Lesson 13

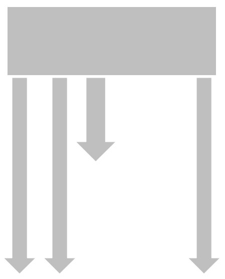
Android Multi-Threading

Victor Matos
Cleveland State University

Notes are based on:
Android Developers
http://developer.android.com/index.html

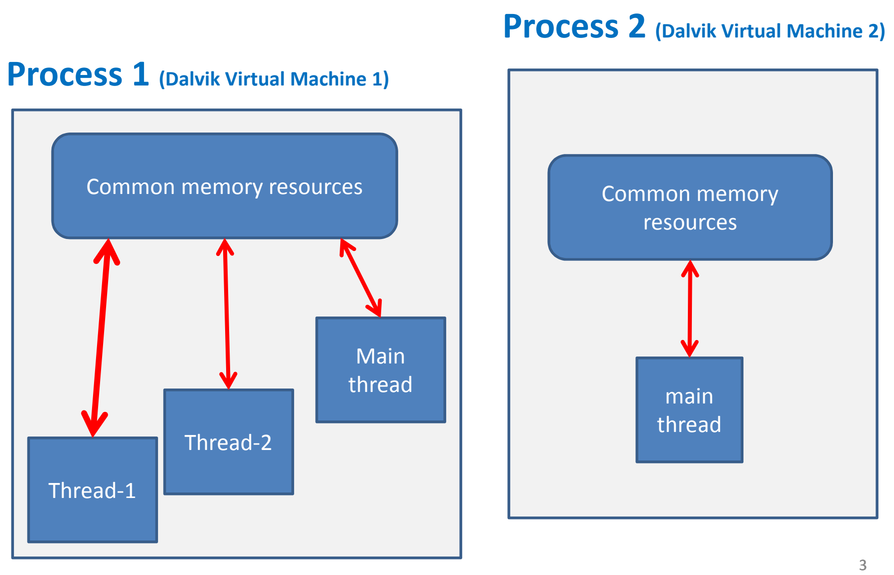
Multi-Threading

Threads
http://developer.android.com/reference/java/lang/Thread.html



1. A Thread is a **concurrent** unit of execution.
2. Each thread has its own call **stack**. The call stack is used on method calling, parameter passing, and storage for the called method's local variables.
3. Each virtual machine instance has at least one **main thread** .
4. Threads in the same VM interact and synchronize by the use of **shared objects** and **monitors** associated with these objects.

Multi-Threading



Multi-Threading

Threads http://developer.android.com/reference/java/lang/Thread.html

There are basically two main ways of having a **Thread** execute application code.

1. Create a new class that *extends* **Thread** and override its **run()** method.

```
MyThread t = new MyThread();
t.start();
```

2. Create a new **Thread** instance passing to it a **Runnable** object.

```
Runnable myRunnable1 = new MyRunnableClass();
Thread t1 = new Thread(myRunnable1);
t1.start();
```

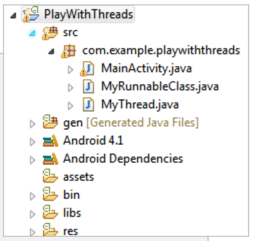
In both cases, the **start()** method must be called to actually execute the new Thread.

Multi-Threading

Threads http://developer.android.com/reference/java/lang/Thread.html

Example1. Creating two threads using different programming styles.

Main Thread


PlayWithThreads
src
com.example.playwiththreads
MainActivity.java
MyRunnableClass.java
MyThread.java
gen [Generated Java Files]
Android 4.1
Android Dependencies
assets
bin
libs
res

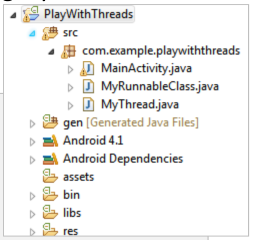
```
public class MainActivity extends Activity {
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Runnable myRunnable1 = new MyRunnableClass();
        Thread t1 = new Thread(myRunnable1);
        t1.start();

        MyThread t = new MyThread();
        t.start();

    }//onCreate
```

Multi-Threading

Threads http://developer.android.com/reference/java/lang/Thread.html

Example1. Creating two threads using different programming styles.

MyRunnable Class


PlayWithThreads
src
com.example.playwiththreads
MainActivity.java
MyRunnableClass.java
MyThread.java
gen [Generated Java Files]
Android 4.1
Android Dependencies
assets
bin
libs
res

```
public class MyRunnableClass implements Runnable {
   @Override
   public void run() {
      try {
         for (int i = 100; i < 105; i++){
            Thread.sleep(1000);
            Log.e("<<runnable>>", “runnable talking: " + i);
         }
      } catch (InterruptedException e) {
         e.printStackTrace();
      }

   }//run

}//class
```

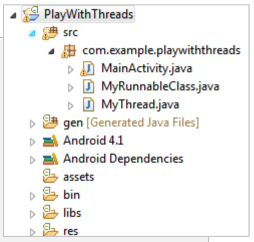
Multi-Threading

Threads http://developer.android.com/reference/java/lang/Thread.html

Example1. Creating two threads using different programming styles.

