Tutorial: IDE Debugging

IDE Debugging Overview

In this tutorial you will debug a simple program using the Intellij IDE debugger.

You will learn

- How to use breakpoints strategically
- How to use conditions within breakpoints
- How much slower programs run with conditional breakpoints active
- How to set a watch with an expression to be evaluated

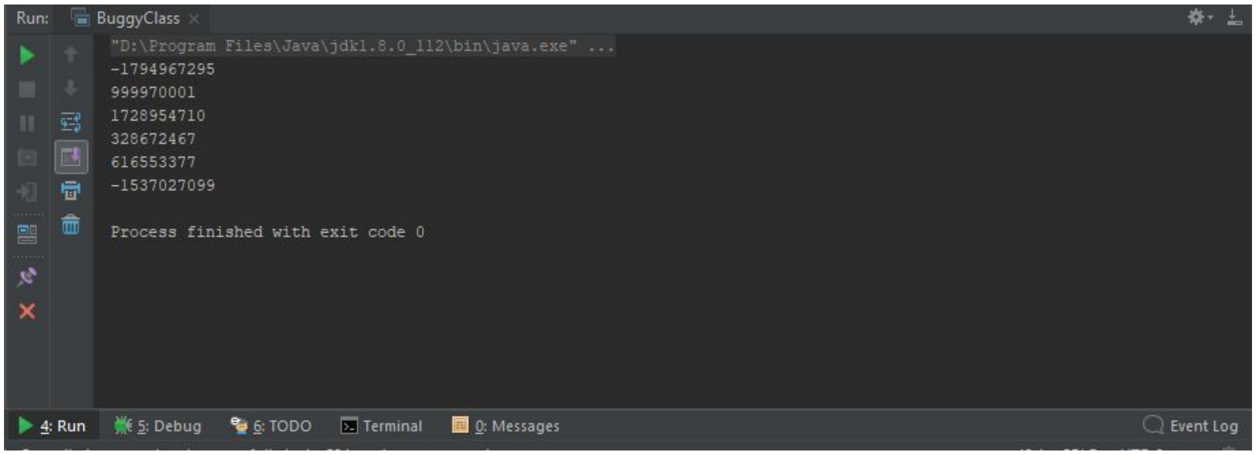
Setup

You will need to download the BuggyClass.java file from the course website. Open it in IntelliJ. Do not change anything in it yet.

Debugging

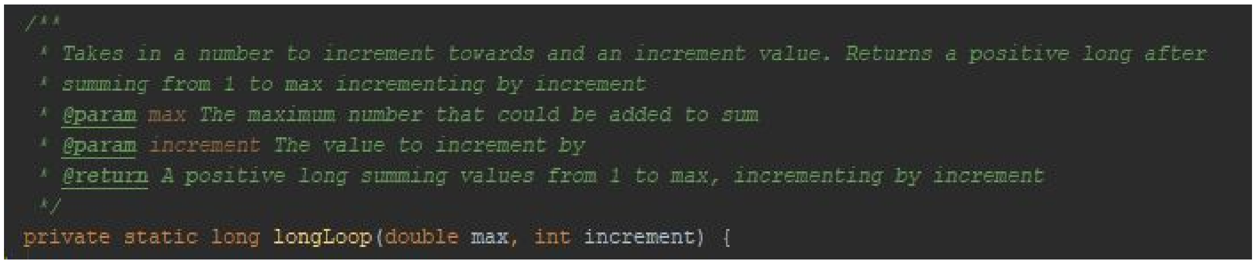
First, let's just run the class and see what the output is. This way we can know what effect our changes are having on the program.

1. Your output should look like this:



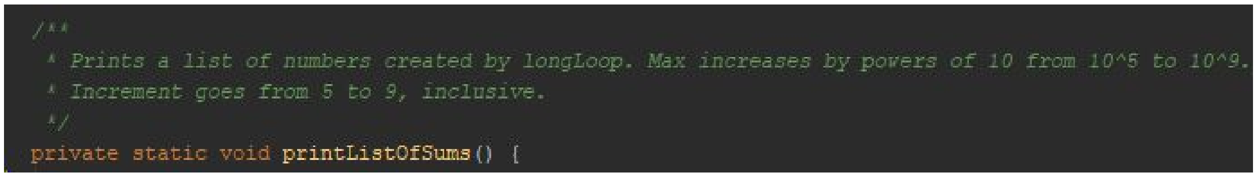
```
"D:\Program Files\Java\jdk1.8.0_112\bin\java.exe" ...
-1794967295
999970001
1728954710
328672467
616553377
-1537027099

Process finished with exit code 0
```

