MOBILE INTELLIGENCE 02

Android - Process & Threads

Minsoo Kim (Pukyong Nat'l Univ.)

Major In Industrial Data System & Engineering

부산대학교 PUSAN NATIONAL UNIVERSITY
Department of Industrial Engineering

국립부경대학교 PUKYONG NATIONAL UNIVERSITY
Department of Industrial and Data Engineering

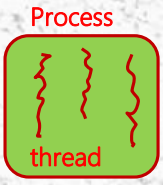
Logistics of this Slide

- Multi-tasking in Android
  - Handling Tasks in the Computer
  - Process and Thread in Android
  - Process Lifecycle management in Android
  - Main Thread (≈ UI Thread)
- Using Thread in Java
  - Creating threads in Java
  - Using anonymous inner-class
- Using Thread in Android
  - Using separate worker thread in Activity
  - Manipulating UI from external Thread
  - Async messaging with Handler & Looper

Multi-tasking in Android

Handling Tasks in the Computer → Running program (≈ process)

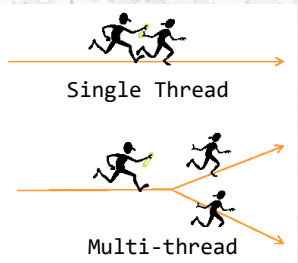
- **Process** and **Thread**



- **Process**: independently executable unit on its own memory space
- **Thread**(≈ **lightweight process**): a series of control flows within a process
  - Reside inside of a process, every process has at least one thread.
  - Threads in the same process share the resources (memory, file, …) of that process.

- **Multi-tasking, Multi-processing, Parallel Processing, Multi-threading**

1. **Multi-tasking**: concurrent exec. of multiple tasks for a specific time period.
   - CPU time scheduling enables multiple tasks to be processed together.
   - Multi-tasking is possible even on a single CPU. ← time-sharing mechanism
2. **Multi-processing**: Two or more CPUs work together to process tasks.
   - CPUs on multiple computers or on a single computer can work cooperatively.
3. **Parallel Processing**: execute a task in parallel on two or more CPUs
   - Reduce the overall time by dividing and executing tasks using multi-core.
   - It is a concept in contrast to **Serial Processing**.
   A. **SIMD**(Single Instruction Multiple Data) type
   B. **MIMD**(Multiple Instruction Multiple Data) type
   C. **MISD**(Multiple Instruction Single Data) type
4. **Multi-threading**: run multiple threads within a process
   - **Synchronization** of tasks over the same resources (esp. data)!



Multi-tasking in Android

Process & Thread in Android System

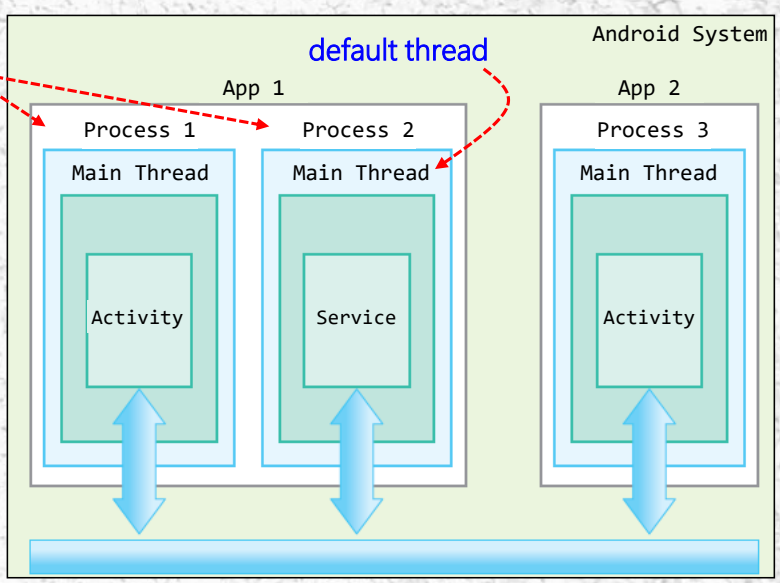
- Utilizes multi-processing of Linux & multi-threading of Java language
  - Run multiple tasks in the foreground & background using the Activity Stack.
- When an app starts, a corresponding new Linux process is created.
  - Basically, all components in an app run **in the same thread in the same process**. If another component of the app is already running as a process, a new component is started within that process and shares the default thread.

Component can also be run as an independent process. & additional threads can be created.

Package name is the default process name.

```
<manifest … package="…">
  …
 <application android:process="name1" … >
  …
  <activity android:process="globalname" … >
    …
  </activity>
  <service android:process=":newPrivateName" … >
    …
  </service>
  …
 </application>
</manifest>
```

Lowercase names are shared as the global process. Names followed by a colon(:) create a new process private to application.