MyThread Class

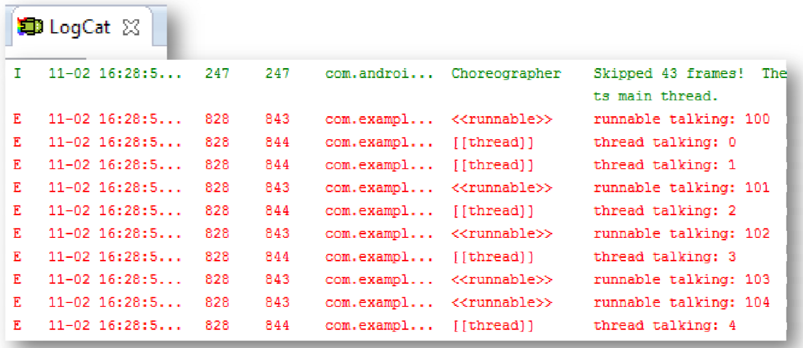
```
public class MyThread extends Thread{

  @Override
  public void run() {
    super.run();
    try {
      for(int i=0; i<5; i++){
        Thread.sleep(1000);
        Log.e("[[thread]]", "Thread talking: " + i);
      }
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }//run
}//class
```



Multi-Threading

Threads http://developer.android.com/reference/java/lang/Thread.html

Example1. Creating two threads using different programming styles.



```
I  11-02 16:28:5...  247   247   com.androi...  Choreographer   Skipped 43 frames!  The
                                                                 ts main thread.
E  11-02 16:28:5...  828   843   com.exampl...  <<runnable>>    runnable talking: 100
E  11-02 16:28:5...  828   844   com.exampl...  [[thread]]      thread talking: 0
E  11-02 16:28:5...  828   844   com.exampl...  [[thread]]      thread talking: 1
E  11-02 16:28:5...  828   843   com.exampl...  <<runnable>>    runnable talking: 101
E  11-02 16:28:5...  828   844   com.exampl...  [[thread]]      thread talking: 2
E  11-02 16:28:5...  828   843   com.exampl...  <<runnable>>    runnable talking: 102
E  11-02 16:28:5...  828   844   com.exampl...  [[thread]]      thread talking: 3
E  11-02 16:28:5...  828   843   com.exampl...  <<runnable>>    runnable talking: 103
E  11-02 16:28:5...  828   843   com.exampl...  <<runnable>>    runnable talking: 104
E  11-02 16:28:5...  828   844   com.exampl...  [[thread]]      thread talking: 4
```

Interleaved execution

Multi-Threading

Advantages of Multi-Threading

1. Threads **share** the process' **resources** but are able to execute independently.

2. Applications **responsibilities** can be **separated**
   - main thread runs UI, and
   - slow tasks are sent to background threads.

3. Threading provides an useful abstraction of **concurrent** execution.

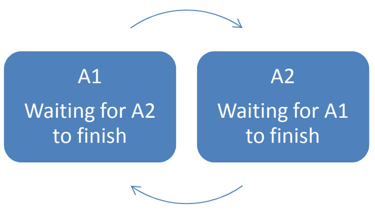
4. A multithreaded program operates ***faster*** on computer systems that have ***multiple CPU**s.*

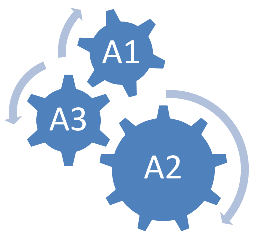
Multi-Threading

Disadvantages of Multi-Threading

1. Code tends to be more **complex**

2. Need to detect, avoid, resolve **deadlocks**





Multi-Threading

Android's Approach to Slow Activities

Problem: An application may involve a time-consuming operation.
Goal: We want the **UI** to be responsive to the user in spite of heavy load.
Solution: Android offers two ways for dealing with this scenario:

1. Do expensive operations in a background ***service***, using *notifications* to inform users about next step
2. Do the slow work in a ***background thread***.

Using Threads: Interaction between Android threads is accomplished using
(a) a main thread ***Handler*** object and
(b) posting ***Runnable*** objects to the main view.

Multi-Threading

Handler Class

http://developer.android.com/reference/android/os/Handler.html

- The ***main thread*** runs a ***message queue*** that takes care of managing the interaction between top-level application objects (activities, intent receivers, etc) and any windows they create.
- You can create your own **secondary threads**, and communicate back with the main application thread through a user-defined **Handler**.
- *Your new Handler is bound to the message queue of the thread in which it is created.*
- The Handler will deliver *messages* and *runnables* to that message queue and execute them as they come out of the message queue.

Multi-Threading

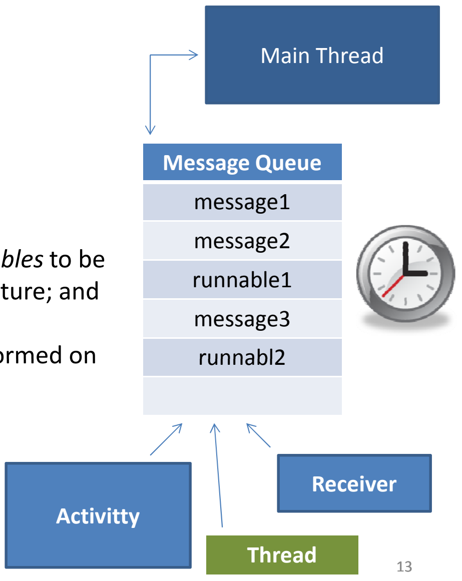
Handler Class

http://developer.android.com/reference/android/os/Handler.html

There are two main uses for a **Handler**:

(1) to schedule *messages* and *runnables* to be executed as some point in the future; and

(2) to enqueue an action to be performed on another thread



Multi-Threading

Threads and UI

Warning

Background threads are not allowed to interact with the UI.

