Content for today*

Slido: https://app.sli.do/event/nzlmzoyg

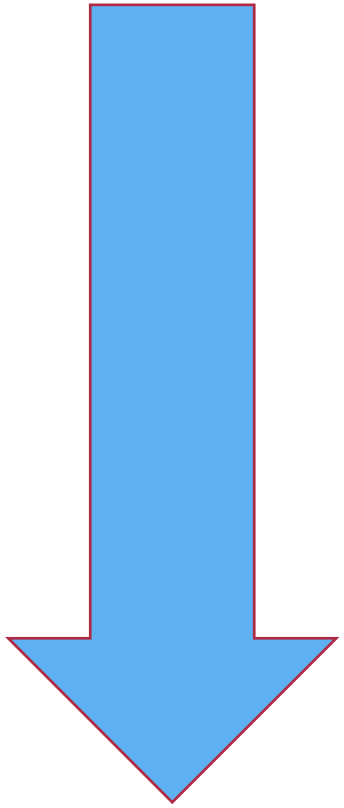
1. Discussing solution to Lab 6 and any issues arising
2. Recap of Threads, Locks, and Conditions
3. **Example code involving Threads, Locks, and Conditions**
   Simpler than, but very similar to, the code required for Lab 7

*No poll so I can have a better chance of getting through all of the above

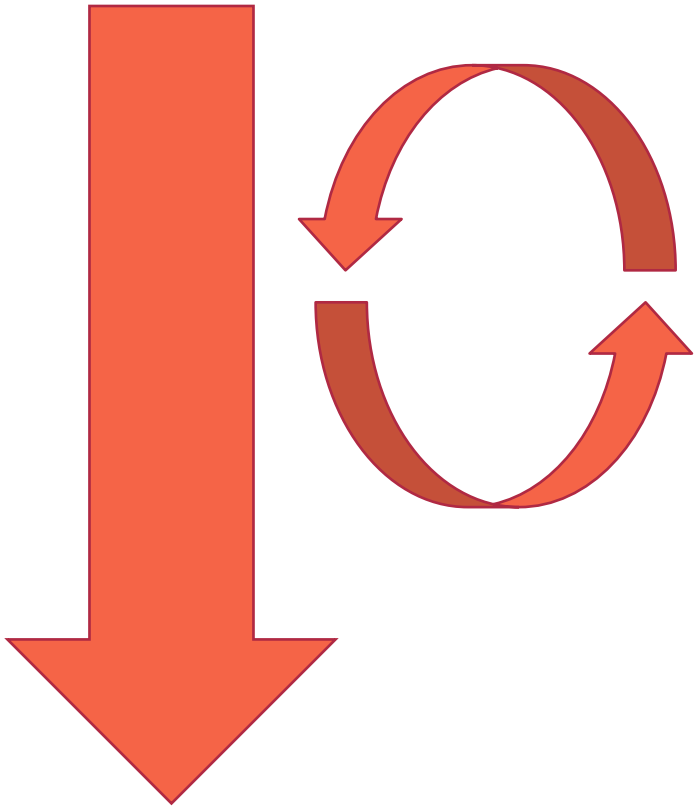
Don't forget: If you have questions about JP2 (or even Level 2 in general), you can attend my office hours from 2-3pm every Friday – link at the top of the JP2 Moodle page.

Programming with Threads

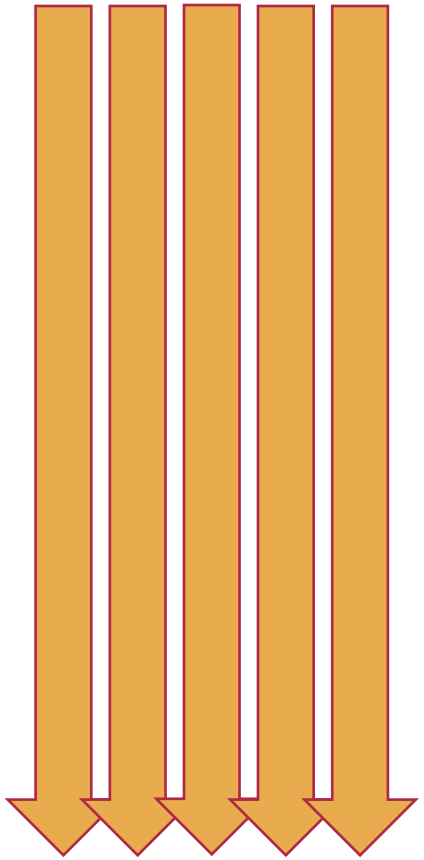
Standard program



Swing



Multiple threads



Creating a Thread

```
public class MyClass implements Runnable {
   // Constructors, fields, other methods, …
    public void run() {
        // … whatever should happen in the thread
    }
}
// In another part of the program
MyClass mc = new MyClass();
Thread thread = new Thread (mc);
```

Things to do with a Thread object

Starting	Starting the thread • thread.start();
Stopping	Telling the thread that it should stop • thread.interrupt();
Waiting	Waiting for the thread to finish • thread.join(); • thread.join(time); // Time in milliseconds

