IEU SE 116 Project Description Spring 2024

Please read the entirety of instructions very carefully before starting the project!!!

Team Structure

This project will be conducted with teams of size four, with few teams of size three. We have already created the teams randomly, and we strongly suggest that you stay with your assigned team to develop professional collaboration skills with help from your instructors. You may find your team assignment at

https://docs.google.com/spreadsheets/d/1t1ZTLJDTEqZyQEPOL67QpccKwG8xiz__Nolbcd Tbf5E/edit#gid=228621852 If you insist, you may form your own teams of size exactly four from students in your section, but then you are on your own for team dynamics in case some members do not do their share for whatever reason. If so, you must update the team sheet by following the embedded instructions by 11:59 pm, April 4, 2024. We will adjust the resulting teams as necessary and publish the final list shortly after.

Version Control System: Git

We want you to learn the common "version control system" software that is used everywhere: Git. A version control system helps us to keep track of our changes in time. It is also used as a collaboration tool, which means it helps you to develop software with other developers. Git is one of many version control systems, but currently the most popular one. Therefore, it is vital for you to learn what Git is, what it does, how it works, and how to use it properly. This project is a great starting point to learn how to collaborate with other users on Git, and how to synchronize your code.

Git software is available at https://git-scm.com/ and it contains documentation, along with videos, at https://git-scm.com/doc or at https://skills.github.com/. However, the documentation also talks about some more advanced things you can do with it. We have already uploaded an offline video to Panopto about Git. Please watch it and start using Git in your project from the very first day.

While Git can be used on your own local system, it is mostly used with a remote (online) server. The most common one is the GitHub, at https://github.com/. Registration is free, so make sure you create a profile if you haven't done so already. You can create a repository for your project, and keep your source code online. Make sure you keep your project private, so that only your team can access it.

Report

A short but formal report should accompany your source code, free of spelling or grammar errors or colloquialisms. In this report, indicate if you have failed to correctly implement any functional requirements, discuss your design (using UML class diagrams) and implementation choices, the challenges you have faced and how you have solved them. **It should also include the execution output of your program with the data provided in the req's.** This can only be done if you keep taking notes during the development. The report must be in PDF format. If you submit in another format, such as "docx" or "pages" or anything else, we will not accept it, and you will not get any points for the entire project.

Submitting Your Project

This project is a team project. One member from each team must submit the following files

- A ZIP file that contains your source code.
- A JAR file that runs your program.
- A PDF file that contains your project report.
- A PDF file that contains screenshots of your Git commits.

The due date is May 19, 23:59. **Late submissions will NOT be accepted.** There will be an oral exam at the end of the semester, where each team member will demo and answer questions about the design and implementation of the project.

Grading

Any software project has a list of requirements. Some of them are "functional" - which are about the features the software should have, and others are "non-functional" - which are about the conditions that the software will be running.

- **Failing any nonfunctional requirement results in a grade of zero**.
- Report is 15 points. Evaluation criteria includes the narration quality, and the software design choices.
- Source code is 15 points. Evaluation criteria includes adherence to basic style guidelines, proper comments, and implementation choices.
- Functional requirements 1-4 are 5 points each, rest are 10 points each, totalling 70.

Non-functional Requirements

Non-functional Requirement 1: The program must be implemented in the Java programming language. Source code should be delivered as a zip file, no other format is permitted.

Rationale: Since we are learning programming using Java, this is expected.

Non-functional Requirement 2: The development must be done on a private Git repository.

Rationale: Version control is important, especially in a team project. While you are free to use any Git server, we strongly suggest that you create a GitHub account. Every time you improve your project, commit your changes to the repository, so that we can see your progress at the end of the semester. Install the "GitHub Desktop" program and you will be ready to go. The repository must be private. No one must be able to see it, but your team. Team members who do not have at least weekly commits will fail this req and receive a 0. Each commit must include the description of the work included in the commit. The deliverables must include your git log screenshots as a pdf file.

Non-functional Requirement 3: The program must be delivered as a JAR file.

Rationale: Instead of several class files, submit the program packed into a JAR file.

Non-functional Requirement 4: The program must be accompanied by a project report in pdf format, as detailed in this document, including the UML class diagrams and program output.

Rationale: The design and implementation decisions must be justified

Non-functional Requirement 5: Each team member must be able to compile and demo the program execution, and answer questions about the design or implementation, in order to demonstrate their participation in the project.

