CS100J, Spring 2001 Project 4

Due Tuesday 3/13

1. Objective

Completing all tasks in this assignment will help you:

- write classes to model a variety of physical objects
- use encapsulation within your classes
- use OOP to simulate a physical system

First skim, and then, carefully read the *entire* assignment before starting any tasks!

2. Background

2.1 Model

A program can model a physical process. A ***model*** is a (usually) simplified representation of another thing. Consider an example of workers removing items from a conveyor belt. An employer might ask a variety of questions about this system to help budget expenses and hiring practices. Some concerns might include the ability of workers to extract items, the amount of time they might work before needing a break, and so on.

2.2 Simulation

Testing with real people and equipment is quite expensive, so computer ***simulations*** provide a cheap alternative. A simulation is a testing of a process using a model. Object-oriented programming assists with the modeling and simulation of a process. A physical system may be modeled as a collection of abstract, computer objects that have the ability to interact. Java provides an idea environment for testing these interactions. Within the physical system lies a process which forms the algorithm used by the program.

2.3 Uncertainty

An interesting consequence of modeling a physical system is the uncertainty of the real system. Humans have a tendency to wish for simple models that predict events with incredible accuracy. When an outcome to a situation is completely known and guaranteed, the situation is ***determinate***. The equation of a straight line $y = mx + b$ is a basic determinate model, where every input x, gives a known value of y.

The “real world” blurs such predictability because of the interaction of multitudes of factors. For instance, an upset stomach might slow one worker down, whereas another might have overloaded on coffee. Granted, if someone could determine all the factors that affect a process, a deterministic relationship might be found. To reduce the effort in forming a model to perform a simulation, vaguely known and relatively unknown quantities are modeled with random values.

Indeterminate uncertainty involves such randomness. Rather than trying to predict a process with 100% certainty, an indeterminate model exhibits a range of outcomes. Given the interaction of the different portions of the model, some outcomes might happen with greater frequency. The more frequent outcomes provide a measure of the how the real system might behave under a variety of inputs. For the model described in the next section, you will write a program to determine such common values and possibly discover some interesting features of the real system.

3. Problem

3.1 The Real World

In many dining halls, students put leftover comestibles and other items (sporks, grungy napkins, and other “delights”) onto trays. The well-fed students place their trays on a conveyor belt which transports everything somewhere mysterious (a place called the washroom/kitchen) for cleaning. Few realize the demise of the trays and the items placed there upon, though. On the other side of the belt, several workers, standing side-by-side, remove the items from the trays, including the trays. Normally, work proceeds smoothly until the dining hall nears closing time. Then,

many students simultaneously finish their victuals and dump their trays. Working frantically, the workers attempt to remove the items from the deluge of trays, but some slip past the last worker. If enough trays pile up at the end of the line, the entire stack topples and a worker must rush to slam the STOP button. But by then, some plates and glasses have met an untimely demise, and tuition rises yet again….*

3.2 A Simplified Model

A program can simulate a group of workers removing items from trays on a conveyor belt moving left to right, as depicted in Figure 1. An equal number of trays appear for an equal number of workers. The simulation has all trays start empty in front of each worker. In the first run, all trays move right one space to the next worker and a new tray appears with random items for the first (left-most) worker. Each worker attempts to remove as many items as possible from their tray. Because the workers get tired from moving items (or bored from waiting to remove items), their efficiency decreases for each run. If the last worker (right-most) fails to remove all contents of their tray, the belt stops. If no items remain, the simulation continues with a new run by repeating the process of shifting all trays to the right, giving the first tray new items, and having the workers extract items from their current tray.

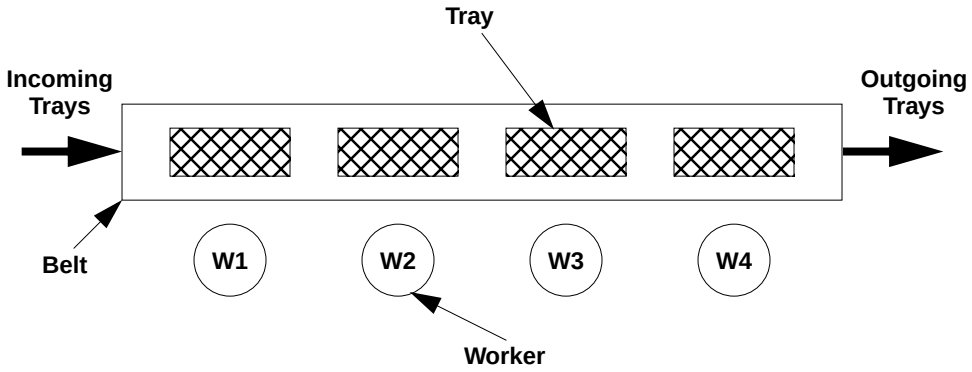

Figure 1: Top View of Simulation

Assumptions:

- Each tray appears with a random amount of items between 0 and 5, inclusive.
- Trays are removed automatically.
- Each worker starts with a random efficiency between 85 and 100%, inclusive. This efficiency affects each worker's capacity of removing a random amount of 2 to 6 items, inclusive. Furthermore, after each run, the efficiency drops by 5%. All workers' efficiencies drop each run to model either work or boredom.
- Workers may choose to skip some items, especially the workers on the left. Otherwise, the workers on the left would get all the grungy trays while the last workers would get the least work! So, the first worker will skip half the items he/she would normally take one half of the time. The second worker will skip one third of the items he/she would normally take one half of the time. The third worker will skip one fourth of the items he/she would normally take half of the time. The last worker will not attempt to skip items he/she would normally take.
- The very first run of the simulation has all empty trays.

4. Things to Do

First write an algorithm and then a program called **`p4sp01.java`** the model described in Section 3.2:

- You may not use inheritance or arrays for this project.
- Write the program for 4 workers. Try to be general where possible, though you will likely be forced to be specific many times.
- You must use an OOP approach with good style that includes writing classes with encapsulation, named constants, and appropriate constructors.
- Optional: Your program may pause after each run.
- Optional: Your code may prompt the user to run another simulation.

*. Yes, this situation is based on personal experience.

Provide output for one of the shorter simulations, but at least 10 runs. If you choose not to pause between runs, your output should look something like the following. The numbers represent the items on each tray, from left to right:

```
Total items taken, so far: 49
Current run: 18
Old contents: 4 2 0 0
New contents: 2 0 0 0
Total items taken, so far: 53
Current run: 19
Old contents: 4 2 0 0
New contents: 3 0 0 0
```

After debugging and running the program, answer these questions about the simulation:

1. On average, how many runs occur before the system stops for the given criteria?
2. What happens to the average runs if you increase amount of items on trays?
3. What happens to the average runs if you decrease amount of items on trays?
4. What happens if all trays contain 0 items? Approximately what is the lower bound of the highest value of items on trays that causes the belt to stop when reaching the last worker?
5. Could you have predicted this lower bound of the highest value? Why or why not?
6. When simulating a process, usually optimal values are sought after. Optimal values are involve an extreme value, like *best*, *worst*, *maximum*, or *minimum*. Attempt to find an optimal value of a range of items on the trays and worker capabilities to produce an average of 200 items removed in no more than about 50 runs.
7. Suggest at least 4 ways to improve the performance of the model and simulation.

5. Submitting Your Work

Submit the following items:

- a title sheet and table of contents
- detailed algorithm
- program **`p4sp01.java`**
- output for “shorter” run
- answers to the questions in Section 5

Follow the submission guidelines stated on the Projects page for CS100J. Number all pages to help the graders find each portion of the assignment.