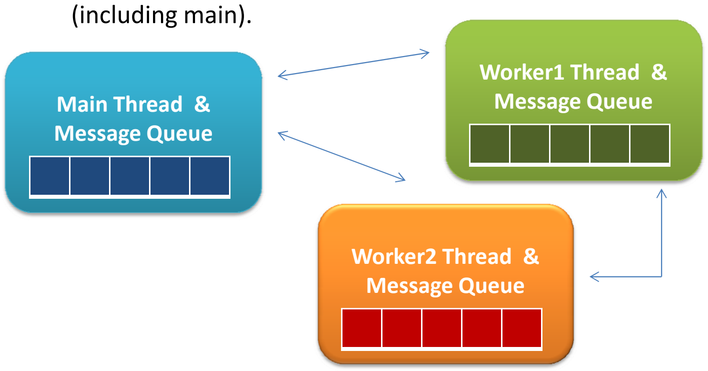
- Only the main process can access the (main) activity's view.
- (Global) class variables can be seen and updated in the threads

Multi-Threading

Observation:

- Typically the main UI thread sets a handler to get messages from the worker threads; however *each worker thread could also define its own handler*.

- A handler in the worker thread creates a local message-queue which could be used to receive messages from other threads (including main).


Main Thread & Message Queue
Worker1 Thread & Message Queue
Worker2 Thread & Message Queue

Multi-Threading

Handler's MessageQueue

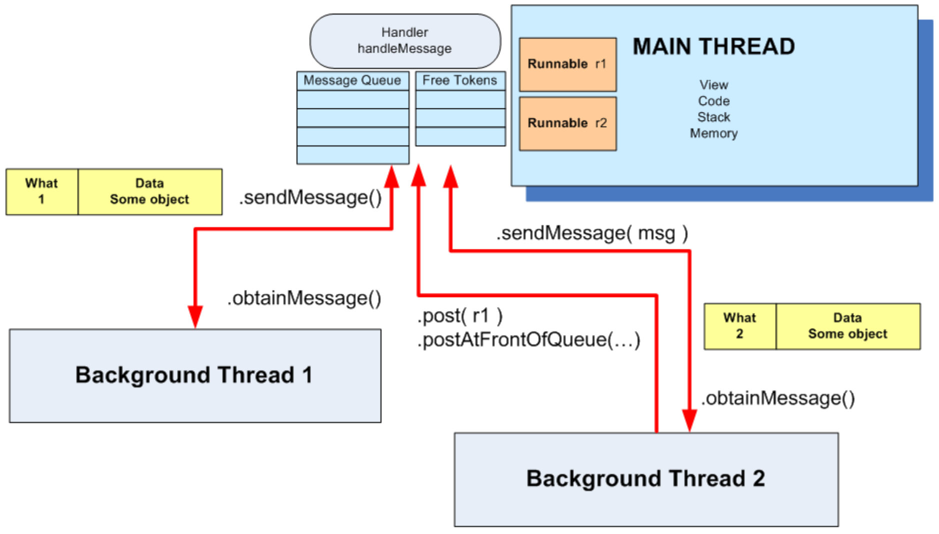
1. A secondary thread that wants to communicate with the main thread must request a message **token** using the *obtainMessage*() method.

2. Once obtained, the background thread can fill data into the message **token** and attach it to the Handler's **message queue** using the *sendMessage*() method.

3. The Handler uses the *handleMessage*() method to continuously attend new messages arriving to the main thread.

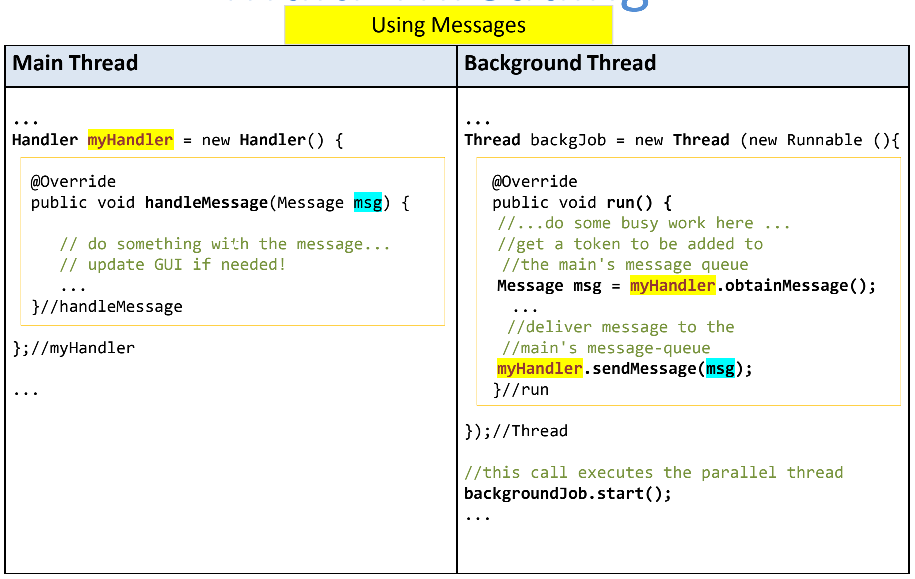
4. A **message** extracted from the process' queue can either return some **data** to the main process or request the **execution** of runnable objects through the *post()* method.

