

§23.1-23.4: Threads

4 April 2007

CMPT167

Dr. Sean Ho

Trinity Western University

Review last time

- Connectionless client/server networking with **UDP**
- **Receiving** a UDP packet (server)
- **Sending** a UDP packet (client)

What's on for today

■ Multithreading

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

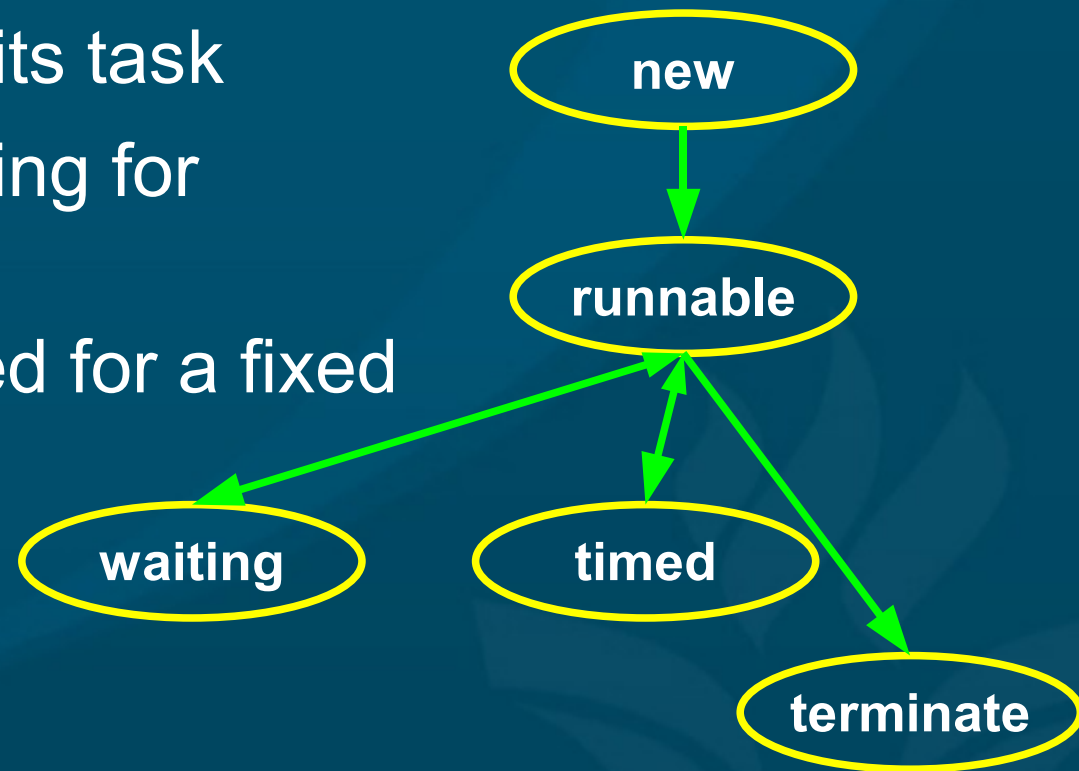
Multithreading

- **Concurrency** is running multiple tasks at the same time
 - Downloading a file, watching a movie, checking email
 - One **server** talking to multiple clients
- **Threads** are individual tasks (objects) that may run concurrently
 - **Executor** (master) thread starts and coordinates worker threads
- Multithreading is built-in to Java ≥ 1.5

Thread state diagram

- Threads can be in one of four **states**:

- **New**: not yet initialized
- **Runnable**: executing its task
- **Waiting**: blocked waiting for another thread
- **Timed** waiting: blocked for a fixed time
- **Terminated**



Task scheduling

- The API allows a program to **create** multiple threads
- But how many threads can run simultaneously depends on how many physical **processors** you have
 - e.g., dual-core, quad-core SMP
- The **scheduler** assigns runnable threads to processors
 - Done by the **operating system**, not the Java VM
 - If more threads than processors, scheduler may **preempt** running threads to allow others to run
 - Each thread has a **priority** (“nice” value)
 - ◆ Lower priority threads might get **starved**

Creating a thread object in Java 1.5

- Class design:
 - Each thread is a separate **object**
 - **Executor** (master thread) is another object
 - ◆ Created from **main()**
- The thread objects should implement the **interface Runnable** (java.lang):
 - Define **method**: public void **run()**
 - Can use **utility** methods in class **Thread** (java.lang)
 - ◆ **Thread.sleep(100);** // timed wait for 100ms

Example: PrintTask

```
import java.util.Random;
class PrintTask implements Runnable {
    private int sleepTime;
    private String name;
    private static Random gen = new Random();
    public PrintTask( String name ) {
        this.name = name;
        this.sleepTime = gen.nextInt( 5000 );
    }
    public void run() {
        System.out.println( name + “: good night!” );
        Thread.sleep( sleepTime );
        System.out.println( name + “: good morning!” );
    }
}
```

Managing threads in Java 1.5

- The **executor** object implements interface **ExecutorService** (java.util.concurrent):
 - Defines **method**: public void **execute()**
- The class **Executors** (java.util.concurrent) provides static methods to **create** executors:
 - ◆ `Executors.newFixedThreadPool(3);`
 - Creates a new **ExecutorService** object that can run up to three **threads** simultaneously
 - If more than three threads are to be executed, the **ExecutorService** object **queues** them up

Example: RunnableTester

```
import java.util.concurrent.*;
public class RunnableTester {
    public static void main( String args[] ) {
        PrintTask task1 = new PrintTask( "Thread 1" );
        PrintTask task2 = new PrintTask( "Thread 2" );
        ExecutorService master =
            Executors.newFixedThreadPool( 3 );
        master.execute( task1 );
        master.execute( task2 );
        master.shutdown();
    }
}
```

- Master fires up worker threads, then **quits**
- Worker threads **continue** running afterward

TODO

- Lab5 due Wed 11Apr:
 - 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