**Report-PROJECT 2**

**CECS 326**

**Avlokita Sharma**

**Video link: https://youtu.be/\_11BjE\_\_uBE**

**The Dining Philosophers problem**

The Dining Philosophers problem is an example of resource allocation amongst concurrent processes and is a well-known synchronization challenge (in this case, threads). In my approach, I control access to the shared resources (forks) and synchronize the philosophers' activities using Java condition variables and the Reentrant Lock class.

Philosopher, DiningPhilosophers, DiningServerImpl, and DiningServer are the four classes that make up the software. A contract for techniques that control the forks and philosopher states is provided by the DiningServer interface. The DiningServerImpl class implements this interface by leveraging condition variables and a ReentrantLock to manage each philosopher's state and restrict access to the forks.

A philosopher is a distinct thread that is represented by the Philosopher class. Between each philosophical thread's thoughts and meals, there is a break. The DiningServerImpl object, which controls access to the forks, is invoked by a philosopher when they wish to dine by calling the takeForks() function. After finishing their meal, the philosopher calls the returnForks() function, which releases the forks for use by other philosophers.

The main() function of the DiningPhilosophers class generates and launches the philosopher threads as well as initializes the DiningServerImpl instance. The philosophers are given 30 seconds to eat and contemplate while the program is running. The main thread terminates the philosopher threads by terminating the executor service after 30 seconds.

To control synchronization and access to shared resources in this implementation, we employ condition variables and a ReentrantLock. By providing mutual exclusion, the ReentrantLock makes sure that only one philosopher at a time can access the shared resources. The philosopher threads are synchronized using condition variables, which enables them to wait until particular circumstances (such as when both forks are available) are met before continuing.

By preventing deadlocks and guaranteeing that philosophers can eat and ponder without breaking the rules of mutual exclusion or starving themselves, the design makes sure that the Eating Philosophers problem is effectively handled.