Debugging Operations

by Sophia

WHAT'S COVERED

In this lesson, you will learn about some of the common errors with operations. Specifically, this lesson covers:

1. The Most Common Java Syntax Error

Before discussing common programming errors related to Java data types and operators, there is one error that is especially important. It involves leaving off the semicolon at the end of a statement. Think of a statement as an instruction telling the program what to do. It is a command or declaration.

Reminder: When reviewing output and errors, your IDE might display error messages in different colors or in plain white text, as shown in the tutorial examples. Focus on the information provided by the error message itself, as the color is not important.

KEY CONCEPT

In Java, not every line ends with a semicolon, but every statement does.

The missing semicolon at the end of the variable declaration in the following code generates a syntax error:

```
public class VariableErrors {
  public static void main(String[] args) {
    String greeting = "Hello, world!"
    System.out.println(greeting);
  }
}
```

The result of VariableErrors.java should look like this:

```
VariableErrors.java:3: error: ';' expected
    String greeting = "Hello, world!"
                                     ^
1 error
error: compilation failed
```

Putting a semicolon at the end of the declaration statement will solve the problem.

EXAMPLE

```
String greeting = "Hello, world!";
```

2. Common Variable Errors

The debugging process is similar to testing code. However, rather than just identifying errors and issues, the objective is working to fix them as you find them. At this point, 'variable names' is a common syntax error that you will most likely encounter early. Some common issues are:

- Using "total amount," "odd~job," and "US$" as variable names. These contain illegal characters such as a space, a tilde (the ~ symbol), and a dollar sign ($) character.
- Spelling or capitalizing variable names differently, such as mixing up *amount*, *Amount*, and *amt*.
- Repeating the data type after a variable is declared (the first time the variable is used).
- Using Java keywords, such as *new* and *class* as variable names. These are keywords that are part of the Java language itself.

Remember, the rules for a variable name require that it:

- Must start with a letter or an underscore character.
- Cannot start with a digit.
- Can only contain letters, numbers, and the underscore character.
- Is case sensitive.

Consider if you put a space in a variable name; Java thinks it is two operands without an operator:

```
public class VariableErrors {
 public static void main(String[] args) {
   String course name = "Intro to Java";
 }
}
```

The result should look like this:

```
VariableErrors.java:3: error: ';' expected
   String course name = "Intro to Java";
                ^
1 error
error: compilation failed
```

From the compiler's perspective, this looks like a declaration of a `String` variable named `course` without an initial value, so a semicolon would be expected after `course`.

EXAMPLE Here is an example of the `String` variable named `course` without an initial value:

```
public class Main {
    public static void main(String[] args) {
        // Declare a String variable named course without an initial value
        String course;
  }
}
```

2a. Java Keywords

IN CONTEXT

Java has 50 **keywords** that are used to recognize the structure of a program. Since they are part of the language, they cannot be used as identifiers (variable names). These are the keywords:

```
abstract continue for new switch
assert default goto package synchronized
boolean do if private this
break double implements protected throw
byte else import public throws
case enum instanceof return transient
catch extends int short try
char final interface static void
class finally long strictfp volatile
const float native super while
```

If we tried to use one of the reserved words as a variable name, we would receive an error message.

```
public class VariableErrors {
 public static void main(String[] args) {
   String class = "Intro to Java";
 }
}
```

Trying to compile this code results in multiple errors. In this case, the error messages are not easy to interpret because using a keyword as a variable name keeps the compiler from parsing the code correctly:

```
VariableErrors.java:3: error: not a statement
   String class = "Intro to Java";
   ^
VariableErrors.java:3: error: ';' expected
   String class = "Intro to Java";
         ^
VariableErrors.java:3: error:  expected
   String class = "Intro to Java";
               ^
VariableErrors.java:5: error: reached end of file while parsing
}
```

The runtime error that you will most likely make would be to use a variable before assigning it a value. You can run into this error if you had a typo in the variable name as well:

```
public class VariableErrors {
 public static void main(String[] args) {
   double principal = 327.68;
   double rate = 0.05;
   double interest = principle * rate;
 }
}
```

In this case, compiling the code results in a clearer error message:

```
VariableErrors.java:5: error: cannot find symbol
   double interest = principle * rate;
                     ^
  symbol:   variable principle
  location: class VariableErrors
1 error
error: compilation failed
```

