ELLZUB001

Assignment 2

Medley Simulation

Introduction

The Medley Simulation is a multi-threaded application designed to simulate a swim medley relay race. This simulation involves several synchronized threads representing swim teams, each with multiple swimmers.

Simulation Structure

The simulation rules enforce that all teams start their race simultaneously and finish at different times based on their performance. This is managed using the CountDownLatch mechanism:

- CountDownLatch: it is used to synchronize the start of the race.
- The latch.countDown(call in the ActionListener for the "Start" button releases the latch, allowing all swimmer threads to begin their race).

Protection Against Data Races

Data races are mitigated using several synchronization mechanisms, ensuring that shared resources are accessed in a thread-safe manner.

Semaphores

Used to control access to critical sections of code, such as starting blocks, ensuring that only one swimmer can use a block at a time.

- Semaphore semaphore = new Semaphore(1, true);
- In the Swimme class, the semaphore.acquire() and semaphore.release() methods are used to control access to the grid block.

Locks

ReentrantLocks: Used to protect access to shared resources such as grid blocks and people locations. This ensures that only one thread can modify or read a resource at a time.

- private final Lock lock;

- In the PeopleLocation class, a ReentrantLock is used to control access to the occupied status, preventing multiple threads from modifying it simultaneously.

Cyclic Barriers

CyclicBarrier: Ensures that all threads reach a certain point before any are allowed to proceed. This is used to synchronize the start of the race for all swimmers.

- CyclicBarrier barrier = new CyclicBarrier(totalswim, new Runnable()) { ... }

- The CyclicBarrier is initialized with the number of teams with multiply by total swimmer of each team and a runnable that will execute when all teams are ready.

Extensions

Helper Class

Added to manage the completion state of threads in a thread-safe manner.

This class provides synchronized access to a shared boolean flag, allowing the main thread to signal when the race should end and ensuring that swimmer threads respect this signal.

GridBlock Enhancements

Enhanced with additional synchronization to handle occupancy status in a thread-safe manner.

The isOccupied field and its associated methods are protected by a ReentrantLock, preventing

data races when checking or updating blocks.

PeopleLocation Enhancements

Enhanced with thread-safe methods to manage occupancy and location status. This ensures that multiple threads interacting with the same PeopleLocation object do not cause race conditions.

Conclusion

The Medley Simulation effectively uses synchronization mechanisms to enforce simulation rules and protect against data races. The use of CountDownLatch, Semaphore, ReentrantLock, and CyclicBarrier ensures that the simulation runs smoothly and accurately. The extensions made, such as the Helper class and enhancements to GridBlock and PeopleLocation, provide additional robustness and ensure that shared resources are managed correctly.