# Oracle Academy
# Java for AP Computer Science A

**8-4**

**Debugging Concepts and Techniques**



**ORACLE**
Academy

# Objectives

- This lesson covers the following objectives:
  - Test and debug a Java program
  - Identify the three types of errors
  - Apply debugging techniques
    - `print` statements
    - Using your IDE's debugger
  - Apply some debugging tips and techniques

# Testing a Java Program

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

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

# Testing a Java Program

- Richie tested it on many sets of data, such as <3,5,9>, <12,1,6>, and <2,7,4>

- 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

- Create a new project and add the `MaxIntegers.java` file to the project

- Observe `MaxIntegers.java`
  - Can you identify what Richie missed in his testing?

ORACLE
Academy

# Identify the Error

- The program fails when it's tested with duplicate values, such as <3,3,3> and <7,2,7>, 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

```java
public static void main(String[] args) {
    int num1 = 3, num2 = 3, num3 = 3;
    int max = 0;
    if (num1 > max) {
        max = num1;
    }//endif
    if (num2 > max) {
        max = num2;
    }//endif
    if (num3 > max) {
        max = num3;
    }//endif
    System.out.println("The max of 3 numbers is " + max);
}//end method main
```
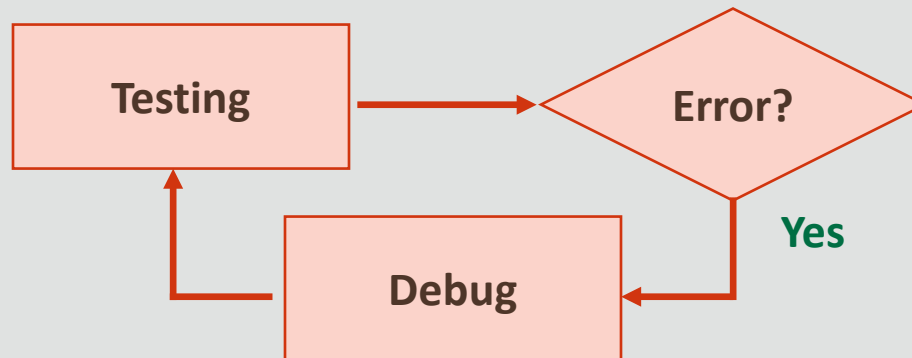
# Testing and Debugging

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

ORACLE
Academy
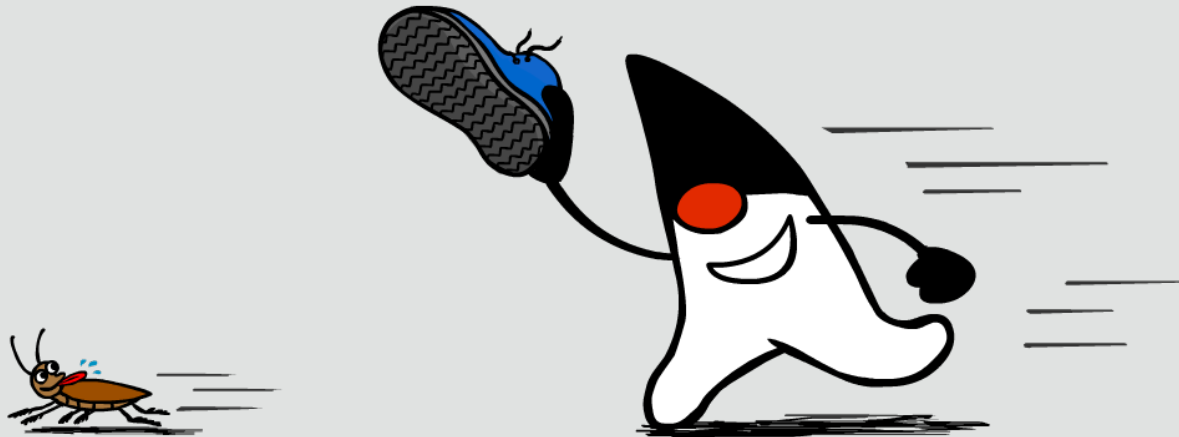
# Testing and Debugging

- Testing:
  - To determine if a code contains errors

- Debugging:
  - To identify an error and fix it

```
Testing ──→ Error?
  ↑            │ Yes
  │            ↓
  └── Debug ←──┘
```

