Kiet Nguyen, #027353519

Thai Tran, #026819986

Report #3

The program is designed to demonstrate the use of synchronization and locking mechanisms in a multithreaded environment to control access to shared resources. It consists of three classes: RoadController, West_village, and East_village.

The RoadController class has a single Lock object called road. This lock is used to control access to the road. The class has two methods, lock() and unlock(), which respectively acquire and release the lock. The main() method creates four West_village and East_village objects each and starts their execution by calling the run() method for each object. The wait() and notify() methods are not used in this program because the Lock interface provides more functionality and better control over locking and unlocking than the basic synchronization provided by the synchronized keyword.

The West_village and East_village classes implement the Runnable interface, allowing instances of these classes to be executed in separate threads. Both classes have two instance variables: villagerNum, which represents the number of villagers, and roadController, which represents the RoadController object used to control access to the road.

In the run() method of both the West_village and East_village classes, the road controller is synchronized, and the lock is acquired using the lock() method. This ensures that only one villager can be on the road at any given time. The villager then performs a random action, which is represented by a randomly generated number between 1 and 3. If the number is 1, the villager is eating a burrito; if it's 2, the villager is taking a quick nap; and if it's 3, the villager is enjoying the scenery. After performing the action, the actionTime() method is called, which randomly determines a sleep time between 1 and 3

seconds. The villager then sleeps for that amount of time using the Thread.sleep() method. After waking up, the lock is released using the unlock() method, and the villager has completed the exchange.

Overall, the program demonstrates the use of synchronization and locking mechanisms in a multithreaded environment to control access to shared resources. Using the Lock interface in the RoadController class provides a more advanced and powerful way of controlling locks than the basic synchronization provided by the synchronized keyword. The program shows how using locks can help prevent race conditions and ensure the proper synchronization of threads in a multithreaded environment.

Video recording link: https://youtu.be/S5KDCa2qWUg