There are three threads that are created HaMo, NaKi, and AkTo that have independent stories that then have a team battle which executes parallel. When they execute simultaneously they share a shared resource sharedBoss that they all attack. If one of the players dies in the separate stories then they do not get to join the team battle at the end of the game.

A race condition that I came across was multiple threads are simultaneously reading and modifying the sharedBoss.hp. This would cause Thread one to read hp as 10 and then Thread 2 also reading hp as 10. That ends up with Thread 1 dealing 20 damage causing hp to do -10 but then Thread 2 also deals 20 damage which should not happen after the health is below 0. The solution was to synchronize the block with a bossLock. I added a synchronized Adventure bossLock above the if statement of seeing if the sharedBoss health is below 0. This made sure that one thread would modify the boss's health at a time. Another race condition was threads printing simultaneously which causes a jumbled output. I yet to fix this but I will be making improvements

One of the thread coordination mechanisms used was the intrinsic locks. I used the lock in order to exclude the shared boss resource. All of the threads use the lock objects which make the boss attacks to prevent data loss. Another mechanism used was Thread,join, I used Thread,join in order for the threads to wait for the main threads to finish. This made sure that the boss battle completes before the program exits. One of the last mechanisms used was the Thread.sleep. I used this in order to have the characters simulate real fighting timing. This also allowed for the reduction of thread contention.