CSC7437 : Global lab.s

J Paul Gibson,
paul.gibson@telecom-sudparis.eu

http://www-public.imtbs-tsp.eu/~gibson/Teaching/CSC7437/

Threads and Processes

/CSC7437-ThreadsAndProcesses.pdf
Processes and Threads are the two fundamental units of execution in a concurrent program.
Processes and Threads

• In Java, concurrent programming is mostly thread-based.

• Processing time for each core in a system is shared among processes and threads through an OS feature called time slicing.

• Concurrency is possible even on simple systems, without multiple processors or execution cores.

https://i4mk.wordpress.com/2013/03/13/processes-vs-threads/
Processes and Threads

The biggest difference between a process and a thread, is that each process has its own address space, while threads (of the same process) run in a shared memory space.

This means that it’s possible to share data amongst threads (e.g. reading and write to the same variables) which should be carefully done, using synchronization on the shared data.

https://i4mk.wordpress.com/2013/03/13/processes-vs-threads/
Processes

Self-contained execution environment.

Independent set of basic run-time resources, such as memory space.

A single application may be implemented by a set of cooperating processes.

Most operating systems support *Inter Process Communication* (IPC) resources.

IPC can also used for communication between processes on different systems.

Most implementations of the JVM run as a single process, with multiple threads executing in parallel.
Threads

Also known as *lightweight processes*.

Creating a new thread requires fewer resources than creating a new process.

Threads exist within a process — every process has at least one.

Threads share the process's resources, including memory and open files.

This has advantages and disadvantages … can you think of them?

Multithreaded execution is essential in Java:
  - every application has at least one thread
  - "system" threads that do memory management, event/signal handling, etc.

In programming, we start with just one thread, called the *main thread*.

Any thread (including the main thread) can create new threads.


Sams 1999

Addison Wesley, 2001
Thread Example

Download the code `Threads.zip` from the web site and import it into Eclipse.
Thread Example

```java
public class ThreadExample {

    public static void main (String[] args) {
        System.out.println("Starting Thread main");
        new SimpleThread("Add1", '1').start();
        new SimpleThread("Add2", '2').start();
        System.out.println("Finishing Thread main");
    }
}
```
Thread Example - typical output

Starting Thread main
Finishing Thread main
String Add2 extended to 2
String Add2 extended to 22
String Add2 extended to 222
String Add1 extended to 1
String Add1 extended to 11
String Add2 extended to 2222
String Add2 extended to 22222
No more increments left for threadAdd2
String Add1 extended to 111
String Add1 extended to 1111
String Add1 extended to 11111
No more increments left for threadAdd1
Thread Example - SimpleThread Code

// see - http://docs.oracle.com/javase/6/docs/api/java/lang/Thread.html

class SimpleThread extends Thread {
    String stringofchars;
    char increment;

    public SimpleThread(String str, char inc) {
        super(str);
        stringofchars = "";
        increment = inc;
    }

    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                sleep((int)(Math.random() * 3000));
            } catch (InterruptedException e) {} 
            stringofchars = stringofchars + increment;
            System.out.println("String " + getName() + " extended to " + stringofchars);
        }
        System.out.println("No more increments left for thread" + getName());
    }
}
IllegalThreadStateException

The runtime system throws an `IllegalThreadStateException` when you call a method on a thread and that thread's state does not allow for that method call. (See the state machine diagram in later slides)

So, when you call a thread method that can throw an exception, you must either catch and handle the exception, or specify that the calling method throws the uncaught exception.

The sleep method can also throw an `InterruptedException`, and so we needed a try/catch in the previous code:

```java
try {
    sleep((int)(Math.random() * 3000));
} catch (InterruptedException e) {} 
```
Sharing Thread Problem

The previous example showed how two independent threads execute concurrently.

Threads can also share data/objects and so their concurrent behaviours are inter-dependent.

We wish to change the previous code so that the 2 threads update the same string of characters.

