PHAS3459
Scientific programming using object-oriented languages

Module 2: Debugging

Ben Waugh & Simon Jolly

Why Bother With Bugs?

- If you write any code, you are guaranteed to spend a good portion of your time weeding out errors in that code.
- Debugging is the process of going through your code and locating and correcting these errors.
- Debugging is laborious, boring, frustrating, time consuming and probably the most important part of writing code.
- Even superficially correct code needs to be debugged to ensure that you're not passing on subtly flawed code to someone else.
- Most software releases from the major software companies, like Apple and Microsoft, do not contain new functionality but simply address the many bugs inadvertently introduced with the previous software…
- There are many types of bug:
  - https://en.wikipedia.org/wiki/Software_bug
  - https://en.wikipedia.org/wiki/Heisenbug
- Your task is to make sure you have a basic grasp of how to find and correct errors in your code.

The Example `BuggyCode` Java Class

- On the course web page, you will find the example Java class `BuggyCode.java`.
- Alongside the `main` method it has two other methods:
  - `increment`: increments a single value a number of times using a `while` loop.
  - `addOne`: adds 1 to every value in a given array using a `for` loop.
- We will use this to learn the basic techniques when debugging Java code.
- Download this class file from the course web page and import it into your `module2` package:
  - Open up the package explorer in Eclipse and right-click on the package “module2”: select “Import”.
  - From the “Import” dialogue, select “General → File System”; click “Next”.
  - Browse to the directory you downloaded `BuggyCode.java` to; select “OK”.
  - Tick the check box next to `BuggyCode.java`; leave all the other check boxes empty. Click “Finish”.
- Once you have imported the class, run it and see what happens…

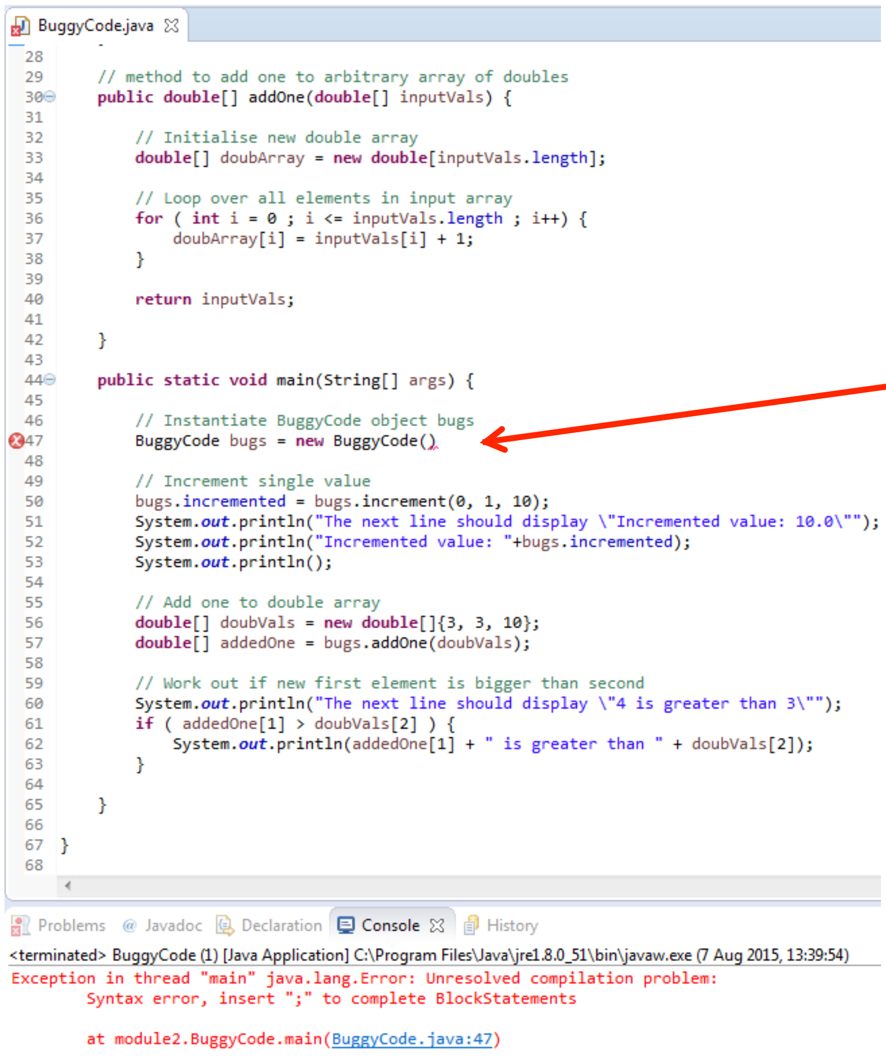
Examining the First Bug

```
Exception in thread "main" java.lang.Error: Unresolved
compilation problem:

    Syntax error, insert ";" to complete BlockStatements
    at module2.BuggyCode.main(BuggyCode.java:47)
```

