1-Solution pseudocode

The Dining Philosopher Problem – The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

Semaphore Solution to Dining Philosopher

Each philosopher is represented by the following pseudocode:

```
process P[i]
  while true do
      {    THINK;
            PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
            EAT;
            PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
      }
```

Examples of Deadlock for Dining-Philosophers Problem

```
void Philosopher
 {
 while(1)
  {
   Wait( take_chopstickC[i] );
   Wait( take_chopstickC[(i+1) % 5] ) ;
   . .
   . EATING THE NOODLE
   .
         Signal( put_chopstickC[i] );
         Signal( put_chopstickC[ (i+1) % 5] ) ;
         .
         . THINKING
        }
      }
```

In the above code, first wait operation is performed on take_chopstickC[i] and take_chopstickC [(i+1) % 5]. This shows philosopher i have picked up the chopsticks from its left and right. The eating function is performed after that.

On completion of eating by philosopher i the, signal operation is performed on take_chopstickC[i] and take_chopstickC [(i+1) % 5]. This shows that the philosopher i have eaten and put down both the left and right chopsticks. Finally, the philosopher starts thinking again.

From the above solution of the dining philosopher problem, we have proved that no two neighboring philosophers can eat at the same point in time. The drawback of the above solution is that this solution can lead to a deadlock condition. This situation happens if all the philosophers pick their left chopstick at the same time, which leads to the condition of deadlock and none of the philosophers can eat.

<u>How did solve deadlock for Dining-Philosophers Problem</u>

To avoid deadlock, some of the solutions are as follows -

- Maximum number of philosophers on the table should not be more than four, in this case, chopstick C4 will be available for philosopher P3, so P3 will start eating and after the finish of his eating procedure, he will put down his both the chopstick C3 and C4, i.e. semaphore C3 and C4 will now be incremented to 1. Now philosopher P2 which was holding chopstick C2 will also have chopstick C3 available, hence similarly, he will put down his chopstick after eating and enable other philosophers to eat.
- A philosopher at an even position should pick the right chopstick and then the left chopstick while a philosopher at an odd position should pick the left chopstick and then the right chopstick.

- Only in case if both the chopsticks (left and right) are available at the same time, only then a philosopher should be allowed to pick their chopsticks
- All the four starting philosophers (P0, P1, P2, and P3) should pick the left chopstick and then the right chopstick, whereas the last philosopher P4 should pick the right chopstick and then the left chopstick. This will force P4 to hold his right chopstick first since the right chopstick of P4 is C0, which is already held by philosopher P0 and its value is set to 0, i.e C0 is already 0, because of which P4 will get trapped into an infinite loop and chopstick C4 remains vacant. Hence philosopher P3 has both left C3 and right C4 chopstick available, therefore it will start eating and will put down its both chopsticks once finishes and let others eat which removes the problem of deadlock.

