1. Study Ben-Ari, Sections 7.1–7.5, 7.8, 7.9, 7.11.

2. Study the paper by Buhr, Fortier, and Coffin, *Monitor Classification*, Sections 1, 2, 3.
   http://www.cslab.pepperdine.edu/warford/cosc450/Monitor-Buhr.pdf

3. Study class handout: Notes on monitors.

4. Study Sestoft, Chapter 16.

5. Implement the dining philosophers problem in Java with a monitor. Algorithm 7.5 in the Ben-Ari text is a solution of the dining philosophers problem with a monitor in which a philosopher picks up both forks at the same time. However, your solution will avoid deadlock by allowing only up to four philosophers to enter the dining room (trattoria) at a time.

Your solution will consist of three classes, each in a separate file as follows:

- **Class Trattoria in file Trattoria.java.**
  Trattoria defines a Java monitor with two private attributes – an array of five booleans named forkAcquired with all values initialized to false, and a counter to keep track of the number of philosophers in the trattoria. Define the following four monitor methods with the synchronized keyword:
  - `enterDiningRoom(int philID)`
    This method should allow only four philosophers to enter the dining room. A fifth philosopher must wait until some other philosopher leaves the dining room. The last statement should output “Philosopher \(p\) entered dining room.” where \(p\) is the value of the philosopher’s id.
  - `exitDiningRoom(int philID)`
    The last statement of this method should output “Philosopher \(p\) left dining room.” where \(p\) is the value of the philosopher’s id.
  - `acquireFork(int philID, int forkID)`
    The last statement of this method should output “Philosopher \(p\) picked up fork \(f\).” where \(p\) is the value of the philosopher’s id, and \(f\) is the value of the fork id.
  - `releaseFork(int philID, int forkID)`
    The last statement of this method should output “Philosopher \(p\) put down fork \(f\).” where \(p\) is the value of the philosopher’s id, and \(f\) is the value of the fork id.

- **Class Philosopher in file Philosopher.java.**
  Philosopher has two attributes, a process id and a reference to Trattoria (Java reference variable). It has no main program, but does define the constructor for Philosopher. The constructor has two parameters, one for the id of the philosopher and one for a reference to the monitor. This class also defines the run method for a Philosopher. Each philosopher should output “Philosopher \(p\) started.” He should then alternately eat and think three times as follows:
  - Enter the dining room with `enterDiningRoom()`.
  - Acquire the right fork with `acquireFork()`.
  - Output “Philosopher \(p\) is going to sleep.”
  - Sleep for 100 msec.
  - Acquire the left fork with `acquireFork()`.
– Output “Philosopher $p$ is going to eat.”
– Sleep between 0 and 99 msec at random.
– Release the forks with releaseFork().
– Leave the dining room with exitDiningRoom().
– Output “Philosopher $p$ is going to think.”
– Sleep between 0 and 99 msec at random.

Class Dining in file Dining.java.

This class consists of a single main program that instantiates a monitor trattoria of class Trattoria. It instantiate an array of five threads for the philosophers and then instantiates and starts each individual philosopher in a loop.

For your convenience, here is a NetBeans project that contains an outline of the main program and run() method for you to complete.

http://www.cslab.pepperdine.edu/warford/cosc450/Dining.zip

Run your program several times to insure that it produces different results each time and does not deadlock. Then, temporarily modify Trattoria so that the test to enter the dining room allows at most five philosophers in the room instead of at most four. Run your program to verify that it deadlocks.

Created a new empty file named a24written.txt with NetBeans as in Assignment 22. Copy and paste the terminal output for the deadlocked run into the text document. Add your explanation of the deadlock to your document, describing the scenario that produced the deadlock. Change the Trattoria back to the correct version before submitting it.

Hand in

Dining.java
Philosopher.java
Trattoria.java
a24written.txt

electronically per the instructions for your course. Hand in the three source files, and not a compressed NetBeans project or a .jar file.

CAUTION: Do not hand in your erroneous listing that deadlocks. Hand in your correct solution of the dining philosophers problem.

CAUTION: a24written.txt must not be a .pdf file or a .rtf file or a .docx file. It must be a .txt file created with the NetBeans editor.

6. Implement the dining philosophers problem in C-- with a monitor. Like the Java solution, the C-- solution will work by limiting the number of philosophers in the room to at most four.

Your solution will consist of a single file with three parts as follows:

monitor Trattoria

The monitor controls access to the room and access to the forks. Therefore, it must have the following attributes:

– An integer count of how many philosophers are in the room, which is initialized to 0.
– An boolean array named `forkAcquired` which is initialized to all false. Because C-- does not have type boolean, it can be an array of integers with 0 for false and 1 for true.

– A condition variable named `OKtoEnter`, whose queue of blocked processes contains those philosophers who are waiting to enter the crowded room.

– A condition variable named `OKtoPickUp`, whose queue of blocked processes contains those philosophers who are waiting to pick up a fork.

All initializations are done in the `init` section of the monitor.

As with the Java solution, your monitor should provide the following methods:

– `enterDiningRoom(int philID)`
– `exitDiningRoom(int philID)`
– `acquireFork(int philID, int forkID)`
– `releaseFork(int philID, int forkID)`

which behave as specified in the Java version.

To prevent interleaving of the output statements, include the following three monitor methods in `Trattoria`:

– `putSleep (int philID)`
  Outputs “Philosopher $p$ is sleeping” where $p$ is the value of the philosopher’s id.

– `putEat (int philID)`
  Outputs “Philosopher $p$ is eating” where $p$ is the value of the philosopher’s id.

– `putThink (int philID)`
  Outputs “Philosopher $p$ is thinking” where $p$ is the value of the philosopher’s id.

### void philosopher (int id)
This is the function that is called in the `cobegin` statement of `main`. The philosopher’s ID is a parameter in the `philosopher()` function as opposed to an attribute of an object. Each philosopher should behave the same as in the Java version. Because this is C--, however, we have no need of any sleep delay statements.

### void main()
This contains a single `cobegin` statement with five calls (not in a loop) to the `philosopher()` function.

For your coding convenience, here is the C-- file with an outline of the solution for you to fill in.

[http://www.cslab.pepperdine.edu/warford/cosc450/dining.cm](http://www.cslab.pepperdine.edu/warford/cosc450/dining.cm)

Name your file

`dining.cm`

and hand it in electronically per the instructions for your course.