**Summary: Hotel Simulation Project**

The Hotel Simulation project aimed to simulate the interactions between guests, front desk employees, and bellhops in a hotel setting. The main goal was to ensure coordination between these objects using semaphores and locks, without using built-in synchronization mechanisms.

The journey to making a fully functional simulation was full of challenges. One of the initial challenges was getting the output to work seamlessly. The simulation would often hang after the last guest retired, which indicated a potential deadlock or a synchronization issue. Debugging it required a solid understanding of concurrency and the Java threading model.

At first, I used the synchronized keyword in Java to manage mutual exclusion. However, after looking at the project requirements, I realized that the use of the synchronized keyword was not allowed. This led to me figuring out if other synchronization mechanisms existed. My next approach was to use the ReentrantLock class. It seemed like a good solution, offering more flexibility than the synchronized keyword. However, I soon realized that ReentrantLock provides mutual exclusion inherently, which again violated the project's constraints.

To address this, I designed a custom lock class, named SimpleLock, to imitate the behavior of ReentrantLock without using any built-in mutual exclusion mechanisms. This custom implementation was key in making sure that the project stuck to the guidelines while still getting the desired synchronization.

The process of transitioning from the synchronized keyword to ReentrantLock, and finally to SimpleLock, was enlightening. It showed the importance of understanding the tools and libraries one uses, especially in the realm of concurrency where nuances can significantly impact the system's behavior.