Simulator

Assignment 2

Mammon Ahmad Alshamali

20/3/2022

Outline

Introduction

The assignment is about building a simulator that simulates processor execution for processes, assuming that we have P processors. The following will be considered:

" A processor can only execute one task at a time.

• The number of processors is fixed during the whole simulation.

• All processors are synchronized, i.e., they all have the same clock.

• Processors are given unique ids as follows: P1, P2, P3, P4, etc.

• Tasks are given unique ids as follows: T1, T2, T3, T4, etc.

• Clock cycles are given unique ids as follows: C1, C2, C3, C4, etc.

• A task is defined by the following:

o Creation time: the clock cycle in which the task was created.

o Requested time: the number of clock cycles the task needs to execute till completion.

o Completion time: the clock cycle in which the task was completed.

o Priority: which can be either high or low.

o State: which can be either one of three states (see below).

• When a task is created, it must be automatically put in a queue.

• The simulator has a scheduler that is responsible for determining the assignment of tasks that are waiting in the queue to processors that are available.

• A task can have only one of three states: either “waiting” in the queue, “executing” on a processor, or “completed”.

• A processor can have only one of two states: “busy” or “idle”.

• If there is a processor that is available, then the schedular must prioritize the assignment of tasks with “high priority” over tasks with “low priority”.

• If there is a tie, i.e., a processor became available and there are two “high priority” tasks waiting, then I will leave it to you (the student) to determine how the scheduler should break ties. The same goes for breaking ties between “low priority” tasks.

• Low priority tasks can be interrupted. For example, if a high priority task is created, and there was a processor that is running a low priority task, then the schedular must remove the low priority task from the processor and put it in the waiting queue, and then assign the high priority task to

the processor.

• If a low priority task is interrupted when executed again, it only needs to run for the remaining

time of its requested time. For example, if a low-priority task is requested to run for 30 cycles, and it was interrupted during the tenth cycle, then when it runs again, it only needs to run for 20 cycles.

• High priority tasks cannot be interrupted.

• For simplicity, we will assume task creation time is negligible. For example, if a task was created during cycle N, and there was an available processor, then this task can be immediately assigned to this processor in the same cycle N.”

The design will be following object-oriented with clean code as much as possible. The details for each class in UML will be on page 16.

Design Data Structure and Algorithm

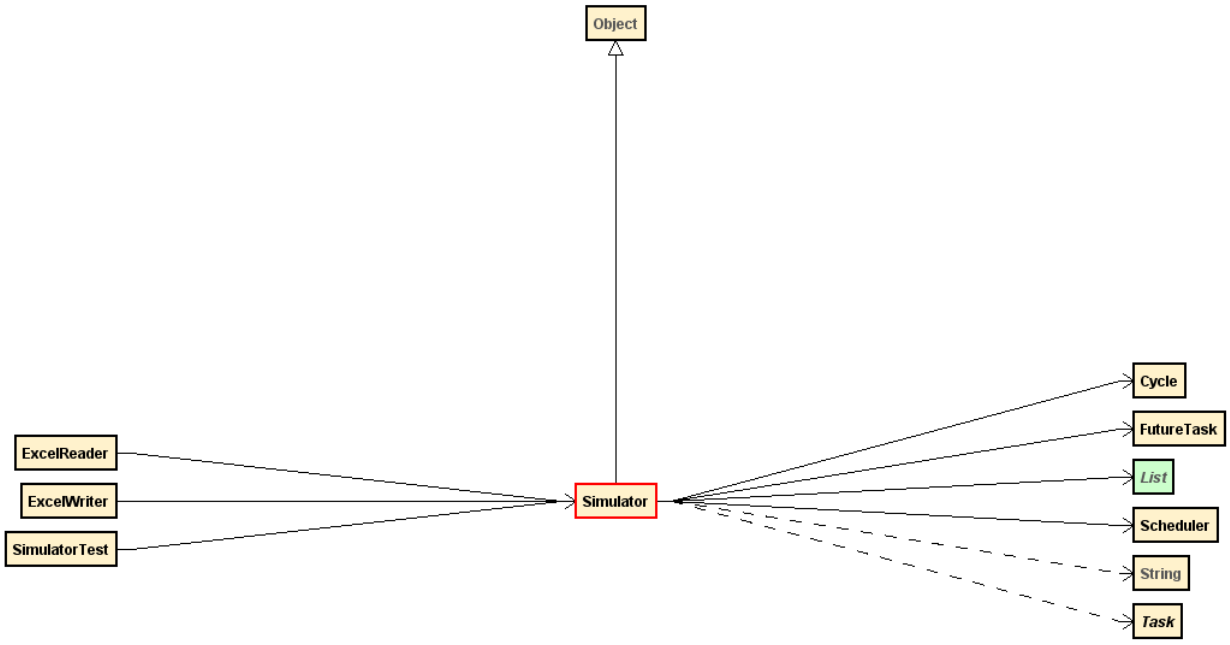
The first thing to be considered is how the simulator will work, the simulator will work as the following:

- Interact with the user by taking the Input which is the number of processors and tasks with tasks details.
- It will create the environment for the simulation.
- Creating processors, cycles, tasks.
- Controlling the schedular to deal with the environment.
- Keep managing the environment until all the tasks and processors are finished.
- Show the result of the simulation like the total simulation time, and completion time for each task.

The design will be explained using a class visualizer, and plant UML

Simulator Class

The simulator act as a mediator between the user and other classes



As it is shown in the figure the simulator will be the mediator for the design it will read the input from an excel file, then simulate. When the simulator finishes the simulation, it will write the results into a new excel file.

Taking the Input and Prepare the Environment:

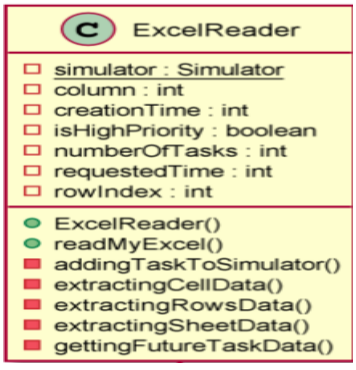
The input will follow this format:

	A	B	C
1	2		
2	10		
3	4	FALSE	44
4	1	TRUE	7
5	2	FALSE	6
6	5	TRUE	15
7	6	FALSE	7
8	7	FALSE	10
9	8	TRUE	20
10	9	FALSE	60
11	10	FALSE	20
12	100	FALSE	20

