Lecture 11 – Threads

08-671
Java for Application Programmers

February 16, 2016

Terry Lee
Assistant Teaching Professor
School of Computer Science

08-671 Lecture Topics

(subject to change – but only a little bit)

#1 Intro
#2 Primitive Types
#3 Java Classes
#4 Reference Types
#5 Loops & Arrays
#6 Methods & Classes
#7 Lists & Maps
#8 File & Network I/O
#9 Swing Interfaces
#10 Swing Actions
#11 Threads
#12 Exceptions
#13 Functional Programming
#14 In-class Written Exam

* Programming Exam – this will be a 3-hour exam

Exam Plan

- Written Exam
  - In-class on Feb 25th (Thursday)
  - Location: BH A51 (Giant Eagle Auditorium)
  - Plan: multiple choice & fill-in the blank, etc.
    - Closed everything. Pencils, erasers and CMU ID
- Programming Exam
  - Date and Time: 5:30pm on Mar 1st (Tuesday)
  - Location: BH A51 (Giant Eagle Auditorium)
  - Plan: same as HW#6, but different
    - Need your laptop. Don't forget your power adapter

Outline

✓ Course Administration

⇢ Questions

Threads

Sample Final Exam Questions

Outline

- ✓ Course Administration
- ✓ Questions
- ⇢ Threads
- Sample Final Exam Questions

Goals

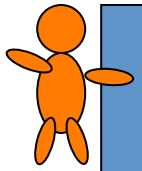
- What it is and why use it?
- How to use it
  - Recipe 1
  - Recipe 2
- Things to watch out

Thread and Java

- A thread is a flow of execution of a task (job) from beginning to end
- One of the powerful features of Java is its support for multi-threading (as mentioned in lecture 1)
- We've been using some of them already

Main Thread

- When you run a Java program, JVM starts a thread called the main thread which looks for the main method of the class you load (you know: **`java IsOdd 4`**)



```
main(String[] args) {
    if (args.length…
    …
}
```

Event Dispatch Thread

- Swing event handling code runs on a special thread called the event dispatch thread which takes care of user interface events. (**`ToDoSwingGUI.java`**)

```
main(String[] args) {
    new ToDoSwingGUI();
    …
}
```

Problem: Continuously Updating

- My app will constantly be displaying status
- I want to be able to let the user interact with the UI to update the display
- Example: `ActionLoop.java`
- To make GUI Applications responsive,
  - Event Dispatch Thread should **only be in charge of short tasks**
  - **Should use separate thread(s) for any long tasks**
  - Example: `MyThreadWindow1.java, MyThreadWindow2.java`

Your Thread

- See `java.lang.Thread` for documentation
- Use `Thread.currentThread()`
- Your thread's name:

```
Thread.currentThread().getName()
```

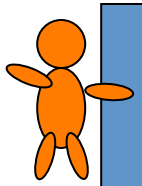
- using `Thread.sleep()`
  - Temporarily suspends the execution of the current thread for a given number of milliseconds to allow other threads (if there are any) to execute

Making More Threads (Recipe 1)

1. Place code for a task into a class that implements `Runnable Interface` and instantiate it

   `MyRunnable r = new MyRunnable();`

2. Use `new Thread(r)` to create a new thread
   - `Object r` must implement `Runnable Interface`
3. Use `start()` method on the thread



```
main(String[] args) {
    MyRunnable r = new MyRunnable(…);
    Thread t = new Thread(r);
    t.start();
    …
}
```

Runnable Interface and its run() method

- The class must implement Runnable Interface

```
public class MyRunnable implements Runnable {
    public MyRunnable(…) {
        …
    }

    @Override
    public void run() {
        …
    }
    …
}
```

Invoking `start()` method

- Do not invoke `run()` method of the Thread or the Runnable object!
  - executes the code in the `run()` method in the current thread
  - No new thread is started
- Call `Thread.start()` method
  - Executes the `run()` method in new thread

Invoking `start()` method

- Does NOT mean the new thread is actually running!
- Also, once a thread is running, it does NOT necessarily keep running!
- Up to JVM
  - Generally speaking, it gives each thread an equal amount of time to perform its task
- Again, **remember a thread may or may not be running at any given time!**