Rationale: This will be a big factor in grading. Being absent during the presentation session is not acceptable. Every team member is responsible for the entire project, not just the parts they may have designed or implemented. If the code does not compile or run, you get zero. If it crashes along the way, you lose substantial points. If any team member cannot sufficiently answer the questions about any aspect of the project (code, design, report, etc.) during the oral exam, that team member will lose substantial points overall. Absent team members from the presentation get zero from the entire project.

Functional Requirements

Project Description

The project involves discrete event simulation of workflow, which can be applied to government offices, banks, hospitals, factories among others. For example, if you need to get a power of attorney document from the notary, you start with the receptionist, have your ID checked at one station, your mental health document if you are above 65 at another station, get your photo taken at one station, the documents printed in one, signed in another, and pay fees at the cashier. Same can be said of a factory, where raw materials move through various stations before they become finished goods.

The fundamental entities in the system are **jobs**. Each job has a duration (minutes) in which it must be completed once it arrives. Each job also has a **job type**. Associated with each job type is a sequence of **tasks** that must be executed one at a time, i.e., The next task in sequence for that job cannot start before the current one finishes.

Each task has a **task type,** and a numeric task size that contributes to the duration of task execution. Some tasks have default task sizes based on the task type, other tasks have varying sizes, even within the same task type. Tasks are executed at **stations.** Some stations can handle a single task type, others can handle multiple task types. Some stations can handle a single task at a time, others can handle multiple tasks up to the capacity of the station. Some stations that can handle multiple tasks concurrently require them to be of the same task type, others do not have that constraint. The duration (number of minutes) of a task execution is calculated by dividing the task size by the station speed for that task type.Some stations operate precisely at constant speed. Others might have speeds that randomly vary uniformly with a certain percentage. E.g., a station that operates at 10 units/min +- 20% will have a speed in the range 8 units/min to 12 units/min, where each speed in that range is equally likely. Note that all tasktype, jobtype, JobID, and stationID strings must start with a letter followed by more letters and/or digits or underscore character '_'.

Workflow Text File Format

Note that MULTIFLAG stands for Y or N, indicating whether multiple task types can be processed concurrently or not. FIFOFLAG stands for Y or N, indicating whether waiting tasks are picked using first come first served strategy or earliest job deadline first strategy. Square brackets indicate optional values that can be safely omitted. Here is the text file format:

(TASKTYPES tasktypeID [defaultSize] … tasktypeID [defaultSize])

(JOBTYPES

(jobtypeID tasktypeID [size] tasktypeID [size] tasktypeID [size]...tasktypeID [size]) …

(jobtypeID tasktypeID [size] tasktypeID [size] tasktypeID [size]...tasktypeID [size]))

(STATIONS

(StationID [maxCapacity] MULTIFLAG FIFOFLAG

(tasktype speed [plusMinus])..(tasktype speed [plusMinus])) …)

Sample Workflow Text File Content and Expected Error Messages and Corrected Version

Below is what a sample file might look like. Line numbers are for reference, they are not part of the file content. Note the syntax and semantic errors for your program to detect and report.

```
1.  (TASKTYPES T1 T2 2 T3
2.        2.5 T4 T5 -4 T1 1 T_1  5 21T)
3.  (JOBTYPES
4.       (J1  T1 1 T2  T3 3)
5.       (J2  T2  T3 T4 )
6.       (J3  T2)
7.       (J2  T21 5 T1 -2) )
8.
9.  (STATIONS
10.  (S1 1 N N T1 2 T2 3 0.20)
11.  (S2  2 N Y T1 2 T2 4)
12. (S3   2 N Y T3 1)
```

Given the above content, the following errors shall be detected by your program:

- Line 1:
  - T5 has a negative task size.
  - T1 is listed twice.
  - 21T is an invalid tasktypeID.
- Line 5:
  - T4 has no default size, either a default size must be declared in TASKTYPE list or the size must be declared within the job
- .Line 7:
  - J2 already declared in line 5.
  - T21 is not one of the declared task types.
  - T1 has a negative task size of -2.
- Lines 9--12: In STATIONS,
  - T5 and T_1 are not executed in any STATIONS even though they are listed as possible task types. This shall raise a warning.
  - There are no STATIONs which execute T4 and/or T21, however, both T4 and T21 are a part of some job type
- .Line 12: ‘)’ missing