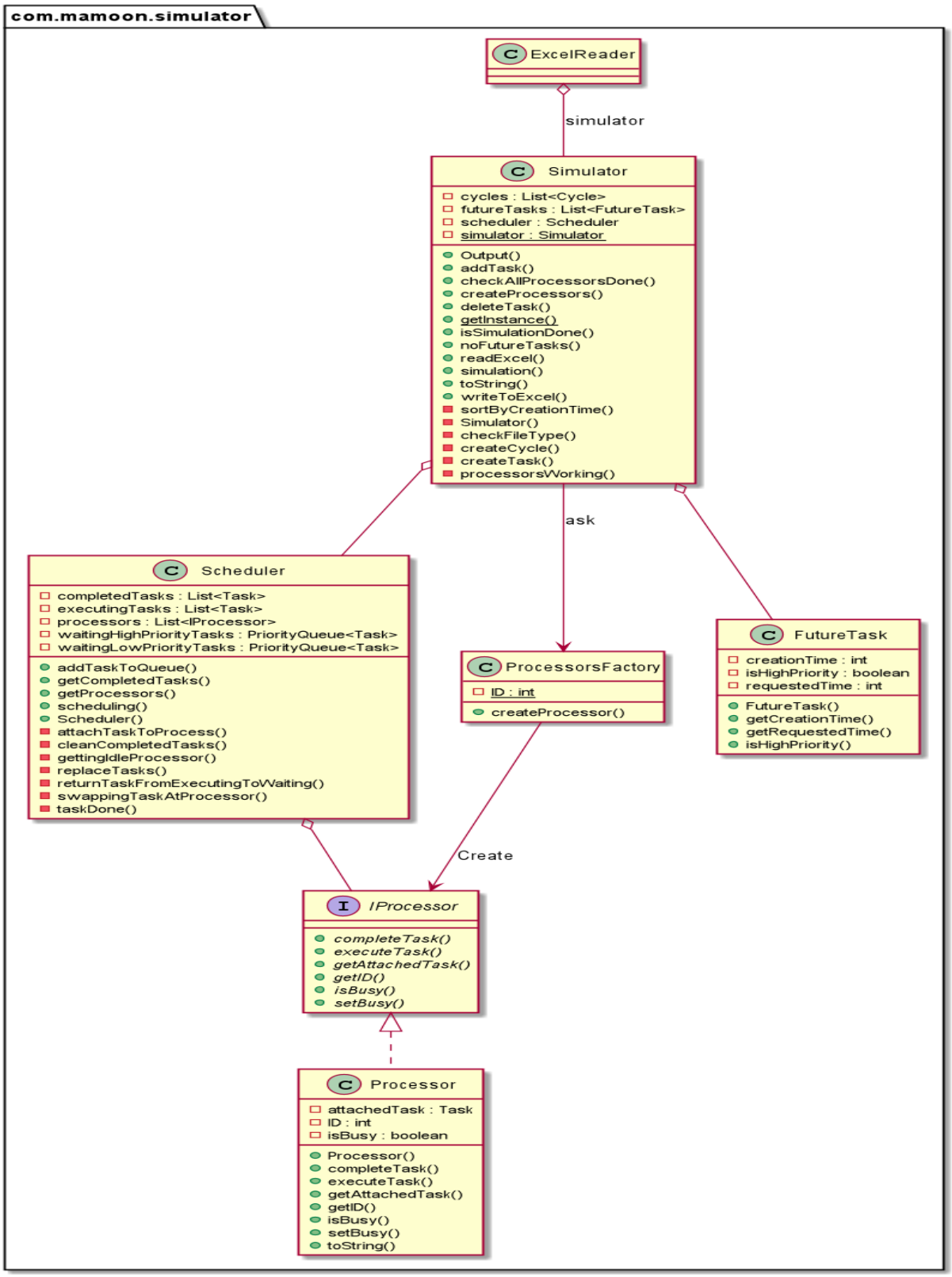
The first row is the number of processors, the second row is the number of tasks. The rest of the rows are the details of each task.

The user writes the input file name then the Simulator will check if the type of the file is correct. After checking if the file type is correct, it will start reading data from the excel file.

So, a class got created to handle reading from an excel file:



And its relation with the simulator will be:



The excel reader will add the number of processors to the simulator and the simulator will ask the processors factory to create the processors. The processors will be created and sent to the schedular class which has an array list of processors.

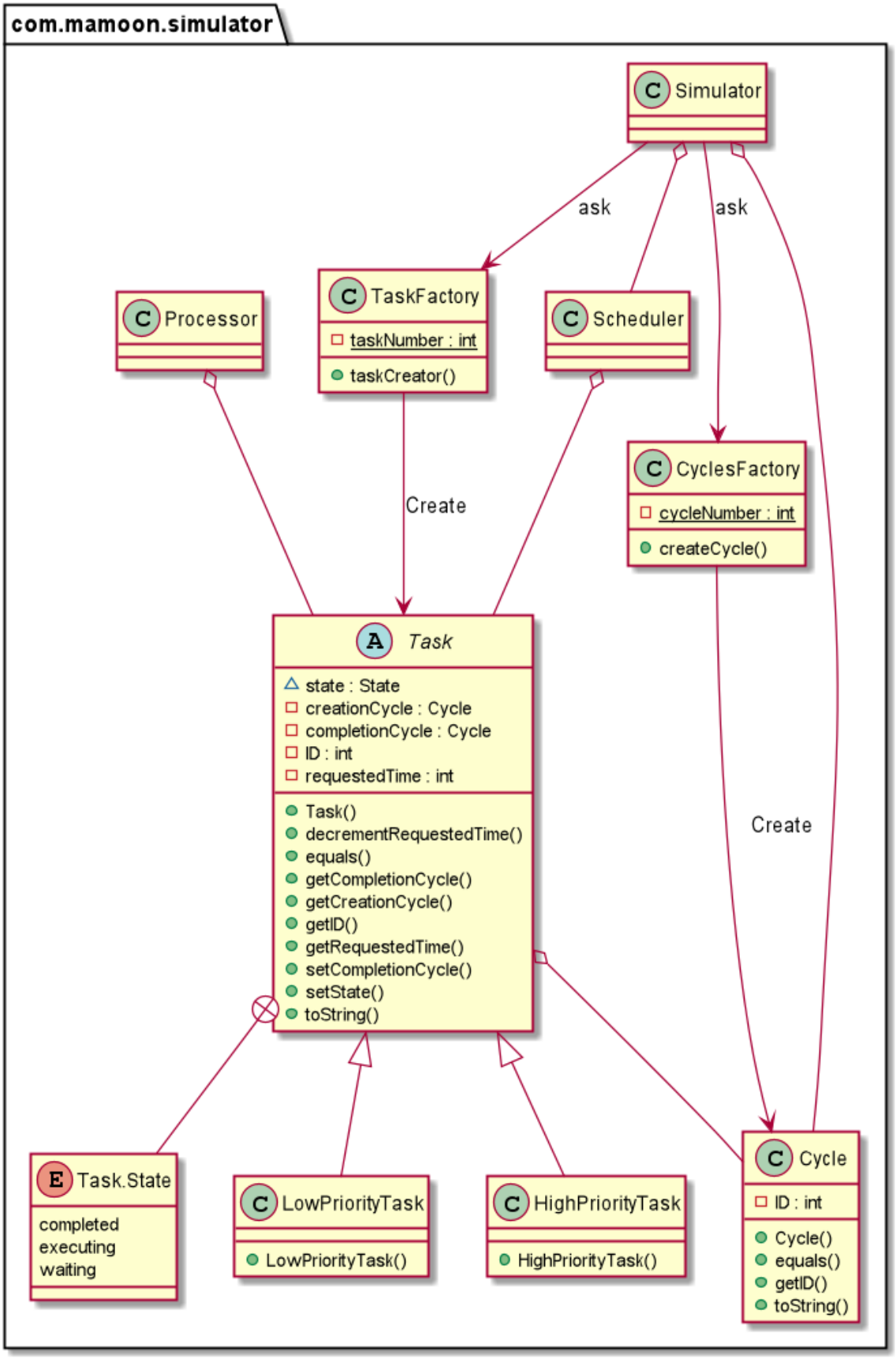
After creating the processors, the details of each task will be added array list in the simulator class where the type of this list is a future task. The list will be sorted based on creation time, every time the add task function is called it will add a task then sort the list.

Now Input is prepared to work with the simulator and processors are created. Tasks and cycles are going to be created dynamically and tasks will be created depending on the generated cycle.

When a cycle is created by the cycles factory it will be added to the array list of cycles in the simulator after that the simulator will check if there a task will be created at this cycle. In case a task or tasks are created, they will be sent to schedular queues.

The factories have been used for tasks, cycles, and processors for very important reasons which are keeping unique IDs for each component and case there will be new types of processors and tasks in the future.

The next figure shows the UML for tasks and cycles and their relationship with simulator and schedular:


com.mamoon.simulator
Simulator
ask
ask
Processor
TaskFactory
taskNumber : int
taskCreator()
Scheduler
Create
CyclesFactory
cycleNumber : int
createCycle()
Task
state : State
creationCycle : Cycle
completionCycle : Cycle
ID : int
requestedTime : int
Task()
decrementRequestedTime()
equals()
getCompletionCycle()
getCreationCycle()
getID()
getRequestedTime()
setCompletionCycle()
setState()
toString()
Create
Task.State
completed
executing
waiting
LowPriorityTask
LowPriorityTask()
HighPriorityTask
HighPriorityTask()
Cycle
ID : int
Cycle()
equals()
getID()
toString()

Simulation and Scheduling

As it is mentioned before, the schedular will receive tasks and put them in queues these queues are waiting low priority queue and waiting high priority queue. Both queues are sorted depending on the requested time. That means the task which has the shortest request time will be attached to a processor first.

