



MULTITREADING

What is a thread?

- A thread is a concurrent unit of execution
- Threads share process's resource but are able to execute independently
- Each thread has a call stack for methods being invoked
- A VM may run several threads in parallel
 - True parallelism for multi-core CPU
- A VM has at least the main thread running when it is started

Why to use threads

- Multi-thread programming is hard, so why to use it?
- If the execution time of the main thread is higher than 5 s, then the OS displays an error message (ANR)
- Slow tasks (like file downloading), cannot run in the main thread; so, in this case you *must* use multiple threads
- In a multi-core CPU, multiple threads can truly run in parallel

How to use multi-tread?

- classical Thread programming
 - Special care must be taken as only main thread can update the UI,
- Android's AsyncTask
 - It's a class that simplifies the interaction with the main thread
- Service
 - Background work can be performed in a background service, using notification to inform the user about the next step

How to create a thread

- There are basically two main ways for a Thread to execute application code.
- One is providing a new class that extends Thread and overriding its run() method.
- The other is providing a new Thread instance with a Runnable object during its creation.
- In both cases, the start() method must be called to actually execute the new Thread.

First method: extending the thread class

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
```

```
public class MainActivity extends Activity {
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        new MyThread().start();
    }
```

Anonymous thread

```
public class MyThread extends Thread{
```

```
    @Override
    public synchronized void start() {
        // TODO Auto-generated method stub
        Log.i("TEST", "Thread started...");
        super.start();
    }
```

The start method may initialize some data, then the run method is called

```
    @Override
    public void run() {
        // TODO Auto-generated method stub
        super.run();
        Log.i("TEST", "Thread running...");
        for (int i=0; i<1000000000; i++) {
            if (i%100000000==0) Log.i("TEST", Thread.currentThread().toString());
        }
    }
}
```

The run method hosts the code to be run in a thread

PID	TID	Application	Tag	Text
26329	26329	com.example.threaddemo	TEST	Thread started...
26329	26356	com.example.threaddemo	TEST	Thread running...
26329	26356	com.example.threaddemo	TEST	Thread[Thread-5595,5,main]
26329	26356	com.example.threaddemo	TEST	Thread[Thread-5595,5,main]
26329	26356	com.example.threaddemo	TEST	Thread[Thread-5595,5,main]

What happens if the Activity is stopped?

```
public class MainActivity extends Activity {  
    MyThread t = null;   
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        t = new MyThread();  
        t.start();  
    }  
    public void btnKill(View v) {  
        t.finish();  
        finish();  
    }  
}  
  
public class MyThread extends Thread{  
    boolean running=false;  
    @Override  
    public synchronized void start() {  
        // TODO Auto-generated method stub  
        Log.i("TEST", "Thread started...");  
        super.start();  
        running=true;  
    }  
    @Override  
    public void run() {  
        // TODO Auto-generated method stub  
        super.run();  
        Log.i("TEST", "Thread running...");  
        for (int i=0;i<1000000000;i++) {  
            if (i%100000000==0) Log.i("TEST", Thread.currentThread().toString());  
            if (!running) return;  
        }  
    }  
    public void finish(){ running=false;}  
}
```

The thread is no longer anonymous

This is a handler of a button in UI

Check the running condition

- A thread has its own lifetime, independent from the creator
- If an activity wants to stop a thread on its ending, it has somehow to stop the thread(s) it launched
- One way is to use a simple boolean flag in run method for returning prematurely

2nd method: Implementing the runnable interface

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(new MyRunnable()).start();
}

public class MyRunnable implements Runnable{

    public void run() {
        // TODO Auto-generated method stub

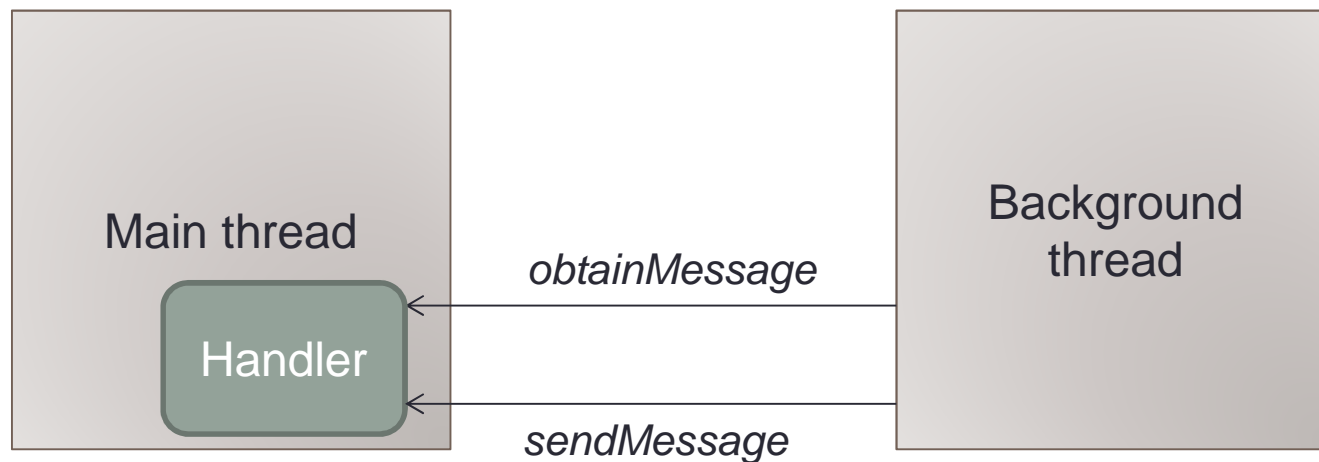
        Log.i("TEST", "Thread running...");
        for (int i=0;i<1000000000;i++) {
            if (i%100000000==0) Log.i("TEST", Thread.currentThread().toString());
        }
    }
}
```

