Hanoi University
Of Technology

Samsung Training Program

Working with Threads and Timers

Pham Van Tien

Contents

- Thread and multithreading concepts
- Usage of threads in Android
- Background threads and UI thread
- Communications: message and runnable objects
- Synchronization
- Timers

Introduction (1/2)

- A thread is a lightweight process, usually being contained in a thread. It represents a thread of execution.
- Thread is a smallest sequence of programmed instructions that can be managed independently by OS scheduler
- Concurrent operations in a program need multiple threading
  - ***Example: a huge movie is being downloaded while the downloaded part is playing back***

Introduction (2/2)

- **In any application, main program should be quickly responsive, whereas slow threads should run in background**
- **Android inherits class Thread and** java.util.concurrent **package from Java to realize multiple threading**

In android, when needed?

- **A multithreading program contain an UI (main) thread and background threads**
- **What background threads are associated with might be:**
  - Communication
  - DB actions
  - FileSystem actions
  - Long running algorithms/calculations

Thread usage

- **Define operations of thread:**

```
private  void  doSomeLongWork ( final  int  incr ) {
  SystemClock . sleep ( 250 );    // should be
   something more useful!
}
```

- **Call thread**

```
private  Runnable  longTask = new  Runnable ()  {
   public  void  run ()  {
     for  ( int  i = 0 ; i < 20 ; i ++)  {
       doSomeLongWork ( 500 );
     }
   }
} ;
```

Handler

- A handler object is created, bound to a background thread for communication with UI thread
- There are two means for communicating with a handler:
  - Messages
  - Runnable object

Message (1/2)

- **Messages are used to communicate between background threads and handlers**
- **Use** sendMessage() **or its variants to send a message to handler**
- **To prepare a message for sending, call** obtainMessage() **to populate it from a pool**

Message (2/2)

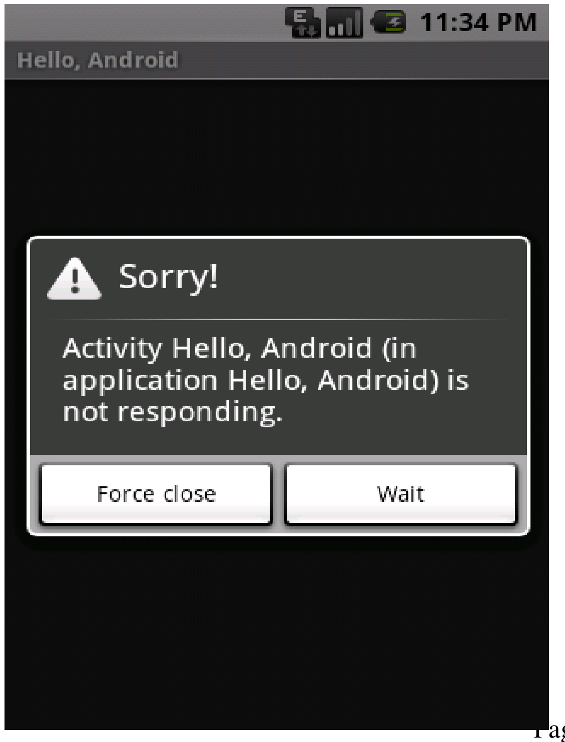
- **Then, the handler processes the message according to what** handleMessage() **implements**
- **Further references are available in Android package** android.os.Handler

Runnable object

- **Can be used as an alternative to messages for communicating with handler**
- **Employs** post(), postAtTime() **and** postDelayed() **methods to pass runnable objects in for actual processing. This is similar to post method in HTTP**

AsyncTask

- It makes background operations transparent to user [who currently interacts with the UI thread]
- It helps alleviate ANR ("Application Not Responding")



Sorry!

Activity Hello, Android (in application Hello, Android) is not responding.

Usage of AsyncTask

To use AsyncTask, you must:

- **Create a subclass of AsyncTask, commonly a private inner class of something that uses the task (e.g. an activity)**
- **Override one or more methods of AsyncTask to perform the background work, plus work associated with the task that needs to be done on the UI thread (e.g. update progress bar)**
- **When needed, create an instance of AsyncTask and** execute() **to have it begin work**

Methods of AsyncTask

There are four methods that can be overridden:

- onPreExecute(): **invoked on the UI thread immediately after the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.**
- doInBackground(): **it can run as long as needed but no infinite loop inside**

Methods of AsyncTask

There are four methods that can be overridden:

- onProgressUpdate(Progress...)**: invoked on the UI thread after a call to** publishProgress(Progress...)**. It displays any form of progress in the user interface while the background computation is still executing.**
- onPostExecute(Result)**, invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.**

Synchronization in Java Multithreading

- Semaphore **is a classic concurrency tool.**
- CountDownLatch **is a very simple yet very common utility for blocking until a given number of signals, events, or conditions hold.**
- CyclicBarrier **is a resettable multiway synchronization point useful in some styles of parallel programming.**
- Exchanger **allows two threads to exchange objects at a rendezvous point, and is useful in several pipeline designs.**

Xml: fork thread

Fork a thread from menu resource:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
   xmlns:android="http://schemas.android.com/apk/res/andro
   id">
 <item android:id="@+id/toast"
   android:title="Raise Toast"
   android:icon="@drawable/toast"
 />
 <item android:id="@+id/run"
   android:title="Run Long Task"
   android:icon="@drawable/run"
 />
</menu>
```

Timer (Java)

- **Android inherits timers defined in** java.util.Timer **package**
- **Commonly used methods include:**
  - void Schedule()**: schedules a task at a specific time**
  - void scheduleAtFixedRate()**: schedule a periodic task**

Timer (android-native)

- **Android also provides count-down timer, defined in package** android.os.CountDownTimer
- **It schedules a countdown until a time in the future, with regular notifications on intervals along the way**

Timer (android-native)

```
new CountDownTimer(30000, 1000) {
   public void onTick(long millisUntilFinished) {
      mTextField.setText("seconds remaining: " +
 millisUntilFinished / 1000);
   }
   public void onFinish() {
      mTextField.setText("done!");
   }
 }.start();
```

Exercises

- Tutorial 8: Sitting in the Background
- Tutorial 9: Life and Times. Thread: pause and resume