# Three Types of Errors

- Errors
  - Compilation errors
  - Logic errors
  - Runtime errors

JavaAPCSA 8-4
Debugging Concepts and Techniques

# 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:
  - `ArrayIndexOutOfBounds`
  - `NullPointerException`
  - `ArithmeticException`

ORACLE
Academy

# Debugging Techniques

- Let's look at two debugging techniques:
  - Using `print` statements
  - Using an IDE debugger

**ORACLE**
Academy

# `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 :

```java
int n = 10;
int sum = 10;
while (n > 1){
    sum = sum + n;
    n--;
}//end while
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

```java
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

# Your IDE's Debugger

- You have already used the IDE 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

- The following slides demonstrate using the debugger in NetBeans

- If you are using a different IDE, consult the documentation for how to do this

JavaAPCSA 8-4
Debugging Concepts and Techniques

# Accessing the Variables Window

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

ORACLE
Academy
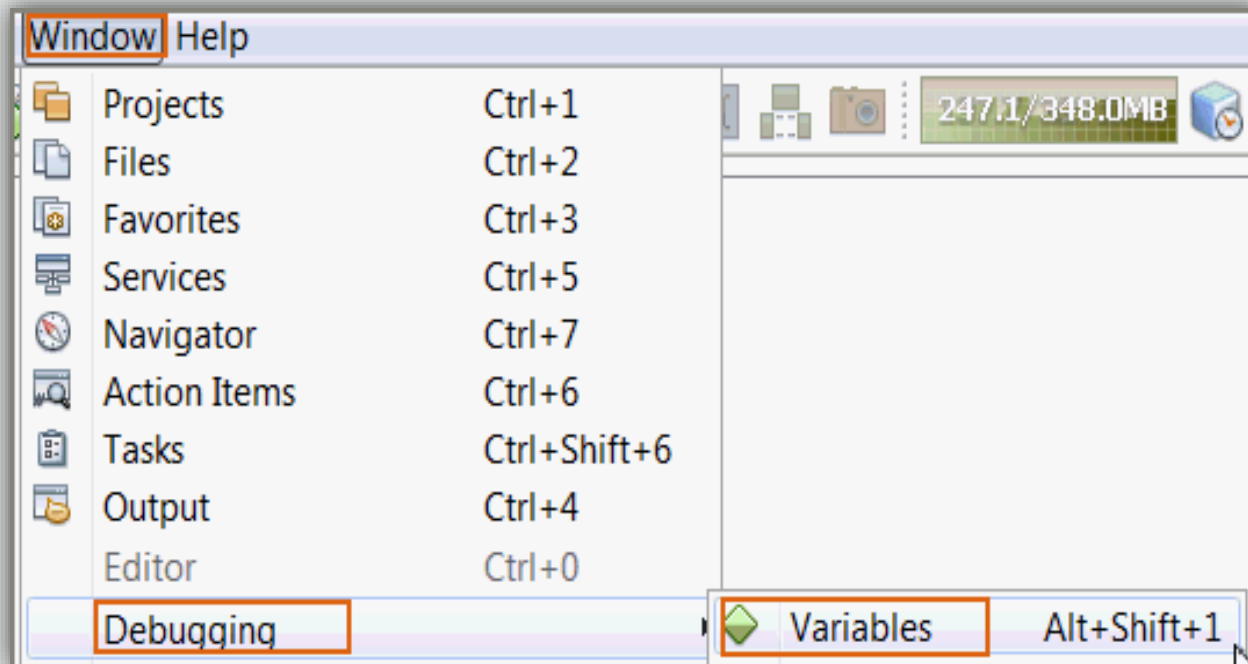
# Exercise 2: Scenario

- 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: x1 and y1
  - Gas station's location: x2 and y2
  - Speed of the car