- Another option to get a thread is implementing the runnable interface
- The interface has a single method, **run**, that hosts the code to be run concurrently
- In this way, the thread could extend other classes (recall java only allows *single* inheritance)

Interacting with the UI

- The background thread may need to update the UI according to its progress
- This means that at *any* time the background thread may need to communicate with the main thread
- This interaction is achieved through a message based mechanism
- The background thread sends a **message** to the main thread after it obtains a token (permission to send)
- The main thread processes the message

Interacting with the UI



- The UI thread creates a Handler object internal to itself
- The working thread uses this object to obtain an empty message and send a message to the UI thread

Example (HandlerThreadDemo)

```
package com.example.handlerthreaddemo;

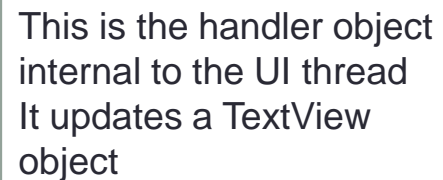
import android.app.Activity;

public class MainActivity extends Activity {

    TextView tv;

    //Create the handler
    Handler handler = new Handler() {

        @Override
        public void handleMessage(Message msg) {
            // TODO Auto-generated method stub
            super.handleMessage(msg);
            //update the UI
            tv.setText("."+tv.getText());
        }
    };
};
```



This is the handler object
internal to the UI thread
It updates a TextView
object

Example (HandlerThreadDemo)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tv = (TextView)findViewById(R.id.tv);
    tv.setText("Progress: ");

    //Create a thread
    Thread backgroundThread=
        new Thread(
            new Runnable() {
                public void run() {
                    // TODO Auto-generated method stub
                    for (int i=0;i<100;i++){
                        try {
                            Thread.sleep(1000);
                            Log.i("TEST","ok..");
                        } catch (InterruptedException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        } //catch

                        //Obtain a message from the handler and send it to
                        //the UI Thread

                        Message msg = handler.obtainMessage();
                        handler.sendMessage(msg);
                    } //for
                } //run
            }
        );
    backgroundThread.start();
}
```

A new thread is started

Send a simple message
that updates the textview

Passing data through the message

- Create a bundle
- Put data in
- Set the data field of the message

```
Bundle b = new Bundle();  
b.putInt("int", 10);  
Message msg = handler.obtainMessage();  
msg.setData(b);  
handler.sendMessage(msg);
```

- Get bundle from the message
- Get the data