Thread States

- New
  - `Thread t = new Thread(r);`
  - Not yet running
- Runnable
  - `t.start();`
  - May or may not be actually running!! It is up to JVM
- Blocked (Timed Waiting)
  - `Thread.sleep(1000);`
- Terminated
  - `run()` completed

Examples (Recipe 1)

- GUIs that print things with multiple threads:

  `MyRunnable.java`

  `MyThreadWindow1.java`

  Notice the main thread in the main method too

Note: Second Recipe

- MyThread extends Thread
- Either public class or private nested class that extends Thread
- Advantages
  - You only need a reference to the “Thread”
  - One less object
  - Forces one thread per runnable object
  - Easier to reason about

Examples (Recipe 2)

- GUIs that print things with multiple threads:
  `MyThread.java`
  `MyThreadWindow2.java`

Additional Examples to compare two recipes

- GUIs that turn things on and off
  `Lights.java` (Recipe 1)
  `Lights2.java` (Recipe 2)
  (* easy to run in terminal)

Reasons to choose First Recipe (`implements Runnable`)

- Decouple the task (Job) from the mechanism (Worker) of running it
- Thread class can be a subclass of something else
  - Java does NOT support multiple inheritance
- Performance requires many threads sharing same job object
- Logic requires many threads sharing same job object

Multi-Threaded and Shared Access

- In most practical multi-threaded applications,
  - Two or more threads need to share access to the same data
- The question is:

  "What happens if two threads have access to the same object and each calls a method that modifies the state of the object?"

- Depending on the order, corrupted objects can result! (Race Condition)

Race Conditions

When a program may or may not execute correctly depending on how the threads are scheduled

"Getting the right answer relies on lucky timing!"

Google "worst software bugs" – look for the race conditions

Synchronize the Access

- To avoid **race conditions**, prevent more than one thread from simultaneously entering a certain part of the code (Critical Section)
- The synchronized statement can be used on objects (or on a method declaration)
  - Only one thread at a time is allowed to enter into synchronized block or synchronized method, etc.

Synchronization Examples

- Consider the following code:

```
List<String> list = new ...;
...
if (list.size() == 0) {
    list.add("First item on list.");
}
```

- The above is not correctly synchronized if there are multiple threads

Issue

Thread 1

```
list.size() -> 0
```

- Now size is 1 but thread 1 does not think like that

```
list.add("First item…");
```

Thread 2

```
list.size() -> 0
list.add("First item…");
```

Synchronization Examples

- The correct code is:

```
List<String> list = new ...;
...

 // protect critical section using synchronized block
synchronized (list) {
    if (list.size() == 0) {
        list.add("First item on list.");
    }
}
```

Example: `Lights2.java`

To witness the issue, comment out "synchronized (b)" in Lights2.java file and run with small number of buttons and large value for milliseconds

```
java Lights2 5 2000
```

Synchronization Examples

- How about this?:

```
public class BankAccount {
    private double balance;
    public void depositAmount(double amount) {
        balance += amount;
    }
    public void withdrawAmount(double amount) {
        balance -= amount;
    }
}
```

Notice that there is one statement in each method

Example: `BankAccount.java, BankTransaction.java, BankTest.java`

Synchronization Examples

- Take a closer look at the instructions in the compiled code!
- The correct code is:

```
public synchronized void depositAmount(double amount) {
    balance += amount;
}
public synchronized void withdrawAmount(double amount) {
    balance -= amount;
}
```

Example: `BankAccount.java, BankTransaction.java, BankTest.java`

What is the Chance?

- This kind of corruption may not happen when you test alone or with only a few people
- May take a few minutes or hours or days or months to occur
- There are few things worse than in the life of programmer than a bug that only shows up once every few hours, days, or months
  - **RACE CONDITION is one of them!**
- **Remember a thread may or may not be running at any given time!**

Are Objects or Methods Thread-safe?

- Check the documentation:
  - Not thread-safe: ArrayList, StringBuilder, etc.
  - Thread-safe: Vector, StringBuffer, etc.
- Also see Documentation on Swing Threading
  - In general, Swing is not thread safe
  - Swing text components provide some support of thread safe operations

Vectors vs. ArrayLists

- Vectors were first
- Vectors are just like ArrayLists
  - But Vectors are synchronized
  - Vectors did not follow the List interface
  - The List interface methods were added for compatibility
- For correct multithreaded access, you usually need to use an ArrayList protected with your own synchronized statements

StringBuffer vs. StringBuilder

- StringBuffers were first
- StringBuilders are just like StringBuffers
  - StringBuffers are synchronized
  - StringBuilders are not synchronized
- For correct multithreaded access, you usually need to use an StringBuilder protected with your own synchronized statements

Use the Unsynchronized Classes?

