CS112 – Spring 2024

Lab03

Instructor: Paul Haskell

INTRODUCTION

In this lab you will write some programs that use Java's built-in numerical data types. You will explore the limits of the data types. Although people who use computers to do mathematical computations often expect perfect results, there are limitations in both the range and accuracy of computer mathematics. A few times when programmers forgot these limitations, the result was a spectacularly famous failure.

Explosion of the Ariane 5

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after lift-off. The rocket was on its first voyage, after a decade of development costing $7 billion. The destroyed rocket and its cargo were valued at $500 million. A board of inquiry investigated the causes of the explosion and in two weeks issued a report. It turned out that the cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than … the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.[1]

Limits of Fixed Point Data Types

Java's built-in data types cannot represent arbitrary large (positive or negative) numbers, and the "smaller" the data type, the smaller the limits. The first assignment in this lab is to calculate the limits for byte, short, and int data types.

One way to find these limits is, for each data type, to create a variable `var1` and initialize its value to 0. Then create a second variable `var2` of the same type.

Now, repeatedly, set `var2` to the value of var1, increase `var1` by 1, and check if `var1` is greater than `var2`. Eventually it will not be greater! Use a `while()` loop to do the checking and incrementing.

How can you tweak the above recipe to find the limits in the negative direction?

For this lab you shall write a program called **Limits.java** that finds and prints the largest positive and "most negative" values that <u>are</u> properly supported by the `byte`, `short`, and `int` data types. (Don't report <u>one past</u> the last supported values.)

[1] https://www-users.cse.umn.edu/~arnold/455.f96/disasters.html. Downloaded 29 June 2022.

The output from your **Limits.java** shall be as follows:

Maximum byte value is <<value>>
Minimum byte value is <<value>>
Maximum short value is <<value>>
Minimum short value is <<value>>
Maximum int value is <<value>>
Minimum int value is <<value>>

Limits of Floating Point Data Types

There are entire graduate courses devoted to the study of the limits and accuracy of mathematical computations done with floating point data types. For this lab, we will take on the first homework problem from the course I took on this subject: we will find (approximately) the smallest positive nonzero number that can be represented with the `float` and `double` data types.

To do this, initialize a `float` (and later a `double`) variable to some positive value, say 1.0. Keep on dividing the number in half and comparing the result against zero. Once the computer cannot tell that the value of your variable is greater than zero, then you have your answer (the previous value, which can be distinguished from zero).

For this lab you shall write a program called **FloatLimits.java** that finds the smallest positive nonzero `float` and `double` values and prints them out. The output from **FloatLimits.java** shall be as follows:

Smallest positive float is <<answer>>
Smallest positive double is <<answer>>

Data Conversion

Here you will demonstrate some of the changes that happen when we convert "larger" data types to "smaller" ones. You shall write a program called **DataConvert.java** that will do such conversions and output the results.

First, initialize a float variable to 2.5. Then create an int variable and initialize the int with the float, via casting. Print "2.5 cast to int gives <<value of the cast>>"

Now set the float variable to -4.5 and then set the int variable to this value with another cast.
Print "-4.5 cast to int gives <<value of the cast>>"

Now create a double variable and initialize it to (1.0/3.0). Assign the value of your double to your float variable via a cast. Print "double 1/3 = <<value of double>> but float 1/3 = <<value of float>>"

Assign your int variable the value 256. Create a byte variable and initialize it with the value of the int, using a cast. Print "byte value of 256 is <<value you got>>"

Now set the integer to 257, set the byte to the int value (via a cast, of course), and print
"byte value of 257 is <<value you got>>"

Next set the integer to 258, set the byte to the int value (via a cast, of course), and print
"byte value of 258 is <<value you got>>".

Finally set the integer to 511, set the byte to the int value, and print
"byte value of 511 is <<value you got>>".

Nested for() loops

Here's a pretty simple final challenge: create a program called **Triangle.java**. This program should use nested for() loops to print out the following pattern of `int` values:

1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
…
15 30 45 60 75 90 105 120 135 150 165 180 195 210 225

Note that the "number of numbers" in each line is equal to the first value in the line. The last line has 15 numbers.

Advice

Please treat the descriptions above as a specification of required behavior.

- You should use the exact filenames specified
- You should use proper class names i.e. in a file called MyProgram.java, the class should be called "MyProgram"
- Use correct capitalization
- Your text printouts should match what is specified exactly: no extra punctuation, no missing or extra spaces, etc.

Put useful comments into your code.

Put your programs into a subdirectory called **Lab03** inside your **MyWork** directory, and remember to push your **Lab03** to GitHub before the deadline. This assignment must be turned in before Sunday Feb 4 at 11:59pm.

Conclusion

In this lab you developed some Java programs on your own, from scratch. You saw some of the limitations of Java's built-in numerical data types. And you wrote a program that performed useful mathematics, to compute the natural logarithm of a number with and without using the built-in Math library.

Rubric

Each program is worth 10 points

- 2 points for a program in the proper directory, with the proper name, that compiles and runs.
- 6 points for correct output.
- 2 points for “good coding style”. For this lab, good coding style means:
    - Some useful comments at the top of each file describing what the program does
    - Some useful comments by key statements in each program describing what the code does
    - Useful variable names
    - Reasonably organized layout of the software