



Java Foundations

8-4

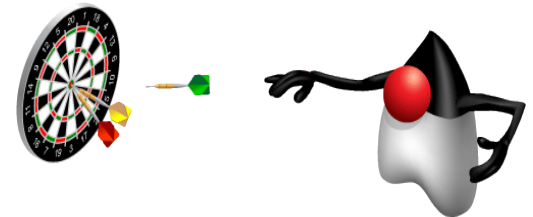
Debugging Concepts and Techniques



Objectives

This lesson covers the following objectives:

- Test and debug a Java program
- Identify the three types of errors
- Apply debugging techniques
 - `print` statements
 - NetBeans debugger
- Apply some debugging tips and techniques



Topics

- Testing and Debugging a Java Program
- Three Types of Errors
- Debugging Techniques: `print` Statements
- Debugging Techniques: NetBeans Debugger
- Debugging Tips

One-
Dimensional
Arrays

ArrayLists

Exception
Handling

Debugging
Concepts and
Techniques



Section 8

Testing a Java Program

Richie wrote a Java program to find the maximum among three integers:

```
public static void main(String[] args) {  
    int num1 = 3, num2 = 3, num3 = 3;  
    int max = 0;  
    if (num1 > num2 && num1 > num3) {  
        max = num1;  
    }  
    if (num2 > num1 && num2 > num3) {  
        max = num2;  
    }  
    if (num3 > num1 && num3 > num2) {  
        max = num3;  
    }  
    System.out.println(" The max of 3 numbers is " + max);  
}
```

Testing a Java Program

- Richie **tested** it on many sets of data, such as $\langle 3, 5, 9 \rangle$, $\langle 12, 1, 6 \rangle$, and $\langle 2, 7, 4 \rangle$.
- The program works for all data.
- However, he was told that the program doesn't work and he couldn't figure out why.



Exercise 1

- Import and open the `DebuggingEx` project.
- Observe `MaxIntegers.java`.
- Can you identify what Richie missed in his testing?

Identify the Error

- The program fails when it's tested with **duplicate values**, such as $\langle 3, 3, 3 \rangle$ and $\langle 7, 2, 7 \rangle$, and it displays the output as zero.
- You identified the error.
- The next step is to fix the error.

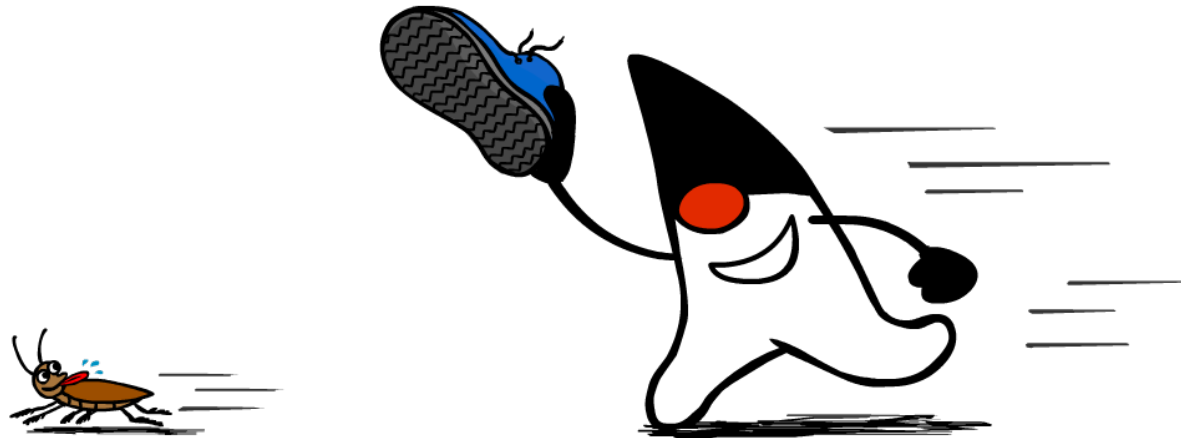
Fix the Error

Modify the program and test it on many data sets, including duplicate values.

```
public static void main(String[] args) {  
    int num1 = 3, num2 = 3, num3 = 3;  
    int max = 0;  
    if (num1 > max) {  
        max = num1;  
    }  
    if (num2 > max) {  
        max = num2;  
    }  
    if (num3 > max) {  
        max = num3;  
    }  
    System.out.println(" The max of 3 numbers is " + max);  
}
```