If the task is executed, it will be moved from waiting queues to executing array list. If it is completed it will be moved from executing to the completed array list.

If a task with high priority is created and all processors are busy, the schedular will replace a low priority task attached to a processor with a high priority task.

The simulation and scheduling will have a kind of repeat actions, because of that templet design pattern has been used in simulator and scheduler

For simulator class, the function which has the templet is a simulation and the templet is

- Create cycle: generate a new cycle using cycles factory to get a unique ID then add the cycle to the list of cycles in the simulator.
- Processors working: decrement requested time for each task attached to a processor.
- Create task: creating a task or tasks because of creation cycle has been generated.
- Scheduling: is a function called by schedular instance with passing the last cycle created in it to start templet for schedular.

For schedular class, the function which has the templet is a Scheduling and the templet is

- Clean completed task: if the requested time for a task attached at the processor is zero the task needs to be released from the processor and put the completed task in the completed list.
- Attach task to a processor: attach task in a queue to an idle processor. It will attach high priority tasks before the low priority task.
- Replace task: replace a low priority task at processor with a high priority task.
- Clean completed task: it is useful if the task needs 0 cycles to finish executing.
- Attach task to a processor: a processor or processors will be free after the previous step; a task needs to be attached to them.

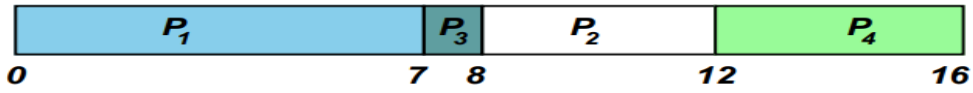
Both templets are going to be performed until there are no future tasks and all processors are idle.

From the previous, the algorithm which has been used for scheduling is the shortest job first with keeping in mind high priority first. As it is written in Operating System Concepts – 8th Edition “Shortest-Job-First (SJF) Scheduling: Associate with each process the length of its next CPU burst

- Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
- The difficulty is knowing the length of the next CPU request
- Could ask the user”

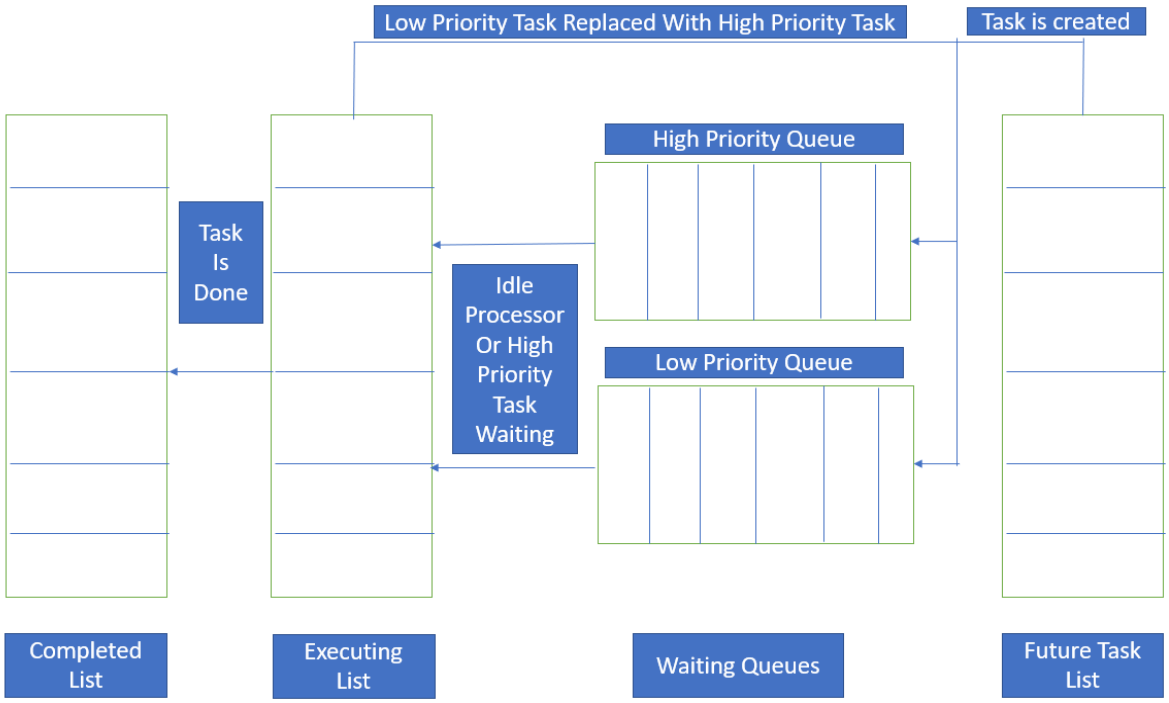
An example from “Operating Systems Steven Hand Michaelmas Term 2010”

