Lecture 02: Thread API

SSC2
Behzad Bordbar
School of Computer Science, University of Birmingham, UK

Recap
Concurrent programming
Thread vs. Processes
Advantages of using threads
Creating threads in Java:
Extending Thread, implementing Runnable
Who has done the na-Exercises?

Contents
- Extending Thread or implementing Runnable
- Thread terminology
- Stopping Threads
- What is the current running thread?
- Stopping threads

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.

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.

But, I haven’t written a method start(). What is going on?
The main() calls t.start().
Because of inheritance start() method of Thread is invoked, which will cause execution of the run(), in your create class, i.e. the method which is overridden in your code.

NEVER CALL run() DIRECTLY.
t.run(); <<< Don't write
Extending Thread or implementing Runnable

There are two general points to mention:
1) Subclassing Thread works only if the class is not extending any other class
2) If extending Runnable you can not have statements within the Runnable implementation of the run().
Because there is no thread in the Runnable instance

Thread.currentThread()

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

  public static native Thread currentThread()

- Example: 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 public final native boolean isAlive();
- 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 lecture01)
- 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-
  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 2003 instead of 2000?

  You can use isInterrupted() to check the status:

  ```java
  public boolean isInterrupted()
  ```

Stopping a thread (continue)

- 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 do we get false at the step 3?

Contents

- Extending Thread or implementing Runnable
- Thread terminology
- Stopping Threads
- Sleep