Examples of starvation for Dining-Philosophers Problem

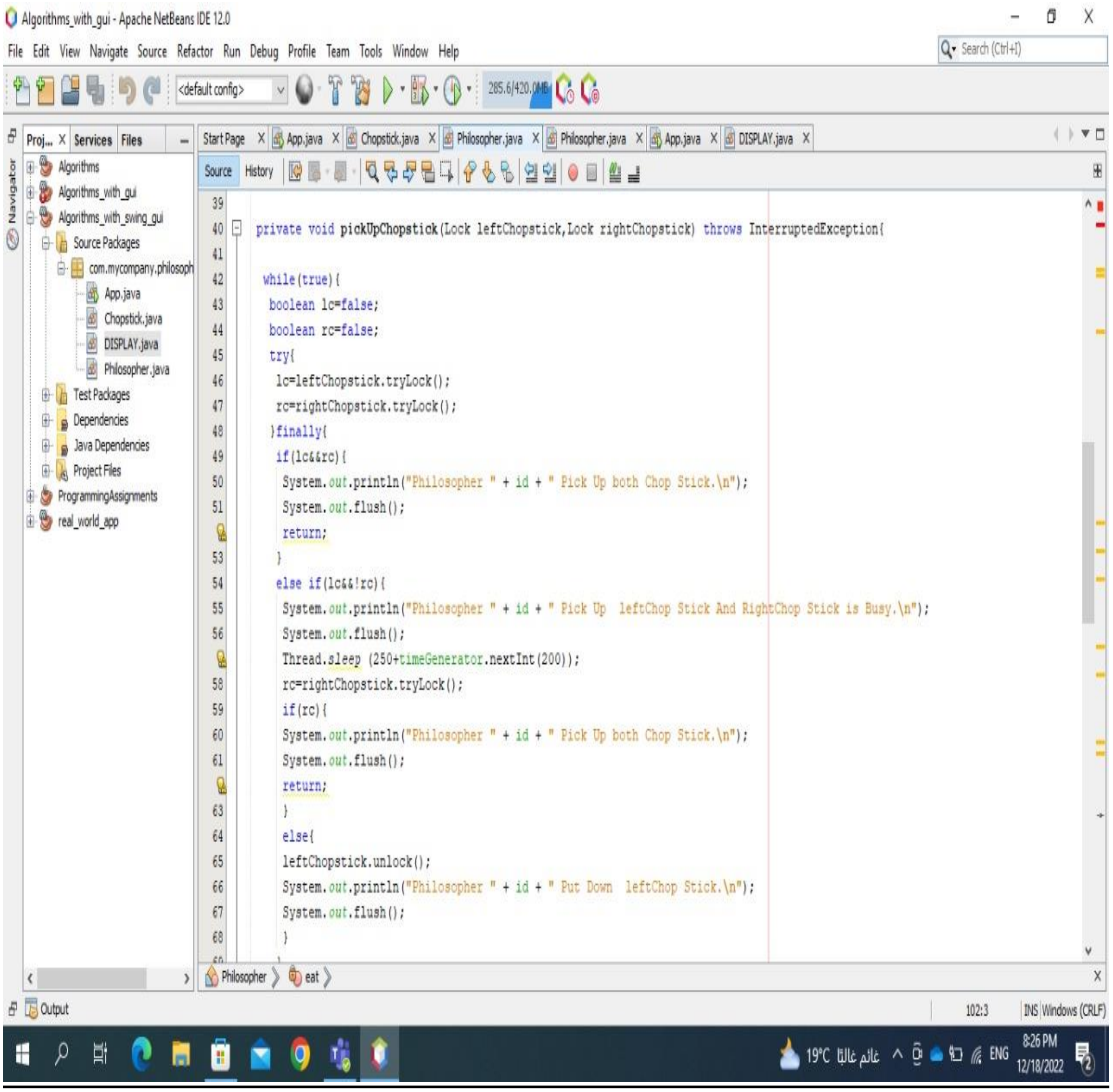
```
while(TRUE)
{
        wait(stick[i]);
        /*
                mod is used because if i=5, next
                chopstick is 1 (dining table is circular)
        */
        wait(stick[(i+1) % 5]);

        /* eat */
        signal(stick[i]);

        signal(stick[(i+1) % 5]);
        /* think */
}
```

However, they each will think for an undetermined amount of time and may end up holding a left fork thinking, staring at the right side of the plate, unable to eat because there is no right fork, until they starve.

<u>How did solve starvation for Dining-Philosophers Problem</u>


```
private void pickUpChopstick(Lock leftChopstick,Lock rightChopstick) throws InterruptedException{

 while(true){
  boolean lc=false;
  boolean rc=false;
  try{
   lc=leftChopstick.tryLock();
   rc=rightChopstick.tryLock();
  }finally{
   if(lc&&rc){
    System.out.println("Philosopher " + id + " Pick Up both Chop Stick.\n");
    System.out.flush();
    return;
   }
   else if(lc&&!rc){
    System.out.println("Philosopher " + id + " Pick Up  leftChop Stick And RightChop Stick is Busy.\n");
    System.out.flush();
    Thread.sleep (250+timeGenerator.nextInt(200));
    rc=rightChopstick.tryLock();
    if(rc){
    System.out.println("Philosopher " + id + " Pick Up both Chop Stick.\n");
    System.out.flush();
    return;
    }
    else{
    leftChopstick.unlock();
    System.out.println("Philosopher " + id + " Put Down  leftChop Stick.\n");
    System.out.flush();
    }
```

Dijkstra's solution uses one mutex, one semaphore per philosopher and one state variable per philosopher. This solution is more complex than the resource hierarchy solution.

<u>**Explanation for real world application and how did apply the problem**</u>

The dining philosopher's problem is a very well known resource sharing problem in the world of computers.

There are 5 philosophers sitting around a round table eating spaghetti and each of them has one chopstick between them. All 5 of them sit around the table and pick up the chopstick placed towards their right. But, here’s the problem. To eat the spaghetti they need both the chopsticks and since everyone picked up the chopstick to their right, nobody gets the left chopstick and hence, nobody can eat.

the dining philosophers problem is an excellent example to explain the concept of deadlock while resource sharing in OS.

Consider the philosophers to be processes and the chopsticks to be a shared resource. Every process needs two resources out of which one it has already acquired and the other is acquired by some other process. Till the other process does not free the resource, this process cannot proceed. Similarly, the other process is dependent on another process for its resource. Since every process is dependent on each other, it forms a circular-wait and the system goes into a deadlock condition.

Here, when each philosopher picks up the chopstick to their right, they also end up picking the left chopstick of the person sitting next to them which leaves every philosopher with just one chopstick and nobody can start eating.

We can apply the problem in real life, for example Cooperating processes that need to share limited resources and Travel reservation: hotel, airline, car rental databases