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

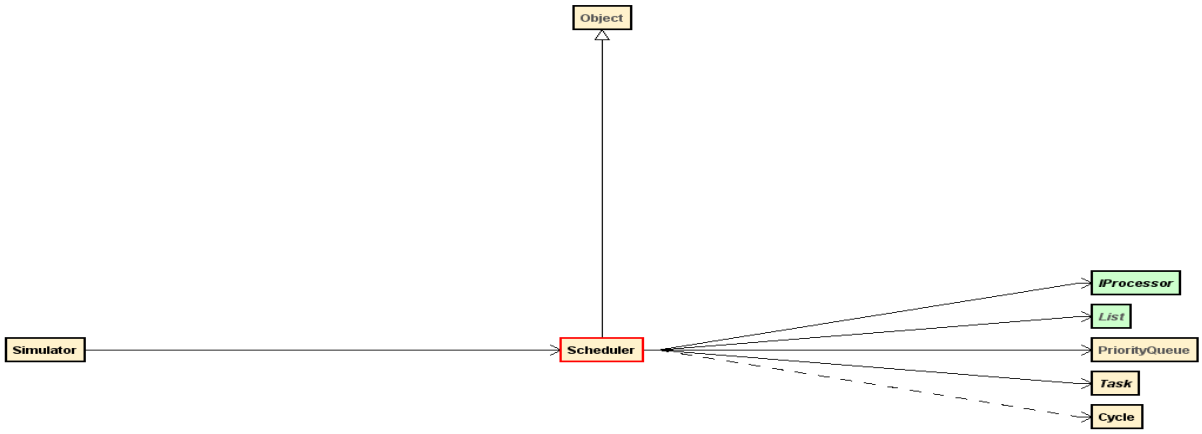


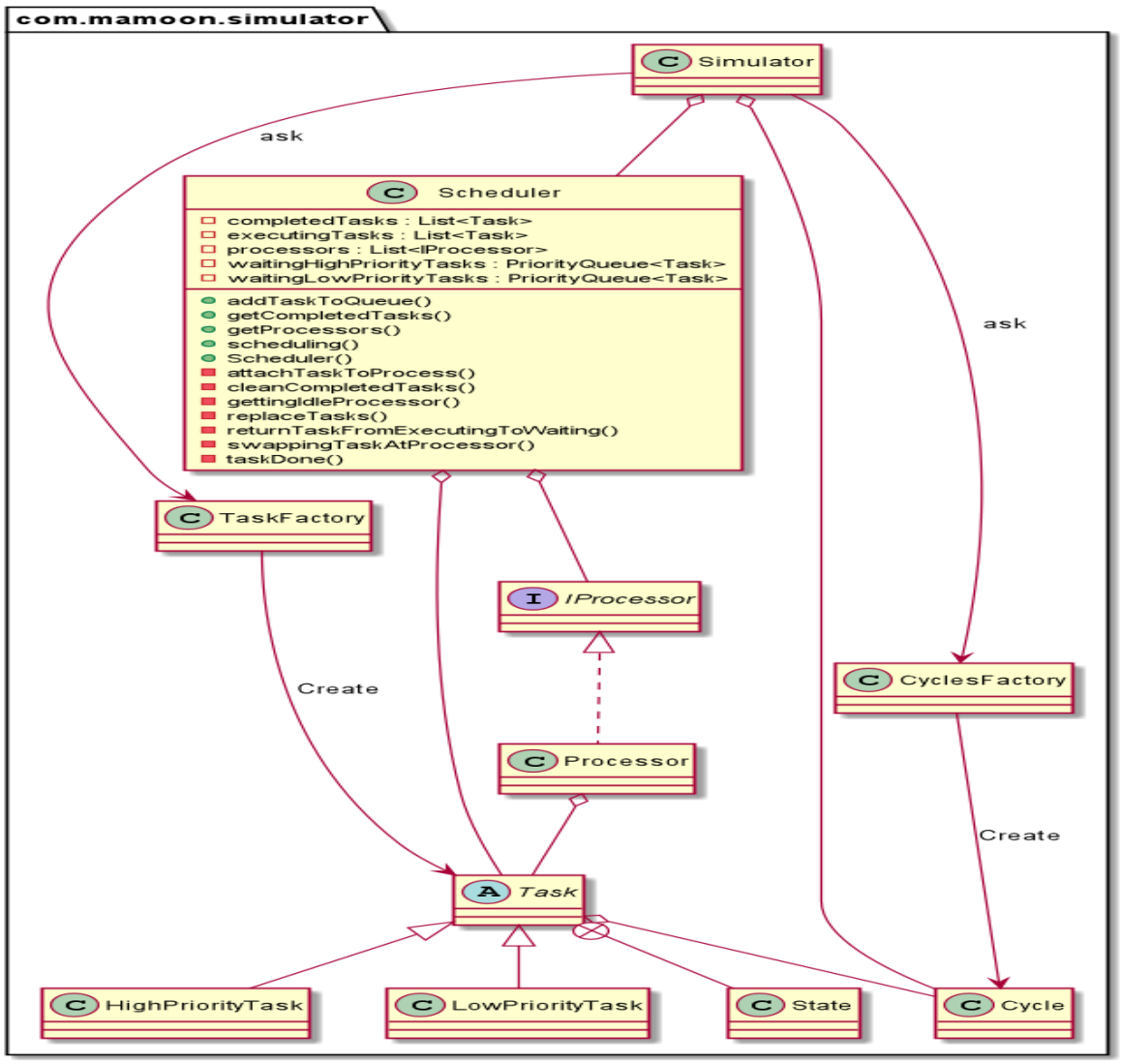
And our simulator will ask the user for task details which make this algorithm suitable for our simulator! And because the simulator mostly will have more than one processor, there will be a very low possibility for starvation.

What about the queues and lists in the scheduler? The next figure answer this:



From a class view, we can see the scheduler like this:





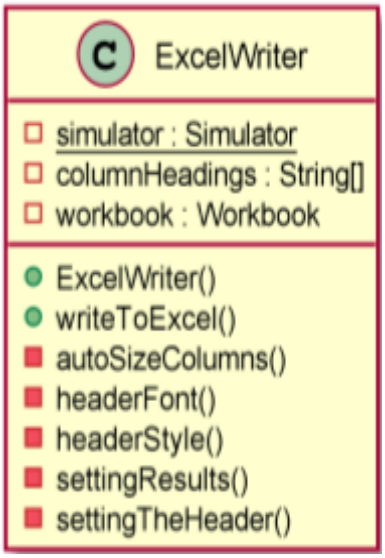
Printing the Output for the User

The output will be in an excel file and follow this format as shown in the next figure:

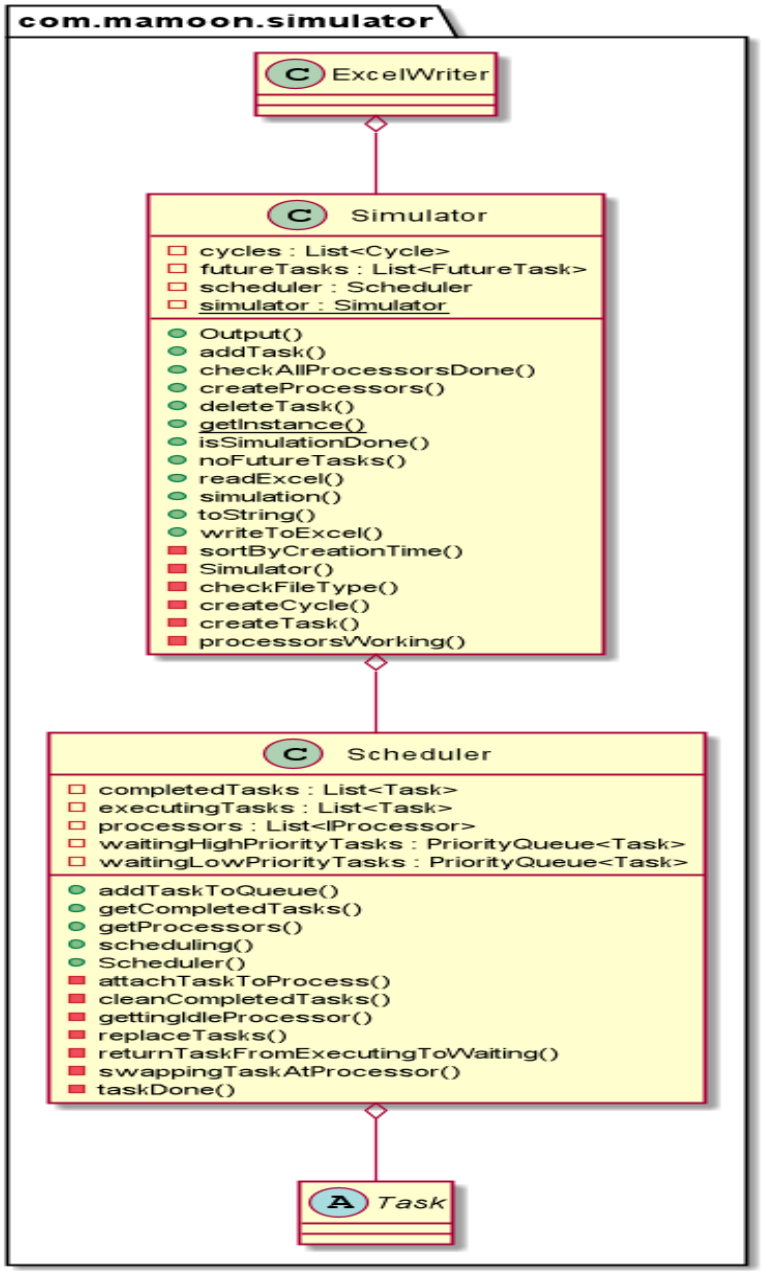
	A	B	C	D	E
1	**Task ID**	**Creation Time**	**Priority**	**Completion Time**	**Total simulation Time**
2	3	5	High	15	
3	4	6	Low	22	
4	6	8	High	28	
5	5	7	Low	32	
6	1	1	Low	39	
7	9	50	High	65	
8	8	10	Low	67	
9	2	1	Low	76	
10	10	100	Low	120	
11	7	9	Low	127	
12					127

As it is shown in the figure the first row shows the header for each column, the output is sorted by completion time. The user should write the name of the excel file. If it is doesn't exist, the simulator will create an excel file for the user.

