CS 351: Assignment 04

Due: Monday, April 10, 2023

Overview

This assignment requires you to implement one discrete-event simulation model using a *process-oriented* perspective in the DESMO-J framework for Java.

Academic Integrity Policy

All work for this assignment must be your own work and it must be completed **independently**. Using code from other students and/or online sources (e.g., Github) constitutes academic misconduct, as does sharing your code with others either directly or indirectly (e.g., by posting it online). Academic misconduct is a violation of the UWL Student Honor Code and is unacceptable. Plagiarism or cheating in any form may result in a zero on this assignment, a **negative score** on this assignment, failure of the course, and/or additional sanctions. Refer to the course syllabus for additional details on academic misconduct.

You should be able to complete the assignment using only the course notes and textbook along with relevant programming language and library documentation (e.g., the Java API specification). Use of additional resources is discouraged but not prohibited, provided that this is limited to high-level queries and not assignment-specific concepts.

Deliverables

You should submit a single compressed archive (either `.zip` or `.tgz` format) containing the following to Canvas:

1. The complete Java source code for your program (i.e., all of your `.java` files). **Do not** include the DESMO-J `.jar` file in your archive, as this can cause problems when uploading to Canvas. Additional source code requirements are listed below:
    - **Your name must be included in a header comment at the top of each source code file.**
    - Your code should follow proper software engineering principles for Java, including meaningful comments and appropriate code style.
    - Your code must not make use of any non-standard or third-party libraries, aside from DESMO-J.
2. A `README` text file that documents any parts of your program that are incomplete and/or not working.

Program 1: `MachineShop` Revisited (12 points)

This problem revisits the machine shop described in Assignment 02 using a process-oriented modeling perspective; refer to that document for a complete system description.

The **updated** goals for this simulation are to determine:

- The minimum, maximum, and average response time for parts.
- The maximum number of parts in the system at any point in time.
- The average and maximum lengths of the queues for processing and inspection.
- **(NEW)** The utilization rates for the machine and inspector (i.e., the percentage of time that each is busy).

In your model, the inputs should be generated according to the following distributions:

Interarrival Times	Exponential with a mean of 7 minutes
Processing Times	Exponential with a mean of 4.5 minutes
Inspection Times	Normal with a mean of 5 minutes and standard deviation of 1 minute (truncated to remove negative values)
Refining Times	Exponential with a mean of 3 minutes
Needs Refining	Bernoulli with $p = 0.25$

Use the appropriate classes in DESMO-J for each of these distributions (see here for a list of classes that are available; **do not** use Java's built-in random number generator). (*Hint:* You might want to use the distributions from the event-oriented implementation first for debugging so that you can compare the outputs from the two models and verify that your process logic is correct; once you have done so, then you can switch to the new distributions.)

Initialize your simulation with the machine and inspector both idle and no parts in the system. Your model class should include a `main` method that runs the simulation once for 24 (simulated) hours and generates a report containing information on all outputs of interest. (*Note:* Recall that a single run of any simulation model is generally not sufficient to provide good estimates for the outputs of interest; later on we will see how we can do multiple runs in DESMO-J.)

Additional Requirements

- You *may* use example code on Canvas and/or from Lab 02 as a starting point for your own work. **However**, if you do this, then **you must ensure** that all comments as well as names for variables, methods, and classes are updated appropriately **and** that any extraneous components are removed. Failure to adequately update any example code that you use will **negatively impact your assignment grade**.
- You **must** use DESMO-J's distributions for generating all random values used in your model (this is important for reproducibility and replication). **Do not** use `Math.random` or the `Random` class in Java.
- All output measures of interest must be tracked using DESMO-J's statistical trackers (`Count`, `Tally`, and `Accumulate`) and `Queue` structures, and these outputs must be included in the report that gets generated after running the simulation.
- Your program should not generate any extraneous print statements or messages to standard out (other than what is produced by DESMO-J's classes already). You may however include additional debugging statements in the debug and trace files produced by DESMO-J.

Grading

At a high level, your assignment grade will depend on the following:

1. Model components
2. Process logic
3. Statistical trackers and outputs

In order to earn points in one of these categories, your program needs to be correct or nearly correct across all preceding categories. So, for example, your program needs to have correct (or nearly correct) model components and process logic in order to earn points for the statistical trackers. This grading scheme is being used because of the dependencies between the categories: it is difficult to assign partial credit for a higher category if one or more preceding categories are significantly incorrect. This means that you should focus your efforts on these categories in order: first make sure that your model includes all or almost all necessary components, then focus on getting the process logic correct, then focus on tracking the right output measures.