A short **PDF** design documentation (**5 pages max**) - which may also get you some partial credit if I cannot get your program to execute as it should. The report should (among other things) describe/provide:

1. what aspects of the project you got working / not working / partly working

Everything listed below is working:

* A delivery of 10 books is made with a probability of 0.01 every tick (configurable).
* Bookshelves have a maximum capacity (configurable, if set to 0 then unlimited).
* 10 assistants by defaults (configurable, minimum 1).
* 6 sections by default (configurable, minimum 1, possibility to generate sections).
* Assistants prioritize empty sections when delivering books. (True ?).
* Real time data reporting.
* Assistants take breaks. By default an assistant can start taking a break after 200 ticks and must take a break after 300 ticks. Each tick when a break is possible the assistant has a 0.01% chance of taking a break. Configurable.
* A lot of parameters are configurable. Launch the program with ‘-h’ option when launching to learn about all of them.
* The mapping between ticks and time is configurable. Default a tick last 10 milliseconds.
* Primitive data printing in the terminal.

1. how you divided the work between you

Most of the work was done side by side. We can however say that some part were mostly done by one or the other as seen below:

Martin Ferrand:

* TimeScheduler.java
* SynchronizedThread.java
* Main.java
* StatsManager.java

Clovis Gilles:

* Assistant.java
* Bookstore.java
* Book.java
* Section.java
* Logger.java
* Makefile

1. how to compile and run the code (on Linux/Ubuntu) e.g., if there is a config file to change parameters

make should work ? (VERIF SI JAVA 19 FONCTIONNE!)

1. what are the considered tasks/dependencies in your program

What does that mean ?

1. what patterns/strategies did you use to manage concurrency

Nearly all objects inherit of SynchronizedThread with work in tandem with TimeScheduler to provide synchronisation between every component. The objective is to make sure that the tick count is consistent between every thread and to make sure that the threads are well handled. This mean every thread have been started (even if news thread are added later on while the program is running), that no thread runs much faster than any other by waiting for all to finish what they were doing (we are aware that we are wasting cpu time of faster thread in this configuration). We are using java.util.concurrent.CyclicBarrier to achieve the synchronisation.

SynchronizedThread is used to create any objects that need to run in a thread that needs to be synchronized (to get correct tick count). Therefore it is an abstract class that inherit from Thread and implementing the run function of Thread that must not be overridden by child class. The function doWork() is abstract and therefore must be implemented by child class, it must perform every operations the thread must do during a single tick. The rest of the methods are utilities to setup the object, exit the thread and other miscellaneous functions.

The rest of concurrency issues are usually dealt with by using the synchronized keyword with provide an easy to use monitor when common resource is modified. It is not used everywhere because sometimes it doesn’t matter if the resource is modified by multiple thread at the same time.

1. how your solution addresses issues like fairness, prevention of starvation etc.

Commencer a jouer du violon tel un maitre artiste.