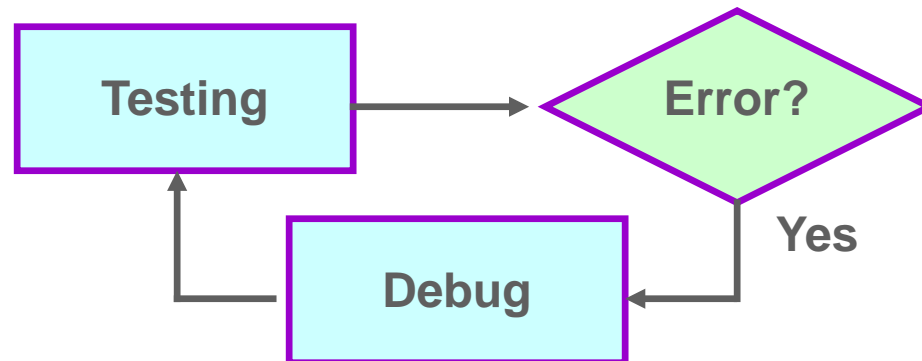
Testing and Debugging

As you observed from the previous example, testing and debugging are important activities in software development.



Testing and Debugging

- Testing:
 - To determine if a code contains errors
- Debugging:
 - To identify an error and fix it



Topics

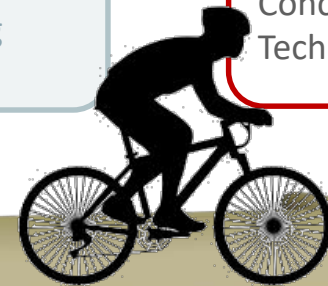
- Testing and Debugging a Java Program
- **Three Types of Errors**
- Debugging Techniques: `print` Statements
- Debugging Techniques: NetBeans Debugger
- Debugging Tips

One-
Dimensional
Arrays

ArrayLists

Exception
Handling

Debugging
Concepts and
Techniques



Section 8

Three Types of Errors

- Compilation errors
- Logic errors
- Runtime errors

Compilation Errors

- Syntax error
- Easiest type of errors to fix
- Examples:
 - Example 1: Missing semicolon

```
int a = 5 // semicolon is missing
```

- Example 2: Errors in expression

```
x = ( 3 + 5 ; // missing closing parenthesis
```

```
y = 3 + * 5 ; // missing argument between '+' and '*'
```

Logic Errors

- Program runs but produces incorrect result.
- Hard to characterize, and so it's hardest to fix.
- Example: Noninitialized variable

```
int i;
```

```
i++; // the variable i isn't initialized
```

Runtime Errors

- These errors occur at run time.
- Java's exception handling mechanism can catch such errors.
- Some of the common exceptions:
 - `ArrayIndexOutOfBoundsException`
 - `NullPointerException`
 - `ArithmeticException`

Topics

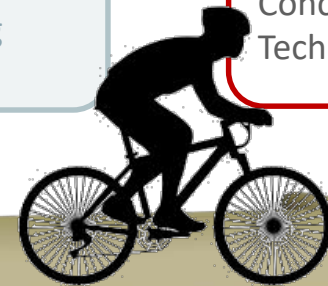
- Testing and Debugging a Java Program
- Three Types of Errors
- Debugging Techniques: `print` Statements
- Debugging Techniques: NetBeans Debugger
- Debugging Tips

One-
Dimensional
Arrays

ArrayLists

Exception
Handling

Debugging
Concepts and
Techniques



Section 8

Debugging Techniques

Let's look at two debugging techniques:

- Using `print` statements
- Using the NetBeans debugger

print Statements: Advantages

- Easy to add
- Provide information
 - Which methods have been called
 - The value of parameters
 - The order in which methods have been called
 - The values of local variables and fields at strategic points

`print` Statements: Disadvantages

- It isn't practical to add `print` statements to every method.
- Too many `print` statements lead to information overload.
- Removal of `print` statements is tedious.

print Statements: Example

- Consider this Java code :

```
int n = 10;  
int sum = 10;  
while (n > 1){  
    sum = sum + n;  
    n--;  
}  
System.out.println("The sum of the integers 1 to 10 is " + sum);
```

- On running this program, it doesn't work correctly.
- To find out what's wrong, you can trace the value of the `n` and `sum` variables by inserting `print` statements.

Modified Program with Additional print Statements for Debugging

```
int n = 10;
int sum = 10;
while (n > 1) {
    System.out.println("At the beginning of the loop: n = " + n);
    System.out.println("At the beginning of the loop:sum="+ sum);

    n--;

    System.out.println("At the end of the loop: n = " + n);
    System.out.println("At the end of the loop: sum = " + sum);
}
System.out.println("The sum of the integers 1 to 10 is " + sum);
```

Output

- Here are the first four lines of the output after the **first iteration** of the loop:

At the beginning of the loop: `n = 10`

At the beginning of the loop: `sum = 10`

At the end of the loop: `n = 9`

At the end of the loop: `sum = 20`

- You can see that something is wrong:

The variable `sum` has been set to 20. Because it was initialized to 10, it's set to `10 + 10`, which is incorrect if you want to add the numbers from 1 to 10.