Multi-Threading



17

Multi-Threading



Using Messages

Main Thread	Background Thread

Main Thread

```
...
Handler myHandler = new Handler() {

   @Override
   public void handleMessage(Message msg) {

      // do something with the message...
      // update GUI if needed!
      ...
   }//handleMessage

};//myHandler

...
```

Background Thread

```
...
Thread backgJob = new Thread (new Runnable (){

   @Override
   public void run() {
    //...do some busy work here ...
    //get a token to be added to
    //the main's message queue
    Message msg = myHandler.obtainMessage();
     ...
     //deliver message to the
    //main's message-queue
    myHandler.sendMessage(msg);
   }//run

});//Thread

//this call executes the parallel thread
backgroundJob.start();
...
```

18

Multi-Threading

Using Post

Main Thread

```
...
Handler     myHandler = new Handler();
@Override
public void onCreate(
            Bundle savedInstanceState) {
 ...
 Thread myThread1 =
        new Thread(backgroundTask,
                   "backAlias1");
 myThread1.start();

}//onCreate

...
//this is the foreground runnable
private Runnable foregroundTask
   = new Runnable() {
  @Override
  public void run() {
   // work on the UI if needed
}
...
```

Background Thread

```
// this is the "Runnable" object
// that executes the background thread

private Runnable backgroundTask
                 = new Runnable () {
  @Override
  public void run() {
    ... Do some background work here
    myHandler.post(foregroundTask);

  }//run

};//backgroundTask
```

Multi-Threading

Messages

To send a Message to a Handler, the thread must first invoke obtainMessage() to get the Message object out of the pool.

There are a few forms of **obtainMessage()**, allowing you to just create an empty Message object, or messages holding arguments

Example

```
//  thread 1 produces some local data
String localData = “Greetings from thread 1”;
//  thread 1 requests a message & adds localData to it
Message mgs = myHandler.obtainMessage (1, localData);
```

Multi-Threading

sendMessage Methods

You deliver the message using one of the **sendMessage...()** family of methods, such as …

- **sendMessage**() puts the message at the end of the queue immediately

- **sendMessageAtFrontOfQueue**() puts the message at the front of the queue
immediately (versus the back, as is the default), so your message takes priority over all others

- **sendMessageAtTime**() puts the message on the queue at the stated time, expressed in the form of milliseconds based on system uptime (SystemClock.uptimeMillis())

- **sendMessageDelayed**() puts the message on the queue after a delay, expressed in milliseconds

Multi-Threading

Processing Messages

To process messages sent by the background threads, your Handler needs to implement the listener

handleMessage(Message msg **)**

which will be called with each message (msg) that appears on the message queue.

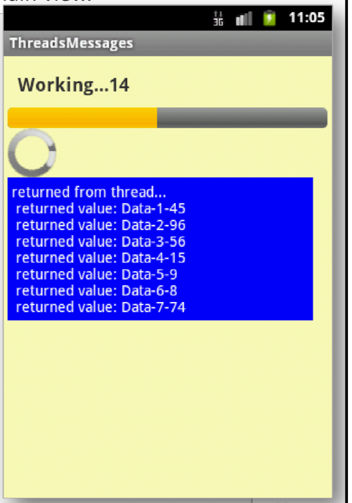
There, the handler can update the UI as needed. However, it should still do that work quickly, as other UI work is suspended until the Handler is done.

Multi-Threading

Example 2. Progress Bar – Using Message Passing Layout 1/2

The main thread displays a horizontal and a circular *progress bar widget* showing the progress of a slow background operation. Some random data is periodically sent from the background thread and the messages are displayed in the main view.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#44ffff00"
    android:orientation="vertical"
    android:padding="4dp" >

    <TextView
        android:id="@+id/txtWorkProgress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="Working ...."
        android:textSize="18sp"
        android:textStyle="bold" />

    <ProgressBar
        android:id="@+id/progress1"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <ProgressBar
        android:id="@+id/progress2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```



Working...14

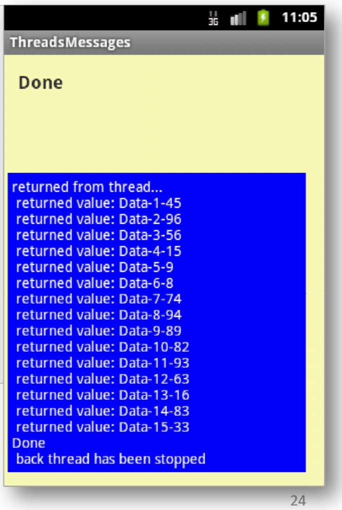
```
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
```

Multi-Threading

Example 2. Progress Bar – Using Message Passing Layout 2/2

The main thread displays a horizontal and a circular *progress bar widget* showing the progress of a slow background operation. Some random data is periodically sent from the background thread and the messages are displayed in the main view.

```
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/txtReturnedValues"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="7dp"
            android:background="#ff0000ff"
            android:padding="4dp "
            android:text="returned from thread..."
            android:textColor="@android:color/white"
            android:textSize="14sp" />
    </ScrollView>

</LinearLayout>
```



Done

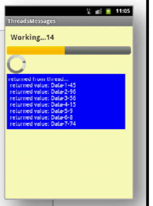
```
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
 returned value: Data-8-94
 returned value: Data-9-89
 returned value: Data-10-82
 returned value: Data-11-93
 returned value: Data-12-63
 returned value: Data-13-16
 returned value: Data-14-83
 returned value: Data-15-33
Done
 back thread has been stopped
```

Multi-Threading

Example 2. Progress Bar – Using Message Passing

Activity 1/5

The main thread displays a horizontal and a circular *progress bar widget* showing the progress of a slow background operation. Some random data is periodically sent from the background thread and the messages are displayed in the main view.


```
11:05
ThreadsMessages
Working...14
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
```