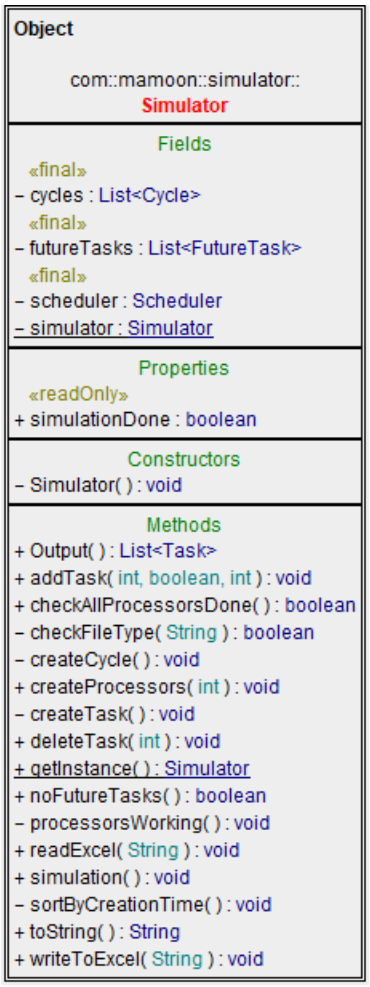
There will be a class to handle writing to an excel file as shown in the figure:

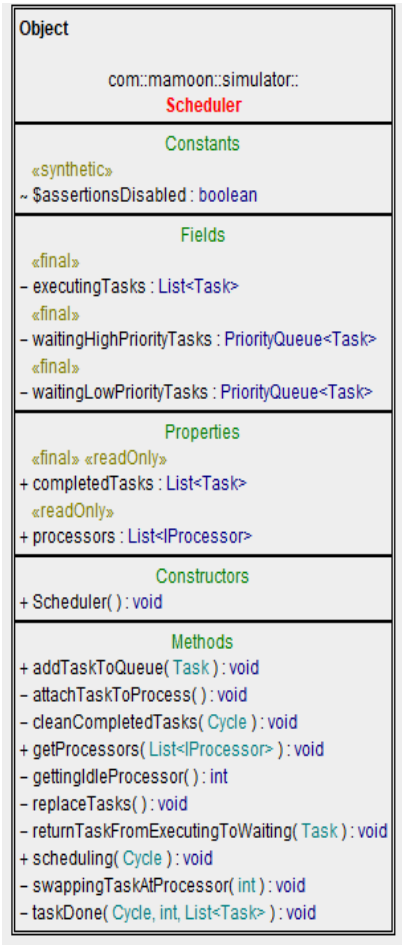


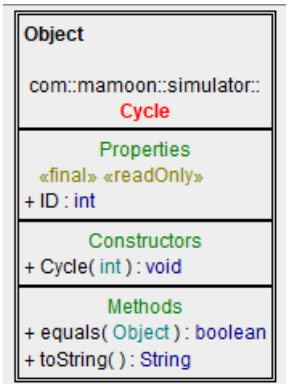
And its relationship with the simulator:

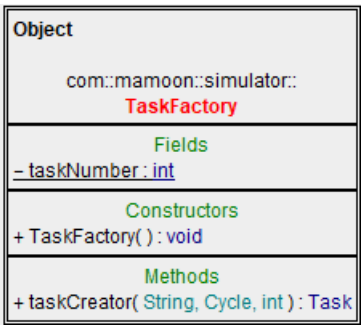


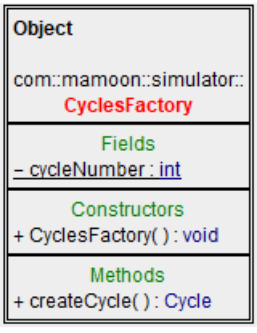
The UML for each class using Class Visualizer:

