

§23.5: Thread Synchronization

11 April 2007

CMPT167

Dr. Sean Ho

Trinity Western University

Review last time

■ Multithreading

- Thread **states**: runnable, wait, timed wait
- **Scheduling**
- Creating **tasks** in Java 1.5
- **Executors** (managers) in Java 1.5

Quiz 6 (10 minutes)

- Describe in words the roles of **DatagramSocket** and **DatagramPacket**. [4]
- What information do we need to **send** a **DatagramPacket** (e.g., arguments to constructor) [4]
- Name and describe in words three of the five **states** in which a thread can be. [4]
- Name the interface and method used to define a new **thread** task. [3]
- Tell me everything you know about **task schedulers**. [5]

Quiz 6: answers #1-2

- Describe in words the roles of **DatagramSocket** and **DatagramPacket**. [4]
 - **Socket**: channel of communication
 - ◆ Server listens (**.receive()**), client talks (**.send()**)
 - **Packet**: payload/content to be communicated
- What information do we need to **send** a **DatagramPacket** (e.g., arguments to constructor) [4]
 - **Data** byte array, **length**, **IP address**, **port**

Quiz 6: answers #3-5

- Name and describe in words three of the five states in which a thread can be. [4]
 - New, runnable, waiting, timed wait, terminated
- Name the interface and method used to define a new **thread** task. [3]
 - Interface **Runnable**, method **.run()**
- Tell me everything you know about **task schedulers**. [5]
 - Decides what **thread** gets the CPU: may **preempt** currently running thread to give CPU to another thread with higher **priority**, prevent **starvation**

Thread synchronization

- Threads are run by the **Executor**
- If two threads wish to modify a **shared** object, we need **synchronization**
 - **Mutual exclusion** (mutex): only **one** thread accesses shared object at a time
 - **Locks**: a way to implement mutex
 - ◆ Thread **asks** for lock before modifying object
 - ◆ If it **gets** the lock, it can modify
 - ◆ If not, **wait** (block) until the lock is freed
 - ◆ **Free** the lock when done modifying



Lock interface

- Any **object** can be a lock if it implements **Lock**
 - ◆ In package `java.util.concurrent.locks`
 - Two **methods**: `.lock()` and `.unlock()`
 - ◆ `.lock()` will **wait** until the lock is freed
 - ◆ If many threads are waiting, **which** one gets it first?
- **ReentrantLock**: can set **fairness** policy
 - **Longest**-waiting thread gets the lock first
- **Deadlock** happens when each thread is waiting on a lock held by another thread

TODO

- Lab5 due tonight:
 - File I/O
 - Store inventory and point-of-sale system
 - Worth 60 points
- Last day for submitting late labs is Fri 13Apr
- Last day of classes is Mon 16Apr
- Final exam is Fri 20Apr 2–4pm