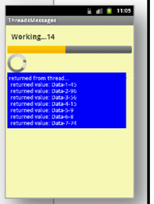
```
public class ThreadsMessages extends Activity {

   ProgressBar bar1;
   ProgressBar bar2;

   TextView msgWorking;
   TextView msgReturned;

   // this is a control var used by backg. threads
   boolean isRunning = false;

   // lifetime (in seconds) for background thread
   final int MAX_SEC = 30;

   //String globalStrTest = "global value seen by all threads ";
   int globalIntTest = 0;
```

Multi-Threading

Example 2. Progress Bar – Using Message Passing

Activity 2/5

```
   Handler handler = new Handler() {
      @Override
      public void handleMessage(Message msg) {

         String returnedValue = (String)msg.obj;

         //do something with the value sent by the background thread here
         msgReturned.append("\n returned value: " + returnedValue );

         bar1.incrementProgressBy(2);

         //testing early termination
         if (bar1.getProgress() == MAX_SEC){
            msgReturned.append(" \nDone \n back thread has been stopped");
            isRunning = false;
         }

         if (bar1.getProgress() == bar1.getMax()){
            msgWorking.setText("Done");
            bar1.setVisibility(View.INVISIBLE);
            bar2.setVisibility(View.INVISIBLE);
         }
         else {
            msgWorking.setText("Working..." + bar1.getProgress() );
         }
      }
   }; //handler
```

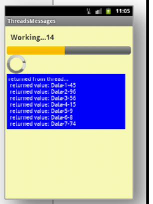

```
11:05
ThreadsMessages
Working...14
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
```

Multi-Threading

Example 2. Progress Bar – Using Message Passing

Activity 3/5

```
@Override
public void onCreate(Bundle icicle) {
   super.onCreate(icicle);
   setContentView(R.layout.main);

   bar1 = (ProgressBar) findViewById(R.id.progress1);
   bar1.setProgress(0);
   bar1.setMax(MAX_SEC);

   bar2 = (ProgressBar) findViewById(R.id.progress2);

   msgWorking = (TextView)findViewById(R.id.txtWorkProgress);
   msgReturned = (TextView)findViewById(R.id.txtReturnedValues);

   //globalStrTest += "XXX"; // slightly change the global string
   globalIntTest = 1;

}//onCreate
```



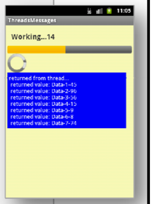
```
Working...14
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
```

Multi-Threading

Example 2. Progress Bar – Using Message Passing

Activity 4/5

```
public void onStart() {
   super.onStart();
   // this code creates the background activity where busy work is done
   Thread background = new Thread(new Runnable() {
      public void run() {
         try {
            for (int i = 0; i < MAX_SEC && isRunning; i++) {
               //try a Toast method here (it will not work!)
               //fake busy busy work here
               Thread.sleep(1000);  //one second at a time

               // this is a locally generated value between 0-100
               Random rnd = new Random();
               int localData = (int) rnd.nextInt(101);
               //we can see and change (global) class variables
               String data = "Data-" + globalIntTest + "-" + localData;
               globalIntTest++;
               //request a message token and put some data in it
               Message msg = handler.obtainMessage(1, (String)data);

               // if thread is still alive send the message
               if (isRunning) {
                  handler.sendMessage(msg);
               }
            }
         }
```



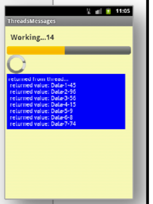
```
Working...14
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
```

Multi-Threading

Example 2. Progress Bar – Using Message Passing

Activity 5/5

```
 catch (Throwable t) {
           // just end the background thread
           isRunning = false;
           }
        }
     });// Tread

     isRunning = true;
     background.start();

  }//onStart


  public void onStop() {
     super.onStop();
     isRunning = false;
  }//onStop
}//class
```



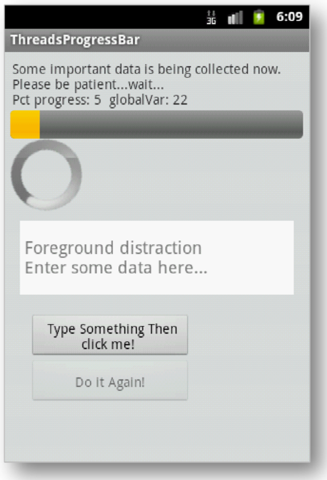
Working...14

```
returned from thread...
 returned value: Data-1-45
 returned value: Data-2-96
 returned value: Data-3-56
 returned value: Data-4-15
 returned value: Data-5-9
 returned value: Data-6-8
 returned value: Data-7-74
```

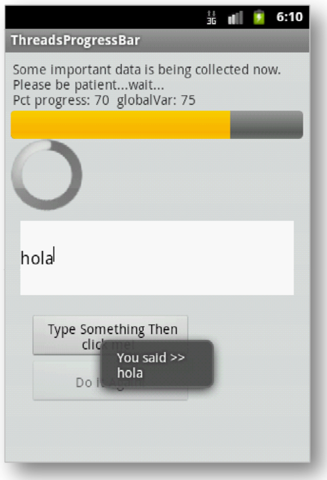
Multi-Threading

Example 3. Using Handler post(...) Method

We will try the same problem presented earlier (a slow background task and a responsive foreground UI) this time using the **posting mechanism** to execute foreground ***runnables***.



