***Assignment 1***

* Bắt buộc phải chạy ra 0
* Thêm chữ Synronized vào hàm increase và decrease trong Data

***Explain the result :***

**Cách 1 : Dùng Synchronized** : đảm bảo rằng mỗi luồng chỉ có thể truy cập và thay đổi biến Data một cách an toàn. Một luồng đang thực hiện tăng hoặc giảm, luồng còn lại phải đợi cho đến khi luồng kia hoàn thành.

**Cách làm :**

* Tạo một lớp : Data
* Viết synchronized method cho các biến tăng và giảm
* Ở hàm main : Tạo và chạy hai luồng song song.

Làm vậy để giúp đảm bảo tính toàn vẹn dữ liệu vì chỉ có một luồng được truy cập biến tại một thời điểm , Tránh race condition đảm bảo rằng giá trị cuối cùng của biến Data cũng như = 0.

**Cách 2 : Dùng ReentrantLock :** Linh hoạt hơn Synchronized

* Cho phép kiểm soát lock/unclock

**Cách làm :**

* Trong class : Data, tạo đối tượng Lock , ReentrantLock để kiểm soát biến
* Trong mỗi phương thức tăng/ giảm, gọi Lock() để đảm bảo chỉ 1 luồng thực hiện
* Sau khi xong thì gọi unclock()

Làm vậy để ReentranLock có thể kiểm soát tốt hơn, kiểm tra xem một luồng có đang giữ khoá hay không

**Cách 3 : Dùng Semaphore :** Kiểm soát số lượng luồng được phép truy cập.

Cách làm :

* Trong class : Data tạo đối tượng : Semaphore chỉ cho phép 1 luồng truy cập tại một thời điểm
* Trong mỗi phương thức tăng giảm, truy cập biến, gọi acquire() để lấy quyền truy cập
* Sau khi xongm gọi release để cho luồng khác tiếp tục

Làm vậy để khi có nhiều luồng hơn cần truy cập tài nguyên bởi Semaphore

Linh hoạt hơn Synchronized and ReentrantLock.

**Vậy sau 3 cách : ta dung cách nào thì giá trị vẫn = 0 vì**

Thread 1 tăng 1.000.000 lần (+1.000.000).

Thread 2 giảm 1.000.000 lần (-1.000.000).

Tổng cộng: 0.

***Trả lời các câu hỏi***

**1. Identify the correctness constraints of the problem**

**-**  **Atomicity of operations**: Each operation (increment or decrement) must be performed atomically.

**- Prevention of race conditions**: Race conditions must be prevented.

- **Correct final value**: After completion, the value of the variable should be 0.

**2. Specify the conditions that each method must wait for**

- **Synchronized Methods**: Condition: A thread must wait if the synchronized method is currently being executed by another thread.

**- ReentrantLock**: Condition: A thread must wait if the lock is currently held by another thread.

**- Semaphore**: Condition: A thread must wait if the semaphore has no available permits (i.e., the semaphore's permit count is 0).

**3. Write down the shared state that you will use to check these conditions**

**- Shared integer variable**: This is the variable that the threads will modify. It starts with an initial value of 0.

**- Locking/synchronization mechanisms**: These depend on the synchronization methods being used

- **Thread states**: To verify that the threads have been properly synchronized.

***Assignment 2***

**Take note** : Dùng được nhiều thread, Thread safe : An toàn , Thêm xóa dung cho nhiều Thread.

**Cách làm :**

Tạo một hang đợi an toàn với luồng ( thread-safe queue ) : có AddLast với removeFirst

Sử dụng ReentrantLock để đảm bảo rằng chỉ một luồng có thể truy cập

***Trả lời các câu hỏi***

1. **Identify the correctness constraints of the problem**

* **Synchronization of operations**: Operations such as adding an element (addLast) and removing an element (removeFirst).
* **No data loss**: Ensuring that no data is lost during the operations.
* **Ensuring correctness of results**: Verifying that the results are accurate and consistent.

