INTRODUCTION TO COMPUTER SCIENCE I
LAB 2: ERRORS AND ARITHMETIC
Friday, February 7, 2020

1 Introduction

There are two parts to this lab. In the first part, you will get some practice *debugging* programs—that is, fixing errors in your code. In the second part, you will write a program to do some geometric computations.

To prepare you for the first part of the lab, you first need some additional information about what actually happens when you click the green arrow in IntelliJ to run your program. There are several steps that happen to translate the Java code that you write into a form that's usable by the computer. The first step is called *compiling* your code. This translated your Java code into an intermediate form called bytecode. When you then *run* the program, another program called the Java Virtual Machine is responsible for further translating your program's bytecode into machine code, which then runs on your computer. When you click the green arrow in IntelliJ, IntelliJ first compiles your program and then, if the compilation is successful, runs it.

There are three types of errors that you can run into when programming:

1. **Compiler errors** are errors in the syntax of your code that can be identified when the compiler runs. If your code has a syntax error the compiler won't be able to recognize your program as valid Java code, and it will tell you so.

2. **Runtime errors** are errors that can't be caught by the compiler, but will cause the program to fail while it is running.

3. **Logical errors** are often the hardest to catch, and the most interesting. Your program might compile successfully and run without crashing, but still not do what you intended it to do. This means that you've made a logical mistake when planning out what your code should do.

All of the errors you'll see in this lab are compiler errors. We'll get lots more practice with runtime errors and logical errors as the course progresses!

2 Understanding Error Messages

Work with a partner. You'll submit one set of files for both of you.

Start by opening up IntelliJ and creating a new project for this lab. Click File → New → Project, then click through to create your project. The only time you won't just click "next" is on the screen that asks you for a project name; here, you can call your project `lab2`.

Go to the course web page and click the link labeled "Error files." This should take you to a page that has links to 12 files whose names start with Err and end with .java. Each of these files has at least one syntax error in it. Your job is to figure out what these errors are and to correct the errors.

One by one, do the following for each of these files:

1. Import the file to IntelliJ. To do this, first create a new Java class by right clicking the "src" folder (if you don't see that right away, click the little arrow next to "lab2" to expand), then select "New" and then "Java class." Give your class the name corresponding to the Err file you're importing, i.e., if you're working on Err1.java, give your class the name Err1.

2. Open the corresponding Err file from the course web page, copy its contents, and paste into your Err file in IntelliJ.

3. Click the green arrow to compile and run the program.

4. The program will not run, and you'll see an error message in the "Messages" window at the bottom of IntelliJ. Read the error message and see if you can figure out what it means. Then try to correct the error in the program, and rerun the program to see if your correction was successful.

5. **After** you have corrected the error, look at the explanation of the problem (see below) and make sure you understand what is happening.

Err1.java

In this example, the compiler gives you the answer. It expects a '`;`' and the compiler tells you exactly where it should be: the numbers in parenthesis right after the word Error tell you the line number and the character within the line where the error occurred.

Err2.java

Again, the compiler tells you exactly what is wrong. It expects a '`)`', and it points to the place.

Err3.java

This time the compiler is reasonably clear about the error, but it is not pointing to the location of the error. Rather, it is pointing to the place where the unclosed String starts. A "literal" is a constant value of a particular type, as opposed to a variable.

Err4.java

This error message is perhaps not as clear. Why is it complaining about a "possible" problem? It turns out that it is only a possible problem because the `double` might contain an integer value, in which case we are fine. But the `double` might contain a value that is not an integer, in which case it cannot be assigned to an `int` variable. There are several possible ways to fix this, including declaring `i` to be a `double` or declaring `j` to be an `int`. It depends on what you really intended.

Err5.java

In this case the compiler is very clear both about the nature of the problem and the location. It cannot find the symbol `j` because no variable called `j` was ever declared.

Err6.java

Again, this is straightforward. The compiler tells you that a variable `i` already has been defined.

Err7.java

Here the compiler seems to be hedging: it tells us that variable `i` "might" not have been initialized. What javac means is that it is not sure that `i` has been initialized (in this case it is clear from looking at the code that `i` has not been initialized; next week we will see some examples of programs in which it will be less clear).

Err8.java

This error message (class, interface, or enum expected) is less helpful. The location is correct, but the message is off base. The compiler is confused by the extraneous word "`Public`", which is does not recognize as a misspelling of "`public`". Java is case sensitive!

