**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.

Initially, I used the synchronized keyword in Java to manage mutual exclusion. However, after revisiting the project requirements, I realized that the use of the synchronized keyword was prohibited. This led to the exploration of alternative synchronization mechanisms. My next approach was to use the ReentrantLock class. It seemed like a viable solution, offering more flexibility than the synchronized keyword. However, I soon recognized that ReentrantLock inherently provides mutual exclusion, 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 leveraging any built-in mutual exclusion mechanisms. This custom implementation was pivotal in ensuring that the project adhered to the stipulated guidelines while still achieving the desired synchronization.

The process of transitioning from the synchronized keyword to ReentrantLock, and finally to SimpleLock, was enlightening. It underscored 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.

In conclusion, this project was a valuable learning experience. It not only deepened my understanding of Java's concurrency mechanisms but also highlighted the importance of thoroughly reading and adhering to requirements. The iterative process of refining the simulation, coupled with the challenges faced, provided practical insights into designing and debugging multi-threaded applications.