The variable was declared with the name `principal`, but the identifier is misspelled as "principle" in the calculation:

```
public class VariableErrors {
 public static void main(String[] args) {
   double principal = 327.68;
   double rate;
   double interest = principal * rate;
 }
}
```
In this case, `rate` has been declared but not assigned a value. Trying to use an uninitialized variable results in an error:

```
src/main/java/VariableErrors.java:5: error: variable rate might not have been initialized
   double interest = principal * rate;
                                 ^
1 error
error: compilation failed
```
Don’t forget that variable names are case sensitive. For example, `Principal` and `principal` would not be the same.

```
public class VariableErrors {
 public static void main(String[] args) {
   double principal = 327.68;
   double rate = 0.05;
   double interest = Principal * rate;
 }
}
```
If you did have the variable defined in both ways in a program, the issue would become a logical error:

```
src/main/java/VariableErrors.java:5: error: cannot find symbol
   double interest = Principal * rate;
                     ^
  symbol:   variable Principal
  location: class VariableErrors
1 error
error: compilation failed
```
You would have to catch it, as you won’t get an error while compiling. Consider the following compiling error:

```
public class VariableErrors {
  public static void main(String[] args) {
    int Rate = 100;
    double principal = 327.68;
    int rate = 0;
    double interest = principal * rate;
    System.out.println(interest);
  }
}
```
In this case, we have `rate` and `Rate`. If we intended to use the `Rate` of 100 but instead used the `rate` of 0, this would run but return the interest value of 0 instead of 32768.

The results would look like this:

```
0.0
```

TERM TO KNOW

Java Keywords
A reserved word with a predefined meaning in the programming language.

3. Common Operations Errors

Many of the common errors in operations occur when you are mixing various data types as part of the operations.

You may have some numbers that you wanted to add, but may have accidentally used quotes around them.

```
public class VariableErrors {
 public static void main(String[] args) {
   String var1 = "3";
   String var2 = "4";
   String var3 = var1 + var2;
   System.out.println(var3);
 }
}
```
The results should look something like this:

```
34
```

Since the variables are strings rather than integers, the `+` operator **concatenates** the values rather than adding them arithmetically. If they are to be added, the variables must be declared as integers (and without quotation marks around them).

Declaring variables as integers looks like this:

```
public class VariableErrors {
  public static void main(String[] args) {
    int var1 = 3;
    int var2 = 4;
    int var3 = var1 + var2;
    System.out.println(var3);
  }
}
```

The results should look like this:

```
7
```

Other problems can exist due to a mixing of two different data types. This can create an unexpected result if one of the two values is a String. This result occurs because any type can be converted to a String. This is the reason that the + sign can be used to concatenate strings and numeric values when using System.out.print() and System.out.println()).

```
public class VariableErrors {
 public static void main(String[] args) {
   int var1 = 3;
   String var2 = "4";
   String var3 = var1 + var2;
   System.out.println(var3);
 }
}
```

The results should look like this:

```
34
```

Something similar happens when using + with `boolean` and `String` values.

```
public class VariableErrors {
  public static void main(String[] args) {
    boolean var1 = true;
    String var2 = "4";
    String var3 = var1 + var2;
    System.out.println(var3);
  }
}
```

The results should look like this:

```
true4
```

Note that two `char` values can't be concatenated to create a `String`, as the following code attempts to do:

```
public class VariableErrors {
  public static void main(String[] args) {
    char var1 = 'A';
    char var2 = 'B';
    String var3 = var1 + var2;
    System.out.println(var3);
  }
}
```

Running this code results in an error:

```
VariableErrors.java:5: error: incompatible types: int cannot be converted to String
    String var3 = var1 + var2;
                       ^
1 error
error: compilation failed
```

You learned in 1.2.2 that char values are handled internally as Unicode values (which is why the error message mentions the int type).

Another common error is a semantic error in the order of operations. For example, if we had 3 grades and we wanted to calculate the average, we would want to add the three grades and divide it by 3. We may do something like the following:

```
public class VariableErrors {
  public static void main(String[] args) {
    float grade1 = 86;
    float grade2 = 100;
    float grade3 = 72;
    float average = grade1 + grade2 + grade3 / 3;
    System.out.println("Average = " + average);
  }
}
```

The results should look like this:

```
Average = 210.0
```

That’s definitely an error due to the order of operations. In this case, the error is that grade3 / 3 is evaluated first. Then, grade1 is added to grade2 and the result of the grade3 / 3. Parentheses are needed to ensure that the grades are added together first before dividing them all by 3:

```
public class VariableErrors {
  public static void main(String[] args) {
    float grade1 = 86;
    float grade2 = 100;
    float grade3 = 72;
    float average = (grade1 + grade2 + grade3) / 3;
    System.out.println("Average = " + average);
  }
}
```

The results should look like this:

```
Average = 86.0
```

It is important to understand that the compiler and the Java Virtual Machine have no way to know what you actually meant to write. You do not get any error messages. You simply get the wrong answer. It is always good to validate and double check any calculations.

TERM TO KNOW

Concatenate
Joining two strings together using the addition (+) operator.

SUMMARY

In this lesson, you have seen examples of the most common errors made when working with Java operators and data types, including **the most common Java syntax error**, which involves leaving off the semicolon at the end of a statement. You have also seen examples of common logical errors, including **common variable errors** and **common operations errors** that may crop up in code that is written without due care. This included using **Java keywords**, which cannot be used as identifiers (variable names). These basics will help you successfully debug problems that may occur in your code as you move ahead with Java.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Concatenate

Joining two strings together using the addition (+) operator.

Java Keywords

A reserved word with a predefined meaning in the programming language.