Blessing Hlongwane Report

The simulation of the swim medley enforces strict rules regarding swimmer order, race transitions, and synchronized starting conditions. Each swim team consists of four swimmers, and they must swim in a specific sequence according to the medley relay order (Backstroke, Breaststroke, Butterfly, and Freestyle). The team-based synchronization is designed to ensure that the next swimmer can only start their section after the previous swimmer has completed theirs. Additionally, no swimmer can dive into the pool until all team members have reached their starting blocks, enforcing both fairness and proper race conditions.

How do we prevent data races?

To prevent data races—where multiple threads access and modify shared resources simultaneously without proper synchronization—the following shared resources are protected:

- **Shared starting blocks**: The location of each swimmer and the shared stadium grid are accessed by multiple threads. To prevent race conditions, synchronization mechanisms are implemented to ensure safe access.
- **Finish counter**: The finish line is a shared resource and ensuring that the final swimmer correctly signals the completion of the race is crucial to avoid miscounts.

Synchronization Mechanisms

Several synchronization mechanisms have been added to the simulation to ensure that the swim race progresses correctly and safely:

a. CountDownLatch

- **Purpose**: CountDownLatch is used to synchronize swimmers at specific race stages, ensuring that the swimmers perform their actions in a coordinated sequence.
- **Where Applied**:
    - A CountDownLatch (stop) was added to ensure that all swimmers reach the +- starting blocks before the race begins. This latch prevents any swimmer from diving in until all four swimmers are ready.
    - Additional CountDownLatch instances (latch, firstlatch, secondlatch) are used to control the start of each swim stroke. For example, the backstroke swimmer starts

first and must finish their part before signaling the next swimmer (Breaststroke) to begin, and so on.

- **Why**: This mechanism ensures proper coordination between swimmers, enforcing the race's sequential structure and preventing swimmers from starting before their teammates complete their part.

b. Synchronized Block

- **Purpose**: The synchronized block prevents multiple threads (swimmers) from diving and racing simultaneously. This is essential in medley relays, where only one swimmer can race at a time.
- **Where Applied**: The dive() and swimRace() methods are wrapped in a synchronized block to ensure that no two swimmers from the same team attempt to swim at the same time.
- **Why**: This protects critical sections of the code where swimmers access the shared StadiumGrid, avoiding race conditions during the swimming phase.

GridBlock Class follows a Java Monitor Pattern

The GridBlock class follows the Java Monitor Pattern, which is a design pattern that uses Java's lock mechanism to ensure that only one thread at a time can execute certain methods of a class. This pattern is particularly useful when dealing with shared data structures, like the stadium grid in this simulation. The GridBlock class acts as a monitor, ensuring that threads (swimmers) can safely update their position without interfering with each other.

Creativity

As part of my creativity, I implemented an image icon and a text, “Winning Team”, flashing at the bottom of the screen, when there is a winner.

Conclusion

The simulation successfully enforces rules around swimmer sequencing, correct starting conditions, and race transitions, all while protecting shared resources and preventing data races. Through the use of CountDownLatch for synchronization and the synchronized keyword to ensure mutual exclusion during critical operations, the simulation adheres to its intended behavior. The GridBlock class employs the Java Monitor Pattern to handle swimmer positioning safely, further contributing to the robustness of the system.