2. Now let’s take a look at what the program is doing and compare that with what it should be doing. Take a look at the documentation for the longLoop method. It should look like this:



```
/**
 * Takes in a number to increment towards and an increment value. Returns a positive long after
 * summing from 1 to max incrementing by increment
 * @param max The maximum number that could be added to sum
 * @param increment The value to increment by
 * @return A positive long summing values from 1 to max, incrementing by increment
 */
private static long longLoop(double max, int increment) {
```

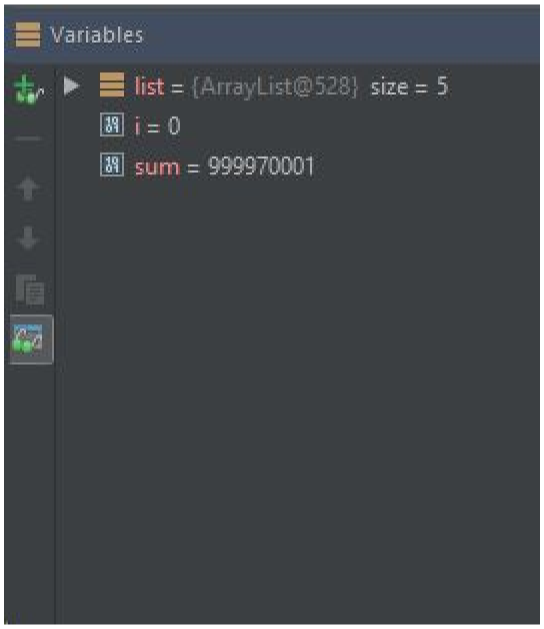
3. Note how it should return a positive long. If we look back to the outputted numbers, some of them were negative. The implementation does not match the documentation, so this method is buggy and incorrect. However we need to know more about how it’s being used to know what is causing incorrect output since some of the printed numbers were positive. Now let’s look at printListOfSums. Its documentation should look like this:



```
/**
 * Prints a list of numbers created by longLoop. Max increases by powers of 10 from 10^5 to 10^9.
 * Increment goes from 5 to 9, inclusive.
 */
private static void printListOfSums() {
```

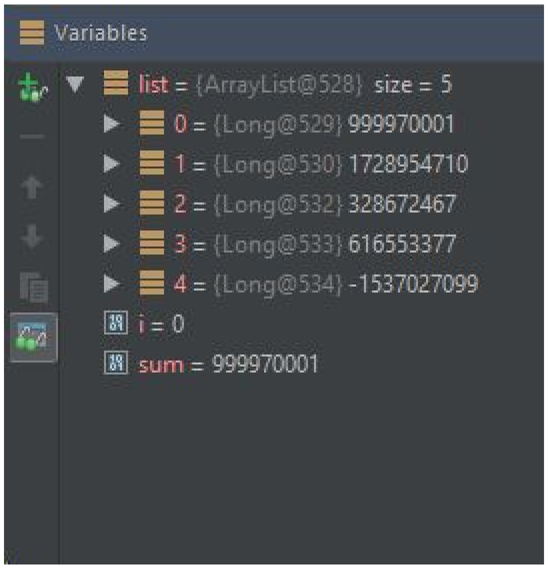
4. It is probably behaving correctly, but let’s double check to see if it can help us fix longLoop. Put a breakpoint on line 41, and run the program in debug mode.

