# Concurrent Programming
COMP 409, Winter 2018

## Assignment 2

**Due date: Wednesday, February 21, 2018**
**6pm**

These instructions require you use Java. All code should be well-commented, in a professional style, with appropriate variables names, indenting, etc. Your code must be clear and readable. **Marks will be very generously deducted for bad style or lack of clarity.**

In this assignment you must use blocking synchronization in all solutions. Do not spin, and do not use `tryLock` or other APIs that allow for a locking or acquire operation to timeout or fail.

All shared variable access must be properly protected by synchronization—there must be *no race conditions,* but also no unnecessary use of synchronization. Unless otherwise specified, your programs should aim to be efficient, and exhibit high parallelism, maximizing the ability of threads to execute concurrently. Please stick closely to the described input and output formats.

1. Suppose you have an $n$ vertex 2D polygon with the property that for each boundary point $p$, a ray extended from the center (origin) through $p$ does not intersect with a similar ray extended from the center to any other boundary point $p'$ (a *star* property). **15**

   Here you will repeatedly modify such a polygon using multiple threads. Define a class/file `star.java` that accepts two command-line parameters $m$ and $c$, such that $m \leq n$, and $c \geq 0$.

   In the program, first create the polygon, representing it by a doubly-linked list of $n$ vertices ($x, y$ pairs and next/prev references). Hard-code a polygon based on the following (circular) sequence of vertices:

   (-1.0,5.0)
   (1.0,2.0)
   (5.0,0.0)
   (1.0,-2.0)
   (-4.0,-4.0)
   (-3.0,-1.0)

   Once the polygon is constructed, launch $m$ threads. Each thread does the following $c$ times:

   (a) Choose a random vertex, $v$.
   (b) Move the vertex to a random point within the triangle formed by $v.prev$, $v$, $v.next$
   (c) Sleep for 30ms.

   Access to each vertex is to be protected by its own lock (or synchronized block). Do not use any other synchronization.

   Once each thread has done $c$ modifications, draw the resulting polygon, scaled down to fit a 1920x1080 bitmapped (png) image and saving it as a file `output.png`.

   Your solution should guarantee that the resulting polygon retains the original *star* property.

2. A cat consists of a body, tail, 2 forelegs which have 5 toes and 2 hindlegs which have 4 toes, a head, 2 eyes, and 6 whiskers[1]. They are created by an assembly process. Infinite bins of each individual part, bodies, tails, legs, toes, heads, eyes, and whiskers, is available. The cat is then assembled by robots doing the following specialized tasks.

[1]A simplification for the assignment; they have more than that.

(a) Toes. A leg is acquired, and (randomly) either 4 toes or 5 toes are acquired and attached, producing either a single foreleg or hindleg. This takes 10–20ms.

(b) Legs. A body is acquired, with or without a tail, and 4 legs are attached to the body to give it 2 forelegs and 2 hindlegs. This takes 30–50ms.

(c) Tail. A body, with or without legs, is acquired along with 1 tail to form a more complete body. This takes 10-20ms.

(d) Eyes. A head, with or without whiskers, is acquired along with 2 eyes to form a more complete head. This takes 10-30ms.

(e) Whiskers. A head, with or without eyes, is acquired along with 6 whiskers to form a more complete head. This takes 20-60ms.

(f) Cat. A head with both eyes and whiskers is attached to a body that has all 4 legs and a tail. This takes 10-20ms.

There are two robots for each task, except for the final cat assembly which only uses one robot. Separate bins exist for each distinct output part (and the final cats). A bin may only be accessed by one robot at a time.

Create a simulation, `catmaker.java` that models this assembly process. Represent each robot by a separate thread, mimicking the assembly work times through sleep. Run the program until 250 complete cats have been produced by the final cat assembler robot; other than this robot, other robots should not keep count of the number of parts produced.

As output, emit proportion of total simulation time that each robot has spent idle (not “working”).

(a) Produce a solution using *monitors*, based on `synchronized` and `wait` / `notify` / `notifyAll`. **15**

(b) Produce a solution using *semaphores*. You may use the `java.util.concurrent.Semaphore` class, or construct your own. **15**

## What to hand in

Submit your assignment to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**. Assignments must be submitted on the due date **before 6pm**.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files. For any written answer questions, submit either an ASCII text document or a .pdf file *with all fonts embedded*. Do not submit .doc or .docx files. Images (plots or scans) are acceptable in all common graphic file formats.

This assignment is worth 10% of your final grade. **45**