Some important data is being collected now.
Please be patient...wait...
Pct progress: 5 globalVar: 22



Some important data is being collected now.
Please be patient...wait...
Pct progress: 70 globalVar: 75

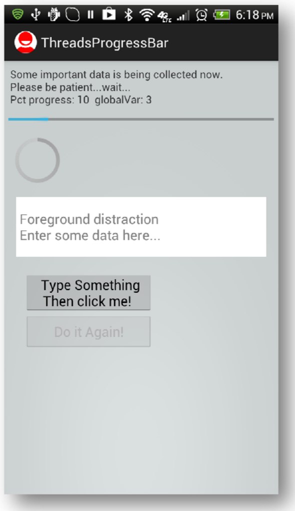
You said >> hola

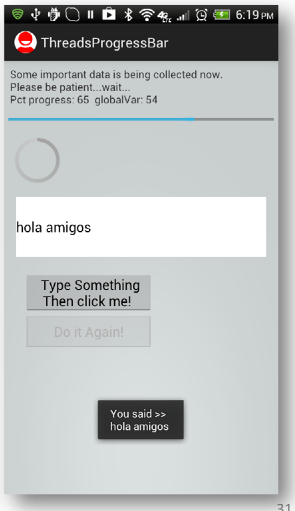
Images obtained from a GingerBread based emulator

Multi-Threading

Example 3. Using Handler post(...) Method

We will try the same problem presented earlier (a slow background task and a responsive foreground UI) this time using the **posting mechanism** to execute foreground ***runnables***.

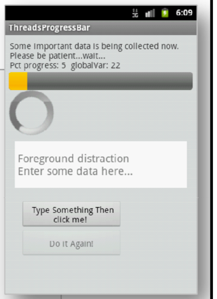

6:18 PM
ThreadsProgressBar
Some important data is being collected now.
Please be patient...wait...
Pct progress: 10 globalVar: 3
Foreground distraction
Enter some data here...
Type Something Then click me!
Do it Again!


6:19 PM
ThreadsProgressBar
Some important data is being collected now.
Please be patient...wait...
Pct progress: 65 globalVar: 54
hola amigos
Type Something Then click me!
Do it Again!
You said >> hola amigos

Images obtained from an IceCream 4.x based device

Multi-Threading

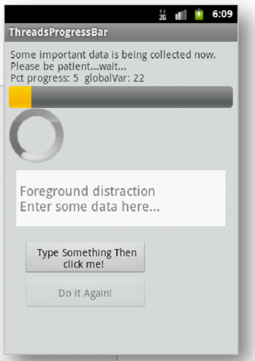
Example 3. Using post - layout: main.xml


6:09
ThreadsProgressBar
Some important data is being collected now.
Please be patient...wait...
Pct progress: 5 globalVar: 22
Foreground distraction
Enter some data here...
Type Something Then click me!
Do it Again!

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#22002222"
    android:orientation="vertical"
    android:padding="6dp" >

    <TextView
        android:id="@+id/lblTopCaption"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="2dp"
        android:text=
            "Some important data is been collected now. Patience please..." />

    <ProgressBar
        android:id="@+id/myBarHor"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="30dp" />

    <ProgressBar
        android:id="@+id/myBarCir"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

Multi-Threading

Example 3. Using post - layout: main.xml


ThreadsProgressBar

Some important data is being collected now.
Please be patient...wait...
Pct progress: 5 globalVar: 22

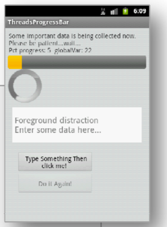
Foreground distraction
Enter some data here...

```
    <EditText
        android:id="@+id/txtBox1"
        android:layout_width="match_parent"
        android:layout_height="78dp"
        android:layout_margin="10dp"
        android:background="#ffffffff"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnDoSomething"
        android:layout_width="170dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:padding="4dp"
        android:text=" Type Something Then click me! " />

    <Button
        android:id="@+id/btnDoItAgain"
        android:layout_width="170dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:padding="4dp"
        android:text=" Do it Again! " />

</LinearLayout>
```

33

Multi-Threading

Example 3. Using post - Main Activity 1/5


ThreadsProgressBar

Some important data is being collected now.
Please be patient...wait...
Pct progress: 5 globalVar: 22

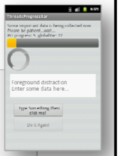
Foreground distraction
Enter some data here...

```
public class ThreadsPosting extends Activity {
   ProgressBar myBarHorizontal;
   ProgressBar myBarCircular;

   TextView lblTopCaption;
   EditText txtDataBox;
   Button btnDoSomething;
   Button btnDoItAgain;
   int progressStep = 5;

   int globalVar = 0;
   int accum = 0;

   long startingMills = System.currentTimeMillis();
   boolean isRunning = false;
   String PATIENCE = "Some important data is being collected now. "
         + "\nPlease be patient...wait... ";

   Handler myHandler = new Handler();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      lblTopCaption = (TextView) indViewById(R.id.lblTopCaption);
```

