PROGRAMMING

Lecture 24

Sushil Paudel

TODAY'S TOPIC

- Bug & Examples
- Debugging in Intellij IDEA
- Viva question samples

BUG

- In computer technology, a bug is a coding error in a computer program.
- The term 'bug' refers to mistakes in software or other aspects of a program.

  These mistakes may produce an error in the form of unexpected results or erratic behavior.
- In the best case, a bug may only affect software performance.
- In the worst case, it may make the software crash

BUG EXAMPLE 1

```java
public void checkEven(int number) {

    boolean isEven = false;

    if (number % 2 != 0) {
        isEven = true;
    }

    System.out.println("Is even: " + isEven);
}
```

BUG EXAMPLE 2

```java
public void add(String s1, String s2) {
    int num1 = Integer.parseInt(s1);
    int num2 = Integer.parseInt(s2);

    int sum = num1 + num2;

    System.out.println("Sum: " + sum);
}
```

BUG EXAMPLE 3

```
private JFrame frame1;
private JFrame frame2;

public void createGUI() {
    frame1 = new JFrame( title: "Frame One");

    JPanel panel = new JPanel();
    panel.add(new JLabel( text: "Hello!"));
    panel.add(new JLabel( text: "Hey!"));

    frame2.add(panel);
    frame2.setSize( width: 200,  height: 200);
    frame2.setVisible(true);
}
```

BUG EXAMPLE 3

```
JFrame frame1 = new JFrame( title: "Frame One");
JLabel name = new JLabel( text: "Km");
JTextField nameField = new JTextField();
JButton convertToMiles = new JButton( text: "Convert to Miles");

frame1.add(name);
frame1.add(nameField);
frame1.add(convertToMiles);

convertToMiles.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Integer kilometer = Integer.parseInt(name.getText());
        double miles = kilometer * 0.62137;
        System.out.println("Miles: " + miles);
    }
});
```

DEBUGGING

- Broadly, debugging is the process of detecting and correcting errors in a program.
- There are different kinds of errors, which you are going to deal with.
- Some of them are easy to catch, like syntax errors, because they are taken care of by the compiler.
- Another easy case is when the error can be quickly identified by looking at the stack trace, which helps you figure out where the error occurred.

DEBUGGING

- However, there are errors which can be very tricky and take really long to find and fix.
- For example, a subtle logic error, which happened early in the program may not manifest itself until very late, and sometimes it is a real challenge to sort things out.
- This is where the debugger is useful.
- The debugger is a powerful tool, which lets you find bugs a lot faster by providing an insight into the internal operations of a program.

DEBUGGING

- This is possible by pausing the execution and analyzing the state of the program by thorough examination of variables and how they are changed line by line.
- While debugging, you are in full control of the things.
- In this manual we are covering a basic debugging scenario to get you started.

EXAMPLE

```
public class AverageFinder {
    public static void main(String[] args) {
        System.out.println("Average finder v0.1");
        double avg = findAverage(args);
        System.out.println("The average is " + avg);
    }

    private static double findAverage(String[] input) {
        double result = 0;
        for (String s : input) {
            result += Integer.parseInt(s);
        }
        return result;
    }
}
```

EXAMPLE

- The program is supposed to calculate the average of all values passed as command-line arguments.
- It compiles and runs without issues, however the result is not what one would expect. For instance, when we pass 1 2 3 as the input, the result is 6.0.
- First of all, you need to think about where the suspected error might be coming from.
- We can assume the problem is not in the print statements. Most likely, unexpected results are coming from our findAverage method. In order to find the cause, let's examine its behavior at runtime.

SET BREAKPOINTS

- To examine how the program operates at runtime, we need to suspend its execution before the suspected piece of code.
- This is done by setting breakpoints.
- Breakpoints indicate the lines of code where the program will be suspended for you to examine its state.

BREAKPOINTS

- Click the gutter at the line where the `findAverage` method is called.