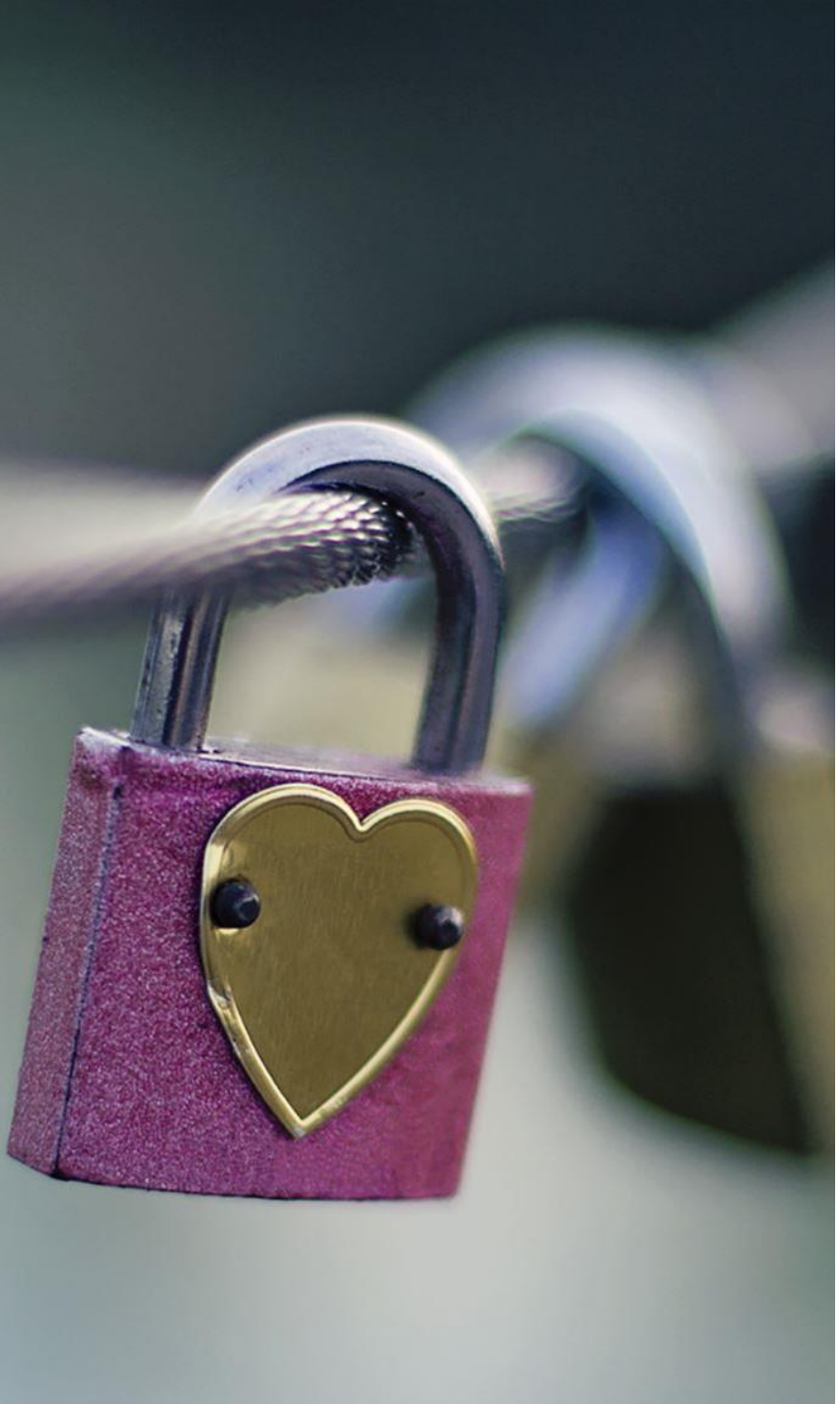
Things to do inside a run() method

Pause for a certain amount of time

```
Thread.sleep(time);
// Always runs on current thread
```

Check whether we have been interrupted, and return as soon as possible if so

- Some things will throw `InterruptedException`
  - `Thread.sleep();`
  - Waiting for locks
  - Calling `join()` on another thread
- Otherwise, check `Thread.interrupted()` periodically (e.g., before or after long-running method calls)



Locks

What does a lock do?

- Ensures that one thread (t1) gets a lock, no other thread can access any of the resources controlled by that lock until t1 releases the lock

Why do you need them?

- To control access to shared resources
- Without locking, threads might
  - *Overwrite shared data in an inconsistent way*
  - *Exceed a resource limitation*

Simple locking: **synchronized** modifier

Fully featured locking: **java.util.concurrent.locks.Lock**

Locks and Conditions

A **Condition** allows a thread to suspend execution until notified by another thread that some state condition may now be true

General use case:

- Check if some state condition is satisfied
- If so, continue without waiting
- If not
  - *Call **await()** on the Condition object – thread will release lock and go to sleep*
  - *When the condition becomes true, another thread will call **signal()** on the condition object, which wakes up the waiting thread*
  - *Waiting thread can continue processing (might be good to double check that state is as expected)*

Message passing example

Implements a one-slot message passing service

Each thread runs the following loop until interrupted:

- If there is no message waiting, put your message in the slot and wait for someone to take it
- If there is a message waiting, remove it from the slot and pause 200ms before notifying the waiting thread

Using a Lock and a Condition

VERY SIMILAR CONCEPTUALLY TO LAB 7 BATTLE BEHAVIOUR