- You want to compute the time it will take for the car from its current position (x1,y1) to reach the gas station (x2,y2)

- A Java program to compute the time by using the time=distance/speed formula is available in the ComputeTime project

# Exercise 2

- Add the file `ComputeTime.java` to the project you created for exercise 1

- Observe `ComputeTime.java`

- Run the program with your IDE's 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 your IDE's debugging features, you identified the bug:

| 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 |

Variables ✕ | Output

- 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

**ORACLE**
Academy

# Identifying the Potential Bug

```java
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);

}//end method main

static double getDistance(double x1, double x2, double y1,double y2){
    return Math.sqrt((x1 - x1) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}//end method getDistance
```

**Potential bug**

ORACLE
Academy

JavaAPCSA 8-4
Debugging Concepts and Techniques

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
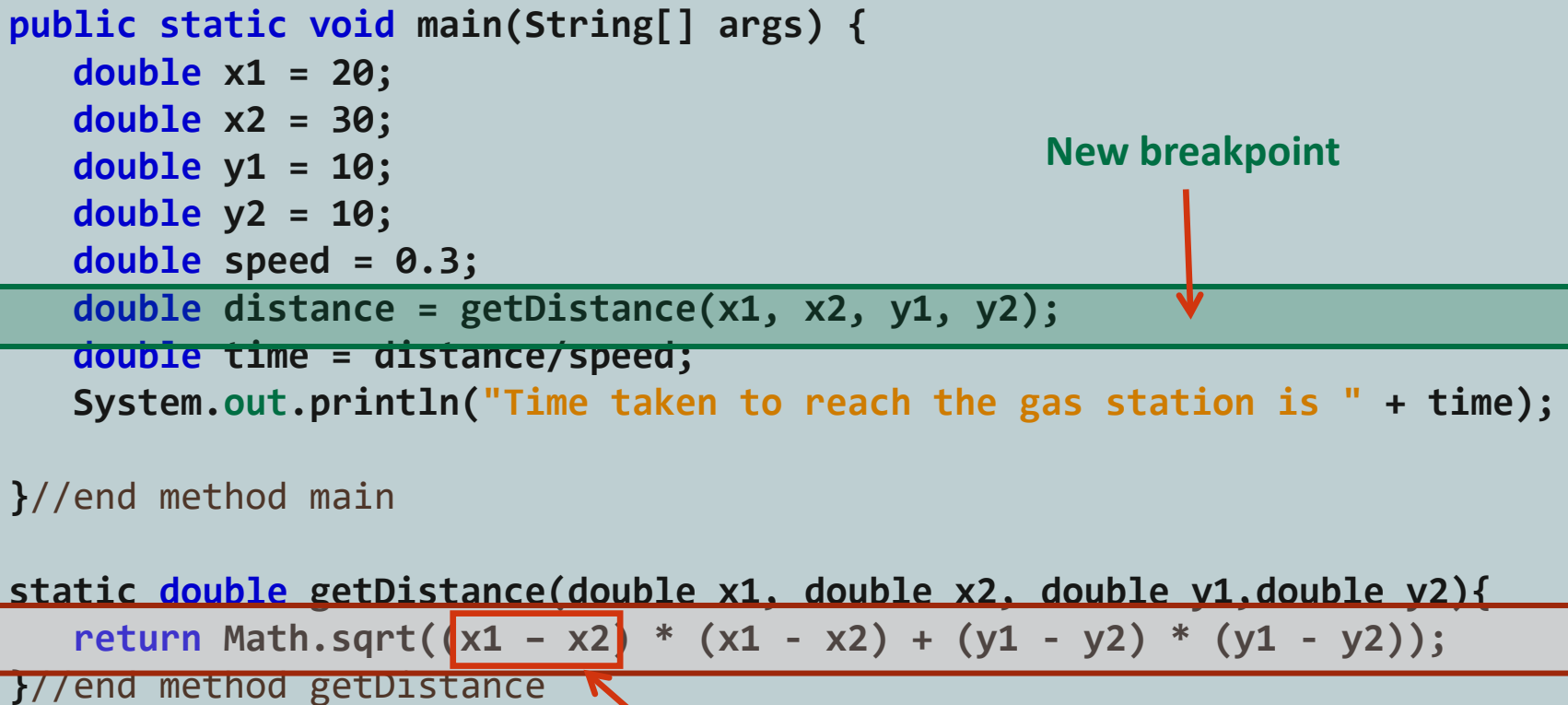
# Fixing the Bug

