CS100M Spring 2004
Assignment 5: The Object Strikes Back!

Due Wednesday, April 14, 23:59

0. Introduction

0.1 Goals

This assignment will help you develop OOP skills. Be sure to develop your code by stubbing classes and methods. Write each method one at a time, testing your code along the way.

0.2 Instructions

Be sure to read the *entire* assignment before answering the questions! Do the tasks in the following sections. You may work with one partner or by yourself for this assignment.

0.3 Submission

Follow the **Submission Format Requirements** at the CMS Info link on the course website. The last section describes what to submit on CMS.

0.4 Grading

All code that you submit must run without warnings and errors. Otherwise, you will receive a zero. This assignment's weight counts as a typical assignment. Some problems have bonus point options.

0.5 Academic Integrity

You must abide by the Code of Academic Integrity, which is provided for CS100M, the Department of Computer Science, and Cornell University on our course website. Refer to the link called Academic Integrity.

1. Task 1

Do Problem 7 pp. 338-339 in Savitch. Call your driver program `Problem1.java`. So, your Main Class will be called `Problem1`. Put your `Temperature` class in the same file as `Problem1`. We have given a partial (stubbed) solution that you must use. However, you may write additional fields and methods as needed, though you will not likely need to do so.

2. Task 2

2.1 Background

DIS.com needs to ship boxes to consumers but is too cheap to hire only three workers. A random number of boxes (between 2 and 4, inclusive) spews forth from a chute into a bin with no boxes initially inside. The lonely workers must take boxes out of this bin and place them in a truck for shipping. The truck has a capacity of 100 boxes. If the truck capacity is reached, it drives off, forcing the operation to shut down. Because of sporadic back pains, each worker has the ability to take a random number of boxes (1 to 4, inclusive) out of the bin for each trip to the bin. If the number of boxes in the bin exceeds the bin's maximum capacity of 20 boxes, the entire operation shuts down. So, the workers must keep obtaining

boxes and putting them in the truck to keep the operation running. DIS.com likes to work hard!

All workers start the shift at their peak efficiency of 100%, but all of this work tires them. So, after every four trips between the bin and truck each worker's efficiency drops by 15% of the previous value. For example, if a worker has completed four trips, her/his efficiency is now 85%, which means he/she will extract 85% of a random number of boxes, between 1 and 4 (inclusive, rounded down) for the next four trips. Because the workers will eventually wear out, they will cease extracting enough boxes, which eventually causes the operation to shut down if the truck has not already filled up.

2.2 Algorithm

You will simulate this problem with a program that has this high-level algorithm for *one* operation. Note that one *simulation* represents one operation of the factory:

- Initialize values.
- Attempt to add boxes to bin from the chute.
- If the bin is not full:
  - Worker1 attempts to remove boxes. Boxes are placed in truck, if there is still room.
  - Worker2 attempts to remove boxes. Boxes are placed in truck, if there is still room.
  - Worker3 attempts to remove boxes. Boxes are placed in truck, if there is still room.
  - Obtain more boxes for bin from the chute.
  - Repeat.
- Otherwise, stop the simulation and report the results:
  - Number of boxes in truck.
  - Number of cycles completed.

So, each time you perform the above algorithm, you have performed one simulation of the factory.

2.3 Non-OOP Solution

In a file called **`Problem2a.java`**, write your solution using only a Main Class and as many static methods and fields as you need.

2.4 OOP Solution

In a file called **`Problem2b.java`**, write an object-oriented solution. Put all of your classes in the same file. The program must prompt the user for a number of simulations to run. The result of each simulation will still be reported, as required in the above algorithm. In addition, after all the simulations have run, the program will report these statistics using the data from these simulations:

- Average number of boxes placed in the truck.
- Minimum number of boxes placed in the truck.
- Maximum number of boxes placed in the truck.
- Average number of completed cycles.

Hint: Use the following class stubs as a starting point:

```
public class Problem2b { }
class MyMath { }
class Truck { }
class Chute { }
class Bin { }
```

```
class Worker { }
```

2.5 Bonus Points

For 5 bonus points, use an array of workers instead of three individual variables.

For 5 bonus points, use an optional command-line argument for the input number of simulations. So, if the user runs the program as:

```
> java Problem2b 10
```

your program will run 10 simulations. Otherwise, if the user supplies no argument:

```
> java Problem2b
```

the program prompts the user instead. Your code must account for an illegal number of input arguments and illegal types to receive the full bonus credit.

For 25 bonus points, modify your program such that it outputs the data for all simulations in a data file. Provide a MATLAB program called **`analysis.m`** that calls your Java program, reads in the data file that **`Problem2b`** generates, and plots it in a meaningful and descriptive way.

3. Submitting Your Work

Submit a zip file called **`a5.zip`** that contains these files:

- **`Problem1.java`**
- **`Problem2a.java`**
- **`Problem2b.java`**
- optional: **`analysis.m`**

Do not submit any other files! As a reminder (see first page), refer to **Submission Format Requirements** on CMS Info on the course website before submitting any work! You will find some differences for Java programs.