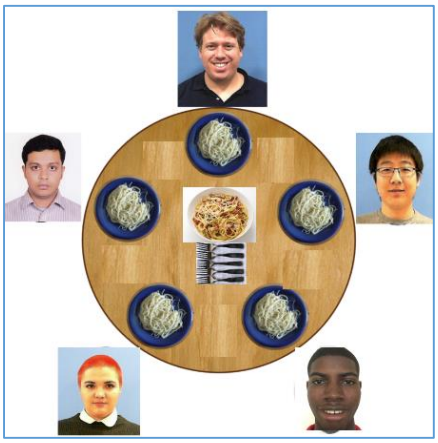
CSE505 – Spring 2018
Assignment 5 (continued)

Problem 3: Dining Programmers Problem

Due Date: **May 10 (11:59 pm)**
You may work in pairs for this assignment.

Problem 3: This is an elaboration of the *Dining Programmers* problem discussed in Lecture 25. As discussed in class, this problem is similar to Dijkstra's *Dining Philosophers* problem but there is an important difference which necessitates a different approach to concurrency control.

Shown below are five programmers around a table. (Resemblance to persons living or deceased is purely intentional – sorry, unintentional!)



These programmers being more practical (and hungry!) than philosophers decided that all five forks shall be placed at the center of the table, as shown in the picture. Each programmer goes through an infinite cycle of three states:

$$\{\text{coding} ; \text{hungry} ; \text{eating}\}^{+}$$

Prior to eating, a programmer must acquire two forks in order to help himself/herself to the spaghetti located at the center of the table. Furthermore, these forks must be acquired one at a time. After eating, the programmer must return both forks back to the center of the table.

By locating the forks at the center of the table it is possible for any two programmers to be eating simultaneously, including adjacent programmers. In this sense the *Dining Programmers* problem is different – and gives more flexibility and concurrency – than the classic *Dining Philosophers* problem.

An uncoordinated approach to acquiring forks will result in a deadlock – each programmer acquires one fork each and is waiting for the second. Hence, the programmers agreed to adopt the following protocol:

1. No programmer may acquire a fork if this is the only remaining fork on the table ***and*** it is the ***first*** fork that the programmer is trying to acquire.
2. However, a programmer may acquire the only remaining fork on the table if it is the ***second*** fork that the programmer is trying to acquire.

Your task is to program a solution to the *Dining Programmers* problem in Java. The file `Concurrency.java` gives the outline of the solution, including the complete definition of classes `Concurrency`, `Programmer`, and `State`. Your task is to complete the definition of methods `pickup` and `drop` in class `Forks` so that no deadlock can arise and maximum concurrency is permitted. You should not modify the classes `Concurrency`, `Programmer`, and `State`. Also the number of forks should be kept at five.

Run your program under JIVE and save the execution trace in a file `DP.csv`. Load this file in the Finite State Machine plugin. Obtain two state diagrams as follows:

1. Obtain a diagram for the field `Forks:1->n`. Save the diagram in the file `forks.svg`.
2. Obtain a diagram for the five fields `States:1->state` … `States:5->state`. Draw the diagram ***after*** performing abstraction with respect to the condition:

   `=E,=E,=E,=E,=E`

   Note: do not put spaces after the commas. Save the resulting state diagram in file `states.svg`.

A correct implementation of the *Dining Programmers* problem will have a maximum of **16**[1] states in the abstracted FSM diagram – it is okay for your run to not get all abstracted states – and no state of the FSM will have more than two components with value `E`.

A screen-cast showing how to install the FSM plugin will be uploaded to Piazza. Prior to this installation, you need to do two things:

1. Install GraphViz on your computer from: http://www.graphviz.com/download
2. Install PlantUML as an Eclipse plugin from: http://files.idi.ntnu.no/publish/plantuml/repository/

WHAT TO SUBMIT:

Make a directory called `A5_UBITId` if working solo or make a directory called `A5_UBITId1_UBITId2` if working as a pair (give UBITId's in alphabetic order).

Put in this directory the files `mystery.py`, `AbsTree.java AbsTree2.java`, `inheritance.png`, `delegation.png`, `Concurrency.java`, `DP.csv`, `forks.svg`, `states.svg`.

Compress the directory, and submit it using the `submit_cse505` command.

Note that `mystery.py`, `AbsTree.java AbsTree2.java`, `inheritance.png`, and `delegation.png` were your solutions to **Problems 1 and 2**.

[1] An abstracted state is one in which every component of the state vector has the value E or ~E. The notation ~E stands for "not eating," i.e., coding or hungry. There are five abstracted states with exactly one E; there are 10 abstracted states with 2 E's; and there is one state with 0 E's. Hence the total number of abstracted states is 16.