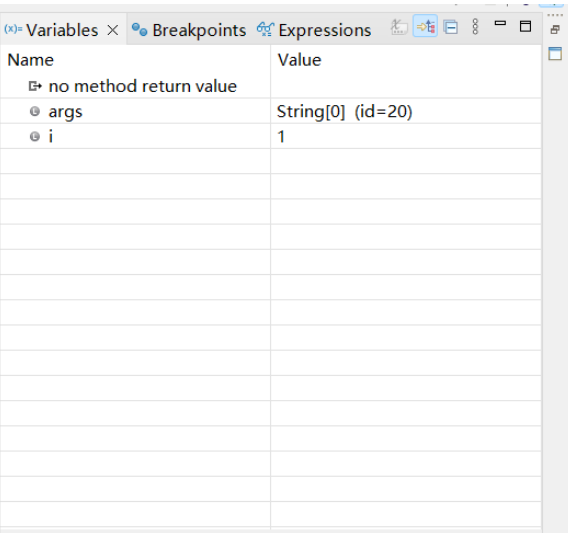
With the understanding of the debugging tools, we can continue the debugging learning process. Next step is to click the Step Over button, we will see:



```
 1
 2
 3 public class hello {
 4
 5     public static void main(String[] args) {
 6
 7         for(int i = 1; i <= 10; i++){
 8             System.out.println("i: " + i);
 9
10             for(int j = 1; j <= 10; j++){
11                 System.out.println("j: " + j);
12             }
13         }
14
15     }
16
17 }
18
```

So, the program finished running line 7 and it is waiting and ready to run line 8.

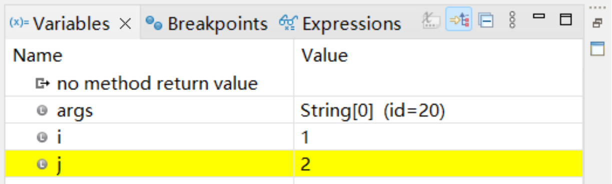
To the right of the window, we may see a Variables tab:



Name	Value
no method return value	
args	String[0] (id=20)
i	1

The Variables is the place to show all the variables and their current values stored in the variables. In this case, integer i has just been initialized to 1. Therefore, we see i's value is 1.

Click Step Over for 4 times (let the inner loop to finish running twice). We can see the Variables as this:



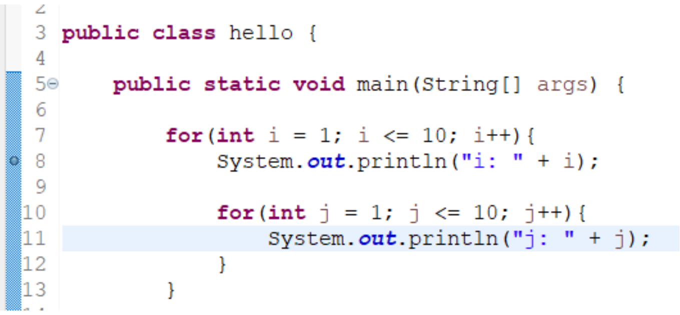
Name	Value
no method return value	
args	String[0] (id=20)
i	1
j	2

That means if the value of a variable has been changed, it will mark in yellow. That will give a eye-catching reminder that the value has been changed.

Therefore, this is the alternative way to debug and see the values in the variables at different lines instead of printing out the value. But now the question is, what if the program is complex and takes a long time to go through each line one by one? To know the answer, we will look at the breakpoints in detail.

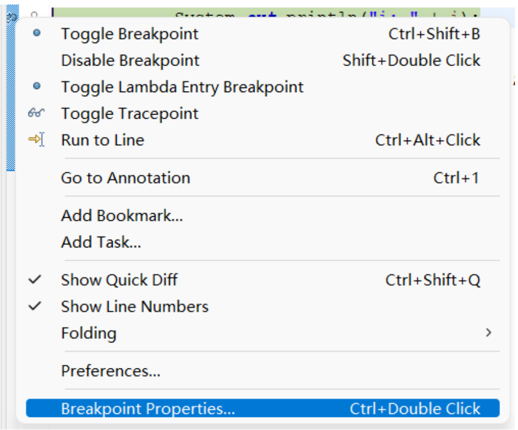
Debugger: More about the Breakpoints

Let's say, in this case, we want to see what the program is doing when i=9 but we don't want to run the debugger line by line until i=9. To do this, we will double click our previous breakpoint at line 5 and add a breakpoint at line 8 (because, again, line 8 means the program has finished running line 7, which means the value of i has been changed to be up-to-dated).

