Homework 3 of CS 3211, 2010, Total 10 marks [Posted on Tuesday March 9]

Please submit in the IVLE workbin by **Monday 29 March 2010 before 11:59 PM**. **Kindly note that there will be no extensions. If you are not finished by the deadline, please submit whatever partial answer you may have - this is better than not submitting at all. Only submissions in the IVLE Workbin will be graded. Submissions sent by e-mail, unfortunately, cannot be considered.**

Upload one single zip file containing all the programs. Also include a README.txt file in the zip which will say which file contains the answer to which question.

If your tutor is Seth, upload your file to the folder ***HW3-Seth***

If your tutor is Dawei, upload your file to the folder ***HW3-Dawei***

If your tutor is Abhik, upload your file to the folder ***HW3-Abhik***

Question 1 [5 marks]

A savings account is shared by several people. Each person may deposit or withdraw funds from the account subject to the constraint that the account balance must never become negative. Develop a Java implementation for this problem. The savings account should be implemented using a monitor.

Question 2 [2 marks]

The process equations we studied in class allow multiple processes to synchronize on a common action. A set of processes with the action `sync` in their alphabets must all perform this action before any of them can proceed. Implement a monitor called `Barrier` in Java with a sync method that ensures that all of N threads must call `sync` before any of them can proceed.

Question 3 [3 marks]

We discussed the Dining Philosopher's problem in the very first lecture. A Java solution of the problem implementing Fork as a monitor is as follows.

```
class Fork {
  private boolean taken=false;
  private PhilCanvas display;
  private int identity;
  Fork(PhilCanvas disp, int id)
    { display = disp; identity = id;}

  synchronized void put() {
    taken=false;
    display.setFork(identity,taken);
    notify();
  }
```

```
synchronized void get()
     throws java.lang.InterruptedException {
    while (taken) wait();
    taken=true;
    display.setFork(identity,taken);
  }
}

class Philosopher extends Thread {
  ...
  public void run() {
    try {
      while (true) {                        // thinking
        view.setPhil(identity,view.THINKING);
        sleep(controller.sleepTime());    // hungry
        view.setPhil(identity,view.HUNGRY);
        right.get();                        // gotright chopstick
        view.setPhil(identity,view.GOTRIGHT);
        sleep(500);
        left.get();                               // eating
        view.setPhil(identity,view.EATING);
        sleep(controller.eatTime());
        right.put();
        left.put();
      }
    } catch (java.lang.InterruptedException e){}
  }
}

// Code to create philosophers and forks
for (int i =0; i<N; ++i)
  fork[i] = new Fork(display,i);
for (int i =0; i<N; ++i){
  phil[i] = new Philosopher(this,i,fork[(i-1+N)%N],fork[i]);
  phil[i].start();
}
```

Using the Java timed wait primitive

`public final void wait(long timeout) throws InterruptedException`

modify the **`Fork`** monitor such that after a wait of 1 second, the call to **`get`** times out and returns the result false. The Philosopher should release the other fork, if it holds it, and try again.