Object Oriented Programming

Week 10 Part 3
Multi-threading: Inter-thread Communication

Lecture

- Inter-thread Communication

Inter-thread Communication

Synchronization Problem

- So far the threads have simply waited for either a second or half a second before waking.
- Often, threads need restart when events happen
- This can be done by polling a shared variable
  - *Polling* is continually checking the value of a variable. A thread sleeps for a while, checks the value, and continue when the value is right
- Guarded blocks make polling efficient

Polling

Loops forever until a Thread sets wait to False

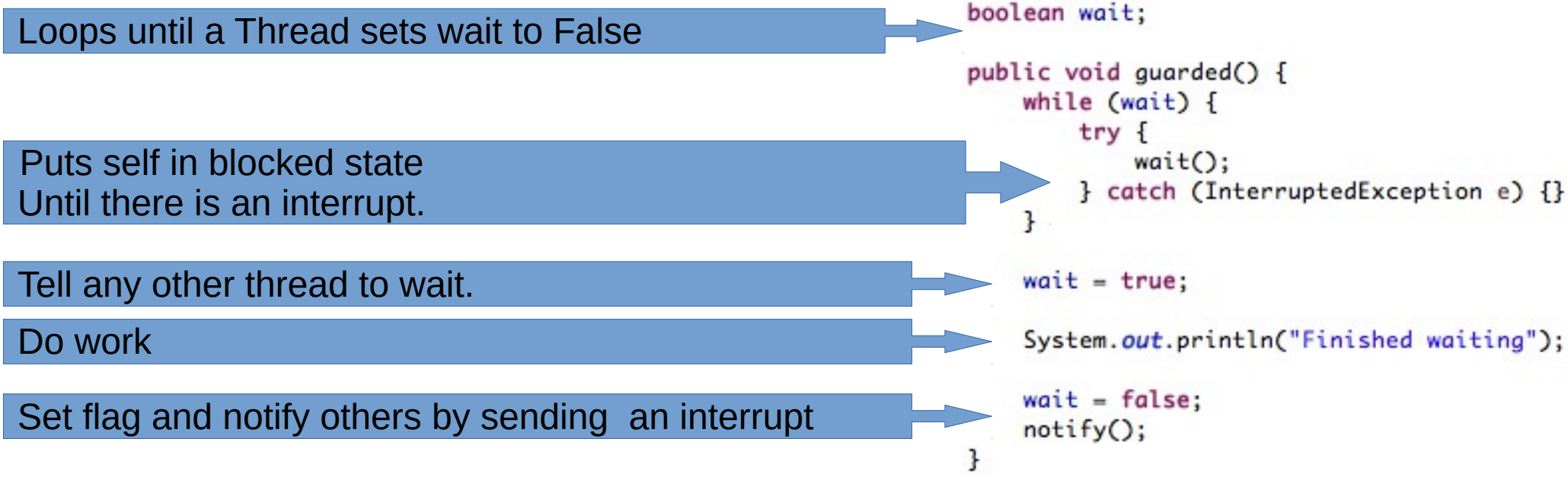
```
public void polling() {
    while (SimpleThreads.wait) {}
    System.out.println("Finished waiting");
}
```

- Polling is inefficient
  - It will run continuously through the threads time slot.
  - It is only preempted when it has used the entire slot doing nothing

Java Inter-thread Communication

- Three methods on Object
  - `void wait()`
    - Halts execution (i.e. permanently blocked)
    - Throws InterruptedException
  - `void notify()`
    - Sends an interrupt to a Thread waiting on the Object's monitor
  - `void notifyAll()`
    - Send an interrupt to all Threads waiting on the Object's monitor

Guarded Block



- Guarded blocks are more efficient
- It does not run until restarted by an interrupt

Using Guarded Blocks

- Let's modify the program so that thread 1 and thread 2 take turns.
- To do this we will use a Guarded Block
  - The guarded block will surround the sharedVar in SimpleThreads
  - If it was the last updater, it blocks waiting for an interrupt
  - If it was not the last updater, it updates the variable, and puts its name in as the last updater.

Example: `SimpleThreads`

Wait if last updater was self

Another thread updated and Sent interrupt

Set last updater to self and Send interrupt to those waiting

```
public static synchronized void incrementAndPrint(int i) {
    while (lastUpdater.equals(Thread.currentThread().getName())) {
        try {
            printMessage(Thread.currentThread().getName()
                    + " waiting");
            SimpleThreads.class.wait();
        } catch (InterruptedException e) {}
    }
    sharedVar = sharedVar + 1;
    printMessage(String.format("loop %d: sharedVar = %d",
            i, sharedVar));
    lastUpdater = Thread.currentThread().getName();
    SimpleThreads.class.notify();
}
```

Example: Output

Thread 2 goes first at ½ sec

Thread 1 goes second at 1 sec

Thread 2 waits at 1 ½ secs

Thread 2 waits at 2 ½ secs

```
main, running: 5 ms, Thread 1 started
main, running: 9 ms, Thread 2 started
Thread 2, running: 511 ms, loop 0: sharedVar = 1
Thread 1, running: 1009 ms, loop 0: sharedVar = 2
Thread 2, running: 1017 ms, loop 1: sharedVar = 3
Thread 2, running: 1523 ms, Thread 2 waiting
Thread 1, running: 2011 ms, loop 1: sharedVar = 4
Thread 2, running: 2012 ms, loop 2: sharedVar = 5
Thread 2, running: 2517 ms, Thread 2 waiting
Thread 1, running: 3012 ms, loop 2: sharedVar = 6
Thread 2, running: 3013 ms, loop 3: sharedVar = 7
Thread 1, running: 4018 ms, loop 3: sharedVar = 8
main, running: 4018 ms, Ending main()
```

Week's Lesson

- Concurrency and threads
- Starting threads
- Synchronization
- Inter thread communication