Completion of the course

Lectures

Monday and Wednesday, 9:00-10:30 on Zoom.

Office hours

Monday and Wednesday, 10:30-11:30 on Zoom. If that timeslot does not suit your schedule, you can make a virtual appointment by email.

Labs

Friday, 10:00-11:00 on Zoom. If you have any questions about the lab, then you can also send me email or post your questions on the forum.

Completion of the course

Second progress report

Submit **before** Tuesday March 24.

Presentations

Monday March 30, 9:00-10:30 and Wednesday April 1, 9:00-10:30 on Zoom.

Final exam

"Take home" exam on Wednesday April 8, 19:00-21:00. The questions will be available online at 19:00. Students have two hours to complete the exam and submit their answers electronically.

Report and code

Submit **before** Thursday April 16.

Completion of the course

- Do the deadlines for the second progress report and the report and code work for you?
- Any other suggestions?

Presentations

- Presentations will be done via Zoom.
- For the groups with two students, both students should present a part.
- Each student should present between 5 and 10 minutes (2 minutes is not enough, 15 minutes is too much).
- After each presentation, both the instructor and the students can ask questions.

Presentations

- Scheduling of the presentations is done by a randomized algorithm.
- If the assigned slot does not fit your schedule, try to swap your slot with others.
- Let us run the code and record the schedule.

Concurrency

EECS 4315

www.eecs.yorku.ca/course/4315/

State space of readers-writers

Number of states

	0	1	2	3	4	5
0	1	30	374	4,046	41,115	400,033
1	24	338	3,833	39,791	391,614	3,711,014
2	356	3,894	40,009	394,027	3,745,232	
3	4,352	42,856	413,962	3,913,381		
4	47,786	452,488	4,234,977			
5	493,298	4,645,734	42,964,550			

Columns: number of readers
Rows: number of writers

The state space explosion problem in action.

The dining philosophers problem

In the dining philosophers problem, due to Dijkstra, five philosophers are seated around a round table. Each philosopher has a plate of spaghetti. A philosopher needs two forks to eat it. The layout of the table is as follows.

The life of a philosopher consists of alternative periods of eating and thinking. When philosophers get hungry, they try to pick up their left and right fork, one at a time, in either order. If successful in picking up both forks, the philosopher eats for a while, then puts down the forks and continues to think.

The dining philosophers problem

```
public class Philosopher extends Thread {
  private int id;
  private Table table;

  public Philosopher(int id, Table table) {
    this.id = id;
    this.table = table;
  }

  public void run() {
    while (true) {
      this.table.pickUp(id);
      this.table.pickUp(id + 1);
      // eat
      this.table.putDown(id);
      this.table.putDown(id + 1);
    }
```

The dining philosophers problem

```
public class Philosophers {
  public static void main(String[] args) {
    final int PHILOSOPHERS = 5;
    Table table = new Table(PHILOSOPHERS);
    for (int id = 0; id < PHILOSOPHERS; id++) {
      (new Philosopher(id, table)).start();
    }
  }
}
```