Topics

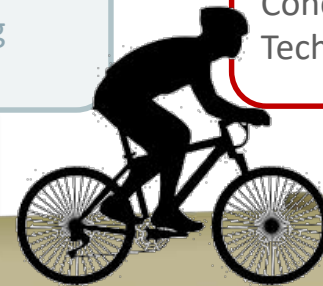
- Testing and Debugging a Java Program
- Three Types of Errors
- Debugging Techniques: `print` Statements
- Debugging Techniques: NetBeans Debugger
- Debugging Tips

One-
Dimensional
Arrays

ArrayLists

Exception
Handling

Debugging
Concepts and
Techniques



Section 8

The NetBeans Debugger

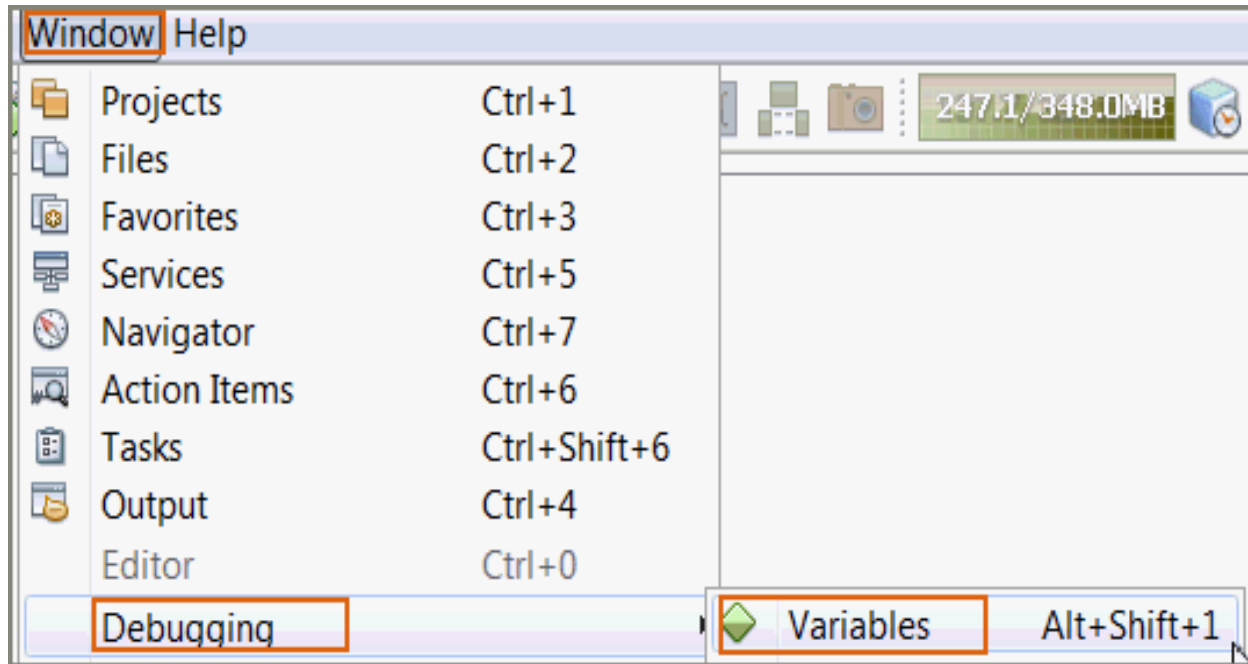
- You have already used the NetBeans graphical-based debugging environment.
- You have used the following features of the debugger:
 - Set breakpoints
 - Trace through a program one line at a time
- Let's use another very important feature to view the contents of variables.

Variables Window

- When you reach a set breakpoint, you can use the Variables window to see the value of the variables at that moment.
- You can find out values of variables without having to put a lot of `print` statements in your program.

Accessing the Variables Window

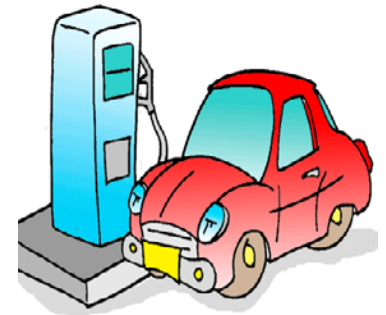
To see the Variable window, in the NetBeans main menu:
Click Window > Debugging > Variables.






