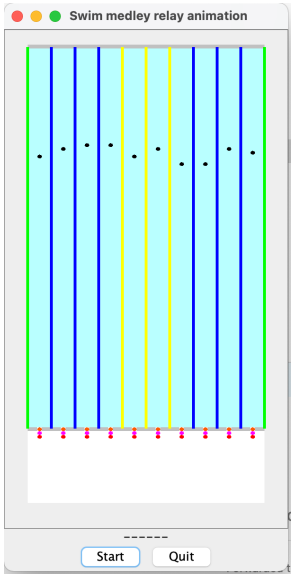
CSC2002S 2024 ASSIGNMENT
Multithreaded CONCURRENT Simulation
Medley Relay Swimming Race
Version 1



1. Objective

In this assignment, your objective is to correct the existing code for a multithreaded Java simulation of a 4x100 medley relay swimming race. You must use **synchronization mechanisms** to ensure that the simulation adheres to the specified synchronization constraints and maintains safety and liveness.

2. Medley simulation

The assignment entails producing a Java simulation of 10 teams of four swimmers taking part in a relay race, ensuring that it follows defined behavioral rules.

In this simulation, the stadium is represented as a **grid** of a specified size**,** containing an **pool**, an **entrance door**, a starting line and a free area.

The simulation begins when the user presses the **Start button**.
Swimmers then enter through the entrance door one by one in the order of their swim stoke, proceed to line up at the starting block, then swim their 100m of the the race as a relay (swimmers start only after their previous teammate has finished): each swimmer swims to the end of the 50m pool and back. Once they have finished their 100m, swimmers exit the pool and stand at the back of the stadium, except for the last member of the team, who stays in the pool once done. You are provided with a video of the working simulation.

You do not need to implement the whole simulation – you are provided with a skeleton Java implementation. The simulation runs, but does not work properly, as **a number of the rules for the simulation are violated**.

Your task is to fix the code, identifying and correcting all the concurrency issues so that all rules are all complied with all the time and the simulation suffers from no safety or liveness problems. Be careful to ensure that compound actions on thread-safe classes are also protected.

3. The behaviour rules for the medley simulation

These rules are listed in a recommended order of trying to enforce them. Start at the beginning and work your way through the list.

1. The **Start button** initiates the simulation – no swimmers enter the stadium until this is pressed.
2. *The Quit button terminates the simulation (it does, this works already).*
3. Only **one swimmer** is allowed on **a grid position** (GridBlock) **at any point in time**.

4. Swimmers move block by block and simultaneously to ensure liveness.
5. After the start button is pressed, swimmers enter through the **entrance door** one **at a time**, in the **race order of their swim stroke** (e.g. Backstroke first)**, arriving swimmers** must **wait** if entrance door is occupied.
6. Swimmers **line up at the starting blocks** for their team in their **race order** (order of the swim stroke).
7. The **race begins only when the all backstroke swimmers** (the first to swim, black circles in the simulation) for every team are in place. If other team members have not arrived yet, that is fine: the race can start. The race starts with the backstroke swimmers swimming to the end of the pool and back.
8. Other members of the team **can only start** their leg of the race when their **previous team member has finished** their leg – this is a relay. So breaststroke waits for backstroke, butterfly waits for breaststroke and freestyle waits for butterfly.
9. The team with the **first freestyle winner home wins the race**.

3.1. Possible extensions

Once you have done the main part of the assignment, you may extend it for a maximum of 3 **bonus marks**, making improvements to the appearance (more realistic) or the behaviour (e.g. second and third places in the race are assigned, or the winning team does a victory dance etc. etc.) of the simulation. You can get creative, but make sure you do the main assignment first!

4. Code Description

The assignment provides a skeleton Java implementation with the following classes:

- MedleySimulation.java: Main class for setting up and starting the simulation, including the GUI.

- CounterDisplay.java: Simple class for displaying/updating text in the main GUI.
- FinishCounter.java: Simple class for tracking whether someone has won the race.

- GridBlock.java: Class representing grid blocks.
- StadiumGrid.java: Class representing the grid for the simulation, made up of grid blocks.

- PeopleLocation.java: Class for storing locations of people (swimmers only, but could add other types) in the simulation. This is what StadiumView reads to display the swimmers.
- StadiumView.java: threaded JPanel class for visualization of the simulation

- SwimTeam.java: Class representing a swim team.
- Swimmer.java: Class representing each swimmer as a thread. Swimmers have one of four possible swim strokes: backstroke, breaststroke, butterfly and freestyle, which have speeds and swimmers have slightly different speeds.

4.1. Running the code

Once you have had a look at the code, run it to see how it is behaving (which will not be at all like the demo provided). To run the code on a Linux (or MacOS) machine, copy the zip file onto the computer, unzip it:

```
unzip PCP_ParallelAssignment2024.zip
```

change into the top folder:

```
cd PCP_ParallelAssignment2024/
```

and then execute the command:

```
make run
```

This will execute the code.

5. Assignment Submission:

You will need to submit your assignment in two places:

1. Submit the **code** to the automarker as a zipped **archive.** Your submission archive must contain:
   - **all the files** needed to run your solution
   - a **Makefile** so that the following command **must run your solution** correctly.
     `make run`
   - **a GIT usage log (**as a .txt file, use `git log –all` to display all commits and save).
     You simulation must **comply with all specified synchronisation requirements.**

2. Submit a short report (**pdf format only**) to the Amathuba assignment.In the report, explain how you enforced simulation rules and how you protected agains data races (there are many!). Detail the synchronization mechanisms you added to each class and their appropriateness. Describe any extensions you made and why they are worthwhile.

5.1. Marking

A rubric will be attached to the assignment on Amathuba as a guide to marking.
Stay updated with any additional assignment information on the PCP MSTeam. You can ask questions on the team too.