The corrected version of above sample is as follows:

```
(TASKTYPES T1 1 T2 2 T3 2.5 T4 T5 4 T_1 5 T21)

(JOBTYPES
     (J1  T1 1 T2  T3 3)
     (J2  T2  T3 T4 1 )
     (J3  T2)
     (J2  T21 5 T1 2) )

(STATIONS
 (S1 1 N N T1 2 T2 3 0.20)
 (S2  2 N Y T1 2 T2 4)
 (S3   2 N Y T3 1)
 (S4  3 Y Y T4 1 T21 2 0.50)  )
```

Specific Functional Requirements

Functional Requirement 1: The program must read from command line names of the workflow file, and the job file. It should properly report if either file does not exist, or is not accessible.

Functional Requirement 2: The program must be able to parse the task type, job type, station information from the workflow file, with the text format specified above. It must print out any syntax errors with a descriptive message and include the line number where the error occurs in the text. It must also verify that all ID's are unique, numeric values are in the appropriate range as required, and print any such semantic errors as well. It should create the appropriate objects based on this information, ready for execution.

Functional Requirement 3: The program must print the workflow information to the console, in an easy-to-read format. You may not use the same syntax as in the input file.

Functional Requirement 4: The program must be able to parse the job file with the text format specified below, one job per line with jobID, jobTypeID, start time, and duration separated with whitespace characters. Note that the deadline to complete a job is computed as start time plus duration. The program must print out any syntax errors with a descriptive message and include the line number where the error occurs in the text. It must also verify that all ID's are unique, numeric values are in the appropriate range as required, and print any such semantic errors as well. It should create the appropriate objects based on this information, ready for execution. Below is sample job file content:

```
Job1 J1       1      30
Job2 J1       2      29
Job3 J2       5      40
Job4 J2       7      35
Job5 J3      10      30
```

Functional Requirement 5: The program must track the state of each job: waiting to start (based on startTime), which task it will execute next, and which station it is waiting to be executed, or which station it is currently in execution, and when the entire job is completed, and how long before or after its deadline. When a task starts or completes, the program must be able to change the state of the job appropriately, and display the resulting state.

Functional Requirement 6: The program must track the state of each station: what task(s) are executing, what tasks are waiting in line to be executed at that station. When a task completes execution at a station, the program must be able to change the state of the station appropriately, and display the result. It should mark the station status as idle, if there are no ongoing or waiting tasks. Otherwise it should implement a strategy to pick which task(s) waiting for that station can move to execution using first-come-first served or earliest-deadline-first depending on the station specification in the input file. <u>Make sure your design is easily extensible to other dispatch strategies such as shortest job first.</u>

Functional Requirement 7: When a task completes, the program must identify the next task in sequence for the respective job, and identify which station it must be routed to for execution. This is a fundamental, critical problem in job scheduling; you are not expected to come up with optimal station assignment to tasks. However, you should absolutely not route tasks to stations that do not have the capability to execute them per their task type. Of the suitable stations, you may choose one randomly, in a round robin fashion, or take into consideration how busy each station is, how fast it is, and how many tasks are already waiting in line to be executed, and job deadlines. If ambitious, you may also consider the frequency of task types versus availability among station capabilities.

Functional Requirement 8: The program must have an event queue in order to manage the event timeline. At each event, the program must print the time, type and details of the event, and the resulting system state. Events include when it is time for a new job to be inserted into the system, per its start time, or when a task completes execution at a station so that - if available- another waiting task can start executing. Rather than tracking time as second one, second 2, second 3, and changing state at each clock tick, in discrete event simulation, one directly moves to the next event: For example at start (i.e., time = 0) all stations are idle. All jobs should have an event indicating their start time at the event queue. Suppose the earliest job has start time = 15. That should be the first event in the queue. The system moves from time zero directly to time 15, dispatches the earliest job to the appropriate station based on its first task, and calculates when that task will finish so as to insert another event into the event queue. Suppose another job starts at time 16, the program will handle it similarly. Suppose at time 20 the first job's first task is complete, triggering the next event. In that event dispatch, the system must route the second task of the first job to the appropriate station queue, and if there is a waiting task at the previous station, it should move to execution, assuming station constraints regarding handling multiple different jobs etc. are met.

Functional Requirement 9: The program must report the average job tardiness for late jobs by job type, and station utilization (percentage time a station is not idle) for each station at the end of execution.