5. Once the breakpoint is hit, take a look at the variables. You should see something like this:



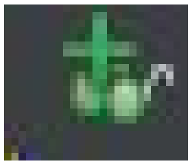
```
Variables
▶ list = {ArrayList@528} size = 5
  i = 0
  sum = 999970001
```

6. Now expand the list variable so we can see its elements by clicking on the arrow next to it. You should see this:



```
list = {ArrayList@528}  size = 5
    0 = {Long@529} 999970001
    1 = {Long@530} 1728954710
    2 = {Long@532} 328672467
    3 = {Long@533} 616553377
    4 = {Long@534} -1537027099
i = 0
sum = 999970001
```

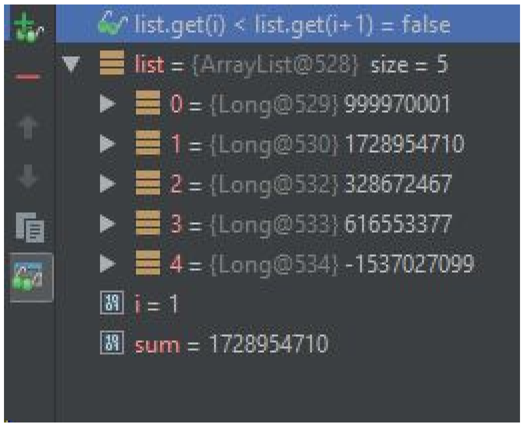
7. We can see that we get one negative number which is not good. Because we are summing numbers inside the longer loop with larger numbers on each iteration, we should expect each element in the list to be larger than the previous element. Let's double check that each variable is smaller than the next one to see if anything weird is going on. To do this, we'll set what is called a watch. To do that, click on the green glasses with the plus sign above them.



8. Type this expression: list.get(i) < list.get(i+1)

9. Hit Enter. This creates a watch that will tell you if the current list element is smaller than the next list element. Now let's step through this loop and see what happens. After

stepping over each statement until the loop repeats, we see something troubling.



```
list.get(i) < list.get(i+1) = false
▼ list = {ArrayList@528} size = 5
    ▶ 0 = {Long@529} 999970001
    ▶ 1 = {Long@530} 1728954710
    ▶ 2 = {Long@532} 328672467
    ▶ 3 = {Long@533} 616553377
    ▶ 4 = {Long@534} -1537027099
  i = 1
  sum = 1728954710
```

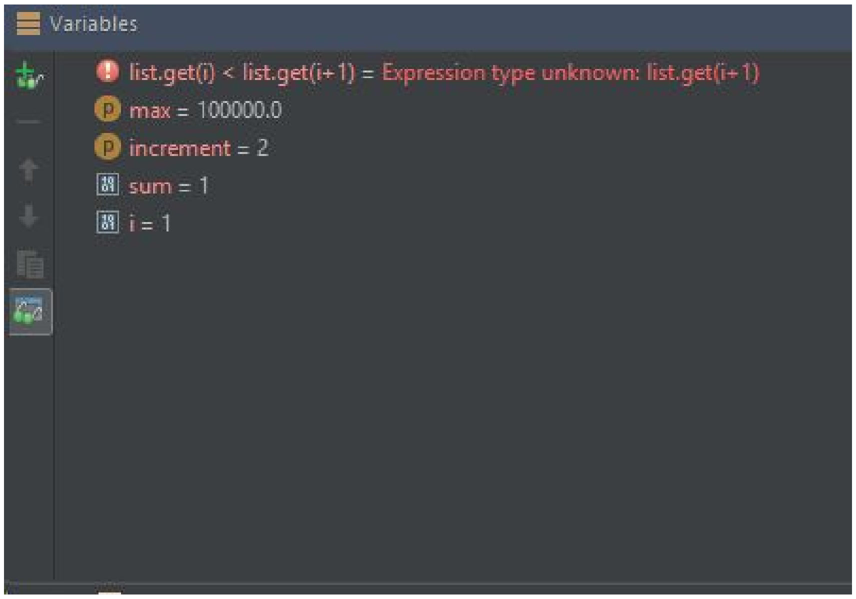
10. The list element is bigger than the next one. Each element should be smaller than the next because we're summing inside of longLoop with bigger numbers each time. If we keep stepping through the loop, we find that there are a couple more times our watch evaluates to false.
    a. Don't worry about the error you get when i == list.size(), we're just out of list items to compare with. It doesn't harm the execution of our program, but we want to leave it there so when we make a change we can see if we're getting what we expect in printListOfSums. One thing to note with watches is you can assign values to in scope variables, and this will change the execution of your program.

Conditional Breakpoints

Now we know something is causing our longLoop calls to make numbers smaller than they should be. Let's step through longLoop to see what happens as we go through.

