

Objectives

After completing the lab you should

1. Be familiar with integer overflow
2. Be familiar with rounding errors
3. Be able to evaluate expressions involving integer division
4. Be able to evaluate expressions involving the modulus operator %
5. Be familiar with using the Scanner class
6. Be able to format floating point numbers
7. Be able to evaluate expressions that involve mixed mode arithmetic.

References: The following references will be used throughout the lab.

- *Course website:* <<http://www.eg.bucknell.edu/~csci203/>>
- *Java API website:* <<http://java.sun.com/javase/6/docs/api/>>
- Chapter 4 of Big Java

Preparation

Prepare for the lab by completing the following steps.

1. Start Eclipse.
2. Create a new Java project named `lab07-xyz01`, where `xyz01` is your login name.
3. Create the `readme.txt` file for this lab, insert the banner file with proper updates.

Introduction

In today's lab we will discuss several issues that arise when doing mathematics in Java. Begin by creating a new class named `Mathematics` with a `main` method where you will do exercises 1 through 6.

Overflow

If you declare `i` to be an `int`, the largest value it is capable of holding is $2^{31} - 1$ which is equal to 2,147,483,647. (See page 135 of your text.) If you try to store something larger in `i`, Java will *not* give you a warning. Instead, it just gives you the wrong answer!

Exercise 1

Create an integer variable `i` with the following value.

```
int i = 987654321 * 1000;
```

Print the value of `i` and the mathematically correct value.

Repeat the example using doubles. That is, create a `double x`.

```
double x = 987654321.0 * 1000.0;
```

Print the value of `x` and the expected value.

Copy the output into your `readme.txt` file and state whether either answer is correct.

Rounding Errors

If I were to ask you to write the decimal representation of $1/3$, you would write

$$1/3 = 0.3333\dots$$

where the ellipsis indicates that the digit 3 repeats forever. If you truncate the decimal representation so that there is a finite number of digits, you have introduced a *rounding error*.

When you store a double on the computer, Java truncates the number to about 15 or 16 decimal places so that it fits in memory. The problem is made worse by the fact that the computer uses a base 2 representation for numbers. This means that some numbers that have a finite representation in decimal, have an infinite representation when converted to base 2. A rounding error occurs almost every time you store a `double` on the computer, or perform a calculation with doubles!

Exercise 2

Create a `double f` as shown.

```
double f = 4.35 * 100.0;
```

Print the value of `f` along with the expected value. Copy the output to your `readme.txt` file and explain what happened.

Conversions

Converting from `int` to `double` is fairly easy in Java. You can do it with a simple assignment. Try the following example.

```
int j = 435;
double y = j;
```

Exercise 3

Type in this example and print the value of `y` along with the expected value. Copy the output into your `readme.txt` file and say whether you received what you expected.

Exercise 4

Java intentionally makes it more difficult to convert from a `double` to an `int`. The reason is that a `double` can be as large as 10^{308} , while the maximum value for an `int` is about 2×10^9 . If you're converting from a `double` to an `int`, there's a good possibility of losing information.

Try to assign the variable `f` to `i` in your program.

```
i = f;
```

Copy the error message you receive into your `readme.txt` file.

If you want to convert from a `double` to an `int`, you must use a *cast*. (We used a cast to convert from `Graphics` to `Graphics2D`.) The cast forces the conversion to occur.

```
i = (int) f;
```

Print out the value of `i` and the expected value. Put the output in your `readme.txt` file and say whether you got what you expected. Explain what happened.

Static Methods

Java has a class named `Math` that contains many useful mathematical operations. For instance, there is a method called `sqrt` that takes the square root of a number.

This method is a *static* method meaning that it does not have an object as the receiver of the message. Instead, you invoke the method by preceding the method name with the name of the class. Here is how you would compute $\sqrt{2}$.

```
double squareRoot2 = Math.sqrt(2);
```

You can compute x^y with `Math.pow(x, y)`.

Exercise 5

Use `Math.pow` to square `squareRoot2` and print the result and the expected value. Put the output in your `readme.txt` file and say whether you got what you expected.

You can use the method `Math.round` to round a double to the closest integer. Use `Math.round` to round `f` to the closest integer and print the result along with the expected result. Note that `Math.round` returns a `long` integer when its parameter is a `double`. Copy your output into your `readme.txt` file.

Page 150 of Big Java lists many of the static methods that are available in the `Math` class.

The `Math` class also has some useful constants. If you need the value of π in a calculation, use `Math.PI`. Similarly, the value of e is available as `Math.E`.

