Java Thread Applications

Exercise

- What is the output of the code on the next slide?
- Is there any deadlock?
    - If so, where is it?

```
public class Deadlock {
   static class Friend {
      private final String name;
      public Friend(String name) {
         this.name = name;
      }
      public String getName() {
         return name;
      }

      public synchronized void bow(Friend bower) {
         System.out.format(
              "%s: %s has bowed to me!%n",
                name, bower.getName());
         bower.bowBack(this);
      }

      public synchronized void bowBack(Friend bower) {
         System.out.format(
              "%s: %s has bowed back to me!%n",
                this.name, bower.getName());
      }
   } // class Friend

   // main method next column
}
```

```
public static void main(String[] args) {
   final Friend alphonse = new Friend("Alphonse");
   final Friend gaston = new Friend(“Gaston");

   Runnable r1 = new Runnable() {
        public void run() {
              alphonse.bow(gaston);
        }
   };
  new Thread(r1).start();

   Runnable r2 = new Runnable() {
        public void run() {
              gaston.bow(alphonse);
        }
   };
   new Thread(r2).start();
}
```

https://docs.oracle.com/javase/tutorial/essential/concurrency/deadlock.html

Definitions

- Starvation - a condition where a thread is unable to obtain regular access to a shared resource. The thread is blocked and unable to make progress.
- Livelock - a condition where a thread is not blocked but is not making progress. Sometimes occurs when two threads are busy responding to the other thread.

Fork.java

```java
/**
 * A Fork object. A fork can alternate between two states: being picked up
 * and being put down. If it is currently picked up, it must be put down
 * before it can be picked up again. Similarly, if a fork is put down, it
 * must be picked up before it can be put down again.
 *
 * @author david
 */
public class Fork {
	/** id for this fork */
	private int id;

	/** flag to indicate if this fork can be picked up */
	private boolean availableFlag = true;

	/** part of an exception message */
	private static final String pickUpMsg = " is not available";

	/** part of an exception message */
	private static final String putDownMsg = " is not being held";

	/**
	 * Creates a Fork with a specified id.
	 *
	 * @param id the specified id
	 */
	public Fork(int id) {
		this.id = id;
	}

	/**
	 * Tests whether this Fork is available.
	 *
	 * @return true if this Fork is available
	 */
	public boolean isAvailable() {
		return availableFlag;
	}

	/**
	 * Picks up this Fork.
	 *
	 * @throws IllegalStateException if the Fork is not available
	 */
	public void pickUp() {
		if (!availableFlag) {
			throw new IllegalStateException(toString() + pickUpMsg);
		}
		availableFlag = false;
	}

	/**
	 * Puts down (releases) this Fork.
	 *
	 * @throws IllegalStateException if the Fork is available
	 */
	public void putDown() {
		if (availableFlag) {
			throw new IllegalStateException(toString() + putDownMsg);
		}
		availableFlag = true;
	}

}
```

Main.java

```
Fork[] forks = new Fork[5];
for (int i = 0; i < 5; i++) {
        forks[i] = new Fork(i);
}

Object waiter = new Object();

Philosopher[] phil = new Philosopher[5];
phil[0] = new Philosopher(names[0], waiter, forks[0], forks[4]);
for (int i = 1; i < 5; i++) {
        phil[i] = new Philosopher(names[i], waiter, forks[i], forks[i-1]);
}

Thread[] th = new Thread[5];
for (int i = 0; i < 5; i++) {
        th[i] = new Thread(phil[i]);
        System.out.println("Philosopher Id: " + phil[i].getId());
}

for (int i = 0; i < 5; i++) {
        th[i].start();
}

try {
   Thread.sleep(sleepTime);

for (int i = 0; i < 5; i++) {
        phil[i].setStopping(true);
        th[i].join();
   }
} catch (InterruptedException ie) {
   ie.printStackTrace();
}
```

Philosopher.java

```
 @Override
 public void run() {
        long startTotalTime = System.nanoTime();
        boolean haveForks = false;
        while (!isStopping()) {
                // think

                // try to get forks

                // eat?
        }
        estTotalTime = System.nanoTime() - startTotalTime;
}
```

// think

```
thoughts++;
long thinkTime = (long)(random.nextFloat() * 10.0);
try {
    Thread.sleep(thinkTime);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

// try to get forks

```
long startMutexTime = System.nanoTime();
synchronized(waiter) {
    if (leftFork.isAvailable() && rightFork.isAvailable()) {
        try {
            leftFork.pickUp();
            rightFork.pickUp();
            haveForks = true;
        } catch (IllegalStateException e) {
            String msg = id + " in illegal state. "
                    + e.getMessage();
            System.out.println(msg);
            setStopping(true);
        }
    }
}
estMutexTime += System.nanoTime() - startMutexTime;
```

// eat?

```
if (haveForks) {
        meals++;
        long eatTime = (long)(random.nextFloat() * 10.0);
        try {
                Thread.sleep(eatTime);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
        try {
                leftFork.putDown();
                rightFork.putDown();
                haveForks = false;
        } catch (IllegalStateException e) {
                String msg = id + " in illegal state. "
                        + e.getMessage();
                System.out.println(msg);
                setStopping(true);
        }
}
```

Output

```
Davids-Air:cs364-2019 david$ java -cp build hw2.Main 2000
Welcome to the diner
Philosopher Id: Plato
Philosopher Id: Socrates
Philosopher Id: Kant
Philosopher Id: Confucius
Philosopher Id: Hadot

The diner is closed.
Philosopher: Plato
    Thoughts: 291,  Meals: 115
    Waiter time (ns): 5475235
    Total time (ns): 2007766370
    Ratio -- Waiter time / Total time: 0.0027
Philosopher: Socrates
    Thoughts: 270,  Meals: 109
    Waiter time (ns): 2552904
    Total time (ns): 2015244809
    Ratio -- Waiter time / Total time: 0.0013
Philosopher: Kant
    Thoughts: 298,  Meals: 104
    Waiter time (ns): 2362471
    Total time (ns): 2016886502
    Ratio -- Waiter time / Total time: 0.0012
Philosopher: Confucius
    Thoughts: 284,  Meals: 120
    Waiter time (ns): 2488524
    Total time (ns): 2019194521
    Ratio -- Waiter time / Total time: 0.0012
Philosopher: Hadot
    Thoughts: 298,  Meals: 106
    Waiter time (ns): 2744126
    Total time (ns): 2023215074
    Ratio -- Waiter time / Total time: 0.0014
Davids-Air:cs364-2019 david$
```