

## CS240 Operating Systems, Communications and Concurrency - Dermot Kelly

### Practical 8 Concurrent Programming in Java

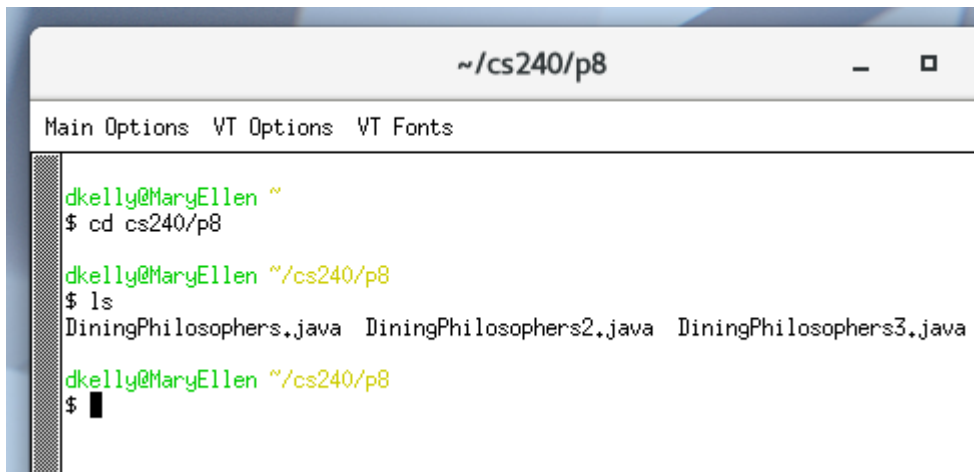
The Dining Philosophers problem is an example of a concurrency problem dealing with the allocation of limited resources among competing processes. The code of a solution to the dining philosophers problem was covered in class and is given again below. Store this code in a single file named DiningPhilosophers.java.

STEP 1: Create a new directory for the practical p8 in your cs240 folder.

STEP 2: Store the file DiningPhilosophers.java in the p8 directory.

STEP 3: Create two copies of this file, one with the name DiningPhilosophers2.java and the other with the name DiningPhilosophers3.java. These files will be where you do your code modifications later.

STEP 4: Edit the file DiningPhilosophers2.java and change the name of the public class to DiningPhilosophers2. Edit the file DiningPhilosophers3.java and change the name of the public class to DiningPhilosophers3.



```
~/cs240/p8
Main Options  VT Options  VT Fonts
dkelly@MaryEllen ~
$ cd cs240/p8
dkelly@MaryEllen ~/cs240/p8
$ ls
DiningPhilosophers.java DiningPhilosophers2.java DiningPhilosophers3.java
dkelly@MaryEllen ~/cs240/p8
$
```

STEP 5: Study the DiningPhilosophers.java program and try to determine what you expect it to do before running, or run it and then look at the program to understand where the output is being generated.

It is most likely you will get a different output sequence each time you run it due to the different scheduling of the threads each time. You can interrupt the execution with CTRL-C. In the output sequence you might notice that a hungry philosopher does not always get the chopsticks immediately if one of its neighbours is still eating and has to wait until they are thinking again. See if you can identify where this happens in the output.

### TO DO

Note that from the way the program is written, the simulation could deadlock if the threads were scheduled in such a way that each philosopher picked up one chopstick. If the code deadlocks then it will stop producing output as all the threads will be blocked. It might never happen but there is a risk it could.

STEP 6: **Modify** the program in **two** ways (as suggested in class) to create simulations that won't ever deadlock while allowing all five threads to continue to operate concurrently. Alter the files Diningphilosophers2.java and DiningPhilosophers3.java with the changes to the existing code you wish to make for the two solutions. You can add additional output statements if you wish to help visualise the synchronisation.