Exercise 2: Scenario

- Let's assume you have a car, and you want to go to the gas station. You have the following details:
 - Car's current position: x_1 and y_1
 - Gas station's location: x_2 and y_2
 - Speed of the car
- You want to compute the time it will take for the car from its current position (x_1, y_1) to reach the gas station (x_2, y_2).
- A Java program to compute the time by using the **time=distance/speed** formula is available in the `ComputeTime` project.



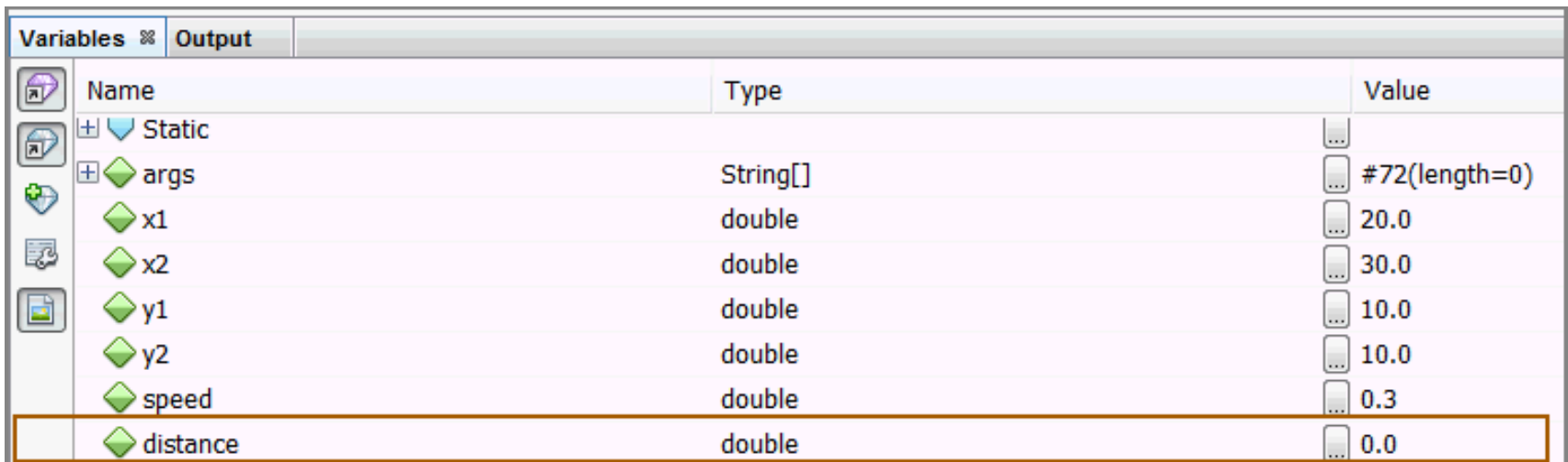


Exercise 2

- Import and open the `DebuggingEx` project.
- Observe `ComputeTime.java`.
- Run the program with the NetBeans debugger to debug this program:
 - Set the breakpoint in the `getDistance` method.
 - Press **Step In** to go to the next line .
 - Observe the values of the `x1`, `x2`, `y1`, `y2`, `speed`, `distance`, and `time` variables.
- Can you identify the bug?

Observe the Value of `distance`

- In the previous exercise using the NetBeans debugging features, you identified the bug:



The image shows the NetBeans IDE's 'Variables' window. It has two tabs: 'Variables' and 'Output'. The 'Variables' tab is active, displaying a table of variables. The table has four columns: 'Name', 'Type', and 'Value'. The 'Name' column also includes a small icon for each variable. The variables listed are: Static (with a plus icon), args (with a plus icon and a diamond icon), x1 (with a diamond icon), x2 (with a diamond icon), y1 (with a diamond icon), y2 (with a diamond icon), speed (with a diamond icon), and distance (with a diamond icon). The 'Type' column shows 'String[]' for args and 'double' for the others. The 'Value' column shows '#72(length=0)' for args and numerical values for the doubles. The 'distance' variable is highlighted with a thick orange border, showing its value as 0.0.

Name	Type	Value
Static		
args	String[]	#72(length=0)
x1	double	20.0
x2	double	30.0
y1	double	10.0
y2	double	10.0
speed	double	0.3
distance	double	0.0

- As you can see, `distance` is 0.0, the formula for computing distance was wrong, and it caused an incorrect return value for the `distance` variable.

