1. Do Ben-Ari Exercise 4.2 to prove Lemma 4.10 and use it to prove that Dekker's algorithm satisfies mutual exclusion. You may hand it in on paper or electronically. If you hand it in electronically it must be a PDF document named

   `a23written.pdf`

   per the instructions for your course.

2. Implement the dining philosophers solution of Ben-Ari, Algorithm 6.11, in Java with semaphores. The source file to modify is `Philosopher.java` in the `cosc450JavaDistr` software distribution.

   Name your class Philosopher with the following three attributes:

   - A private integer `processID` with a constructor to initialize its value
   - A private static fair semaphore `room` allocated with `new` and initialized to 4
   - A private static array of semaphores `fork` allocated with `new`

   In the main program, first allocate each fork semaphore in a loop, initializing it with the Java Semaphore constructor. Next, allocate an array of five philosophers (with `new`). Then, initialize the array of philosophers in a loop (with `new`) giving each philosopher its proper processor ID. Start each philosopher process in a loop and `join()` them in a loop.

   In your implementation of `run()`, loop through the activity of thinking and eating three times.

   - For p1, output "Philosopher *n* is thinking." where *n* is the philosopher id (0..4) on a new line. Then, sleep with a random 20 ms delay.
   - Immediately after p2, output "Philosopher *n* entered the dining room." on a new line.
   - Put a random 20 ms delay between p3 and p4 to try to force a deadlock.
   - For p5, output "Philosopher *n* is eating." on a new line. Then, sleep with a random 20 ms delay.
   - Immediately after p8, output "Philosopher *n* left the dining room." on a new line.
   - At the conclusion of the meal, output "The Philosophers are finished eating." on a new line.

   Run your program a few times to verify that several philosophers are eating concurrently. Show the following in the documentation section at the bottom of the source file.

   (1) Copy the output from one of the runs and paste it into the documentation section.
   (2) From the output in (1), what was the maximum number of philosophers in the room and who were they when the maximum occurred?
   (3) Change the initial value of the room semaphore from 4 to 5, allowing all five philosophers into the room at once. Run your program a few times to verify that it deadlocks. Copy the output from one of the deadlock runs and paste it into the documentation section.
   (4) Experiment with the length of the random delay between p3 and p4. What is the ms delay that will cause a deadlock to occur in roughly half of the runs?
   (5) Comment out the random delay between p3 and p4. Can you ever get a run to deadlock?
   (6) Explain your answer to (5).

   For this problem, hand in

`Philosopher.java`

CAUTION: Do not hand in your erroneous listing with the room semaphore initialized incorrectly. Hand in your correct solution of the dining philosophers problem with the original 20 ms delay between p3 and p4.