CPSC 150L Lab 5
Loops

Fall 2017

1 Introduction

When programming, you will often need to perform the same operation many times over. One way to do so is to write each operation explicitly. This method, however does not scale when you do not know how many times you need to repeat the operation. Therefore, we use *loops* to perform repetitive tasks. Fork the lab 5 repository from the 150 students group on Gitlab and clone your fork. After doing so, use the `git add .`, `git commit -m "message"`, `git push` workflow to backup your code. You will submit your code to WebCAT multiple times to ensure that you get credit for each exercise.

2 Exercises

2.1 Summation

Create a class called `LoopMethods.java`. Within this class, declare the following method.

Code:

```
public static int sumSeries(int start, int end)
```

This method should take integer starting and end values (called `start` and `end` respectively) and return the sum of all integers from `start` through `end` (inclusive). You may assume that `end` $\geq$ `start`.

2.1.1 Testing

Test your code against `LoopMethodsTest.java`. You may also create a `main` that passes user input to `sumSeries` to check your results. If you need to know the correct answer, notice that

$$\sum_{i=m}^{n} i = \left(\sum_{i=1}^{n} i\right) - \left(\sum_{i=1}^{m-1} i\right) = \frac{n(n+1)}{2} - \frac{(m-1)m}{2}.$$

Exercise 1 Complete

Run:

```
git add .
git commit -m "Completed exercise 1"
git push origin master
```

2.2 Factorial

Create the following method in `LoopMethods.java`.

Code:

```
public static int factorial(int n)
```

This takes a *natural number* n (a non-negative integer) as input and returns the *factorial* of n, denoted $n!$. Calculating $n!$ is similar to calculating the sum from 1 to n, which is what

you did in the previous example. To calculate $n!$, we do the following,

$$n! = 1 \cdot 2 \cdot \ldots \cdot n$$

That is, we take the product of all natural numbers from 1 to n. Additionally, we define $0! = 1$. For example,

$$6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720.$$

The difference between this and summing from 1 to n is that we are multiplying instead of adding.

2.2.1 Testing

Test your code against `LoopMethodsTest.java`.

Additionally, you may create a `main` that passes user input to `factorial` to check your work. You may make up your own test cases (use a calculator to check your answer). Furthermore, all arithmetic for this program should use integers. This will be a problem for large n since $n!$ is a rapidly increasing function (for example, $14! = 87,178,291,200$ which is more than 40 times as large as `integer`'s maximum value of 2,147,483,647).

Exercise 2 Complete

Run:

```
git add .
git commit -m "Completed exercise 2"
git push origin master
```

---

3 Common Mistakes

Some common mistakes for this lab are as follows.

1. Ensure that you chose the right type of loop. If your loop is inherently count-controlled, use the `for` loop, otherwise use the `while` loop.
2. If you are using a count-controlled loop, make sure your bounds for the loop are correct.
3. If you need a variable outside of a loop, declare it outside of the loop.

4 Tutorial

4.1 Count-Controlled Loops

There are many ways of writing a loop, and that presents a problem for the programmer when trying to determine the type of loop to use. In this exercise, we will investigate the use of one of the more common loops, a count controlled loop.

This simply means that you create a counter that determines how many times a particular operation repeats. You have likely seen something like this before, but lets try a simple one. A count-controlled loop looks as follows.

Code:

```
int i = 1;
while (i <= 6) {
     System.out.println("Try Again");
     i++;
}
```

In this code snippet, the line with `while` in it is the entry point of the loop. The condition in the parentheses is what tells the loop whether to terminate or keep going. The loop will run while the condition in the parentheses evaluates to `true` and the loop will terminate once it

is `false`. In this case, the loop will run while `i` is less than or equal to 6 and will terminate once `i` is greater than 6.

Write a simple program and insert this code in the main routine. Guess what it will do before running it and then compile and execute it to see if your guess was correct.

Modify the number 6 to be 35 and run it again. Did it behave as expected? Making a count-controlled loop execute a particular number of times is not that hard and you can use this example as a starting point.

A simpler way of writing a count controlled loop is the `for` loop. The previous code snippet looks as follows when written with a `for` loop.

Code:

```
for (int i = 1 ; i <= 6 ; i++) {
    System.out.println("Try Again");
}
```

A `for` loop has three statements in the parentheses separated by semicolons. The first statement in the parentheses tells us that `i` is the *index* of the loop. The second statement states that the loop will execute while `i` is less than or equal to 6, and will terminate once `i` is greater than 6. The third statement tells the loop what to do to `i` at the end of each iteration; in this case, the loop will increment `i`.