Identifying the Potential Bug

```
public static void main(String[] args) {  
  
    double x1 = 20;  
    double x2 = 30;  
    double y1 = 10;  
    double y2 = 10;  
    double speed = 0.3;  
    double distance = getDistance(x1, x2, y1, y2);  
    double time = distance / speed;  
    System.out.println("Time taken to reach the gas station is " + time);  
  
}  
  
static double getDistance(double x1, double x2, double y1, double y2) {  
    return Math.sqrt((x1 - x1) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
}  
}
```


Potential bug

Fixing the Bug

- Because you identified the bug, you can change the location of the breakpoint to where the `getDistance()` method is called.
- This saves having to step through code that you already looked at.
- So let's modify the code and rerun the debugger with the new breakpoint to see what we get.





















Rerunning the Debugger

```
public static void main(String[] args) {  
  
    double x1 = 20;  
    double x2 = 30;  
    double y1 = 10;  
    double y2 = 10;  
    double speed = 0.3;  
    double distance = getDistance(x1, x2, y1, y2);  
    double time = distance / speed;  
    System.out.println("Time taken to reach the gas station is " + time);  
}  
  
static double getDistance(double x1, double x2, double y1, double y2) {  
    return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
}  
}
```



The diagram illustrates a Java code editor with two annotations. A blue arrow points from the text "New breakpoint" to the line `double distance = getDistance(x1, x2, y1, y2);`, which is highlighted in red. Another blue arrow points from the text "Modified code" to the expression `(x1 - x2) * (x1 - x2)` in the `getDistance` method, which is enclosed in a red box.

Observing the Variables

Variables 		Output	
 Name	Type	Value	
  args	String[]	 # / 2 (length=0)	
 x1	double	 20.0	
  x2	double	 30.0	
 y1	double	 10.0	
 y2	double	 10.0	
 speed	double	 0.3	
 distance	double	 10.0	
 time	double	 33.33333333333336	

- We fixed the bug!
- The `distance` variable is now reporting a value of 10.0, and the `time` variable is now reporting a value of 33.33.

Topics

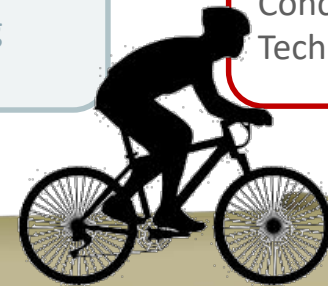
- Testing and Debugging a Java Program
- Three Types of Errors
- Debugging Techniques: `print` Statements
- Debugging Techniques: NetBeans Debugger
- Debugging Tips

One-
Dimensional
Arrays

ArrayLists

Exception
Handling

Debugging
Concepts and
Techniques



Section 8

Single Versus Double Equals Operator

Assignment (=) versus Comparison (==) Operator

1. Comparison operator

`if (x = 0)` instead of `if (x == 0)`

Look for it in `if`, `for`, and `while` statements.

2. Assignment operator

`int x == 1;` instead of `int x = 1;`

Misplaced Semicolon

Check for the semicolon after the `if` statement or the `for/while` loop statements.

```
if (x == 0); {  
    <statements>  
}
```

instead of

```
if(x == 0) {  
    <statements>  
}
```

```
while(<boolean expression>); {  
    <statements>  
}
```

instead of

```
while(<boolean expression>) {  
    <statements>  
}
```

Invoking Methods with Wrong Arguments

- Method call parameter types must match method definition parameter types.
- For example:

- Given a method definition:

```
void methodName(int x, char y) { }
```

- Invoke this method:

```
methodName(a, b)
```



a must be an int and b must be a char.

Boundary Conditions

- It's important to test the **boundary conditions**.
- The rationale behind testing them is that errors tend to occur near the boundary values of an input variable.
- For example, boundary condition for:
 - Input data (test with valid versus invalid)
 - Loops (beginning and ending of loops)

Testing Boundary Conditions for Loops

- This allows for boundary case tests like “less than” and “greater than” for loop iteration conditions to be accurately tested.
- For example, given this loop:

```
if ( num >= 50 && num <= 100 ) {  
  //do stuff  
}
```

- To test boundary conditions, you would test with numbers near 50 and 100, that is, 49, 50, 51, 99, 100 and 101.



Exercise 3

- Import and open the `DebuggingEx` project.
- Observe `BoundaryTesting.java`.
- Validate the input by executing the program with the following boundary test values for year and month:

Year	Month
1582	2
1583	0
1583	13
1583	1
1583	12

Summary

In this lesson, you should have learned how to:

- Test and debug a Java program
- Identify the three types of errors
- Apply debugging techniques
 - `print` statements
 - NetBeans debugger
- Apply some debugging tips and techniques