- 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
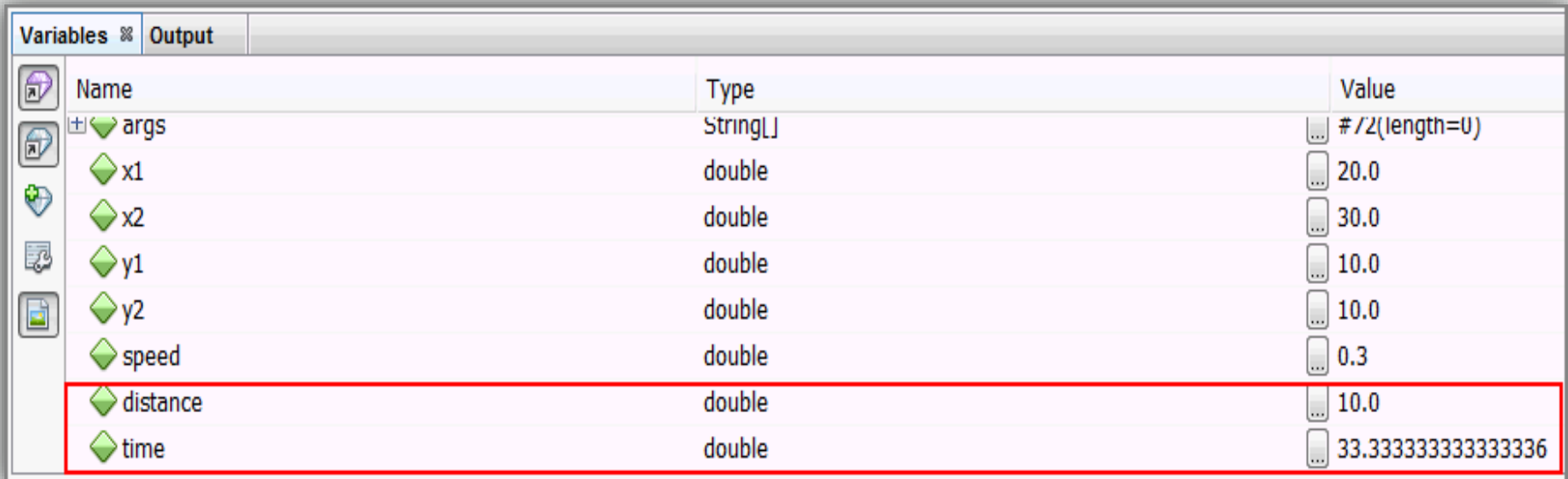
# Rerunning the Debugger

```java
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);

}//end method main

static double getDistance(double x1, double x2, double y1,double y2){
    return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}//end method getDistance
```

**New breakpoint**

**Modified code**

**ORACLE**
Academy

# Observing the Variables

- 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

| Variables ⚹ Output | | |
|---|---|---|
| **Name** | **Type** | **Value** |
| ⊞ 🔶 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 | 10.0 |
| 🔶 time | double | 33.333333333333336 |

ORACLE
Academy

# 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;`

ORACLE
Academy

# 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>
}
```

**ORACLE** Academy

# 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) I
  - 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
}//endif
```

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

# Exercise 3

- Add the file `BoundaryTesting.java` to the project you created for exercise 1

- 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
    - Using your IDE's debugger
  - Apply some debugging tips and techniques

ORACLE
Academy