Debugger Refresher

Let's walk through the key functions of the debugger!

Suppose you have the following method. You may already see where the issue lies, but for this example, we'll go through the steps of using the debugger.

```
public class sum {
    2 usages  new *
    public static int sumMethod(int num1, int num2){
        return num1 - num2;
    }
}
```

Now, here's your test case for the method:

```
    public void sumTest(){
        int number1 = 5;
        int number2 = 4;

        int expectedSum = number1 + number2;

        assertEquals(expectedSum,sumMethod(number1,number2));
    }
}
```

Looks pretty good, right? But—surprise! Your tests are failing, and you can't figure out why. Time to break out the debugger! 😎

0. Breakpoints

Breakpoints are placed where you want the code to pause. They allow you to inspect everything that has happened in memory up to (but not including) that specific line of code.

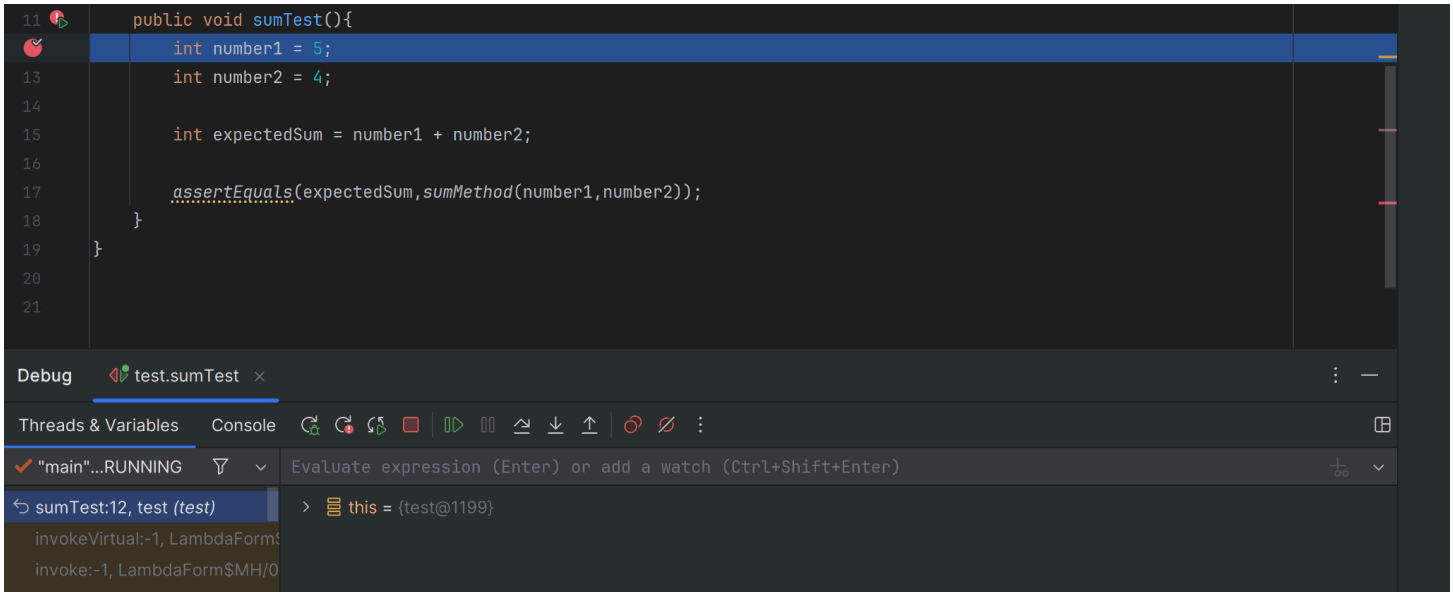
To add a breakpoint, click on the line number (also known as the gutter). In tests, this helps track the actual values being passed through your code.

```
11      public void sumTest(){
            int number1 = 5;
13          int number2 = 4;
14
15          int expectedSum = number1 + number2;
16
17          assertEquals(expectedSum,sumMethod(number1,number2));
18      }
19  }
```

