INTRODUCTION TO COMPUTER SCIENCE I
PROJECT 2
Writing and calling methods

Onto our second project! We now not only know how to perform arithmetic calculations, but we also have learned a bit about creating small chunks of code to which we can assign names—chunks of code that we can then use by calling on those names. By *defining* and *calling* these *methods*, we can divide our computation into smaller pieces that we can re-use in a variety of ways. For this project, we're going to write and use methods in a few different ways.

1 Stage I: Programming methods to do unit conversions

The core of this project will be *unit conversions*—translating a measurement in one unit into some other unit. In particular, we will use a few different measurements of *length*, and a few of *time*. Here are the units we will use:

- **Measurements of length:**
  - *meter (m):* The canonical unit.
  - *foot (ft):* 0.3048 m
  - *inch (in):* $\frac{1}{12}$ ft
  - *yard (yd):* 3 ft
  - *smoot:* 1.7018 m
  - *barleycorn:* $\frac{1}{3}$ in
- **Measurements of time:**
  - *second (s):* The canonical unit.
  - *minute (m):* 60 s
  - *jiffy:* $\frac{1}{60}$ s
  - *helek:* $\frac{10}{3}$ s

1.1 Getting started with a new program

Now that you have some units with which to work, it is time to start a new program for yourself. Specifically:

1. Login to `remus/romulus`.
2. Create a directory for this project and change into it:

```
$ mkdir project-2
$ cd project-2
```

3. Copy the beginnings of a new source code file, being sure not to forget the **trailing space and period characters**:

```
$ cp ~sfkaplan/public/COSC-111/project-2/Converter.java .
```

4. Open the new source code file in *Emacs*:

```
$ emacs Converter.java &
```

You will see, in this source code file, the beginnings of a new program named `Converter`. It contains, for starters, two complete methods named `convertInchToFoot` and `convertFootToInch`. Given the conversion factor from one to the other, you should, as these methods show, easily be able also to convert in the opposite direction.

Your first task: For each of the conversions listed above, write a method to perform that conversion **and** its inverse (e.g., inches to feet **and** feet to inches). Each method should follow the same form as `convertInchToFoot`. Its caller should provide a `double` value; the method should return the conversion as another `double` value; for converting from unit *Foo* to *Quux*, the method should be named `convertFooToQuux`, using that exact form of capitalization.[1]

After writing these methods, you may wish to `test them`. It is always a good idea to determine, with pen and paper, what the results should be for a few different inputs, and these methods are no different. How do you test your methods? Write for yourself a `main` method—just as we have since the first day of class—and call on your various conversion methods. Print the results to the screen to see if they come out correctly.

2 Stage II: Your program using your conversion methods

Now that you have tested your conversion methods, let's put them to use. In order to do so, we must commence a new scavenger hunt. To proceed, you need to find two pieces of information. Specifically:

1. Go the college's *Museum of Natural History*. Near it (**not in** it, find the meteor that crashed at Canyon Diablo. The placard next to the meteor indicates that this rock created *Meteor Canyon*. Record the **diameter** and **depth** of that canyon.

[1]Why does the choice of name matter? It will later, when my program relies on your methods. If your methods are not named correctly, my program will not work correctly, and therefore, you will get stuck with that part of the project.

2. Head to the gym,[2] and find the pool. Find the *oldest team or pool record for swimming (**not** diving) amongst the men's and women's teams*. Note both the **distance of the event** and the **time** (and, just for fun, the year and the name).

Your next task: Armed with these piece of information, you should go back to your `Converter` program. Change its `main` method to do the following:

1. Allow the user to enter the **diameter and depth** of Meteor Crator in *meters*.
2. Calculate and print these lengths in *barleycorns*. Note that you **should not yourself calculate the conversion factor from meters to barleycorns.** Instead, you should **compose methods that you've written**, based on the conversion factors provided above, and let the machine do the "heavy lifting" in this conversion.
3. Allow the user to enter both the **distance and time**, using the native units of *yards* and *seconds*, of the record setting swim.
4. Calculate and print the **mean velocity of that swimmer** in units of $\frac{\text{smoots}}{\text{jiffy}}$. Again, compose the existing methods that you wrote based on the conversions listed above.

Record these two results to a precision of thousandths (three digits after the decimal point)—you will need them for the next stage.

3 Stage III: My program using your conversion methods

Your penultimate task: Now we will do something different with your program. Using some professorial kung-fu, I have written a program that can reach into your `Converter` program and use its methods. To use my program, do the following from your `project-2` directory:

1. Copy my program into your directory, as follows, being sure to note that this is a `.class` file (already compiled), and not forgetting the trailing space and period:

```
$ cp ~sfkaplan/public/COSC-111/project-2/MysteryConversion.class .
```

2. Run my program in your directory:

```
$ java MysteryConversion
```

When my program runs, it will prompt you for the **diameter and depth of Meteor Crator** in *barleycorns*, and then the **mean velocity of the swimmer** in $\frac{\text{smoots}}{\text{jiffy}}$. When it does so, enter the values that you obtained in the previous stage, **rounding to the nearest thousandth** (three digits after the decimal point) for each.

My program will then **use your conversion methods** to calculate another value. If you've written your methods correctly, you will see the correct output—otherwise, it will be incorrect, in which case you should go back and carefully test your methods again.

How will you know if it is correct? Write down your output, again to the thousandths, and move on to the next stage...

[2]Don't worry—not all of our projects will involve trips to the gym. It just happens to be a good source of arbitrary numbers, among other things.

4 Stage IV: Claiming your reward

Open a web browser, and enter the following web address:

```
https://www.cs.amherst.edu/~sfkaplan/courses/fall-2012/COSC-111/projects/XYZ.html
```

Except that `XYZ` is not what you should really write at the end of that web address. Instead, you should replace `XYZ` with whatever value my program emitted in the previous stage. If you have entered the correct value for `XYZ`, then your web browser will automatically be **redirected to a map**.

The map is marked (quite clearly) with a location. You should go to that location. When you get there, you should ask for the "CS 111 treasure."

Your final task: Once you have claimed your prize, log back into `remus/romulus`. In your `project-2` directory, use *Emacs* to open a file named `final-answer.txt`. In that file, tell what what your prize was, thus proving that you completed the hunt. Save the file and close your *Emacs* window.

5 How to submit your work

Use the `cs111-submit` command to turn in your work. Specifically, you should submit your source code file (`Converter.java`), as well as the decription of your final reward (in `final-answer.txt`):

```
cs111-submit project-2 Converter.java final-answer.txt
```

This assignment is due by the start of the next lab, September 20/21