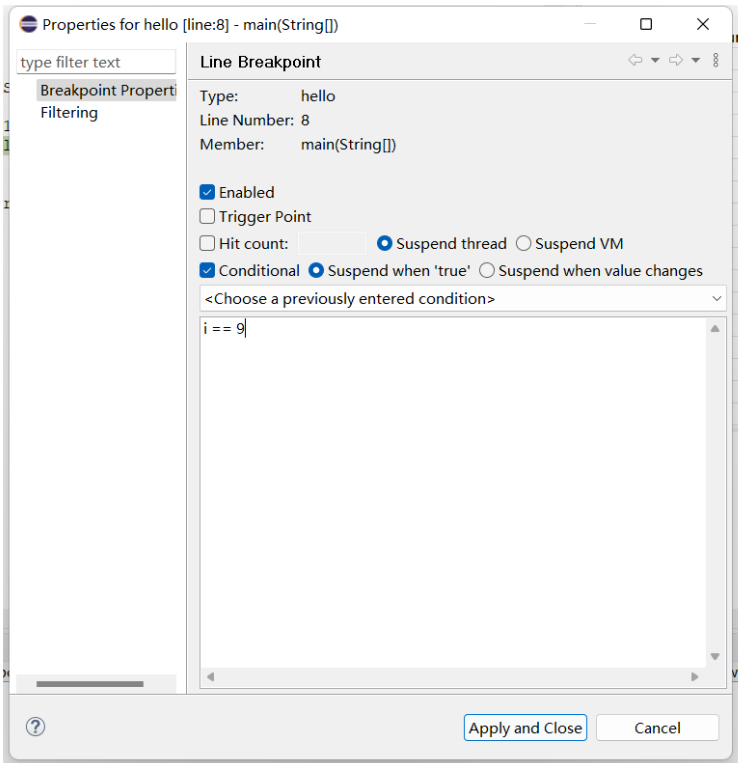


```
 3 public class hello {
 4
 5     public static void main(String[] args) {
 6
 7         for(int i = 1; i <= 10; i++){
 8             System.out.println("i: " + i);
 9
10             for(int j = 1; j <= 10; j++){
11                 System.out.println("j: " + j);
12             }
13         }
```

And then, right click the breakpoint and go to Breakpoint Properties.


Toggle Breakpoint Ctrl+Shift+B
Disable Breakpoint Shift+Double Click
Toggle Lambda Entry Breakpoint
Toggle Tracepoint
Run to Line Ctrl+Alt+Click
Go to Annotation Ctrl+1
Add Bookmark...
Add Task...
✓ Show Quick Diff Ctrl+Shift+Q
✓ Show Line Numbers
Folding ›
Preferences...
Breakpoint Properties... Ctrl+Double Click

Select "Conditional", and put "i == 9" in the text entry.


Properties for hello [line:8] - main(String[])

Line Breakpoint

Type: hello
Line Number: 8
Member: main(String[])

☑ Enabled
□ Trigger Point
□ Hit count: ● Suspend thread ○ Suspend VM
☑ Conditional ● Suspend when 'true' ○ Suspend when value changes

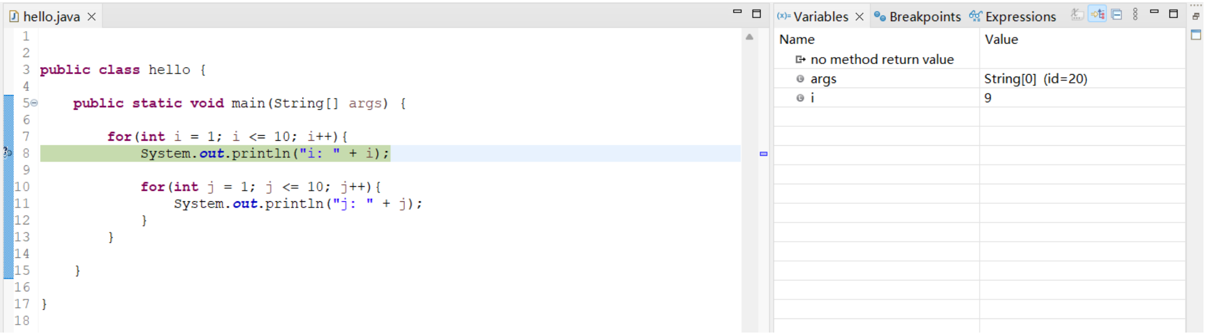
<Choose a previously entered condition>

```
i == 9
```

The text entry box is where we set up the condition that the breakpoint should hold the debugging process. In this case, it means that the debugging process would stop when i has a value of 9.

Inside the text entry box, it should be the condition just like we write the if statement, but without "if", "else if", or "else". Also, feel free to use "&&", or "||" to express logical relations.

Before running the new debugger, we will click ■ to terminate the previous debugger. And then, again click ✻ to run the new debugger. Then, click resume ▶ button, we will see the program stop when i=9.



```
public class hello {

    public static void main(String[] args) {

        for(int i = 1; i <= 10; i++){
            System.out.println("i: " + i);

            for(int j = 1; j <= 10; j++){
                System.out.println("j: " + j);
            }
        }

    }

}
```

Name	Value
no method return value	
args	String[0] (id=20)
i	9

Now, feel free to use Step Over to debug specific lines of code or variables.

Conclusion

So here is the basic use of debugger. Using these techniques, we can solve many problems instead of printing out the variables. And you can also speed up the debugging process by skipping unrelated steps and slow down to where you want to debug.

There are many more related features in debugger, and they are also super helpful, such as Expressions. Please feel free to play around and search for documents if you are interested.