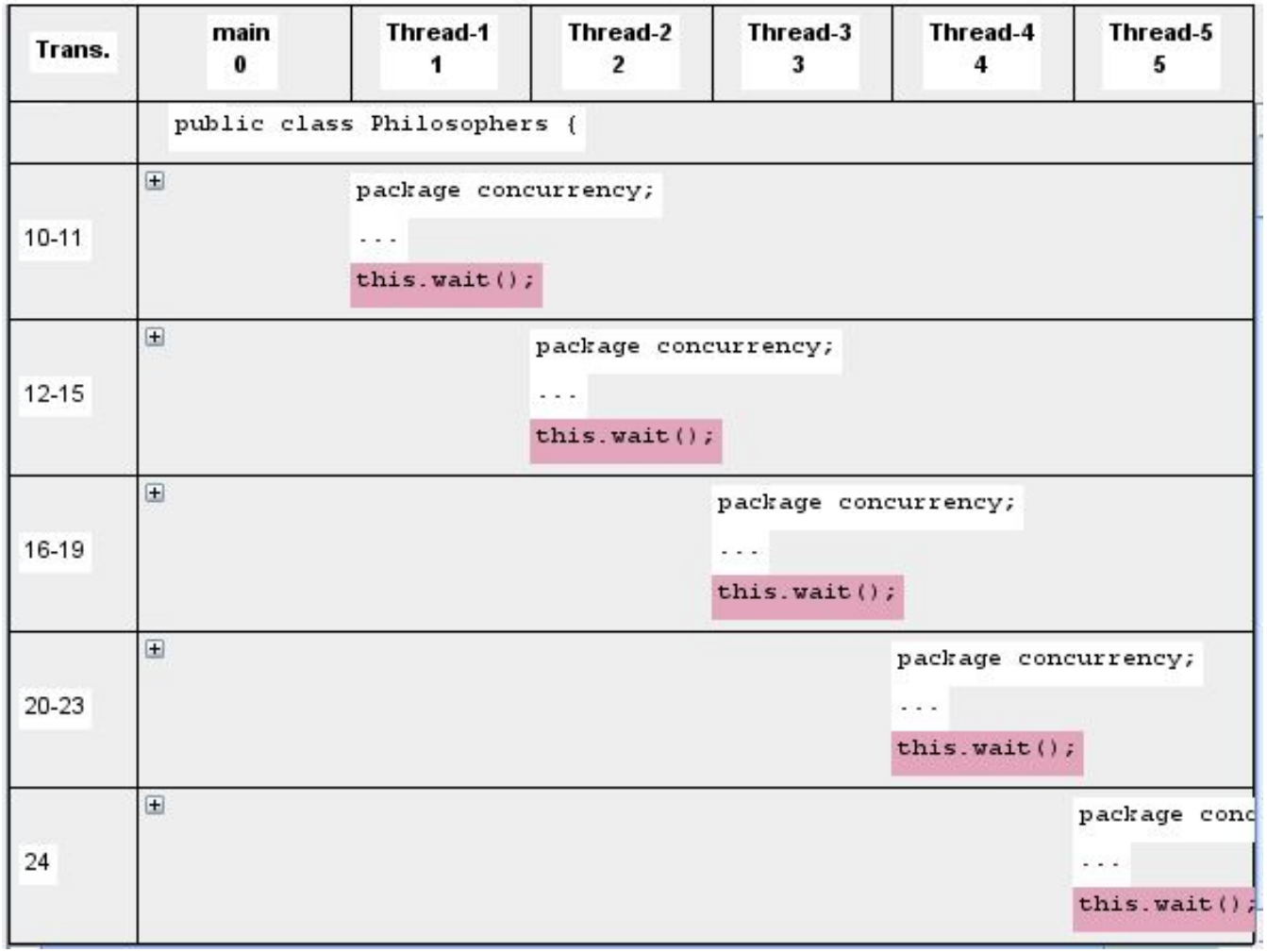
The dining philosophers problem

```
public class Table {
  private int size;
  private boolean[] pickedUp;

  public Table(int size) {
    this.size = size;
    this.pickedUp = new boolean[size]; // all false
  }
```

The dining philosophers problem

```
  public synchronized void pickUp(int id) {
    while (this.pickedUp[id % this.size]) {
      try {
        this.wait();
      } catch (InterruptedException e) {}
    }
    this.pickedUp[id % this.size] = true;
  }

  public synchronized void putDown(int id) {
    this.pickedUp[id % this.size] = false;
    this.notifyAll();
  }
}
```

Deadlock



All five philosophers pick up their left fork first and then all wait for their right fork.

Solutions

- One left handed philosophers (picks up left fork first) and four right handed philosophers (pick up right forks first).
- Only allow at most four philosophers to sit down at the table.
- Keep track of each philosopher (thinking, hungry, eating).

Bounded buffer

The bounded-buffer problem, also known as the producer-consumer problem, is a classic example of concurrent access to a shared resource. A bounded buffer lets multiple producers and multiple consumers share a single buffer. Producers write data to the buffer and consumers read data from the buffer.

Producer

```
public class Producer extends Thread {
  private BoundedBuffer buffer;

  public Producer(BoundedBuffer buffer) {
    this.buffer = buffer
  }

  public void run() {
    this.buffer.put(Math.random());
  }

}
```

Consumer

```
public class Consumer extends Thread {
  private BoundedBuffer buffer;

  public Consumer(BoundedBuffer buffer) {
    this.buffer = buffer
  }

  public void run() {
    System.out.println(this.buffer.get());
  }
}
```

ProducersAndConsumers

```
public class ProducersAndConsumers {
  public static void main(String[] args) {
    final int CAPACITY = 2;
    BoundedBuffer buffer = new BoundedBuffer(CAPACITY);

    final int PRODUCERS = 2;
    for (int p = 0; p < PRODUCERS; p++) {
      (new Producer(buffer)).start();
    }

    final int CONSUMERS = 2;
    for (int c = 0; c < CONSUMERS; c++) {
      (new Consumer(buffer)).start();
    }
  }
}
```

BoundedBuffer

```
public class BoundedBuffer {
  private double[] content;
  private int front;
  private int rear;

  public BoundedBuffer(int capacity) {
    this.content = new double[capacity];
    this.front = 0;
    this.rear = 0;
  }

  public void put(double value) { ... }

  public double get() { ... }
}
```