CS100J Spring 2001: Project 4 Grading Guide

Notes

- Please carefully review all notes written on your grading form and project.
- Find the codes for these notes below.
- Try to understand why you received the note so that you may avoid it on your next project.
- * means the item is worth twice

Scores

- Bonus may be applied for exemplary work or doing additional tasks
- Give 1 bonus point for each valid and justified response to Question 5 and 6.
- Let c and s be the number of correctness and style: see table, below
- For each program not included, remove one correctness and style point

category	Points					
	0	1	2	3	4	5
correctness	nothing turned in	$c \geq 16$	$12 \leq c \leq 15$	$8 \leq c \leq 11$	$4 \leq c \leq 7$	$0 \leq c \leq 3$
style	nothing turned in	$s \geq 17$	$13 \leq s \leq 16$	$9 \leq s \leq 12$	$4 \leq s \leq 8$	$0 \leq s \leq 3$

1. General

(s1a) grading form provided for each partner, as coversheet
(s1b) grading form properly filled out and signed
(s1c) title sheet and table of contents provided (OK if unified)
(s1d) pages numbered and properly bound
(s1e) all work typed
(s1f) lines of text/code not chopped off or misaligned

2. Algorithm

(s2a) written in pseudocode as a series of steps – not an essay
(s2b) included all the steps to be executed in a run
(s2c) gives high-level description

3. Program Correctness

(c3a) simulation continues as long as tray 4 is emptied by worker 4.
(c3b) each worker starts with a random starting efficiency between 85 and 100% (both inclusive).
(c3c) the efficiency should always be within the range specified and should be able to take any value in the range:
- Code, like `Math.random()*(high-low+1)+low` or `(Math.random()*(100-85+1)+85)/100`, will not work – they can achieve values > `high`,
- Code, like `(int)(Math.random()*(100-85+1)+85))/100`, cannot get values like 90.5% (i.e., 0.905).

(c3d) the run # is displayed for each run.
(c3e) *both old contents and new contents are shown for each run.
(c3f) the total # of items taken so far is displayed after every run.
(c3g) the first run printed shows old contents trays 2,3,4 as 0 – only tray 1 may be non-empty. If the output included does not show runs starting from the beginning, check to make sure that the first run displayed is after the trays have shifted and a new tray has entered with a random # of items on it.
(c3h) trays 1,2,3 shifted right in proper order: tray4=tray3, tray3=tray2, tray2=tray1
(c3i) tray 1 enters with random # of items
(c3j) capacity of a worker is a random # between 2 and 6 (both inclusive) multiplied by efficiency.
(c3k) workers may choose to skip items half the time - the choice is random.
(c3l) code, like `items-= items*(1-MyMath.random(0,1))/2`, will not work because integer division has precedence – items will never be decremented i.e., no skipping takes place
(c3m) the # of items skipped depends on position.
(c3n) check that the # of items extracted is at most the # of items on the tray.
(c3o) worker efficiency decreases by 5% of previous efficiency - not 5% points.
(c3p) * no syntax errors (as much as you can make out) – program should compile.
(c3q) output should be correct.
(c3r) **program output matches with the code - no FAKING of output.

4. Style

(s4a) *separate `Worker` class provided.

(s4b) *separate **`Tray`** class provided.
(s4c) **`Belt`** class provided.
(s4d) a main simulation/driver class provided.
(s4e) class such as **`MyMath`** provided which contains code for random # generation.
(s4f) **`Belt`** class has 4 Trays as members i.e., **`Tray`**s participate in has-a relationship with **`Belt`**.
(s4g) *simulation class sets up the model by creating **`Worker`**s, **`Belt`**.
(s4h) ***`private`** members are used.
(s4i) efficiency and position are private member fields of **`Worker`** class.
(s4j) #_of_items on a tray is a private member of **`Tray`** class and Total_#_of_items extracted is a private member of the **`Belt`** class.
(s4k) *getters and setters used.
(s4l) method provided to change worker efficiency.
(s4m) methods for getting #_of_items on a tray, filling a tray, removing items from tray.
(s4n) **`static`**/**`final`** variables used to represent ranges
(s4o) code is properly indented.
(s4p) comments included for major lines of code and methods.
(s4q) *modular code – methods used to subdivide larger tasks and/or perform repetitive tasks
(s4r) clear, readable code – avoiding needless complication/redundancy.
(s4s) output included with program.
(s4t) output shown is for >=10 runs.

5. Discussion

(c5a) answers to each question typed separately – no essays
(c5b) (1) average # of runs: 18-24; # of items extracted: 35-50. This could vary depending on when the data is rounded e.g., **`items=(int)(efficiency*(Math.random(6-2+1)+2))`** can give more runs than **`items=(int)(effciency*(int)(Math.random(6-2+1)+2))`**
(c5c) (2) average # of runs decreases as #_of_items on a tray increases
(c5d) (3) average # of runs increases as #_of_items on a tray decreases
(c5e) (4a) simulation continues forever if all trays contain 0 items because # of items extracted is 0
(c5f) (4b) approximate lower bound is 17 +/- 2 is fine.
(c5g) (5) difficult to predict: Suppose each worker has capacity = 4, starts with efficiency 1. Average efficiency of workers = (1+.95+.95*.95+.95*.95*.95)/4=0.9275 (since worker 1 has eff. 1 in run 1, worker 2 has eff. 0.95 in run 2 and so on). So # of items that can be extracted = 4 (workers)*4 (capacity)*0.9275 (efficiency)=14.84. If each worker starts with efficiency 0.85, # of items that can be extracted = 12.6. We haven't taken into account workers' willingness to skip items which can only decrease # of items extracted. So the predicted lower bound doesn't match the lower bound obtained.
(c5h) (6) grading is flexible: since an average of >= 4 items have to be removed per run, the range for #_of_items on tray should be >= 0-10 or so. It is preferable to have the lower end point as 0. Even after 25 runs due to efficiency loss at each run, the efficiency reduces to < 28% of original value, so it is very unlikely that the simulation runs for 50 runs. The simulation most probably would run for 30-35 runs so the extraction rate should be ~ 6-8 items per run. Also the average efficiency of a worker in the first 25 runs(assuming he/she starts with 100%) is 57.8%. So roughly speaking, the maximum capacity should be 14-16 items per run and the worker capacity and efficiency ranges need to be narrow. The ranges obtained were: #_of_items on tray: 0-15, worker capacity: 13-18, worker efficiency : 90-100%.
(c5i) (7) grading is flexible: if the answer is reasonable and there is some justification, credit is awarded. Ways could be more realistic efficiency function, more accurate data to model worker capacities and efficiency, better way of modeling uncertainty than a simple random distribution, factoring in different worker skills/abilities/attitudes, differences in types of items, and more. Essentially, imagine the real-world situation and ask what's been glossed over/approximated in the model.

6. Miscellaneous

(c6a) New contents displayed AFTER extracting items and BEFORE shifting items
(c6b) miscellaneous