Operating Systems and Networks

Lecture 07:
Threads in Java
Threads vs processes
Threads in java
sleep
InterruptedException
Threads vs. processes

Because threads within the same process share resources, there is no expensive context switching

- Changes made by one thread to shared system resources will be seen by all other threads.
- Two pointers having the same value point to the same data.
- Reading/ writing to the same memory locations is possible, and therefore requires explicit synchronization by the programmer.

Your job becomes harder
Defining and starting a thread in Java

Normally, two different objects required. One, the thread itself, knows how to execute code, and the other, the run-object, knows what code to execute.

In Java a thread is an object an instance of Thread. There are two ways to create a new thread:

1) Extend the class Thread
2) Write a class that implement Runnable interface and use it in the Thread constructor.
Extend `java.lang.Thread` and override `run()`

```java
public class MyThread extends Thread{
    public void run(){
        // Write the code for to override run()
        System.out.println("This is my first thread");
    }
}
```
A run-object to test:

```java
public class TestMyThread {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}
```

**Exercise:** Modify `run()` in `MyThread.java` to do something else.
Implementing `Runnable` interface

1. Write a class, implementing `Runnable`, whose `run` method is the code you want to be executed by a thread.

2. Create an instance of that class. This is the run-object.

3. Create an instance of `Thread`, using the run-object as constructor parameter.

4. Call the `start()` method on the thread. This starts the thread executing `run()` on its run-object.
implement the interface and use the class in the Thread constructor.

```java
public class HelloRunnable implements Runnable{
    //implement run method here
    public void run(){
        System.out.println("Thread by implementing Runnable");
    }
}
```
Extending Runnable interface (continue)

run-object to test:

```java
public class TestHelloRunnable {
    public static void main(String[] args) {
        HelloRunnable ht = new HelloRunnable();
        Thread t = new Thread(ht);
        t.start();
    }
}
```

*Exercise:* Modify `run()` in `HelloRunnable.java` to do something else.
Exercises

These are unassessed exercises.

- Check out Threads API.

2) Run `MyThread5Times.java`. Do you see any change in the order of the printed sentences?
Thread.currentThread()

- To identify which thread within the JVM is running the code use the static method

```java
public static native Thread currentThread()
```

Exercise: create a Thread which change its name randomly every 5 second a few times. Write a test to identify which thread is running.

(See currentThreadExample folder for a solution)
Thread terminologies

- After the thread has been created (spawned), but before start has been called on it, the thread is new.
- Between the call of start and termination of run, it is alive. You can use isAlive() to check.
- When run has terminated, the thread is terminated (or dead).
- An alive thread that is able to execute its run method is runnable. (Don't confuse this with the Runnable interface)
Thread terminologies (continue)

- There may be many runnable threads in a JVM, but only one at a time can be actually running (discussion about single processor ...)

- However, there are also various reasons why an alive thread might not be runnable.
  - might have put itself to sleep for a fixed time
  - might be waiting for something to be done by other threads.

Such a thread is suspended.

State of a thread: new, alive, terminated, suspended
Thread.sleep

- The static void method `Thread.sleep(long millis)` suspends the current thread for the given time.
- During that time it is not runnable, but waiting.
- ALWAYS place `Thread.sleep()` in a try/catch. Otherwise, you get a compile error, a checked exception
  `InterruptedException` will be thrown.
Beginners error

- It is tempting to squash the exception by catching it but not handling-

```java
try {
    Thread.sleep(1000);  //sleep 1 second
} catch (InterruptedException e) {
    //Poor coding- must handle exception
}
```

Checked exceptions must be handled

Joshua Bloch: you are not only ignoring the fire alarm, but also turning it off so nobody else knows it rang. …We will return to this later.
Stopping a thread

- The only safe way to stop a thread is for it to stop itself.
- A thread t1 can interrupt running of a thread t2 by invoking its `interrupt()` method:

```java
public void interrupt()
```
- Call `interrupt()` on a thread to signal that it should stop itself. The effect of this is to "set the interrupt status" of the thread, i.e. sets a flag in the destination thread indicating it has been terminated and returns right away.
Example

- Write a thread called `Sleepy` which sleeps for a period of time, but its sleep is interrupted, causing it to throw an `InterruptedException`.

(See the folder “sleepy” for a solution)

Question: Why the elapsed time is 2001 or 2002... instead of 2000?

You can use `isInterrupted()` to check the status:

```java
public boolean isInterrupted()
```
The run method can check the interrupt status by calling Thread.interrupted().

```java
public static boolean interrupted()
```

This returns a boolean result to say whether or not the interrupt status (of the currently executing thread) is set.

At the same time, it also resets the interrupt status so calling Thread.interrupted() a second time will return false (unless there has meanwhile been yet another call of interrupt()).
Example: Thread.Interrupted()

Write some code to experiment with interrupting a current thread.
(see interruptedExample folder)

Question: We have interrupted the thread at step 2. Why we get false at the step 3?
Why sleep throw this exception?

- To make sure the thread knows if interrupt() is called on it.
- If a thread is asleep, it cannot be calling Thread.interrupted() to check the interrupt status (because is asleep!). Therefore
  1) If the current thread has its interrupt status set when it tries to call Thread.sleep(), then it is not allowed to go to sleep - it stays runnable.
2) If a thread is asleep when `interrupt()` is called on it, then it is immediately woken up - it becomes runnable again.

- In each case, an `InterruptedException` is thrown and the interrupt status is reset. (It does not need to be set any more, because the `InterruptedException` means the thread now knows about the interrupt.)
To sum up

- A thread is *interrupted* if some other thread calls `interrupt()` on it.
- Normally, when a thread is interrupted it should try to stop itself (by finishing its run method).
- There are two ways a thread can find out it has been interrupted: either by calling `interrupted()`, or by getting an `InterruptedException`.