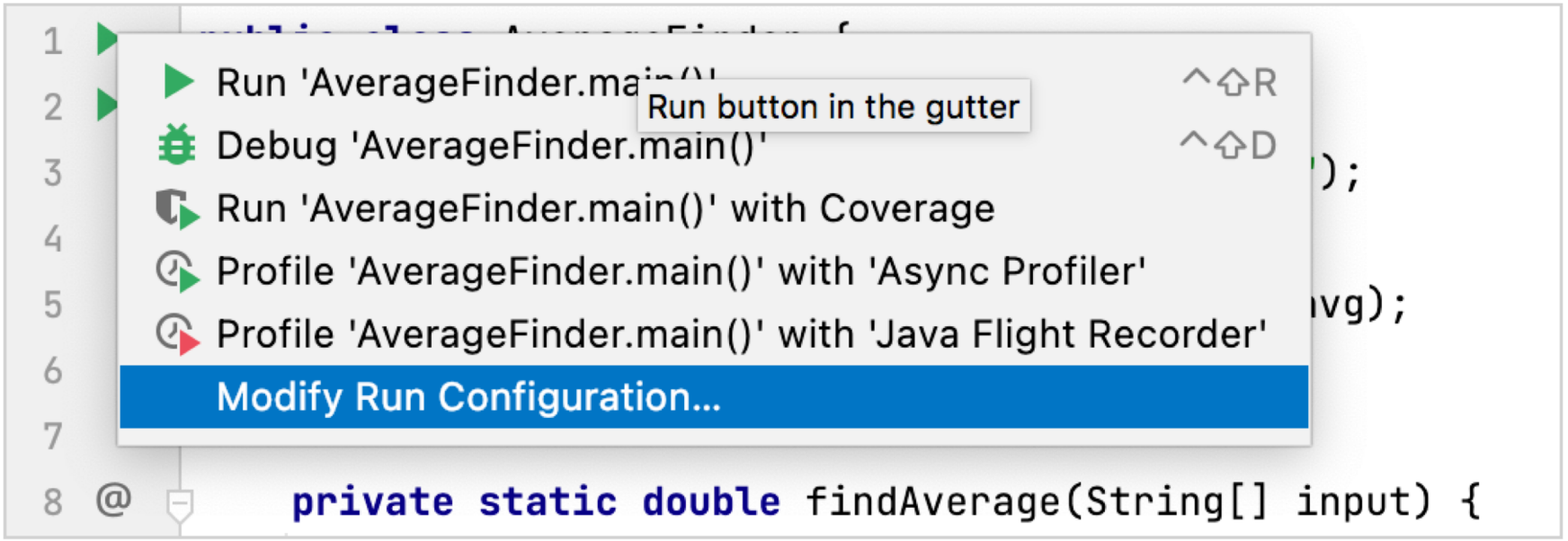
```
1  public class AverageFinder {
2      public static void main(String[] args) {
3          System.out.println("Average finder v0.1");
4          double avg = findAverage(args);
5          System.out.println("The average is " + avg);
6      }
7
8      private static double findAverage(String[] input) {
9          double result = 0;
10         for (String s : input) {
11             result += Integer.parseInt(s);
12         }
13         return result;
14     }
15 }
```

DEBUG MODE

- Now let's start the program in debug mode.
- Since we are going to pass arguments for running and debugging the program, make sure the run/debug configuration has these arguments in place.