- What happens when you first run the code? It doesn't run!
- How do you find out why?
  - Looking in the Console gives you all the information you need!
  - It tells you what the problem is ("`Syntax error`"), where it is ("`module2.BuggyCode.main(BuggyCode.java:47)`") and how to fix it ("`insert ";" to complete BlockStatements`").

Correcting the First Bug



BuggyCode.java

```
28
29      // method to add one to arbitrary array of doubles
30      public double[] addOne(double[] inputVals) {
31
32          // Initialise new double array
33          double[] doubArray = new double[inputVals.length];
34
35          // Loop over all elements in input array
36          for ( int i = 0 ; i <= inputVals.length ; i++) {
37              doubArray[i] = inputVals[i] + 1;
38          }
39
40          return inputVals;
41
42      }
43
44      public static void main(String[] args) {
45
46          // Instantiate BuggyCode object bugs
47          BuggyCode bugs = new BuggyCode()
48
49          // Increment single value
50          bugs.incremented = bugs.increment(0, 1, 10);
51          System.out.println("The next line should display \"Incremented value: 10.0\"");
52          System.out.println("Incremented value: "+bugs.incremented);
53          System.out.println();
54
55          // Add one to double array
56          double[] doubVals = new double[]{3, 3, 10};
57          double[] addedOne = bugs.addOne(doubVals);
58
59          // Work out if new first element is bigger than second
60          System.out.println("The next line should display \"4 is greater than 3\"");
61          if ( addedOne[1] > doubVals[2] ) {
62              System.out.println(addedOne[1] + " is greater than " + doubVals[2]);
63          }
64
65      }
66
67  }
68
```

```
<terminated> BuggyCode (1) [Java Application] C:\Program Files\Java\jre1.8.0_51\bin\javaw.exe (7 Aug 2015, 13:39:54)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	Syntax error, insert ";" to complete BlockStatements

	at module2.BuggyCode.main(BuggyCode.java:47)
```

- So the syntax error is occurring because there's a missing ";" at the end of line 47.
- The editor will also show you where the error exists.
- If you add a ";" at the end of line 47, the code should now run without syntax errors.
- But it still has other bugs…

Bug Number 2

- Running the code again will produce another error:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3
    at module2.BuggyCode.addOne(BuggyCode.java:37)
    at module2.BuggyCode.main(BuggyCode.java:57)
```

- This is not a syntax error: we are able to run the code!
- However it's clear that there's something wrong…
- Again, the Console tells you what the bug is: "`java.lang.ArrayIndexOutOfBoundsException: 3`"
- It also tells you where it is, including the line of code that produced the actual error:
  - "`module2.BuggyCode.addOne(BuggyCode.java:37)`"
- …plus the line where that code is called from the main method:
  - "`module2.BuggyCode.main(BuggyCode.java:57)`"
- However, this time, it doesn't tell you what the solution is! You only get a clue from the error message ("`Array Index Out Of Bounds Exception: 3`").
- How do we find out what the error is?

Your Friend the "`print`" Statement…

- Enter the humble "`print`" statement! Or in Java parlance, **`System.out.println`**.
- Print statements are the easiest and most common way of debugging code.
- The aim is to work out what the program is doing at the point where there seems to be an error.
- This normally means checking the values of various variables to make sure the code is doing what you think it should.
- Putting in "`System.out.println`" statements at suitable locations will allow to display the values of the necessary variables.
- Lets try this with the `addOne` method…

Identifying the `addOne` Bug

- Modify the `addOne` method in the following way:

```java
public double[] addOne(double[] inputVals) {

  // Initialise new double array
  double[] doubArray = new double[inputVals.length];
  System.out.println("Length of input array: "+inputVals.length);

  // Loop over all elements in input array
  for ( int i = 0 ; i <= inputVals.length ; i++) {
    System.out.println("Value of i: "+i);
    doubArray[i] = inputVals[i] + 1;
    System.out.println("Value of inputVals[i]: "+inputVals[i]);
    System.out.println("Value of doubArray[i]: "+doubArray[i]);
  }

  return inputVals;
```

- Note that we've added in some more print statements to display the values of some variables:
  – These lines tell us what the current value is of the array we're reading from, `inputVals[i]`, and the array we're filling, `doubArray[i]`.
  – This line tells us what the value is of the iterator, `i`.
  – This line tells us how many elements there are in `inputVals`.
- What happens when we run this code?

Modified `addOne` Output

You should get the following output after adding the `print` statements:

```
The next line should display
  "Incremented value:10.0"
Incremented value: 0.0

Length of input array: 3
Value of i: 0
Value of inputVals[i]: 3.0
Value of doubArray[i]: 4.0
Value of i: 1
Value of inputVals[i]: 3.0
Value of doubArray[i]: 4.0
Value of i: 2
Value of inputVals[i]: 10.0
Value of doubArray[i]: 11.0
Value of i: 3
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3
    at module2.BuggyCode.addOne(BuggyCode.java:39)
    at module2.BuggyCode.main(BuggyCode.java:61)
```

- So, what's going on here?
- At the start we've identified there are 3 elements in `inputVals`:
  - We should already know this as we initialised it with the values {3,3,10} in the `main` method.
  - Now we know that the code does too!
- We can see that, as the code intends, each individual value of in `doubArray` is one higher than `inputVals`. So far so good…
- But now we see the problem! There are only 3 values in `inputVals`, but `i` is running from 0 to 3!

Correcting the `addOne` Bug

- This indexing bug arises because arrays in Java are *zero-indexed*: if an array has 3 elements, the first element is element 0, the second is element 1 and the third is element 2, NOT 1, 2 and 3!
- This is what "**`Array Index Out Of Bounds`**" means: `addOne` is trying to reference `inputVals[3]`, but the maximum index is `inputVals[2]`.
- If you look at the **`for`** loop, you'll see that `i` runs from 0 to 3 (`i <= inputVals.length`).
- To get it to run from 0 to 2, modify the **`for`** loop so that it reads:

```
for ( int i = 0 ; i < inputVals.length ; i++)
```

- What happens if you run the code now? You shouldn't get any errors displayed in the console!
- Go ahead and comment out the "`print`" statements you've added in (you may need them later).

Bugs Without Errors

- However, the code still isn't behaving as expected! There are clues in the Console:

```
The next line should display "Incremented value: 10.0"
Incremented value: 0.0

The next line should display "4 is greater than 3"
```

- These lines are added to show what the methods should be doing.
- So neither method is functioning as it should!
- Here you have to be careful: bugs in your code don't often result in errors in the console.
- So you have to make sure you know what the code is expected to do and what it's doing at each stage.
- Carefully placed "`print`" statements will help with this, but make sure you comment out "`print`" statements that you no longer need. Otherwise you will just confuse the user with too much information.
- Lets go ahead and debug the "`increment`" method…

The `increment` Method

- Here’s the `increment` method in full:

```
// increment a double, initVal, by the value incVal, a number of times given by numSteps
public double increment(double initVal, double incVal, int numSteps) {

  // Initialise necessary variables
  double finalVal = initVal;
  int i = 0;

 // Loop numSteps times, incrementing initVal each time
  while ( i <= numSteps ) {
    initVal += incVal;
    i += 1;
  }

  return finalVal;

}
```

- Here’s what the code is doing:
  - Take an initial value, `initVal`.
  - Increment it by the value `incVal`.
  - Do this `numSteps` times.
  - Return this incremented value as `finalVal`.
- So when we call `increment` from the “`main`” method and ask it to add 1 to 0, 10 times, why does it return 0? Let’s debug…

Debugging the `increment` Method

- Add the following `print` statements to the `increment` method:

```
// increment a double, initVal, by the value incVal, a number of times given by numSteps
public double increment(double initVal, double incVal, int numSteps) {

  // Initialise necessary variables
  double finalVal = initVal;
  int i = 0;
  System.out.println("Initial values of initVal: "+initVal+"; finalVal: "+finalVal);

  // Loop numSteps times, incrementing initVal each time
  while ( i <= numSteps ) {
    initVal += incVal;
    System.out.println("For iteration "+i+", incVal = "+incVal+", initVal = "+initVal);
    i += 1;
  }

  System.out.println("Final values of initVal: "+initVal+"; finalVal: "+finalVal);

  return finalVal;

}
```

- What do these `print` statements tell you about how the coded is functioning?
- Is everything as you expect?
- Can you see where the bug is?

The `increment` Method Output

You should get the following output adding the `print` statements:

```
Initial values of initVal: 0.0; finalVal: 0.0
For iteration 0, incVal = 1.0, initVal = 1.0
For iteration 1, incVal = 1.0, initVal = 2.0
For iteration 2, incVal = 1.0, initVal = 3.0
For iteration 3, incVal = 1.0, initVal = 4.0
For iteration 4, incVal = 1.0, initVal = 5.0
For iteration 5, incVal = 1.0, initVal = 6.0
For iteration 6, incVal = 1.0, initVal = 7.0
For iteration 7, incVal = 1.0, initVal = 8.0
For iteration 8, incVal = 1.0, initVal = 9.0
For iteration 9, incVal = 1.0, initVal = 10.0
For iteration 10, incVal = 1.0, initVal = 11.0
Final values of initVal: 11.0; finalVal: 0.0
The next line should display "Incremented value: 
10.0"
Incremented value: 0.0

The next line should display "4 is greater than 3"
```

- So what can we identify?
- The starting value is correct ("0").
- The code correctly sets the final value to be equal to the initial value ("0").
- The `while` loop increments the initial value by the correct amount each time ("1").
- So the code seems to be working. But the final value is wrong ("0"). Why?
- The code returns `finalVal` but increments `initVal`!

Correcting the `increment` Method

- Modify the `while` loop so that it increments `finalVal`:

```java
// increment a double, initVal, by the value incVal, a number of times given by numSteps
public double increment(double initVal, double incVal, int numSteps) {

  // Initialise necessary variables
  double finalVal = initVal;
  int i = 0;
  System.out.println("Initial values of initVal: "+initVal+"; finalVal: "+finalVal);

  // Loop numSteps times, incrementing initVal each time
  while ( i <= numSteps ) {
    finalVal += incVal;
    System.out.println("For iteration "+i+", incVal = "+incVal+", finalVal = "+finalVal);
    i += 1;
  }

  System.out.println("Final values of initVal: "+initVal+"; finalVal: "+finalVal);

  return finalVal;

}
```

- Now `finalVal` should have the correct value! Except it returns "11" instead of "10"…
- The print statements you've added give you all the information you need: the `while` loop is increment one step too many.
- Modify the `while` loop: `while ( i < numSteps )`.

Debugging the `addOne` Method Again

Comment out your `print` statements. The code should now give the following output:

```
The next line should display "Incremented value: 10.0"
Incremented value: 10.0

The next line should display "4 is greater than 3"
```

- So there’s still something missing!
- Go ahead and debug both the `addOne` method, using the `print` statements you included previously, as well as the code that calls it from the `main` method.
- Make sure you know what each step of the code is doing…

Summary

- Debugging is frequently a slow, painful, tedious process.
- Unfortunately it's probably what you'll spend most of your time doing, particularly as your code becomes more complex…
- There are a number of specific debugging tools built-in to various programming IDEs:
  - Eclipse has its own built-in debugger to help you step through code.
  - If you end up writing code professionally, you'll probably come across other IDEs which have their own type of debugger.
- However, the most universal technique is still the humble `print` statement!
- The key to debugging: make sure your code is doing what you expect it to do at every stage…