Err9.java

This may seem like an oddly phrased error message (class Err9a is public, should be declared in a file named Err9a.java). The real error is that the name of our file is Err9.java, and the name of the class contained within this file must also be Err9 (alternatively, the file name and the class name could be Err9a.java and Err9a respectively). The reason for the phrasing of the error message is that it is possible to have a non-public class, for which there are different rules. You do not need to know about this yet, but you should be aware that it may be difficult to understand some of the error messages at your current level of knowledge.

Err10.java

We see two errors, so let's handle the first one first. This is a little misleading because it points to the right place, but has the wrong expectation about what it wants to see. What is missing is a '{', not a ';'. Once you have corrected the first error, the other error message should disappear: the additional error was not real, and only appeared because the first error caused the compiler to be confused about how to read the rest of the program. Advice: when in doubt, fix the first error, recompile, and see what happens.

Err11.java

Indeed, ')' is an illegal start of an expression, but the error message isn't particularly helpful. Instead, the problem is that javac expects an expression after the '+', but instead there is a ')'.

Err12.java

The first error message is technically correct; "`System.out.println("hello");`" is not a statement, but the compiler doesn't tell you that the problem is that there is a '{' instead of a '('. Once we fix this error, the compiler comes up with an entirely different error. This is common. You fix an error and the compiler can understand the program better, so it finds another error.

When you are done, **put both of your names in a comment at the top of Err1.java** and save the file. If you do not put your name in the file and you are working from your partner's account, I will have no way of knowing that you have completed this part of the assignment.

Submit all of your java files to the "Lab 2A: Errors" assignment on Moodle. You should only make one submission for both you and your partner.

3 Geometry Computations

Work **on your own** for this part of the assignment.

Create another Java class in your lab2 project, and call this one Lab2B. Copy the Lab2B code from the course web page into your Lab2B file in IntelliJ.

Compile and run the Lab2B program using the green arrow. When you run the program, it should prompt you to enter the base and height of a rectangle, and then it should print out the area.

You may notice that if you type in something that is not a number when the program asks you to enter the base or the height, the program will stop running and an error message will print. This is an example of a runtime error: the compiler didn't know that the user wasn't going to type in a number, but when the program ran and the user didn't behave as expected, the program couldn't continue running. We will see later how to handle this; for now you can assume that your user is well behaved and will enter sensible numbers.

Your job is to calculate various geometric properties of different shapes. The table on the next page lists the shapes and the values you should compute, as well as the data you will need to read from the keyboard. For each shape, you should read from the keyboard the values listed in the "Input" column, compute all of the values listed in the "Value" column using the given formulas, and then print the results. For example, the output for the right triangle might look something like:

```
Please enter the length of the base:
3
Please enter the height:
4
Right triangle area is: 6.0
Right triangle hypotenuse is: 5.0
Right triangle perimeter is: 12.0
```

Your program should read in *new* input values for each shape (i.e., the user should be able to enter

one base and height for the rectangle, then a different base and height for the triangle). So in the end, your program should read in 9 different values from the keyboard (one for each of the quantities in the "input" column) and print out 11 different results (one for each of the quantities in the "value" column).

Shape	Input	Value	Formula
Rectangle	Base (b) Height (h)	Area (A)	$A = bh$
		Perimeter (P)	$P = 2b + 2h$
Triangle	Base (b) Height (h)	Area (A)	$A = \frac{bh}{2}$
Right Triangle	Base (b) Height (h)	Area (A)	$A = \frac{bh}{2}$
		Hypotenuse (c)	$c = \sqrt{b^2 + h^2}$ *
		Perimeter (P)	$P = b + h + c$
Circle	Radius (r)	Area (A)	$A = \pi r^2$ **
		Circumference (C)	$C = 2\pi r$
Regular polygon	# sides (n) Side length (l)	Perimeter (P)	$P = nl$
		Apothem (a) ***	$a = \frac{l}{2\tan(\pi/n)}$ ****
		Area (A)	$A = \frac{aP}{2}$

***Note: you can use** `Math.sqrt(x)` **to compute the square root of a variable** `x`**.**
****Note: you can use** `Math.PI` **to get the value of** π**.**
*****Note: the apothem is the distance from the center of any side of a regular polygon to the polygon's midpoint.**
******Note: you can use** `Math.tan(x)` **to compute the tangent of a variable** `x`**.**

When you are done, submit your program to the "Lab 2B: Geometry" assignment on Moodle.

This assignment is due on Thursday, February 13, 11:59 pm.