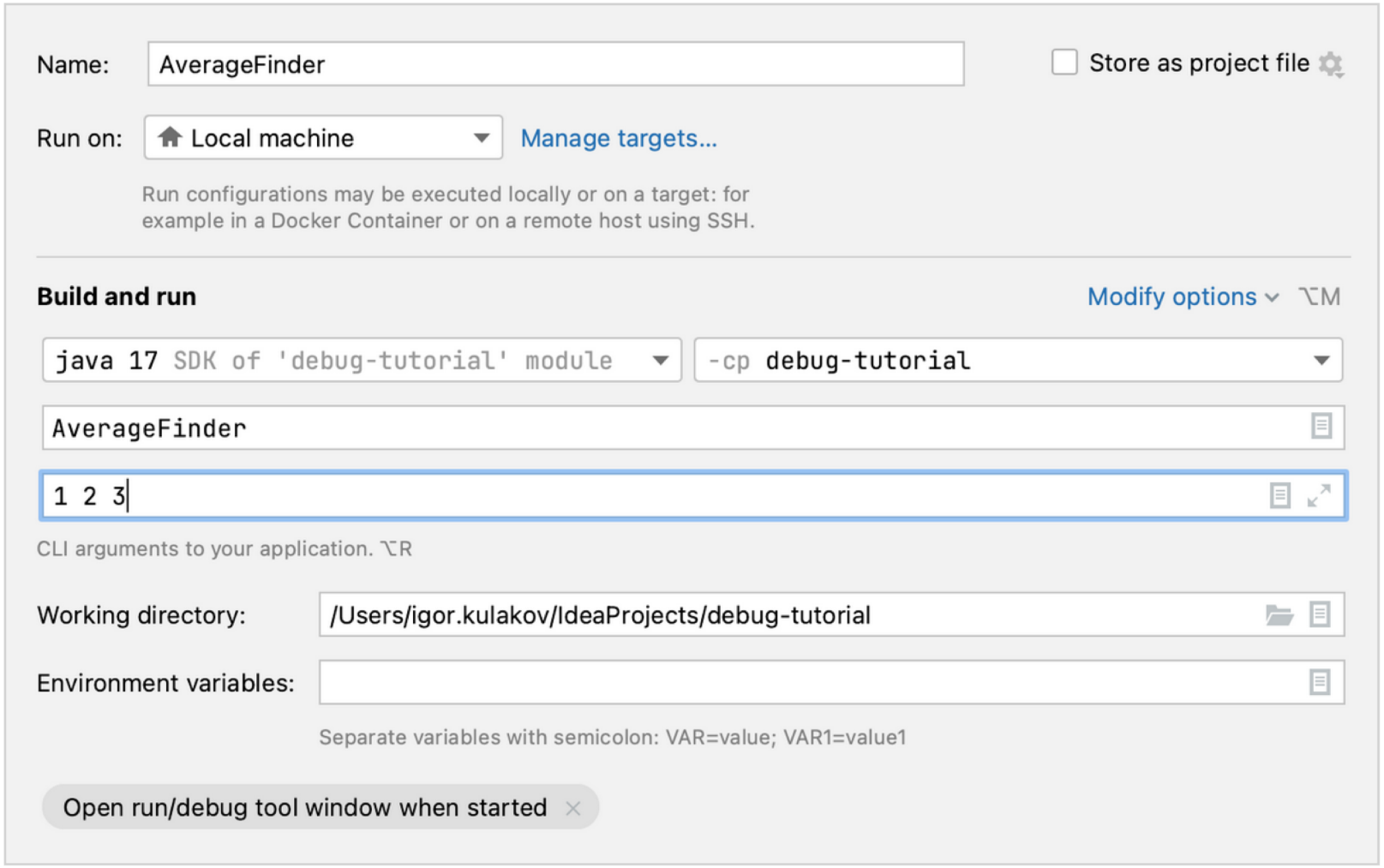
STEP 1

1. Click the **Run** icon in the gutter, then select **Modify Run Configuration**.



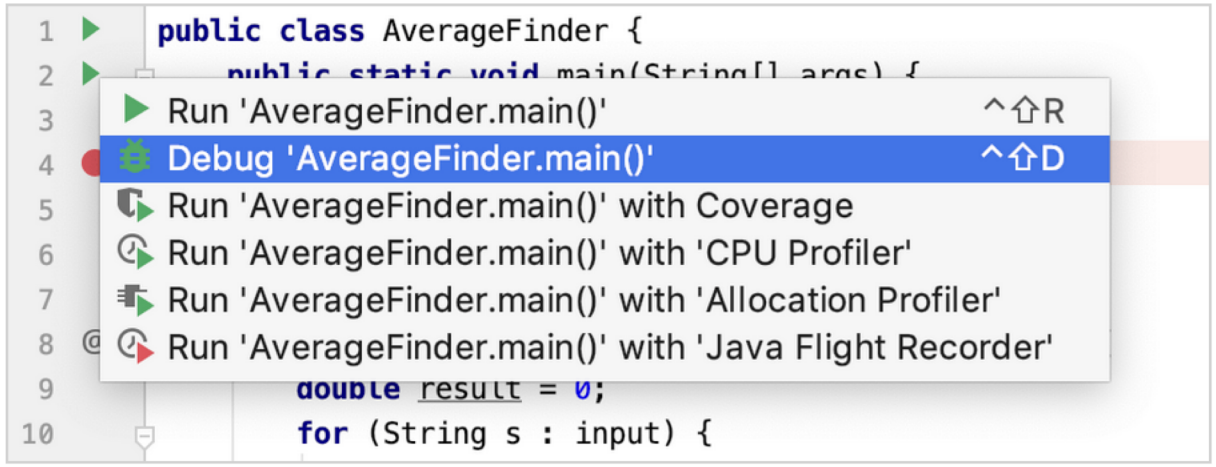
STEP 2

2. Enter arguments in the **Program arguments** field.

