Seminar one - report

Name: Meron Habtemichael
Course: Operating system

Observation 1

How do you exit the java command shell application (without exiting the window)?

- We can exit the java command shell application by entering ‘end’ as an input. The program actually terminates by calling the command `System.exit(0);`

Observation2

Can you first start an editor such that notepad and then another application without first exit the first application? Can you explain why?

- No, because the CPU is blocked by the process we just initiated.

Observation 3

When you start, for example, notepad and then exit the command shell, is the notepad program still running? Why?

- Because we have system called the OP to create a whole new process for the application.

Task 1

- ❖ How did you implement the task?
  - ➢ First, I learned three ways of implementing threads(like in the demo class) and all the concepts needed.
  - ➢ Created a separate Thread to run concurrently when we are accessing the Command line.
  - ➢ The Thread has a contractor to take a command input. That is again stored as a variable and used to in the createProcess command.
  - ➢ Since we want to open as many applications as possible. We need a separate Thread to manage each process. So in the while loop, I replaced the createProcess(commandLine) with a new Thread object each time a user enters the command.
- ❖ Why did you solve it in the way you did it?
  - ➢ I think that’s how I viewed the solution of the problem to be the easiest to meet the requirements.
  - ➢ It was tricky to get the hang of the problem at first. But once you break it into a plan and follow simple steps with the execution it was okay.
- ❖ What difference in behavior did you notice?
  - ➢ I realized how servers were able to handle multiple clients at one time. Using multiple Threads. And always having plus one Thread listening for new command which is the main Thread creating all these new Thread objects.
  - ➢ The operating system is all that we write programs to communicate with.

Task 2

- ❖ How did you implement the task?
  - ➢ First created a static list of strings called errorList
  - ➢ Then I added every caught error into the list when creating a process from the input command.
  - ➢ Then if a user types in “Showerrlog”, loop throughout the list and display the error history.
- ❖ Why did you solve it in the way you did it?
  - ➢ That was straightforward.

➢ Using a static list was ideal in case the application gets bigger and we form other classes. That can be accessed or shared easily.

❖ What difference in behavior did you notice?

➢ IOException happens pretty often.

Task 3 - stopwatch

❖ How did you implement the task?

➢ Created a method counter to keep up with the counting of everything. Basic rules of conversion of milliseconds to seconds.

➢ The thread sleep time was 10 milliseconds so I represented the millisecond variable to contain a maximum of 100 values ie 10 * 100 = 1000.

➢ Created a separate thread to implement the stopwatch functionality.

❖ Why did you solve it in the way you did it?

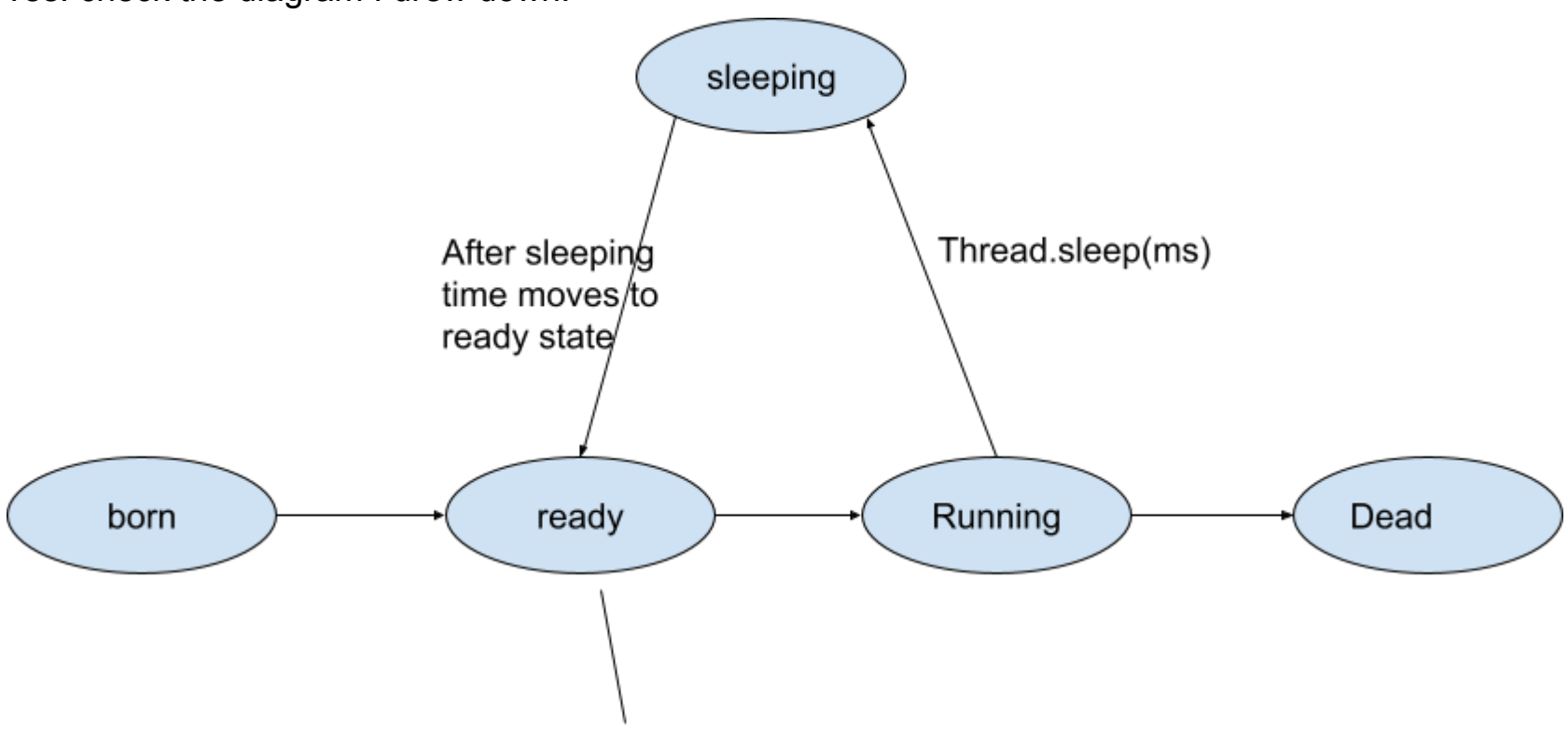
➢ I used my way because that is convenient and fast from my perspective. And it meets the requirement listed in the task description.

➢ I tried to follow simple and clear steps.

❖ What difference in behavior did you notice?

➢ I noticed that having parallel threats running can be really powerful. The thread library is rich in methods and functionalities to use to solve a problem.

- *What does the Thread.join() do? What happens if we remove it from the main method?*
  - The Thread.join() method allows the stopwatch Thread to complete its execution. This helps the next line to be executed after the thread has finished its job. In that case, we were able to print that next coming line after waiting for the Thread to die first.
- *Compare the real elapsed time using a stopwatch on your phone with your Java implementation. Does the elapsed time differ, why?*
  - Yes. check the diagram I drew down.



Here it deals with Thread scheduler. When this thread is moving to running state is totally dependent to this Thread scheduler. This consumes some time each time the stopwatch loops here. So it makes the stopwatch Thread a little slower that real time.