We will do this using a SharedString class
Sharing Thread Problem

Your task is to code the class SharingThread

```java
class SharedString {

public SharedString(){\textit{str} ="";}

public String \textit{str};

public void add (char c){\textit{str} = \textit{str} + c;}

public String toString () {return \textit{str};}
}

public class SharingThreadsTest {

    public static void main (String[] args) {

        SharedString soc = new SharedString();
        new SharingThread("SharingAdda", soc, 'a').start();
        new SharingThread("SharingAddb", soc, 'b').start();
    }
}
```
Sharing Thread Problem

We want the output from this code to produce, typically:

Shared String extended by SharingAddb to b
Shared String extended by SharingAddb to bb
Shared String extended by SharingAdda to bba
Shared String extended by SharingAddb to bbab
Shared String extended by SharingAddb to bbabb
Shared String extended by SharingAdda to bbabba
No more increments left SharingAddb
Shared String extended by SharingAdda to bbabbaba
Shared String extended by SharingAdda to bbabbabaa
Shared String extended by SharingAdda to bbabbabaaa
No more increments left SharingAdda

**TO DO:** Your task is to code the `class SharingThread extends Thread {}` to provide this behaviour
Thread State Machine Model

The `start()` method creates the system resources necessary to run the thread, schedules the thread to run, and calls the thread's `run()` method.

The next state is "Runnable" rather than "Running" because the thread might not actually be running when it is in this state.

[https://bitstechnotes.wordpress.com/2017/12/16/java-thread-state-diagram/](https://bitstechnotes.wordpress.com/2017/12/16/java-thread-state-diagram/)
Threads and Synchronization Issues

Threads can share state (objects)

This is very powerful, and makes for very efficient inter-thread communication

However, it makes two kinds of errors possible:

• thread interference, and
• memory inconsistency.

Java provides a synchronization “tool” in order to avoid these types of errors.
Thread Interference

Interference happens when two operations, running in different threads, but acting on the same data, *interleave*. This means that the two operations consist of multiple steps, and the sequences of steps overlap. Because they are unpredictable, thread interference bugs can be difficult to detect and fix.

Consider a simple class called Counter

```java
class Counter {
    private int c = 0;
    public void increment() {c++;}
    /*
     * Multiple steps of c++
     * 1. Retrieve the current value of c.
     * 2. Increment the retrieved value by 1.
     * 3. Store the incremented value back in c.
     */
    public void decrement() {c--;
    public int value() {return c;
}
```

If a Counter object is referenced from multiple threads, interference between threads may give rise to unexpected behaviour.
Thread Interference

<table>
<thead>
<tr>
<th>Thread A</th>
<th>Thread B</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>17 + 1 = 18</td>
<td>17 + 1 = 18</td>
</tr>
</tbody>
</table>

- Read
- Write

Integer

17

Time

18

18
Memory inconsistency

If the two increment statements had been executed in the same thread, it would be safe to assume that the values written would be “18” and “19”. But, in this example, the values printed out might well be “18” and “18”, because there's no guarantee that thread A's change to counter will be visible to thread B — unless the programmer has established a happens-before relationship between these two statements.

There are several actions that create happens-before relationships. The simplest technique/tool is to use synchronization.
Synchronized methods, example:

```java
public class SynchronizedCounter {
    private int c = 0;
    public synchronized void increment() {c++;
    public synchronized void decrement() {c--;
    public synchronized int value() {return c;
}
```

Two invocations of synchronized methods on the same object cannot interleave. When one thread is executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block (suspend execution) until the first thread is done with the object.

When a synchronized method exits, it automatically establishes a \textit{happens-before} relationship with any subsequent invocation of a synchronized method for the same object. This guarantees that changes to the state of the object are visible to all threads.

Synchronization is effective for keeping systems \textit{safe}, but can present problems with \textit{liveness}
NOTE: Java Constructors cannot be synchronized

http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html

Note that constructors cannot be synchronized — using the synchronized keyword with a constructor is a syntax error. Synchronizing constructors doesn't make sense, because only the thread that creates an object should have access to it while it is being constructed.

Warning: When constructing an object that will be shared between threads, be very careful that a reference to the object does not "leak" prematurely. For example, suppose you want to maintain a List called instances containing every instance of class. You might be tempted to add the following line to your constructor:

```
instances.add(this);
```

But then other threads can use instances to access the object before construction of the object is complete.
Making Concurrent Telephones Using Threads

Step 1: Write a producer thread that produces a random sequence of bits for transmission to a consumer

Step 2: Write a consumer thread that accepts data from a producer and records them

Step 3: Write a Telephone class which implements a producer and a consumer.

Step 4: Simulate a connection between 2 Telephones by connecting them in a bidirectional producer-consumer composition.

TODO - change your POTS design to incorporate independent Telephone threads