- Implementing barrier:
    - We know that the philosophers come to the dining table in a random amount of time, 1 to 10 seconds. Therefore, first I created a random object to call nextInt() function which takes the bound value 11 as an argument. I call the nextInt() function inside a loop because the arrival times are in range [1,10], therefore nextInt() function generating the value 0 is not acceptable. Whenever it generates a nonzero arrival time for the philosopher, I break out of the loop and make the philosopher sleep that amount of time. The sleep function that is called on the thread (philosopher object) takes milliseconds as an argument, therefore I passed the random arrival time x 1000 to obtain the milliseconds unit.
    - Whenever I do anything about threads, sleeping, acquiring, releasing; I do that inside a try block and handle the exception.
    - After the sleep, I called the PutPlate_GUI() function on the table object so that I can put the table of the arriving philosopher on the table.
    - Then, inside a loop, I release the barriers of other philosophers. Then that philosopher tries to acquire its barrier for 5-1=4 times, which is the number of philosophers other than herself. If the number of arriving philosophers at that moment is smaller than 4 then there are still waited for philosophers, making N-1 acquires will block until remaining philosophers come and make a release on the barrier of that philosopher. When all barriers are released, I call StartDining_GUI() function.
- Dining stage:
    - In a continuous loop, we let the philosopher think, get hungry, eat, think again, and repeat this process. So initially, the state of the philosopher is thinking.
    - I got a random amount of thinking time, 0-to-10. After that amount of time, I called the takeForks() function implemented by me. Inside that function:
        - Acquire the mutex to indicate the presence in the critical region
        - Call the Hungry_GUI() function
        - Update the state of the philosopher as HUNGRY
        - Test if the neighboring philosophers are eating or not. If they are eating, the philosopher waits in a hungry state, else update her state as EATING
        - Call the ForkTake_GUI() function
        - Release the semaphore, then I also release the mutex for the critical region and acquire the semaphore
    - I call Eating_GUI() and StopEating_GUI()
    - I call putForks() function implemented by me. Inside that function:
        - Acquire the mutex
        - Call the ForkPut_GUI() function
        - Update the state as THINKING
        - Test if the neighboring philosophers are hungry and their other neighboring philosopher is not eating, if so release their semaphores to signal that they can eat now
        - Release the mutex in order to indicate the exit from the critical region
    - I repeat this process in an infinite loop, until the user exit from the GUI.