1. Set a breakpoint on line 24 and run the program in debug mode.

2. If you look at the variables, we've got an interesting problem.



Variables

```
list.get(i) < list.get(i+1) = Expression type unknown: list.get(i+1)
max = 100000.0
increment = 2
sum = 1
i = 1
```

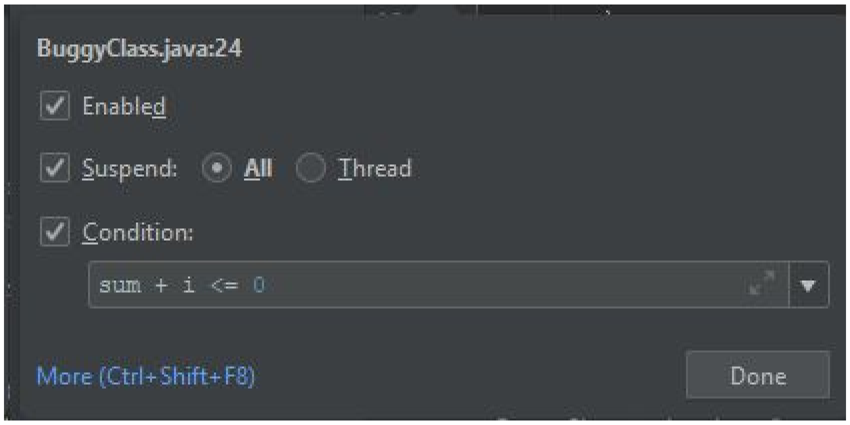
   a. The variable max is 100,000. Because we increment by 2, that leaves us with about 50,000 loop iterations where something could go wrong. Since we don't want to step through 50,000 loop iterations, we need to have a different approach to debug this method.

3. Right click on the breakpoint you created on line 24.

4. Check the box next to Condition.

5. Put the following expression in the text field: sum + i <= 0

6. This lets us know if we're about to get a negative number when we add i to sum. You should see a dialog window like this now:



BuggyClass.java:24

[✓] Enabled

[✓] Suspend: [● All] ○ Thread

[✓] Condition:

```
sum + i <= 0
```

7. You might be thinking "This is crazy, we're adding a positive number to a positive number, how are we supposed to get a negative number?" You're right, in math that is

crazy, but in computer science we are crazy and this can happen. It’s called overflow. In programming languages, primitive data types (think ints, doubles, and other number types) have maximum and minimum values. Well, when you try to increase a primitive number past it’s maximum value it will wrap around to the minimum value (or vice versa for decreasing the number below the minimum value, this is called underflow). This happens because of how the numbers are represented in binary and how the arithmetic happens in binary. You’ll learn more about this in CS 224. That is probably happening in longLoop, but we should check for that. This breakpoint condition will check for that.

8. Now click the Done button, and continue executing your program by hitting the Resume Program button on the left or by hitting F9. Note how much more slowly your program executes now with the conditional breakpoint. This is because it is computationally expensive to insert this condition at runtime.

9. After it runs for a while, possibly minutes, it should break when i = 92,681 and sum = 2,147,396,601.

10. Just to double check that our condition is working the way we think it should, create another watch with this expression: sum + i

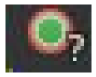
11. We do get a large negative number. This means we are overflowing sum. To fix this we just need to make sum a data type with a larger maximum value. If we look at the return type for longLoop, it is returning a long. However sum is an int, so we can change it to a long and see if that fixes the problem. Make that change.

