MIDDLE EAST TECHNICAL UNIVERSITY
Department of Computer Engineering

Homework#2 **5710546/01**

- Submit thru *online.metu.edu.tr* by the end of April 25, 2004.
- Write a Java program to simulate the Dining Philosophers.
- You are expected to follow the object-oriented approach using encapsulation, polymorphism, abstraction, associations etc. as much as possible.

Dining Philosophers

A common problem in concurrent programming is coordinating access to a shared resource by enforcing a synchronization condition between multiple concurrent processes. A classic example is the Dining Philosophers problem, which consists of some number of philosophers, say *#dp*, seated at a round table thinking and eating concurrently. For this version of the Dining Philosophers problem, they are eating from a shared bowl in the middle of the table using a pair of chopsticks. Each Philosopher has a left chopstick shared by his left-neighbor and a right chopstick shared by his right-neighbor, so there are *#dp* chopsticks total. In order to eat, a Philosopher must obtain both left and right chopsticks exclusively. If a chopstick is not available a Philosopher must wait for it to be released by his neighbor before being allowed to eat. A Philosopher is allowed to eat for no more than 1 second before relinquishing both chopsticks to think again, so as not to starve the other Philosophers. After a random number of milliseconds (again no more than 1 sec.) of thinking, a Philosopher may try to eat again but has to reacquire both chopsticks.

A philosopher acquires a chopstick when it is available, or waits. When finished eating, a philosopher releases a chopstick, which notifies his neighbor who might be waiting on the chopstick. Your main program must create *#dp* Chopstick objects and *#dp* Philosopher thread objects (where *#dp* is the first command-line argument to your *main* method), and assign to each Philosopher a pair of Chopsticks on construction. Then, create each Philosopher thread and start it running in your main procedure. When a Philosopher threads runs, it starts with thinking. The program should terminate when each Philosopher gets to eat *#ne* times (where *#ne* is the second command-line argument to your *main* method).

The output of your program should be similar to:

```
Philosopher-n is thinking.
…
Philosopher-n is eating.
…
Philosopher-n is thinking.
…
Philosopher-n is eating.
```

where n is the id of the Philosopher, $0 \leq n < \#dp$.

Make sure that no starvation of any Philosopher ever happens, and no program deadlock occurs.