34

Multi-Threading

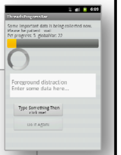
Example 3. Using post - Main Activity 2/5



```
   myBarHorizontal = (ProgressBar) findViewById(R.id.myBarHor);
   myBarCircular = (ProgressBar) findViewById(R.id.myBarCir);

   txtDataBox = (EditText) findViewById(R.id.txtBox1);
   txtDataBox.setHint(" Foreground distraction\n Enter some data here...");

   btnDoItAgain = (Button) findViewById(R.id.btnDoItAgain);
   btnDoItAgain.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
         onStart();
      }// onClick
   });// setOnClickListener

   btnDoSomething = (Button) findViewById(R.id.btnDoSomething);
   btnDoSomething.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
         Editable text = txtDataBox.getText();
         Toast.makeText(getBaseContext(), "You said >> \n" + text, 1).show();
      }// onClick
   });// setOnClickListener

}// onCreate
```

Multi-Threading

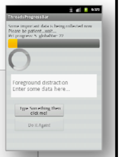
Example 3. Using post - Main Activity 3/5



```
@Override
protected void onStart() {
   super.onStart();
   // prepare UI components
   txtDataBox.setText("");
   btnDoItAgain.setEnabled(false);

   // reset and show progress bars
   accum = 0;
   myBarHorizontal.setMax(100);
   myBarHorizontal.setProgress(0);
   myBarHorizontal.setVisibility(View.VISIBLE);
   myBarCircular.setVisibility(View.VISIBLE);

   // create background thread were the busy work will be done
   Thread myBackgroundThread = new Thread( backgroundTask, "backAlias1" );
   myBackgroundThread.start();

}
```

Multi-Threading

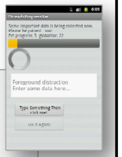
Example 3. Using post - Main Activity 3/5



```
@Override
protected void onStart() {
   super.onStart();
   // prepare UI components
   txtDataBox.setText("");
   btnDoItAgain.setEnabled(false);

   // reset and show progress bars
   accum = 0;
   myBarHorizontal.setMax(100);
   myBarHorizontal.setProgress(0);
   myBarHorizontal.setVisibility(View.VISIBLE);
   myBarCircular.setVisibility(View.VISIBLE);

   // create background thread were the busy work will be done
   Thread myBackgroundThread = new Thread( backgroundTask, "backAlias1" );
   myBackgroundThread.start();

}
```

Multi-Threading

Example 3. Using post - Main Activity 4/5



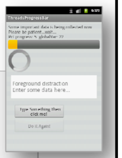
```
// FOREGROUND
// this foreground Runnable works on behave of the background thread
// updating the main UI which is unreachable to it
private Runnable foregroundRunnable = new Runnable() {
   @Override
   public void run() {
      try {
         // update UI, observe globalVar is changed in back thread
         lblTopCaption.setText( PATIENCE
                     + "\nPct progress: " + accum
                     + "  globalVar: " + globalVar);

         // advance ProgressBar
         myBarHorizontal.incrementProgressBy(progressStep);
         accum += progressStep;

         // are we done yet?
         if (accum >= myBarHorizontal.getMax()) {
            lblTopCaption.setText("Background work is OVER!");
            myBarHorizontal.setVisibility(View.INVISIBLE);
            myBarCircular.setVisibility(View.INVISIBLE);
            btnDoItAgain.setEnabled(true);
         }
      } catch (Exception e) {
         Log.e("<<foregroundTask>>", e.getMessage());
      }
   }
}; // foregroundTask
```

Foreground runnable is defined but not started !

Background thread will requests its execution later

Multi-Threading

Example 3. Using post - Main Activity 5/5

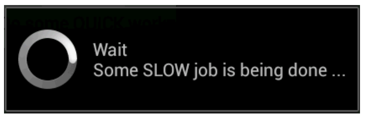


```
   // BACKGROUND
   // this is the back runnable that executes the slow work

   private Runnable backgroundTask = new Runnable() {
      @Override
      public void run() {
         // busy work goes here...
         try {
            for (int n = 0; n < 20; n++) {
               // this simulates 1 sec. of busy activity
               Thread.sleep(1000);
               // change a global variable from here...
               globalVar++;
               // try: next two UI operations should NOT work
               // Toast.makeText(getApplication(), "Hi ", 1).show();
               // txtDataBox.setText("Hi ");

               // wake up foregroundRunnable delegate to speak for you
               myHandler.post(foregroundRunnable);
            }
         } catch (InterruptedException e) {
            Log.e("<<foregroundTask>>", e.getMessage());
         }

      }// run
   };// backgroundTask

}// ThreadsPosting
```

Tell foreground runnable to do something for us...

Multi-Threading

Using the AsyncTask class



Wait
Some SLOW job is being done ...

1. The **AsyncTask** class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.
2. An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.
3. An asynchronous task is defined by

3 Generic Types	4 Main States	1 Auxiliary Method
Params, Progress, Result	**onPreExecute, doInBackground, onProgressUpdate onPostExecute.**	**publishProgress**

Multi-Threading

```
AsyncTask <Params, Progress, Result>
```

AsyncTask's generic types	
Params:	the type of the input parameters sent to the task at execution.
Progress:	the type of the progress units published during the background computation.
Result:	the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type **Void**

Note:
Syntax “**String ...**” indicates (Varargs) array of String values, similar to “**String[]**”