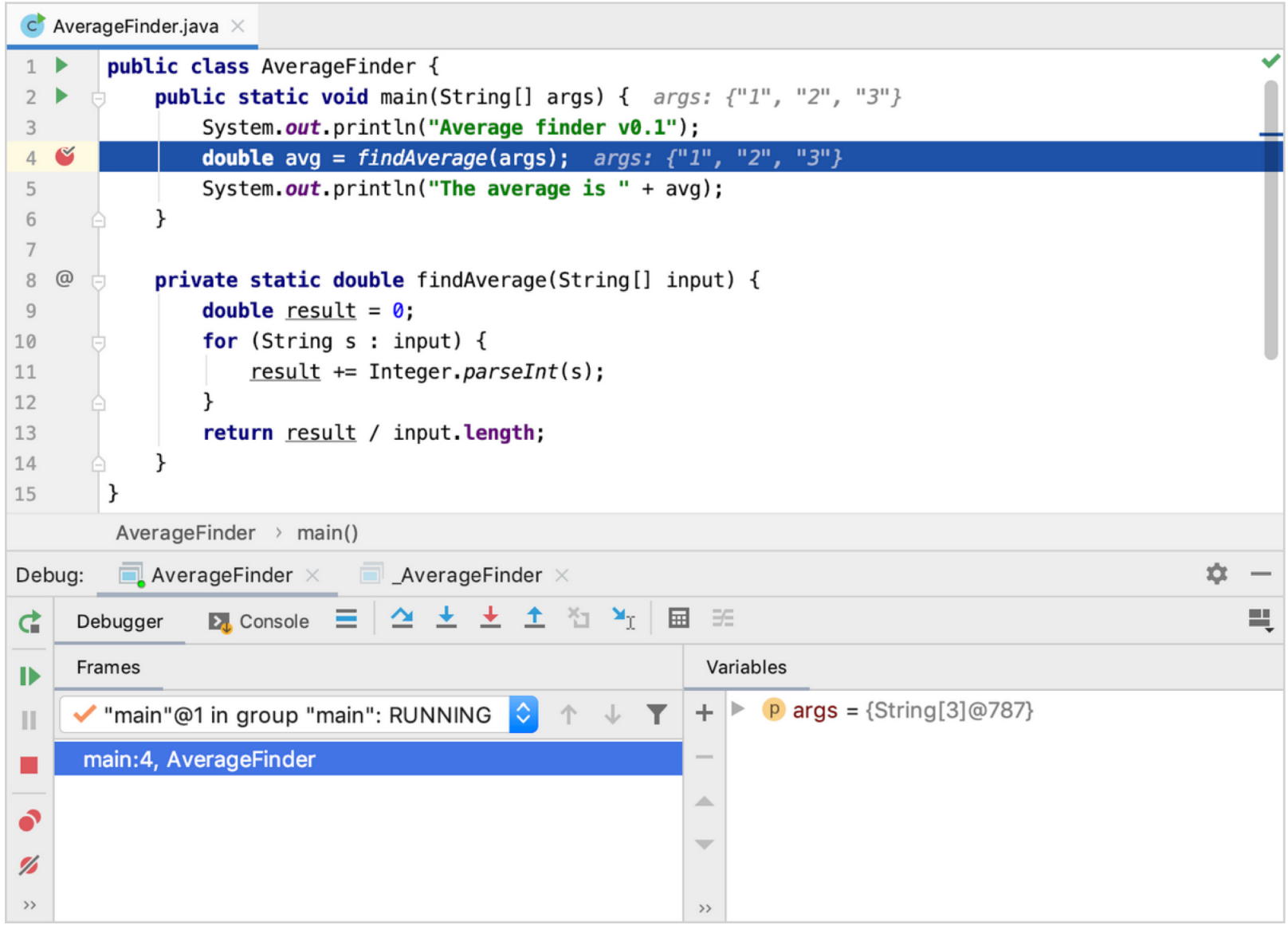


STEP 3

3. Click the **Run** button near the `main` method. From the menu, select **Debug**.



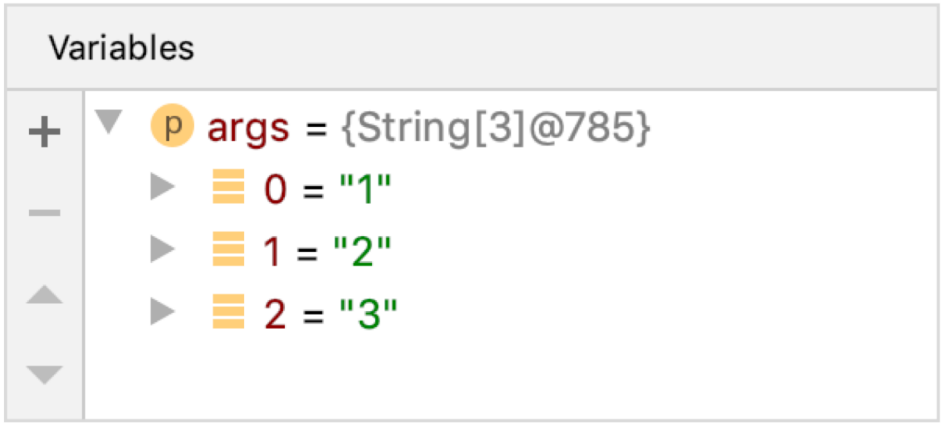
ANALYSE THE PROGRAM STATE



