CS180 Lab 4: Numeric types and Exceptions

Chapter from textbook relevant for this lab: 3
Lab for week: 4
Lab created by: James Wilkinson

Learning objectives

1. Understand how to convert between numeric types
2. Understand how precision changes between numeric types
3. Understand how to round integers and floating point numbers
4. Understand what exceptions are and how to handle them
5. Understand how to calculate exponential
6. Understand how many small calculation errors can sum up to large errors

Files

`Lab04.student.tar.gz`

Setup

1. Create a `lab04` directory in your `CS180` directory
2. Extract the contents of the `Lab04.student.tar.gz` file into the `lab04` directory

Rounding Practice

Learning Objectives

a. Understand how precision changes between numeric types
b. Understand how to convert between numeric types
c. Understand how to round integers and floating point numbers
d. Understand what exceptions are and how to handle them

Notes

Rounding down can be done with `Math.floor()` . `Math.floor()` will return a double whose result has been rounded down to the nearest integer. i.e. `Math.floor(32.4)` will return `32.0`.

Rounding up can be done with `Math.ceil()` . `Math.ceil()` will return a double whose result has been rounded up to the nearest integer. i.e. `Math.ceil(32.4)` will

return `33.0`.

Exercises

The answers for the following exercises should be typed into `Answers.txt`.

a. Compile and run the `Lab04Rounding.java` program. Observe its output.

b. (Exercise 4.1) Modify the program such that the **final calculation** is rounded **up** to the **closest integer**, compile, run and type the number 1 at the prompt. Copy the final line of output into your Answers.txt file.

c. (Exercise 4.2) Change the variables such that the result of **every calculation** is rounded **down** to the **closest integer**. Compile, run and type the number 10 at the prompt. Copy the final line of output into your Answers.txt file.

d. (Exercise 4.3) The previous exercise should have given you an interesting result. This was caused by trying to divide by zero and then store that result in a double. If you had tried to divide an integer by zero, you would get a very different result. To see this, uncomment the lines **under** "// Exercise 4.3a" by deleting the `//` preceding each line, recompile and run the program with any valid input.

   i. You should have run into a divide-by-zero exception at this point. There are several lines of code called a `try-catch` block that can be used to deal with this. Comment out the lines of code for Exercise 4.3a by typing `//` in front of each line and then uncomment the lines **under** "// Exercise 4.3b" by deleting the `//` preceding each line. Compile and run the program again typing the number `10` at the prompt.

   ii. Change the code for Exercise 4.3b so that instead of printing an error and exiting, it instead sets `myVar` to `-1` inside of the `catch` block and continues. Compile, run, type the number `10` at the prompt and copy the final line of output into your Answers.txt file.

e. (Exercise 4.4) Comment out Exercise 4.3b. Now, working with the code from Exercises 4.1 and 4.2, change the variables such that the result of **every calculation** is rounded **down** to the **closest hundredth**, i.e. 0.987 = 0.98, 1.174 = 1.17, 0.38574 = 0.38. Compile, run and type the number `7` at the prompt. Copy the final line of output into your Answers.txt file.

   HINT: To do this, multiply by 100 before rounding, round down and then divide by 100.0. Note that you should use 100.0 and not 100.

Salami Slicing

Learning Objectives

a. Understand how to calculate exponential

b. Understand how many small calculation errors can add up to large errors

Notes

a. You will need to calculate an exponential. You can do this with the `Math.pow()`

method. This method returns a double and is used with the following syntax: `Math.pow(base, exponent)`

b. The formula to calculate the future value of a compounding interest balance is as follows:

```
F = P * (1 + I/R)^(R*Y)
```

where `F` is the future value,
`P` is the starting principle,
`I` is the interest rate,
`R` is the compounding periods per year, and
`Y` is the years to mature.

c. Sections that you should edit are marked `// TODO`. **Do not** edit code in other areas.

d. (optional) Visit http://en.wikipedia.org/wiki/Salami_slicing to learn more about salami slicing

Steps (be sure to run your program after each step to test for errors)

a. Compile and run the `Lab04Salami.java` program and observe its output.

b. Modify the section labeled *"Get starting parameters from user"* to get input from the user that matches the appropriate variables.
   i. A `Scanner` object has already been created for you named `scan`.
   ii. Get the interest as an integer. ie. 5% should be typed by the user as 5 without the `%` sign. You will want to save `interest` as a `double` so 5% interest would be saved as as `0.05`.

c. Calculate the `futureValue` for each account using the compound interest formula.

d. Calculate the total `futureValue` by multiplying the `futureValue` by the number of `accounts`.

e. Calculate the `adjustedValue` for each account
   i. Set the starting `adjustedValue` to equal the `principle`.
   ii. Inside the `for` loop (which you will learn more about at a later time), calculate the `adjustedValue` with the following formula:

```
A = A*(1+I/R)
```

`A` is the `adjustedValue`
`I` is the interest rate
`R` is the periods per year

f. Calculate the total `adjustedValue` by multiplying the `adjustedValue` by the number of accounts.

g. Calculate the difference by subtracting `adjustedValue` from `futureValue`.

h. Test your program with the first "No slices" test case below. Your output should match the provided output.

i. (Exercise 4.5) Run your program again with the second "No slices" test case below. Type the three missing values from the table into your Answers.txt file.

j. Inside the `for` loop, round each result **down**...
   i. ...**to the nearest penny**, compile and compare against the first "Small slices" test case. Your output should match the provided output.

(Exercise 4.6) Run your program again with the second "Small slices" test case. Type the three missing values from the table into your Answers.txt file.

ii. ...**to the nearest dollar**, compile and compare against the first "Big slices" test case. Your output should match the provided output.

(Exericse 4.7) Run your program again with the second "Big slices" test case. Type the three missing values from the table into your Answers.txt file.

Test Cases

	# Accounts	Principle	Interest Rate	Periods	Years	Future Value per Account	Adjusted Value per Account	Total amount sliced
No slices	10	500	5	4	5	**$641.02**	**$641.02**	**$0.00**
	50,000	10,000	6	12	10			
Small slices	500,000	10,000	7	4	10	**$20,015.97**	**$20,015.71**	**$131,715.93**
	5,000,000	1,000	7	12	1			
Big slices	5,000	25,000	6	12	5	**$33,721.25**	**$33,688.00**	**$166,269.07**
	25,000	5,000	6	12	5			

Grading

Criteria	**Points**
Four exercises from "Rounding Practice" [3 pts each for a total of 12 points]	
Three test cases from "Salami Slicing" [4 pts each for a total of 12 points]	
Running the final version of the Salami Slicing (Big slices) program gives correct output for the last test case [12 points]	
Total Points [36 points]	

Lab turn-in

1. At the terminal, change to parent directory of your lab 4 directory.
2. Type the following command replacing **`mylab4`** with your lab 4 directory name and **`labID`** with

your lab section's code:

```
turnin -v -c cs180=labID -p lab04 mylab4
```

3. Verify your submission by typing the following again replacing **labID** with your lab section's code:
   `turnin -c cs180=labID -p lab04 -v`

<End of CS 180 Lab 4>