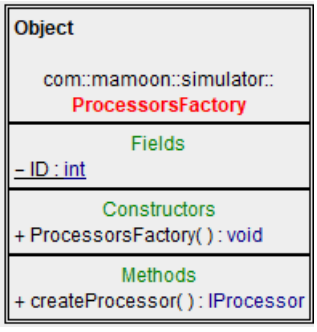


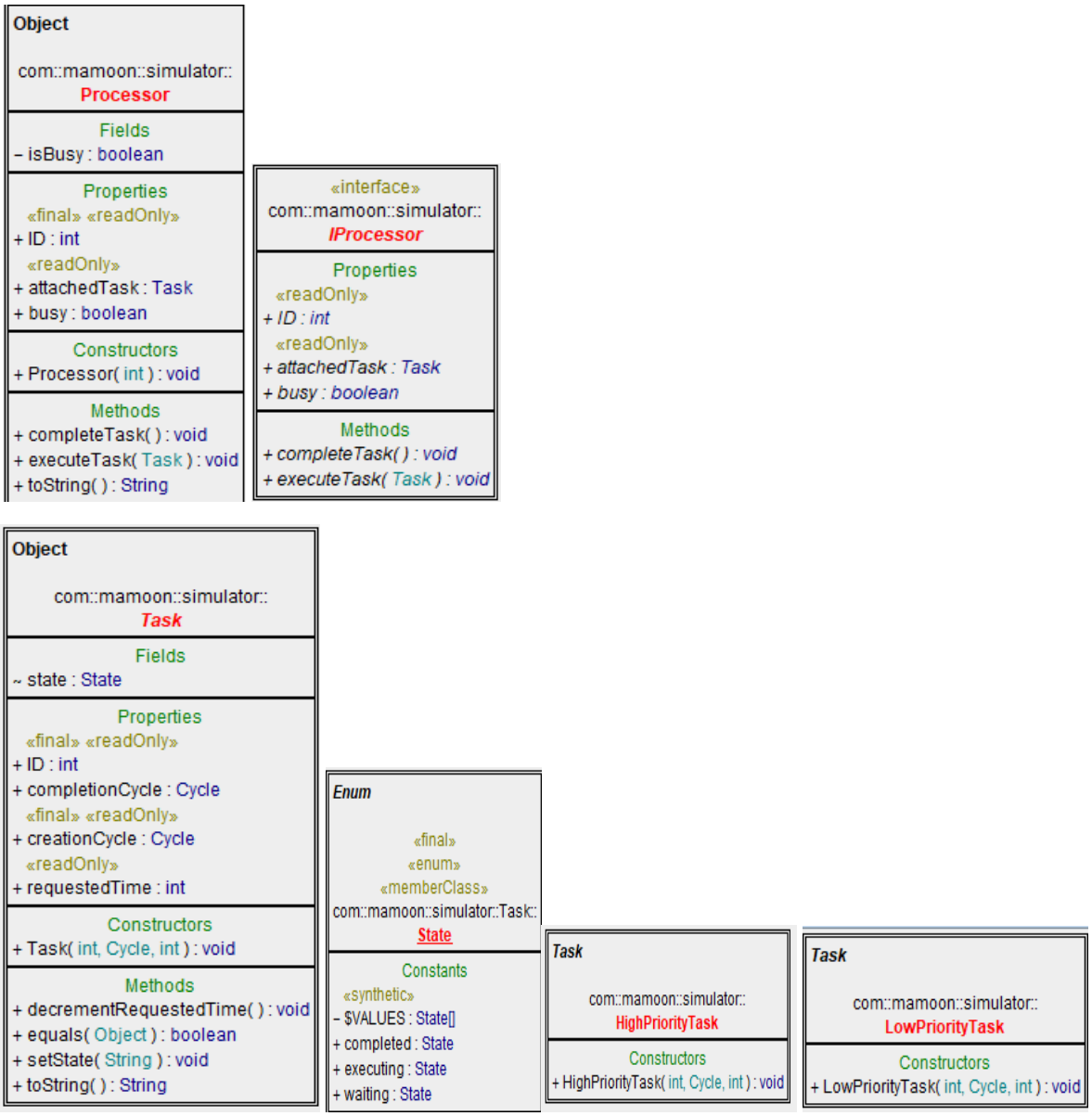


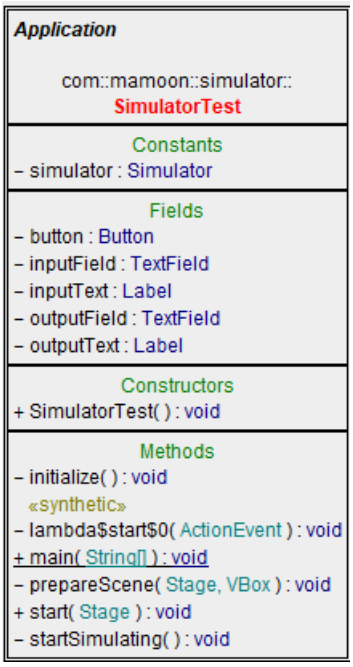


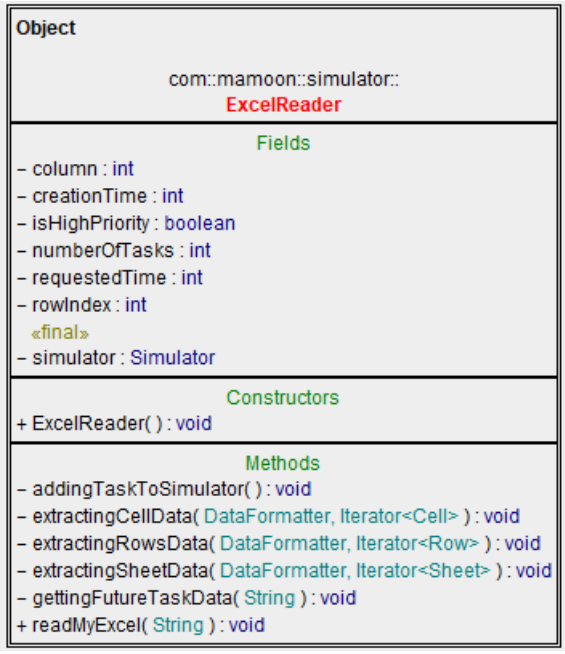


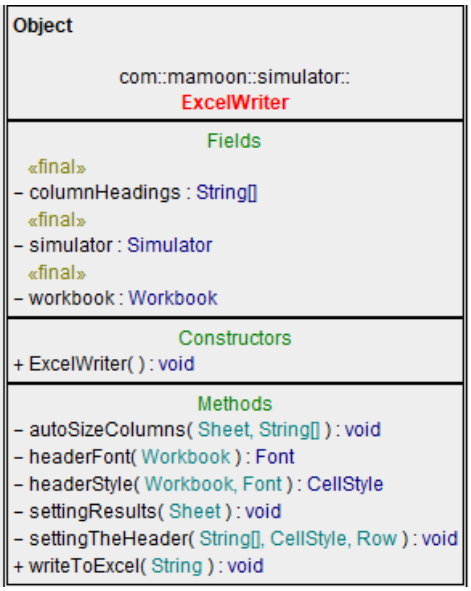




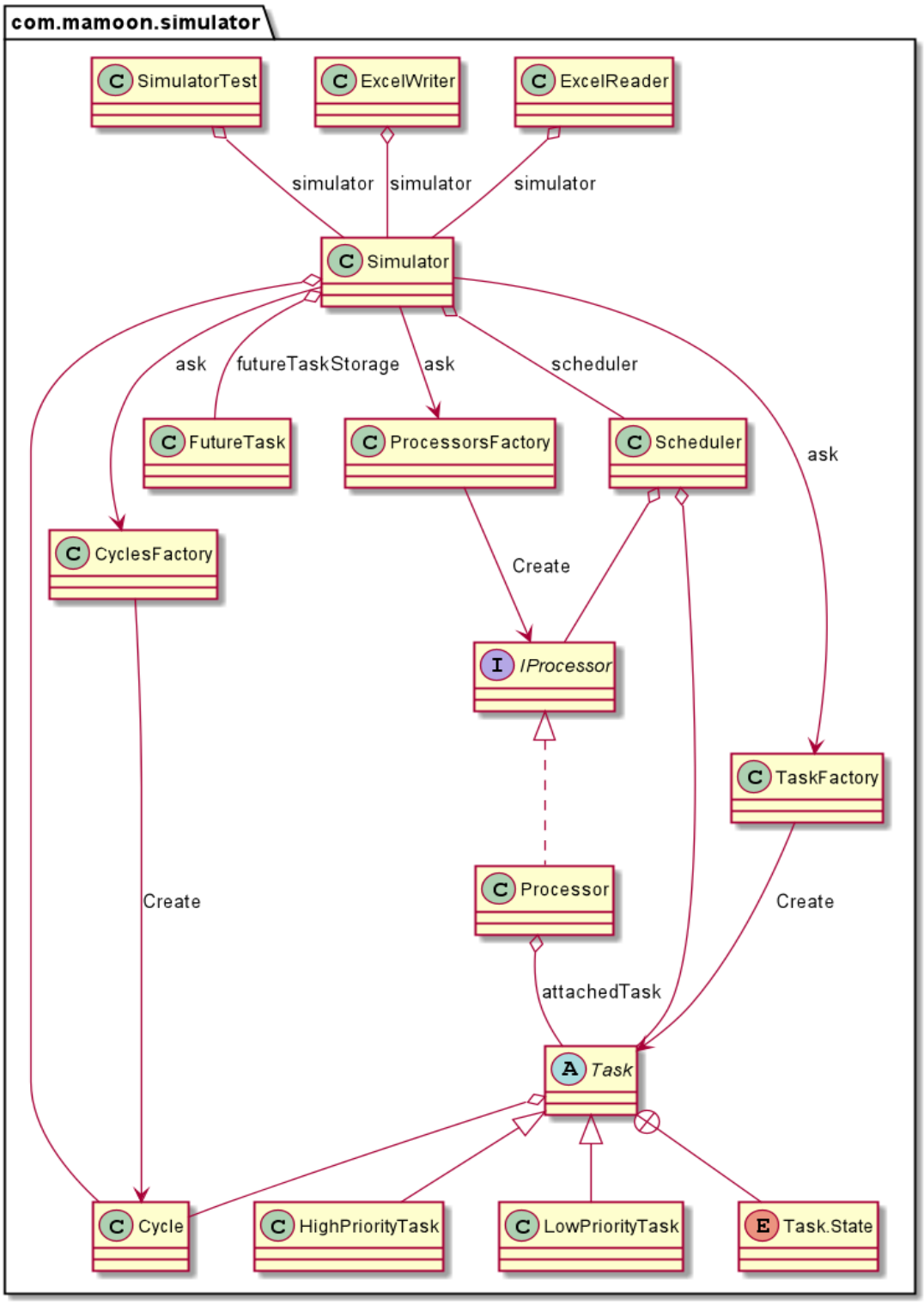

Object
com::mamoon::simulator::
Processor
Fields
– isBusy : boolean
Properties
«final» «readOnly»
+ ID : int
«readOnly»
+ attachedTask : Task
+ busy : boolean
Constructors
+ Processor(int) : void
Methods
+ completeTask( ) : void
+ executeTask(Task) : void
+ toString( ) : String
«interface»
com::mamoon::simulator::
IProcessor
Properties
«readOnly»
+ ID : int
«readOnly»
+ attachedTask : Task
+ busy : boolean
Methods
+ completeTask( ) : void
+ executeTask(Task) : void
Object
com::mamoon::simulator::
Task
Fields
~ state : State
Properties
«final» «readOnly»
+ ID : int
+ completionCycle : Cycle
«final» «readOnly»
+ creationCycle : Cycle
«readOnly»
+ requestedTime : int
Constructors
+ Task(int, Cycle, int) : void
Methods
+ decrementRequestedTime( ) : void
+ equals(Object) : boolean
+ setState(String) : void
+ toString( ) : String
Enum
«final»
«enum»
«memberClass»
com::mamoon::simulator::Task::
State
Constants
«synthetic»
– $VALUES : State[]
+ completed : State
+ executing : State
+ waiting : State
Task
com::mamoon::simulator::
HighPriorityTask
Constructors
+ HighPriorityTask(int, Cycle, int) : void
Task
com::mamoon::simulator::
LowPriorityTask
Constructors
+ LowPriorityTask(int, Cycle, int) : void


Application
com::mamoon::simulator::
SimulatorTest
Constants
- simulator : Simulator
Fields
- button : Button
- inputField : TextField
- inputText : Label
- outputField : TextField
- outputText : Label
Constructors
+ SimulatorTest( ) : void
Methods
- initialize( ) : void
«synthetic»
- lambda$start$0(ActionEvent) : void
+ main(String[]) : void
- prepareScene(Stage, VBox) : void
+ start(Stage) : void
- startSimulating( ) : void