STEP7: Compile DiningPhilosophers2.java and execute it. Then compile DiningPhilosophers3.java and execute it.

```
// Store these three classes in the file DiningPhilosophers.java
// Study the class Semaphore and the class Philosopher included below.
// A Semaphore object maintains a private integer which can only be accessed by the
// operations acquire and release. These are declared as synchronized which means the
// methods execute indivisibly on the semaphore's value when they are invoked by
// different threads.
// The Philosopher class extends the Thread class and defines a new
// run() method for it which is executed when the thread is started.
// Methods of the same name as the class in which they are defined are known as
// constructors. These are used to initialise instances of objects when they are first
// created.
// The class DiningPhilosophers contains the main method and this is where execution
// begins.
```

```
public class DiningPhilosophers {
    public static void main(String args[]) {
        Semaphore chopSticks[];
        Philosopher workerThread[];

        // Create an array of five Semaphore Object Reference Handles
        chopSticks = new Semaphore[5];

        // Create five Binary Semaphore Objects and assign to the array
        for (int i=0; i<5; i++)
            chopSticks[i] = new Semaphore(1); // Semaphore initial value=1

        // Create an array of five Philosopher Thread Object Reference Handles
        workerThread = new Philosopher[5];

        // Create and initiate five Philosopher Thread Objects
        for (int i=0; i<5; i++) {
            workerThread[i] = new Philosopher(i, chopSticks);
            workerThread[i].start();
        }
    }
}
```

```

// The Philosopher class implements a run() method defining the behaviour of a Philosopher thread
class Philosopher extends Thread {
    private int myName;
    private Semaphore chopSticks[];

    // This is the constructor function which is executed when a Philosopher
    // thread is first created
    public Philosopher(int myName, Semaphore chopSticks[]) {
        // 'this' distinguishes the local private variable from the parameters
        this.myName = myName;
        this.chopSticks = chopSticks;
    }
    //
    // This is what each philosopher thread executes
    //
    public void run() {
        while (true) {
            System.out.println("Philosopher "+myName+" thinking.");
            try {
                sleep ((int)(Math.random()*20000));
            } catch (InterruptedException e) {}

            System.out.println("Philosopher "+myName+" hungry.");
            chopSticks[myName].acquire(); // Acquire left
            chopSticks[(myName+1)%5].acquire(); // Acquire right

            System.out.println("Philosopher "+myName+" eating.");
            try { // Simulate eating activity for a random time
                sleep ((int)(Math.random()*10000));
            } catch (InterruptedException e) {}

            chopSticks[myName].release(); // Release left
            chopSticks[(myName+1)%5].release(); // Release right
        }
    }
}

/* The Semaphore class contains methods declared as synchronized. Javas locking mechanism will
ensure that access to Semaphore methods is mutually exclusive among threads that invoke these
methods.*/
class Semaphore {
    private int value;

    public Semaphore(int value) {
        this.value = value;
    }

    public synchronized void acquire() {
        while (value == 0) {
            try {
                // Calling thread waits until semaphore is free
                wait();
            } catch (InterruptedException e) {}
        }
        value = value - 1;
    }

    public synchronized void release() {
        value = value + 1;
        notify();
    }
}

```

# What to submit on moodle.

Open two xterminals and two editor windows.

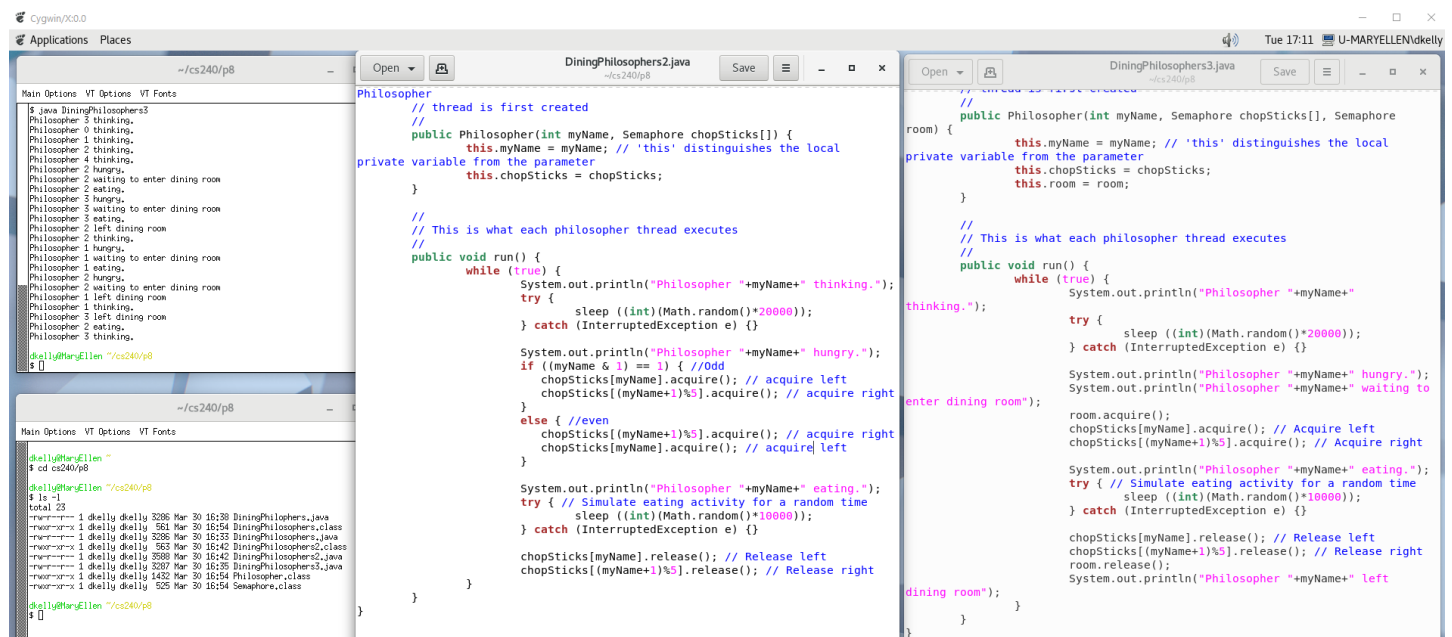
In the editor windows show only the Philosopher class code from both DiningPhilosophers2 and DiningPhilosophers3 programs that you altered.

