2. Each group must write an implementation of one of the three problems presented in miniproject.txt/3 - The MiniProject.pptx (see module for Session 05), in multi-threaded Java. Your solution must be free of deadlock; you may choose any deadlock prevention or deadlock avoidance technique you like. Send your solution to kaje@ucn.dk, including code and an explanation of why the program is deadlock-free. DEADLINE: April 26th, 2021

In our case, each Philosopher has obtained his left fork but is unable to obtain his right fork because his neighbor has already obtained it. This is known as the circular wait, because it is one of the situations that leads to a deadlock and stops the mechanism from progressing.

We could prove this by running the code a few times and we saw that the code occasionally only hangs. Here's an example output that illustrates the aforementioned problem:

```
Philosopher 1 8487540546530: Thinking
Philosopher 2 8487542012975: Thinking
Philosopher 3 8487543057508: Thinking
Philosopher 4 8487543318428: Thinking
Philosopher 5 8487544590144: Thinking
Philosopher 3 8487589069046: Picked up left fork
Philosopher 1 8487596641267: Picked up left fork
Philosopher 5 8487597646086: Picked up left fork
Philosopher 4 8487617680958: Picked up left fork
Philosopher 2 8487631148853: Picked up left fork
```

We found out that it works perfectly without deadlock if except for one Philosopher, who aims for his right fork first, the rest of the Philosophers aim for their left fork first.

```
public class DiningPhilosophers {

    public static void main(String[] args) throws Exception {

        final Philosopher[] philosophers = new Philosopher[5];
        Object[] forks = new Object[philosophers.length];

        for (int i = 0; i < forks.length; i++) {
            forks[i] = new Object();
        }

        for (int i = 0; i < philosophers.length; i++) {
            Object leftFork = forks[i];
            Object rightFork = forks[(i + 1) % forks.length];

            if (i == philosophers.length - 1) {

                // The last philosopher picks up the right fork first
                philosophers[i] = new Philosopher(rightFork, leftFork);
            } else {
                philosophers[i] = new Philosopher(leftFork, rightFork);
            }

            Thread t
              = new Thread(philosophers[i], "Philosopher " + (i + 1));
            t.start();
        }
    }
}
```

The following output demonstrates one of the situations in which all Philosophers have a chance to think and eat without triggering a deadlock:

```
Philosopher 1 88519839556188: Thinking
Philosopher 2 88519840186495: Thinking
Philosopher 3 88519840647695: Thinking
Philosopher 4 88519840870182: Thinking
Philosopher 5 88519840956443: Thinking
Philosopher 3 88519864404195: Picked up left fork
Philosopher 5 88519871990082: Picked up left fork
Philosopher 4 88519874059504: Picked up left fork
Philosopher 5 88519876989405: Picked up right fork - eating
Philosopher 2 88519935045524: Picked up left fork
Philosopher 5 88519951109805: Put down right fork
Philosopher 4 88519997119634: Picked up right fork - eating
Philosopher 5 88519997113229: Put down left fork. Back to thinking
Philosopher 5 88520011135846: Thinking
Philosopher 1 88520011129013: Picked up left fork
Philosopher 4 88520028194269: Put down right fork
Philosopher 4 88520057160194: Put down left fork. Back to thinking
Philosopher 3 88520067162257: Picked up right fork - eating
Philosopher 4 88520067158414: Thinking
Philosopher 3 88520160247801: Put down right fork
Philosopher 4 88520249049308: Picked up left fork
Philosopher 3 88520249119769: Put down left fork. Back to thinking
```