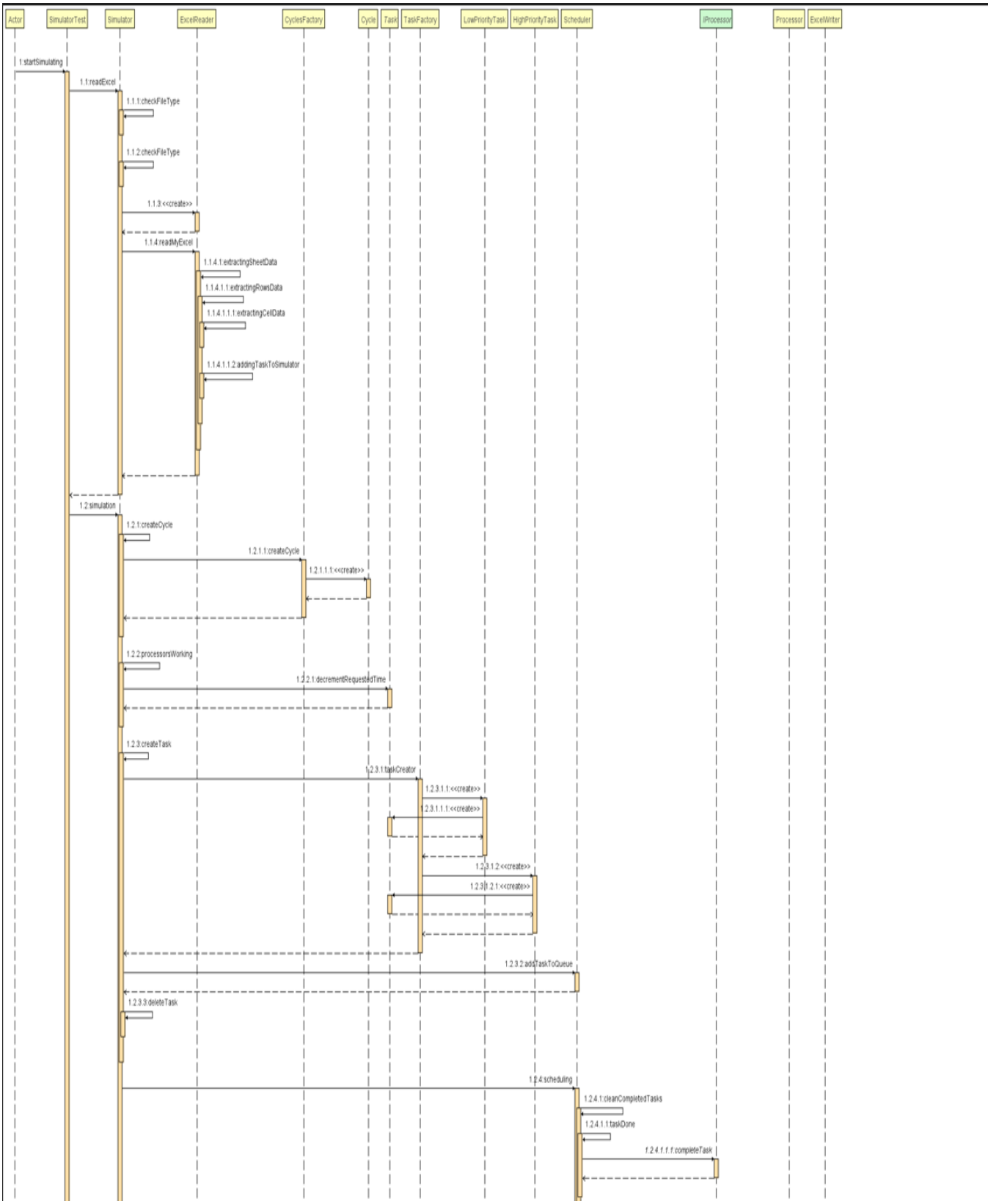

Object
com::mamoon::simulator::
ExcelReader
Fields
- column : int
- creationTime : int
- isHighPriority : boolean
- numberOfTasks : int
- requestedTime : int
- rowIndex : int
«final»
- simulator : Simulator
Constructors
+ ExcelReader( ) : void
Methods
- addingTaskToSimulator( ) : void
- extractingCellData(DataFormatter, Iterator<Cell>) : void
- extractingRowsData(DataFormatter, Iterator<Row>) : void
- extractingSheetData(DataFormatter, Iterator<Sheet>) : void
- gettingFutureTaskData(String) : void
+ readMyExcel(String) : void

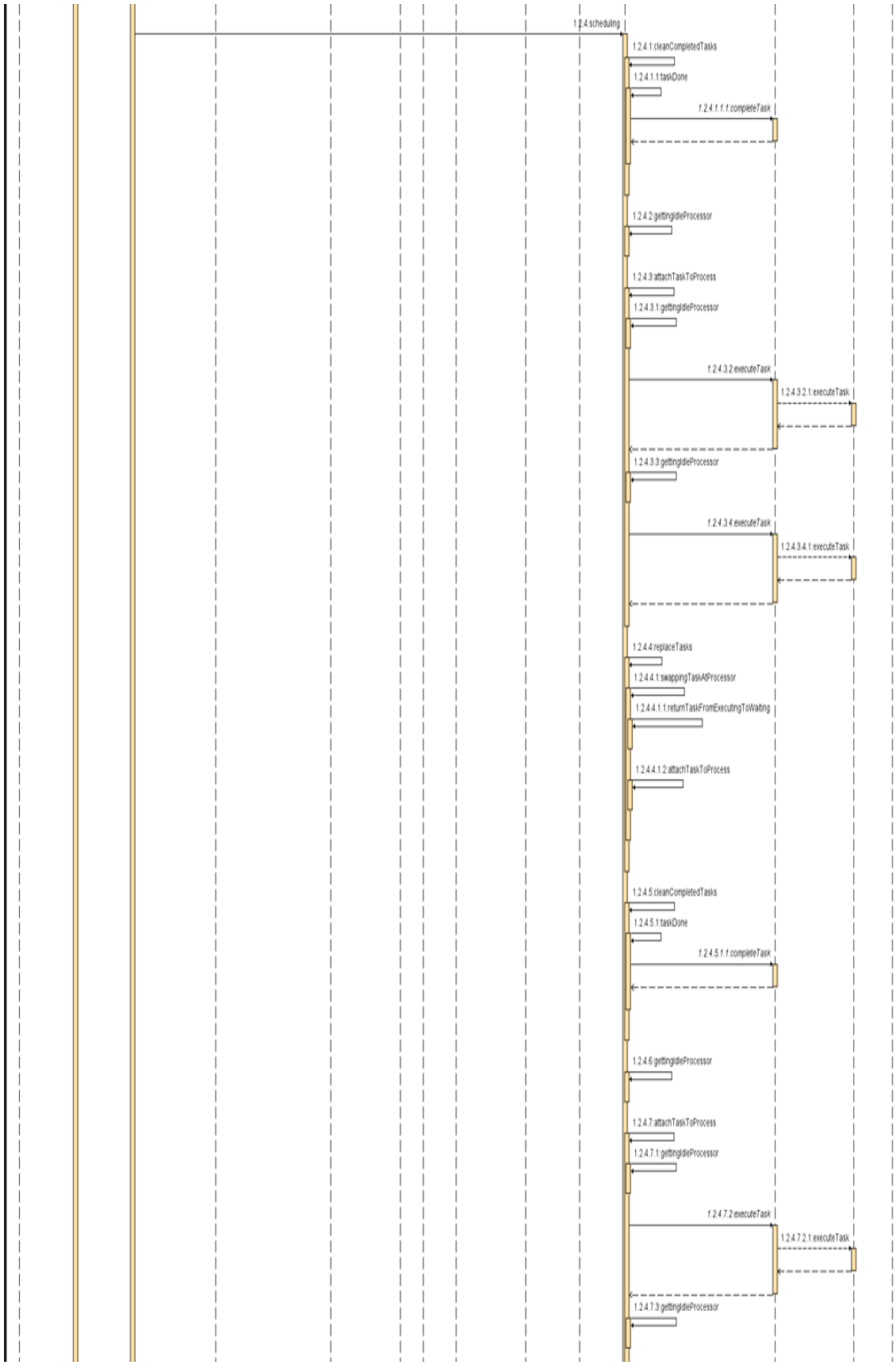

Object
com::mamoon::simulator::
ExcelWriter
Fields
«final»
- columnHeadings : String[]
«final»
- simulator : Simulator
«final»
- workbook : Workbook
Constructors
+ ExcelWriter( ) : void
Methods
- autoSizeColumns(Sheet, String[]) : void
- headerFont(Workbook) : Font
- headerStyle(Workbook, Font) : CellStyle
- settingResults(Sheet) : void
- settingTheHeader(String[], CellStyle, Row) : void
+ writeToExcel(String) : void