IN DETAIL

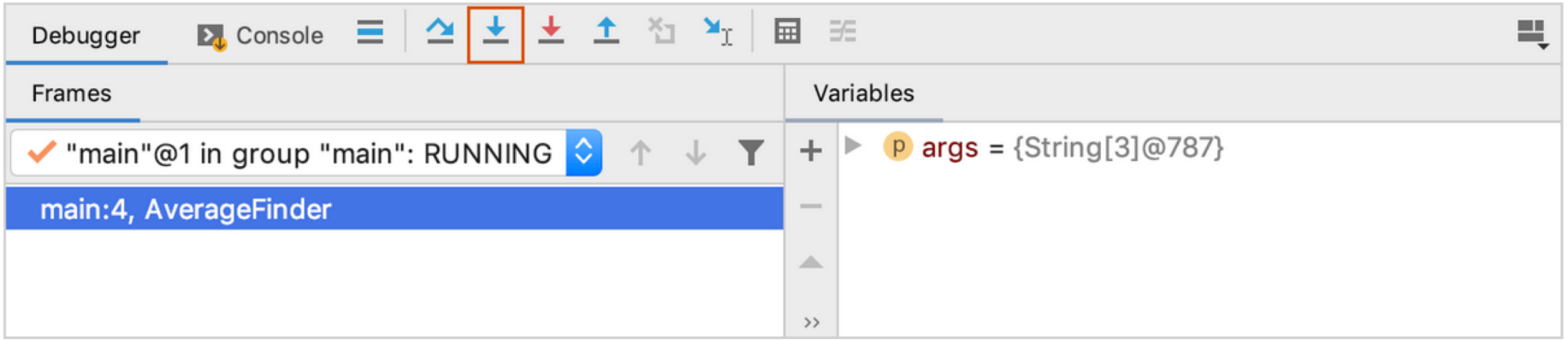
```
System.out.println("Average finder v0.1");
double avg = findAverage(args);  args: {"1", "2", "3"}
System.out.println("The average is " + avg);
```