- Often no synchronization is required because only one thread is using an instance of a class
- Often, when multiple threads using the same instance, you may still need to add EXTERNAL synchronization

Example: `SyncArrayListDemo.java`, `SyncVectorDemo.java`

Stopping Threads

- Do NOT ever call `Thread.stop()`
  - Caller can't determine or know whether the thread is at a safe point to be stopped
  - Thus, it is deprecated
- One option is to use a method to tell the thread to stop, then let the thread exit `run()`

```
private void setFinish() { finished = true; }

public void run() {
    while (!finished) { … }
}
```

Organizational Tip!

- Calling `Thread.sleep()` method in a loop and try/catch
- Do NOT do the following because thread may continue to execute even though it is being interrupted

```
while (…) {
    …
    try {
        Thread.sleep(mySleepTimeInMillis);
    } catch (InterruptedException e) {
        …
    }
    …
}
```

Organizational Tip!

- Instead, do the following

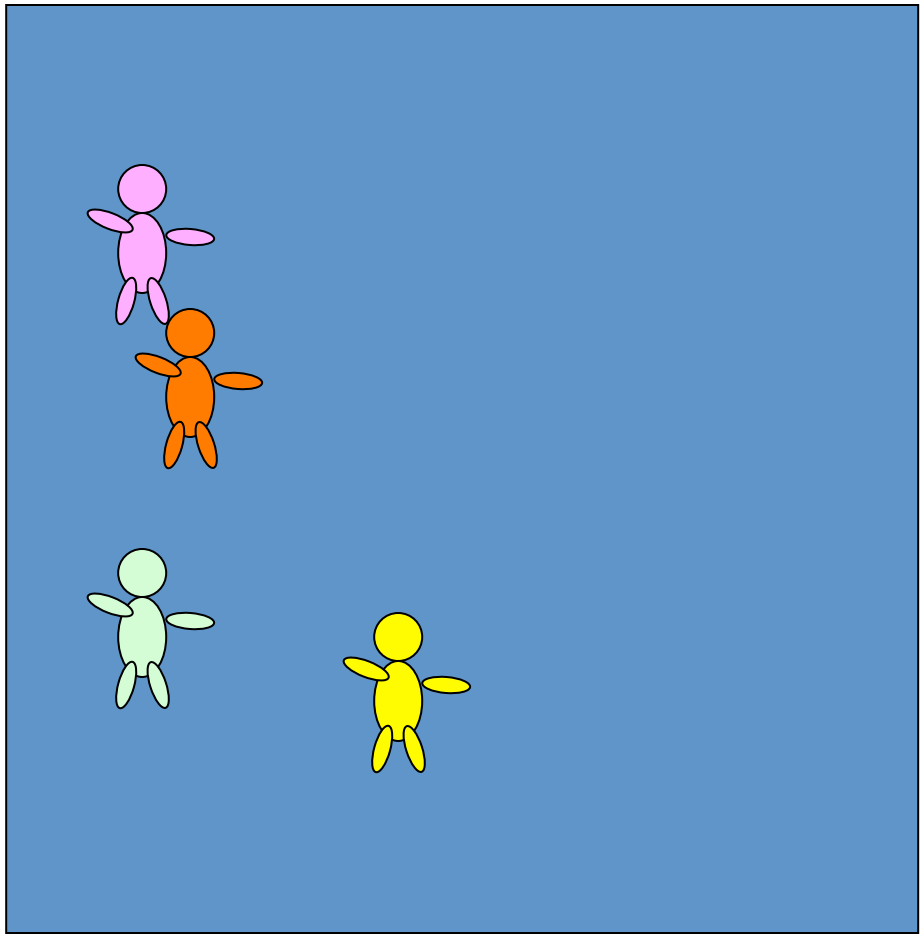
```
try {
    …
    while (…) {
        …
        Thread.sleep(mySleepTimeInMillis);
    }
    …
} catch (InterruptedException e) {
    …
}
```

Question for You

- What's the difference between a thread and a process?