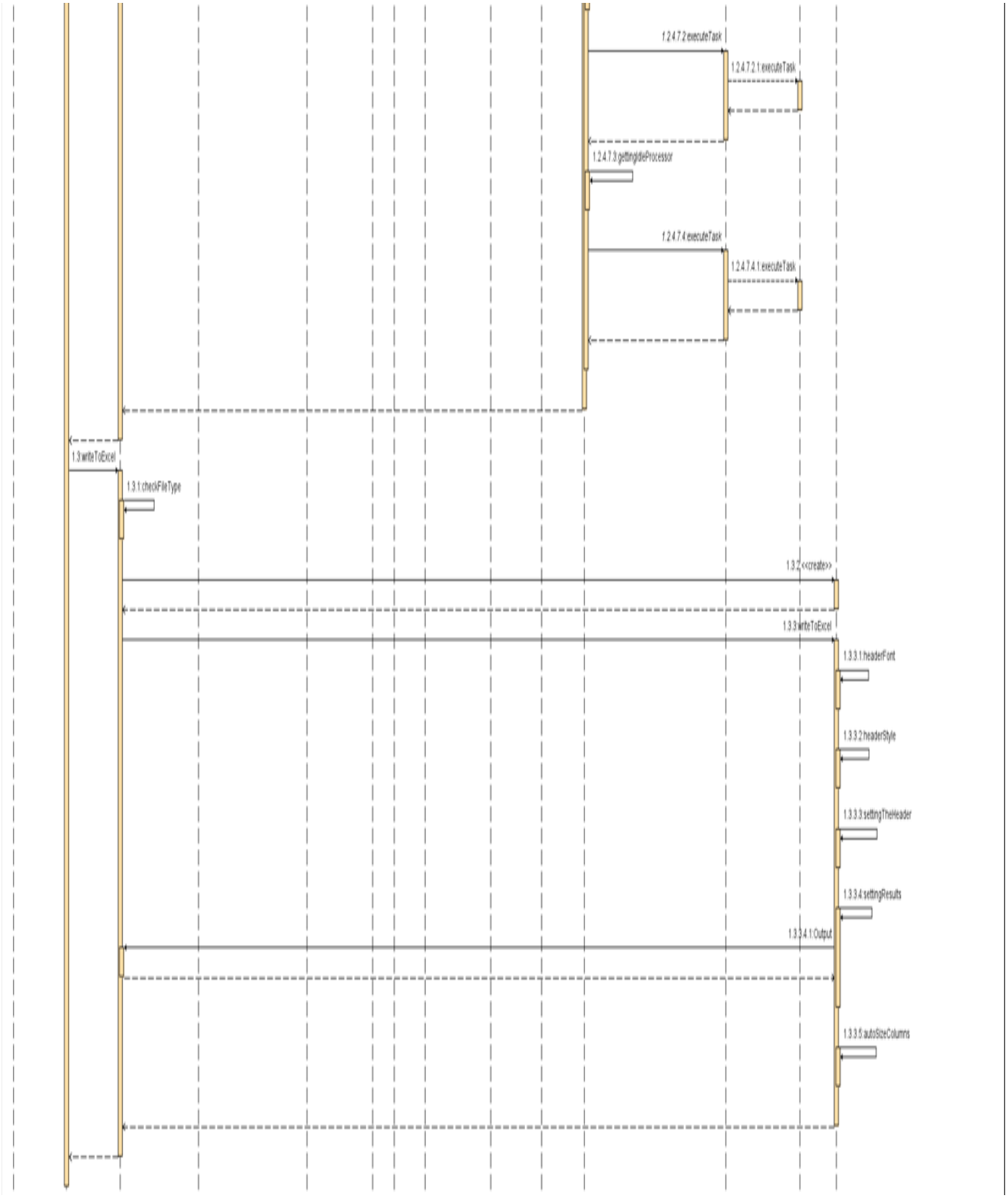
Simulator Class Diagram



Simulator Sequence diagram

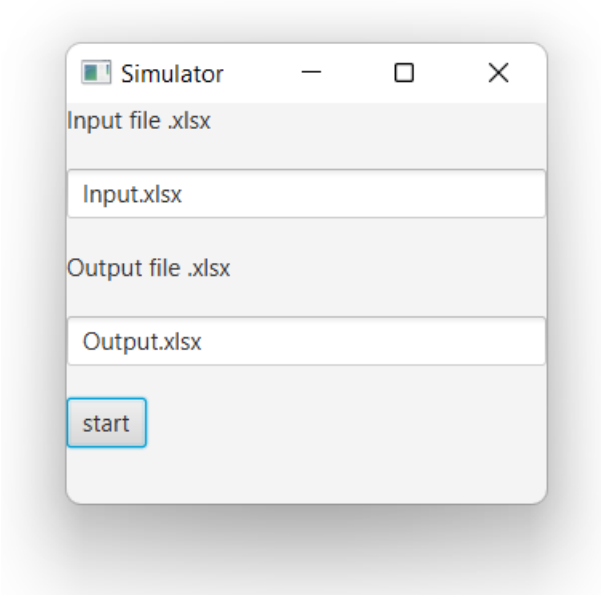



1.2.4:scheduling
1.2.4.1:cleanCompletedTasks
1.2.4.1.1:taskDone
1.2.4.1.1.1:completeTask
1.2.4.2:gettingIdleProcessor
1.2.4.3:attachTaskToProcess
1.2.4.3.1:gettingIdleProcessor
1.2.4.3.2:executeTask
1.2.4.3.2.1:executeTask
1.2.4.3.3:gettingIdleProcessor
1.2.4.3.4:executeTask
1.2.4.3.4.1:executeTask
1.2.4.4:replaceTasks
1.2.4.4.1:swappingTaskAtProcessor
1.2.4.4.1.1:returnTaskFromExecutingToWaiting
1.2.4.4.1.2:attachTaskToProcess
1.2.4.5:cleanCompletedTasks
1.2.4.5.1:taskDone
1.2.4.5.1.1:completeTask
1.2.4.6:gettingIdleProcessor
1.2.4.7:attachTaskToProcess
1.2.4.7.1:gettingIdleProcessor
1.2.4.7.2:executeTask
1.2.4.7.2.1:executeTask
1.2.4.7.3:gettingIdleProcessor


1.2.4.7.2:executeTask
1.2.4.7.2.1:executeTask
1.2.4.7.3:gettingIdleProcessor
1.2.4.7.4:executeTask
1.2.4.7.4.1:executeTask
1.3:writeToExcel
1.3.1:checkFileType
1.3.2:<<create>>
1.3.3:writeToExcel
1.3.3.1:headerFont
1.3.3.2:headerStyle
1.3.3.3:settingTheHeader
1.3.3.4:settingResults
1.3.3.4.1:Output
1.3.3.5:autoSizeColumns

Working With User

To make this jar file work without needing the black screen, a simple graphical user interface has been created. JavaFX has been learned and used for the simulator; the next figure shows the user interface design:



The first text field is to enter the input file name.

The second text field is to the output file name.

The start button will start the simulating and once it is activated it will be disabled.

If the simulation is done the application will close and the output file will be generated with results.

Note that both should end with ".xlsx" the simulator will not accept another format.

The Input file should be in the same folder as the jar file. The output file will be also in the same folder for the jar file.

References

1- https://www.cl.cam.ac.uk/teaching/1011/OpSystems/os1a-slides.pdf
2- http://www.facweb.iitkgp.ac.in/~isg/OS/SLIDES/ch6-CPU_Scheduling.pdf
3- Schoology assignment.
4- The slides are provided in Schoology.
5- https://github.com/chargeahead/ExcelRead
6- https://github.com/chargeahead/ExcelProjectWrite
7- https://www.youtube.com/watch?v=FLkOX4Eez6o&list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG&index=1
8- https://www.class-visualizer.net/
9- https://plantuml.com/class-diagram