You can also get information about all variables that are currently in scope in the **Variables** panel.



DEBUGGING STEP 1

1. To step into a method, click the **Step Into** button or press F7.

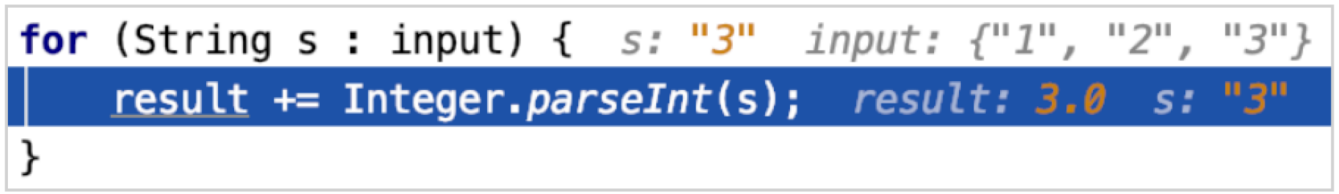


Another line gets highlighted in the editor because we advanced the execution point one step forward.

STEP 2

2. Continue stepping with **Step Over** `F8`. Notice how it is different from **Step Into** – it also advances the execution one step forward, but it doesn't visit any methods like `Integer.parseInt()` along the way.

   Let's keep stepping and see how the local variable `result` is declared and how it is changed with each iteration of the loop.



   Right now the variable `s` contains the value `"3"`. It is going to be converted to an Integer and be added to `result`, which currently has the value of `3.0`. No errors so far. The sum is calculated correctly.

STEP 3

3. Two more steps take us to the return statement and we see where the omission was. We forgot to divide the sum by the number of values. This was the cause of incorrect method return.

```
double result = 0;  result: 6.0
for (String s : input) {  input: {"1", "2", "3"}
    result += Integer.parseInt(s);
}
return result;  result: 6.0
```

To continue the program execution after it has been suspended, go to the main menu and select **Run | Debugging Actions | Resume**, or press ⌥⌘R

STEP 4

4. Let's correct the error.

```
return result / input.length;
```

VIVA QUESTION SAMPLES

- Constructor vs Methods
- Types of constructor
- Difference between static and instance variable
- What is return type in method?
- What if we add return type in constructor?
- How to call static method vs instance method?
- Difference between int and long?
- How to compile and run the program using command prompt?

VIVA QUESTION SAMPLES

- Default value of int vs Integer
- Final variable usage
- Method overriding vs method overloading
- Types of polymorphism
- Down casting vs upcasting
- Array vs Array List
- Interface vs Abstract Methods

VIVA QUESTION SAMPLES

- What is the use of layout manager?
- Which interface do you use for Button click action?
- Difference between frame and panel?
- What do you change the text of label?
- How to convert the value of text field to Integer?
- Difference between GridLayout and GridbagLayout?
- How to compare the values of two text fields?

VIVA QUESTION SAMPLES

- What is the use of JTable?
- How many panels can be added in a frame?
- How to clear the text fields?
- What will happen if you don’t initialize the Jframe?
- Can you reuse same component in different panels?
- What is the recommended naming convention for components like label, text fields, etc?

REFERENCE

- https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html#running-program

THANK YOU!

Any questions?