In one xterminal window give a long directory listing of your p8 folder.

In the other xterminal window, show the output from one of the programs shown in your editor windows.

Take a screenshot and upload to moodle.

Remember that your picture must be readable so do not compress it.



```
Philosopher
// thread is first created
public Philosopher(int myName, Semaphore chopSticks[]) {
    this.myName = myName; // 'this' distinguishes the local
    private variable from the parameter
    this.chopSticks = chopSticks;
}

// This is what each philosopher thread executes
//
public void run() {
    while (true) {
        System.out.println("Philosopher "+myName+" thinking.");
        try {
            sleep ((int)(Math.random()*20000));
        } catch (InterruptedException e) {}

        System.out.println("Philosopher "+myName+" hungry.");
        if ((myName & 1) == 1) { //Odd
            chopSticks[myName].acquire(); // acquire left
            chopSticks[(myName+1)%5].acquire(); // acquire right
        }
        else { //even
            chopSticks[(myName+1)%5].acquire(); // acquire right
            chopSticks[myName].acquire(); // acquire left
        }

        System.out.println("Philosopher "+myName+" eating.");
        try { // Simulate eating activity for a random time
            sleep ((int)(Math.random()*10000));
        } catch (InterruptedException e) {}

        chopSticks[myName].release(); // Release left
        chopSticks[(myName+1)%5].release(); // Release right
    }
}

}

}

DiningPhilosophers3.java
// thread is first created
public Philosopher(int myName, Semaphore chopSticks[], Semaphore
room) {
    this.myName = myName; // 'this' distinguishes the local
    private variable from the parameter
    this.chopSticks = chopSticks;
    this.room = room;
}

// This is what each philosopher thread executes
//
public void run() {
    while (true) {
        System.out.println("Philosopher "+myName+"
thinking.");
        try {
            sleep ((int)(Math.random()*20000));
        } catch (InterruptedException e) {}

        System.out.println("Philosopher "+myName+" hungry.");
        System.out.println("Philosopher "+myName+" waiting to
enter dining room");
        room.acquire();
        chopSticks[myName].acquire(); // Acquire left
        chopSticks[(myName+1)%5].acquire(); // Acquire right

        System.out.println("Philosopher "+myName+" eating.");
        try { // Simulate eating activity for a random time
            sleep ((int)(Math.random()*10000));
        } catch (InterruptedException e) {}

        chopSticks[myName].release(); // Release left
        chopSticks[(myName+1)%5].release(); // Release right
        room.release();
        System.out.println("Philosopher "+myName+" left
dining room");
    }
}

}

}

Main Options VT Options VT Fonts
$ java DiningPhilosophers3
Philosopher 3 thinking.
Philosopher 0 thinking.
Philosopher 1 thinking.
Philosopher 2 thinking.
Philosopher 4 thinking.
Philosopher 2 hungry.
Philosopher 2 waiting to enter dining room
Philosopher 2 eating.
Philosopher 3 hungry.
Philosopher 3 waiting to enter dining room
Philosopher 3 eating.
Philosopher 2 left dining room
Philosopher 2 thinking.
Philosopher 1 hungry.
Philosopher 1 waiting to enter dining room
Philosopher 1 eating.
Philosopher 2 hungry.
Philosopher 2 waiting to enter dining room
Philosopher 1 left dining room
Philosopher 3 left dining room
Philosopher 2 eating.
Philosopher 3 thinking.
dell@MaryEllen ~/cs240/p8
$ ls -l
total 23
-rw-r--r-- 1 dell dell 5285 Mar 30 16:38 DiningPhilosophers.java
-rw-r--r-- 1 dell dell 551 Mar 30 16:54 DiningPhilosophers.class
-rw-r--r-- 1 dell dell 5285 Mar 30 16:33 DiningPhilosophers2.java
-rw-r--r-- 1 dell dell 563 Mar 30 16:42 DiningPhilosophers2.class
-rw-r--r-- 1 dell dell 5588 Mar 30 16:42 DiningPhilosophers3.java
-rw-r--r-- 1 dell dell 5287 Mar 30 16:35 DiningPhilosophers3.class
-rw-r--r-- 1 dell dell 1432 Mar 30 16:54 Philosopher.class
-rw-r--r-- 1 dell dell 525 Mar 30 16:54 Semaphore.class
dell@MaryEllen ~/cs240/p8
$
```