Multi-Threading

Using the AsyncTask class

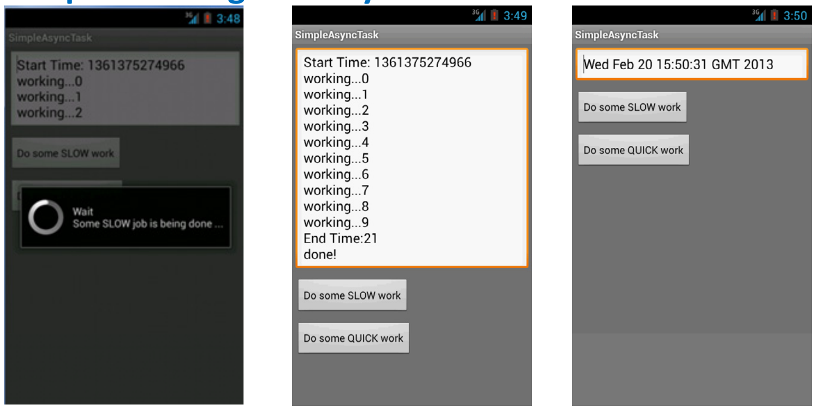
```
private class VerySlowTask extends AsyncTask<String, Long, Void> {

1       // Begin - can use UI thread here
        protected void onPreExecute() {

        }

        // this is the SLOW background thread taking care of heavy tasks
        // cannot directly change UI
2       protected Void doInBackground(final String... args) {
        ... publishProgress((Long) someLongValue);
        }

        // periodic updates - it is OK to change UI
        @Override
3       protected void onProgressUpdate(Long... value) {

        }

        // End - can use UI thread here
4       protected void onPostExecute(final Void unused) {

        }
}
```

Multi-Threading

AsyncTask's methods
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(Params...), invoked on the background thread immediately after *onPreExecute*() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use *publishProgress(Progress...)* to publish one or more units of progress. These values are published on the UI thread, in the *onProgressUpdate(Progress...)* step.
onProgressUpdate(Progress...), invoked on the UI thread after a call to *publishProgress(Progress...)*. The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
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.

Multi-Threading

Example 4: Using the AsyncTask class



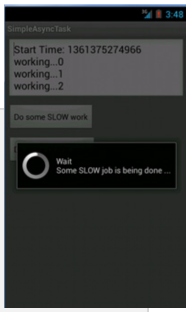
```
Start Time: 1361375274966
working...0
working...1
working...2
working...3
working...4
working...5
working...6
working...7
working...8
working...9
End Time:21
done!
```

```
Wed Feb 20 15:50:31 GMT 2013
```

The main task invokes an AsyncTask to do some slow job. The AsyncTask methods do the required computation and periodically update the main's UI. In our the example the background activity negotiates the writing of the lines in the text box, and also controls the circular progress bar.

Multi-Threading

Example 4: Using the AsyncTask class



SimpleAsyncTask

Start Time: 1361375274966
working...0
working...1
working...2

```java
public class Main extends Activity {
Button btnSlowWork;
Button btnQuickWork;
EditText txtMsg;
Long  startingMillis;

@Override
public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     txtMsg = (EditText) findViewById(R.id.EditText01);

     // slow work...for example: delete all data from a database
     btnSlowWork = (Button) findViewById(R.id.Button01);
     this.btnSlowWork.setOnClickListener(new OnClickListener() {
          public void onClick(final View v) {
               new VerySlowTask().execute();
          }
     });

     btnQuickWork = (Button) findViewById(R.id.Button02);
     // delete all data from database (when delete button is clicked)
     this.btnQuickWork.setOnClickListener(new OnClickListener() {
          public void onClick(final View v) {
               txtMsg.setText((new Date()).toString());
          }
     });
}// onCreate
```

Multi-Threading

Example 4: Using the AsyncTask class

```java
private class VerySlowTask extends AsyncTask <String, Long, Void> {

     private final ProgressDialog dialog = new ProgressDialog(Main.this);

     // can use UI thread here
     protected void onPreExecute() {
          startingMillis = System.currentTimeMillis();
          txtMsg.setText("Start Time: " + startingMillis);
          this.dialog.setMessage("Wait\nSome SLOW job is being done...");
          this.dialog.show();
     }

     // automatically done on worker thread (separate from UI thread)
     protected Void doInBackground(final String... args) {
       try {
          // simulate here the slow activity
          for (Long i = 0L; i < 3L; i++) {
               Thread.sleep(2000);
               publishProgress((Long)i);
               }
       } catch (InterruptedException e) {
               Log.v("slow-job interrupted", e.getMessage())
       }
       return null;
     }
```

Multi-Threading

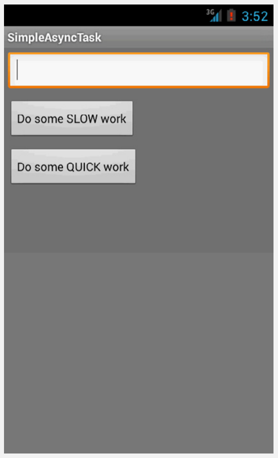
Example 4: Using the AsyncTask class

```
     // periodic updates - it is OK to change UI
     @Override
     protected void onProgressUpdate(Long... value) {
          super.onProgressUpdate(value);

          txtMsg.append("\nworking..." + value[0]);
     }

     // can use UI thread here
     protected void onPostExecute(final Void unused) {

          if (this.dialog.isShowing()) {
               this.dialog.dismiss();
          }

          // cleaning-up, all done
          txtMsg.append("\nEnd Time:"
                    + (System.currentTimeMillis()-startingMillis)/1000);
          txtMsg.append("\ndone!");
     }

 }//AsyncTask

}// Main
```

Multi-Threading

Example 4: Using the AsyncTask class

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/EditText01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="7dp" />

    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="7dp"
        android:text="Do some SLOW work" />

    <Button
        android:id="@+id/Button02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="7dp"
        android:text="Do some QUICK work" />

</LinearLayout>
```



Multi-Threading

Questions