```
Bundle b = msg.getData();  
int i = b.getInt("int");  
tv.setText(tv.getText()+ "."+i);
```

AsyncTask

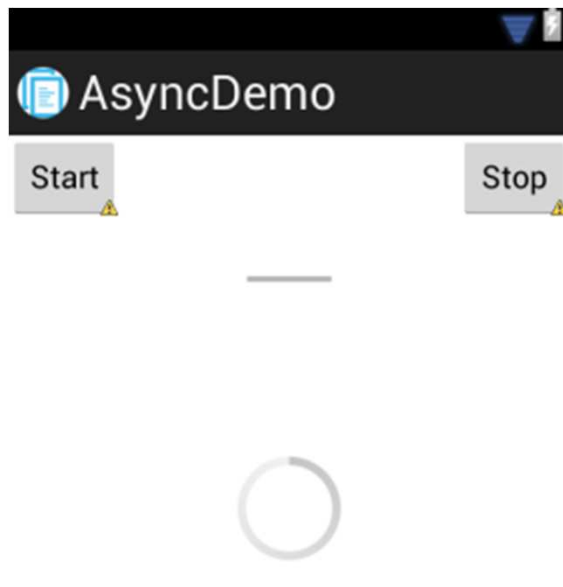
- AsyncTask is a class that simplifies the interaction with UI
- This class allows to perform background operations and publishes the results on the UI thread, without having to manipulate threads and/or handlers.
- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.

AsyncTask

When an asynchronous task is executed, the task goes through 4 steps:

1. `onPreExecute ()`, invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. `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.
3. `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.
4. `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.

Example: (AsyncDemo)



- ProgressBar (Large)
- ProgressBar (Normal)
- ProgressBar (Small)
- ProgressBar (Horizontal)

Select from eclipse widgets

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginRight="14dp"
        android:onClick="btnStart"
        android:text="Start" />
    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="74dp"
        android:max="99" />
    <Button
        android:id="@+id/btnStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:onClick="btnStop"
        android:text="Stop" />
    <ProgressBar
        android:id="@+id/rotatingBar"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/progressBar"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="83dp" />
</RelativeLayout>
```


Example

```
package com.example.asyncdemo;

import android.os.Bundle;

public class MainActivity extends Activity {

    private ProgressBar rotatingBar, progressBar;
    private AsyncTask at;

    public void onCreate(Bundle savedInstanceState) {}

    public void btnStart(View v) {}

    public void btnStop(View v) {}

    public void onStop() {}

    private class MyAsyncTask extends AsyncTask<String, Integer, String> {
        protected String doInBackground(String... p) {}

        protected void onCancelled() {}

        protected void onPostExecute(String result) {}

        protected void onPreExecute() {}

        protected void onProgressUpdate(Integer... values) {}
    }
}
```

Example

```
package com.example.asyncdemo;

import android.os.Bundle;

public class MainActivity extends Activity {

    private ProgressBar rotatingBar, progressBar;
    private AsyncTask at;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        rotatingBar = (ProgressBar) findViewById(R.id.rotatingBar);
        progressBar = (ProgressBar) findViewById(R.id.progressBar);
        rotatingBar.setVisibility(View.GONE);
    }

    public void btnStart(View v) {}

    public void btnStop(View v) {}

    public void onStop() {}

    private class MyAsyncTask extends AsyncTask<String, Integer, String> {
        protected String doInBackground(String... p) {}

        protected void onCancelled() {}

        protected void onPostExecute(String result) {}

        protected void onPreExecute() {}

        protected void onProgressUpdate(Integer... values) {}
    }
}
```

Constant	Value	Description
visible	0	Visible on screen; the default value.
invisible	1	Not displayed, but taken into account during layout (space is left for it).
gone	2	Completely hidden, as if the view had not been added.

Example

```
public class MainActivity extends Activity {  
  
    private ProgressBar rotatingBar, progressBar;  
    private AsyncTask at;  
  
    public void onCreate(Bundle savedInstanceState) {}  
  
    public void btnStart(View v) {  
        at = (new MyAsyncTask()).execute("go");  
    }  
  
    public void btnStop(View v) {  
        at.cancel(true);  
    }  
  
    @Override  
    public void onStop() {  
        super.onStop();  
        Toast.makeText(this, "Activity Stopped", 1).show();  
    }  
  
    private class MyAsyncTask extends AsyncTask<String, Integer, String> {  
        protected String doInBackground(String... p) {}  
  
        protected void onCancelled() {}  
  
        protected void onPostExecute(String result) {}  
  
        protected void onPreExecute() {}  
  
        protected void onProgressUpdate(Integer... values) {}  
  
    }  
}
```

The three types used by an asynchronous task are the following:

1. **Params**, the type of the parameters sent to the task upon execution.
2. **Progress**, the type of the progress units published during the background computation.
3. **Result**, the type of the result of the background computation.



Example

```
private class MyAsyncTask extends AsyncTask<String, Integer,String> {
    @Override
    protected String doInBackground(String... p) {
        // TODO Auto-generated method stub

        for (int i=0;i<100;i++)
        { try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        publishProgress(i);
        if (isCancelled()) break;
    }

    return "finished";
}

protected void onCancelled() {}

protected void onPostExecute(String result) {}

protected void onPreExecute() {}

@Override
protected void onProgressUpdate(Integer... values) {
    // TODO Auto-generated method stub
    super.onProgressUpdate(values);
    progressBar.setProgress(values[0]);
}
}
```

publishProgress generates integers to mean the actual progress....