Concurrency:
Multi-core Programming
& Data Processing

Lab 7

-- Signaling and conditions --

Guarded blocks

- **Option 1:** poll shared variable continuously in a loop

```
while(!stop){}
```

- Works, but bad practice!

Guarded blocks

- **Option 1:** poll shared variable continuously in a loop

```
while(!stop){}
```

- Works, but bad practice!

- **Option 2:** `wait()` inside the loop

```
while(!stop) {
  try{
    wait();
  }catch (InterruptedException e) {}
```

- Important: <u>always</u> in a loop – check if your condition still holds

Wait – notify(All)

- `wait()` and `notify()` are applied to objects used as locks
- Thread + lock + wait =>
  - lock is released, thread inactive
  - thread wakes up on notify-like operation (or for no reason at all...)
  - Has to re-acquire the lock before exiting `wait()` method
- Thread + lock + notify =>
  - All other threads waiting on same lock are notified (`notifyAll()`)
  - One random thread waiting on same lock is notified (`notify()`)

Recommendation

- Don't use them directly if not absolutely necessary.
  - Use high-level concurrency utilities instead.

Conditions

- **Option 3:** Condition interface
- Operations:
  - `await()` - thread waits until signaled
  - `signal(All)()` - wakes up one (all) thread(s)
  - created from a lock object with `lock.newCondition()`

Conditions

- **Option 3:** Condition interface
- Operations:
  - `await()` - thread waits until signaled
  - `signal(All)()` - wakes up one (all) thread(s)
  - created from a lock object with `lock.newCondition()`
- Wait-and-notify++

Conditions

- **Option 3:** Condition interface
- Operations:
  - `await()` - thread waits until signaled
  - `signal(All)()` - wakes up one (all) thread(s)
  - created from a lock object with `lock.newCondition()`
- Wait-and-notify++
  - Condition + Lock = love (Lock not compatible with wait-and-notify – they are used internally by the Lock implementation)

Conditions

- **Option 3:** Condition interface
- Operations:
  - `await()` - thread waits until signaled
  - `signal(All)()` - wakes up one (all) thread(s)
  - created from a lock object with `lock.newCondition()`
- Wait-and-notify++
  - Condition + Lock = love (Lock not compatible with wait-and-notify – they are used internally by the Lock implementation)
  - More than one Condition per Lock => target groups of threads individually

Exercise

- Implement the producer-consumer problem with wait() and notify() constructs.
- Modify the implementation to use Conditions.
- Hint: you can use as starting point the implementation with semaphores found on the forum on ILIAS.