1. **Specify the conditions that each method must wait for**

* **addLast method**: Condition: The addLast method must wait if the queue is full or if a lock is currently held by another thread.
* **removeFirst method**: Condition: The removeFirst method must wait if the queue is empty or if a lock is currently held by another thread.

1. **Write down the shared state that you will use to check these conditions**

* **Queue**: Operations such as addLast and removeFirst verify the correctness of the operations.
* **Locks**: Using synchronization mechanisms like ReentrantLock, synchronized.
* **Semaphore or Condition Variables**: To manage thread synchronization and ensure correct access to shared resources.
* **Thread states**: Ensuring that the threads operate correctly and do not experience deadlocks or conflicts, such as a thread being blocked when the queue is empty or full.

***Assignment 3***

Hàng chờ đầy thì mới được lấy ra

* Khi await trên một biến condition . Thì phải có khóa.
* Signal phải khác signal All .

4 điều cần nhớ :

1. Lock() 🡪 await , signal
2. Có cả 2 hàm signal và signalAll : cho all các thread trên signal chạy.
3. Khi await thì java thì trả cái khóa lại, cho các cái khác chạy
4. Khi hết Await và block. Nếu chưa có khóa thì tiếp tục chờ.

Nếu có 100 cái thread chạy. Nhưng chỉ chạy được 1 cái vì f có khóa mới chạy

***Trả lời các câu hỏi***

1. **Identify the correctness constraints of the problem**

* **Synchronization of add and remove operations**: When the add and remove operations are performed simultaneously, there should be no contention or unsynchronized changes to the state of the queue.
* **Waiting when full or empty**: The thread performing the add operation must wait until there is available space in the queue. Similarly, when the queue is empty, the thread performing the remove operation must wait until new elements are added.
* **No data loss**: Ensuring that no data is lost during the operations.
* **Ensuring correctness of results**: Verifying that the results are accurate and consistent.

1. **Specify the conditions that each method must wait for**

* **add method**: The thread performing the add operation must wait until there is available space in the queue to add an element. Condition: The add method must wait when the queue is full.
* **remove method**: When the queue is empty, the thread performing the remove operation must wait until an element is added. Condition: The remove method must wait when the queue is empty.

1. **Write down the shared state that you will use to check these conditions**

* **Queue: Operations such as add and remove will directly operate on this queue.**
* **Queue Size**: Monitoring the number of elements in the queue to determine when the queue is full and when it is empty.
* **Synchronization mechanisms (Locks, Semaphores, or Condition Variables)**: The add and remove operations are performed safely in a multi-threaded environment, using synchronization mechanisms such as ReentrantLock, synchronized, or Condition.
* **Condition Variables**: Can be used to make the thread performing the add operation wait when the queue is full and the thread performing the remove operation wait when the queue is empty.
* **Thread states**: Ensuring that the add and remove operations are performed correctly and that threads are not blocked or conflicting incorrectly.

***Assignment 4***

1. **Correctness constraints:**
   * All threads must wait at the barrier until the required number of threads (parties) arrives.
   * Only when all threads have reached the barrier can they proceed together.
2. **Conditions that each method must wait for:**
   * The await() method must wait until the total number of threads that have called it equals the specified parties value.
3. **Shared state to check these conditions:**
   * An integer counter to track the number of threads that have reached the barrier.
   * A Lock to ensure mutual exclusion.
   * A Condition to make threads wait and signal them when the barrier is tripped.

***Assignment 5***

 **Correctness constraints:**

* A maximum of 3 cars can be on the bridge at any time.
* All cars on the bridge must be traveling in the **same direction**.
* Cars must wait if the bridge is full or if cars are going in the opposite direction.

 **Conditions that each method must wait for:**

* **arriveBridge(int direction)** must wait if:
  + There are already 3 cars on the bridge, or
  + The cars currently on the bridge are going in the opposite direction.
* **exitBridge()** doesn't wait but may signal waiting cars after a car leaves.

 **Shared state to check these conditions:**

* An integer to count the number of cars currently on the bridge.
* An integer or boolean to track the **current direction** of traffic on the bridge.
* A Lock to ensure thread-safe access.
* A Condition to make cars wait when conditions are not met and signal them when they can proceed.