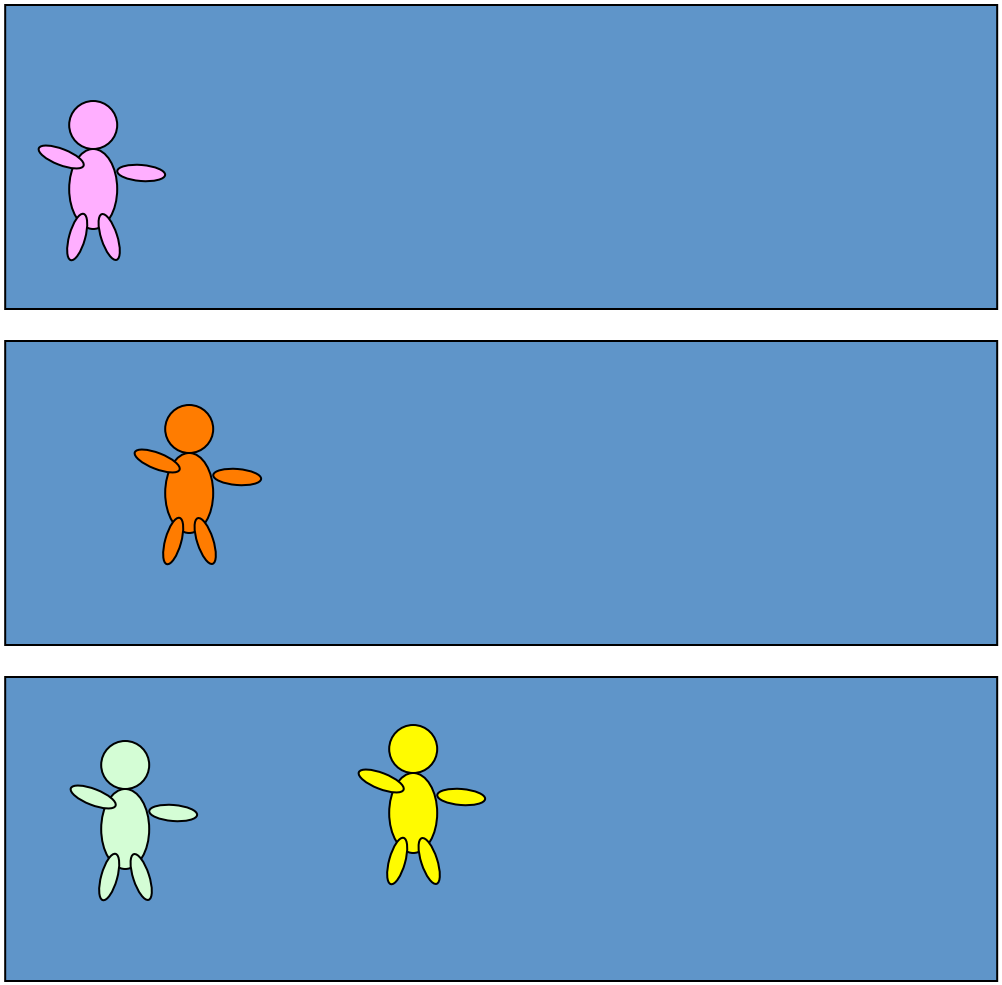
Threads

- One shared address space
  - Lots of little men concurrently executing in the *same* address space
  - Can run without or with minimal OS or hardware support (for threads)



Processes

- Lots of separate address spaces
  - Each process has separate memory
  - Each can have one or more threads
  - Requires OS support and usually hardware support



Outline

- ✓ Course Administration
- ✓ Questions
- ✓ Swing Recipes
- ✓ Threads
- ⇢ Sample Final Exam Questions

Sample Final Exam Questions

- What does the synchronized modifier mean for a method declaration?
  - How is this related to the synchronized statement?

These are the same

```
// synchronized method
public synchronized void f() {
    …
}
```

is equivalent to

```
public void f() {
    // synchronized block
    synchronized (this) {
        …
    }
}
```

Sample Final Exam Questions

- What is a race condition?
- What does it mean for an object to be immutable?
  - **Why is this important?**

Homework #7

- Go out today at 1:30 pm
- Due on Monday
  - no extensions!
- Need more time
  - start early!

Next Lecture

- Exceptions
- Thread stacks
- Recursion
- Look at memory model for objects in depth