Bug Fixed

1. Rerun the program, either by hitting Stop (ctrl + F2) and Rerun (ctrl + F5) or just hitting Rerun.

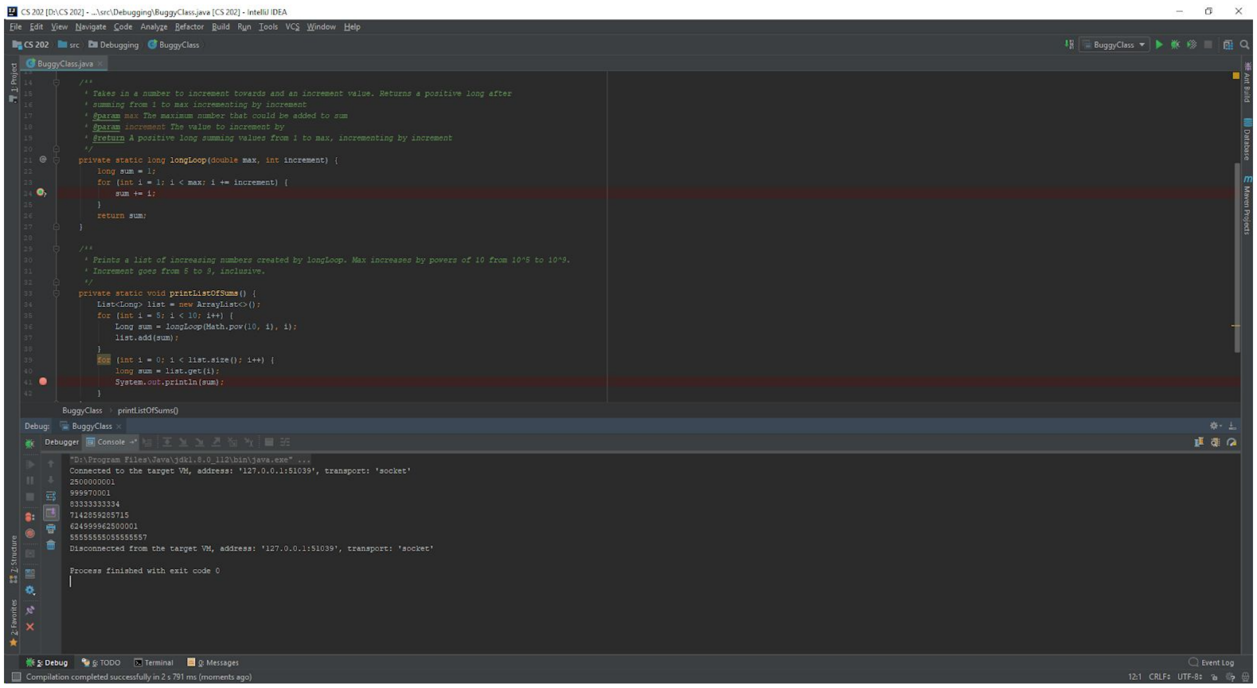
2. It’ll run for awhile, but your conditional breakpoint shouldn’t trigger this time. Once it hits your breakpoint on line 41, we can be confident that we aren’t overflowing anymore.

3. To confirm this, let printListOfSums run all the way through and step through its second loop like we did before. To speed things up, disable your conditional breakpoint by right clicking on it and clicking on the box next to Enabled. Hit done, and it should now look like this:



4. This means we can re enable it if we need to without recreating it. If we check our first watch while we step through the loop where the numbers get printed out, we see that each number is smaller than the next. Now we can be confident that we fixed the bug.

5. Take a screenshot like the one below of the final output along with the rest of your IntelliJ window to submit on Learning Suite to receive credit for this tutorial.


CS 202 [D:\CS 202] - ...\src\Debugging\BuggyClass.java [CS 202] - IntelliJ IDEA

```java
    /**
     * Takes in a number to increment towards and an increment value. Returns a positive long after
     * summing from 1 to max incrementing by increment
     * @param max The maximum number that could be added to sum
     * @param increment The value to increment by
     * @return A positive long summing values from 1 to max, incrementing by increment
     */
    private static long longLoop(double max, int increment) {
        long sum = 1;
        for (int i = 1; i < max; i += increment) {
            sum += i;
        }
        return sum;
    }

    /**
     * Prints a list of increasing numbers created by longLoop. Max increases by powers of 10 from 10^5 to 10^9.
     * Increment goes from 5 to 9, inclusive.
     */
    private static void printListOfSums() {
        List<Long> list = new ArrayList<>();
        for (int i = 5; i < 10; i++) {
            Long sum = longLoop(Math.pow(10, i), i);
            list.add(sum);
        }
        for (int i = 0; i < list.size(); i++) {
            long sum = list.get(i);
            System.out.println(sum);
        }
```

```
"D:\Program Files\Java\jdk1.8.0_112\bin\java.exe" ...
Connected to the target VM, address: '127.0.0.1:51039', transport: 'socket'
2500000001
999970001
83333333334
7142859285715
624999962500001
55555555055555557
Disconnected from the target VM, address: '127.0.0.1:51039', transport: 'socket'

Process finished with exit code 0
```

Compilation completed successfully in 2 s 791 ms (moments ago)