Formatting Numbers

Java's default behavior is often not acceptable when printing numbers. For example, consider the following code.

```
double change = 2.10;
System.out.println("Your change is $" + change);
```

This will print

```
Your change is $2.1
```

which, although mathematically correct, is not how one expects dollar amounts to be formatted.

Java has a method `printf` which solves this problem. When you use `printf`, you must provide a *format specifier*. To print our previous example in a more pleasing manner, we will specify that the number has width 4 and 2 places after the decimal.

```
System.out.printf("Your change is $%4.2f\n", change);
```

This produces the desired output.

```
Your change is $2.10
```

Note that `printf` does not automatically print an end of line character. If you want one, you must say so explicitly as I have done in this example.

Exercise 6

There is a list of format specifiers on page 168 of Big Java. Use the information on this page to determine which format specifier you need to print the integer 1,234,567 with commas, and add the necessary code to your program. Add the output to your `readme.txt` file. Your output should look like this.

```
Testing printf: 1,234,567
Expected:      1,234,567
```

Be sure to put a new line character at the end.

Mixed Mode Arithmetic

Mixed mode arithmetic is arithmetic that is done between numbers of different types. For example, adding an integer and a double (a number with a decimal point.) Mathematically, it makes no difference if the numbers are different types, but on a computer the results can be surprising.

Import the file `FahrCels.java` from

```
~csci203/2009-fall/student/labs/lab07
```

The following exercises are designed to get you familiar with mixed mode operations and understand the principles and issues behind these operations in a programming language such as Java.

Include the answers to the following exercises in your `readme.txt` file. Notice that the word “why” appears in each question. Be sure to answer appropriately.

Exercise 7

Run the `FahrCels` program, entering the value 40 for both temperatures. Why does the program return two different answers?

Exercise 8

If the output expressions in the program are changed as follows

```
System.out.println(intFahr + " = " + (intFahr - 32) * (5 / 9)
    + " Celsius");
....
System.out.println(doubleFahr + " = " + (doubleFahr - 32.0) * (5 / 9)
    + " Celsius");
```

the value printed by both statements will be 0. Why is this?

Exercise 9

What is printed (and why?) if some parentheses are removed from the output expressions? In other words, the output statements become the following:

```
System.out.println(intFahr + " = " + (intFahr - 32 * 5 / 9)
    + " Celsius");
....
System.out.println(doubleFahr + " = " + (doubleFahr - 32.0 * 5 / 9)
    + " Celsius");
```

Note carefully the placement of the parentheses!

Exercise 10

If the output statement using `intFahr` is changed as follows

```
System.out.println(intFahr + " = " + (intFahr - 32.0) * 5 / 9
    + " Celsius");
```

what will the output be if the value of 40 is entered? Why?

Dialog Boxes for Input and Output

In this section you will change your Fahrenheit to Celsius program so that it uses dialog boxes for input and output.

Before introducing the dialog boxes, clean up the code a bit. Remove the code that does the conversion using integers. It should be clear that integers are not the appropriate primitive type to be using for this calculation. You can also remove references to the `Scanner` class.

You should also remove the *magic numbers* from the code. Highlight the 32 and select **Refactor** → **Extract Constant...** Name the constant

```
FAHRENHEIT_FREEZING_POINT
```

Create a constant for $5/9$ too. First surround it with parentheses and make sure you have decimal points.

```
(5.0 / 9.0)
```

Then, highlight the expression and extract it to a constant. Call it

```
CONVERSION_FACTOR
```

Test your program and make sure it still works.

Exercise 11

The static method `JOptionPane.showInputDialog` will present an input dialog. For example, to prompt for the temperature in Fahrenheit you could use the following statement.

```
String temperature = JOptionPane
    .showInputDialog("Fahrenheit temperature:");
```

Note that it returns a string. The static method `Double.parseDouble` will convert it to a double.

```
double fahrenheit = Double.parseDouble(temperature);
```

Once you have called `showInputDialog`, you *must* exit your program using

```
System.exit(0);
```

Insert this line at the end of `main` and verify that it's still working.

The static method `JOptionPane.showMessageDialog` can be used to display the output instead of printing it in the console. Introduce a line similar to the following which will display the conversion results in a dialog.

```
JOptionPane.showMessageDialog(null, "Celsius: " + celsius);
```

What to Submit

Your `readme.txt` file should include the the results from Exercise 1 through Exercise 10. Your `FahrCels.java` file should contain your solution to Exercise 11. Make sure your `readme.txt` file is in your lab directory, and drag your lab directory to the CSCI 203 lab drop box: [CSCI203 Lab drop_box](#)