1. Debugging

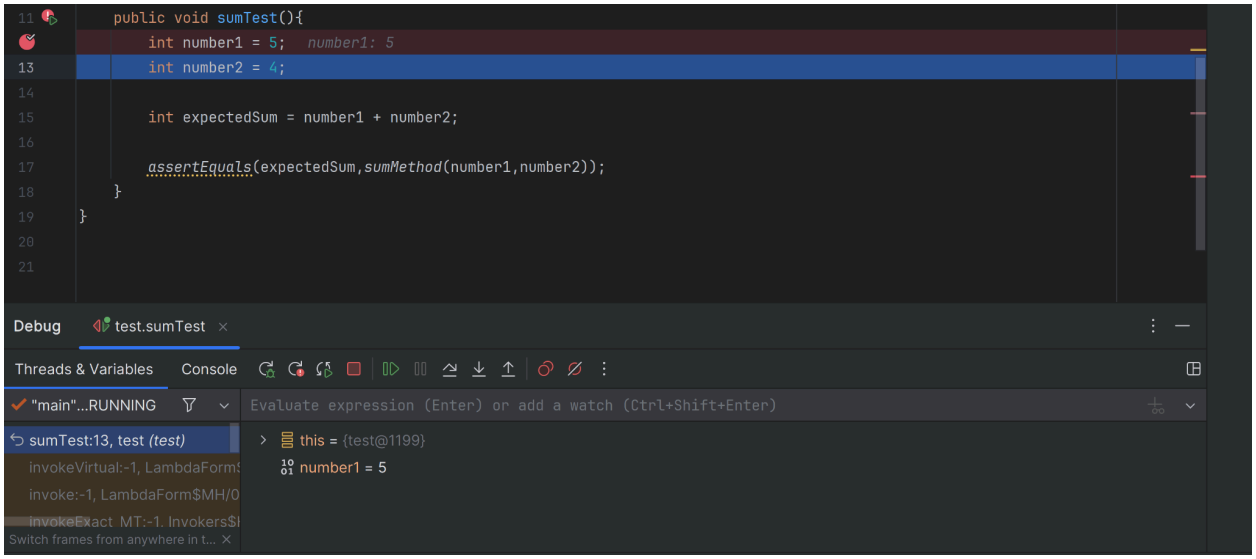
Once we establish where we want our breakpoint, we want to hit the bug icon in the corner. This will start the debugging process and you are able to see all your values on the bottom of your screen.

> Your screen will now look something like this, with the line the debugger is on being highlighted in blue.



2. Step Over

The **Step Over** button allows you to move to the next line of code. The debugger updates, showing how the previous line impacted memory.

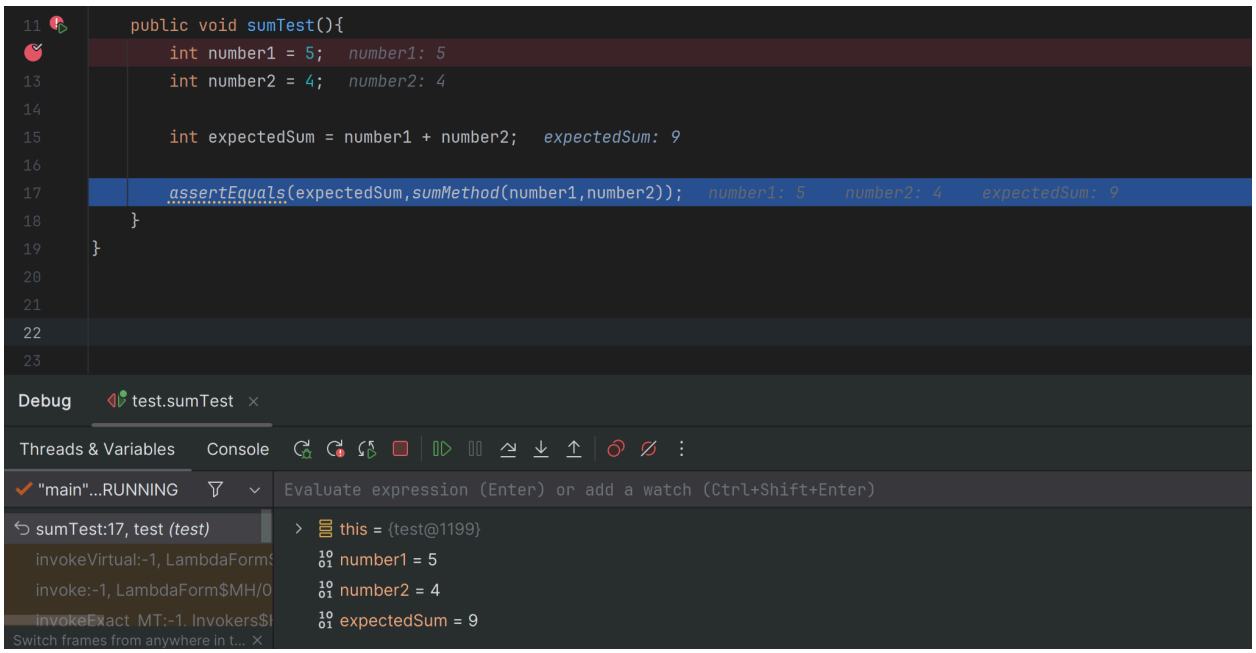


Notice how the value of number1 is now present, and the blue line has moved down one? That's what happens when you step over!

3. Step Into

The **Step Into** button lets you dive into a method call, allowing you to see how that method operates with the given inputs.

Let's step over a few times to get to the nitty gritty of our code.



Let's step over a few times to reach the critical part of our code. At this point, all the variables from our tests are loaded into memory, and we're ready to call our method. How do we see how the code behaves with these test inputs? By using **Step Into**!

Important Note:
If multiple methods are being called on the same line, **Step Into** will highlight each method and let you choose which one to enter. If you press the button again without selecting, it will step into the first method.

```
assertEquals(expectedSum,sumMethod(number1,number2));
```

After stepping into the desired method, we find the issue—we were subtracting instead of adding. Oops!

```
    public static int sumMethod(int num1, int num2){   num1: 5    num2: 4
        return num1 - num2;   num1: 5    num2: 4
    }
}
```

4. Step out of

The **Step Out** button is essentially the opposite of **Step Into**. It takes you back to where you were before pressing step into, allowing you to continue from there.