Multi-tasking in Android

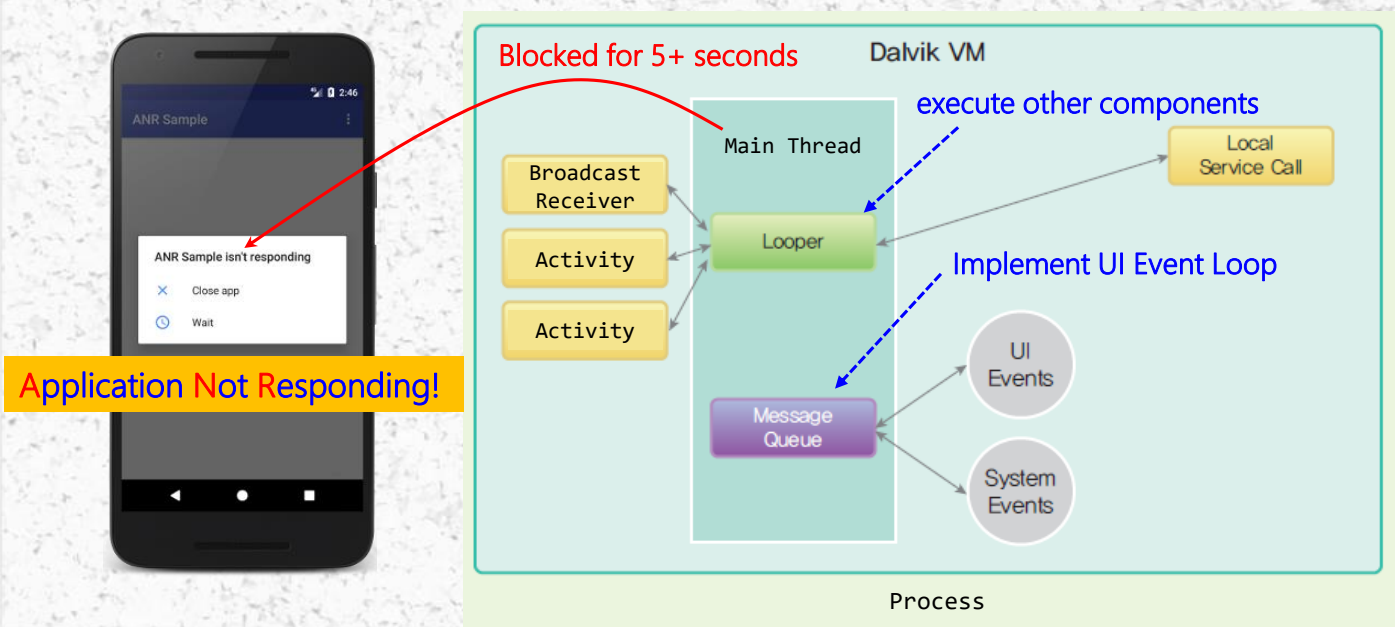
- Process Lifecycle management in Android
  - Android removes processes based on priority to free up memory.
    - There are 5 level of importance hierarchy.

1. **foreground process**: the process the user is currently working on.
   - Have an activity interacts with the user or have a service connected to it.
   - Foreground service that is started by calling startForeground() method.
   - Having a service executing onCreate(), onStart(), onDestroy() method.
   - Having a broadcast receiver executing onReceive() method.
2. **visible process**: a process that does not have a foreground component but still can affect what the user sees on the screen.
   - Having an activity that is not in the foreground but affects the user.
     - e.g: previous activity visible at the back of the dialog window.
   - Having a service connected to a visible application
3. **service process**: a process running a service started with startService().
   - Though not visible, it's still handling something important to the user.
     - e.g.: play music at the background, download file at the background, …
4. **background process**: a process of invisible activity with a onStop() call.
5. **empty process**: a process with no active component
   - A process kept for the purpose of caching to quickly execute components

Multi-tasking in Android

- **Main Thread** (≈ **UI Thread**): default thread of application execution
  - **single thread model**: main thread is responsible for event handling of UI components, UI drawing, and all system calls from the components.
    - Rule 1: prevent UI Thread from blocking ← can cause **ANR exception**!
    - Rule 2: never manipulate Android's UI toolkit outside of the UI Thread.

  → Use worker thread for long tasks & manipulate UI only at the main thread.



Using Thread in Java

- Creating threads in Java
  1. Inherit Thread class and provide run() method at the subclass.
     - Execute run() method by calling Thread.start() as a separate thread.
  2. Pass an object implementing Runnable interface to a Thread object.
     - Usable when cannot inherit Thread by already inheriting another class.
     - Mainly adopted as an anonymous inner-class for convenient use of thread
  - ❖ Directly calling run() is processing the work in an existing thread!

```
public class Worker extends Thread {
  ...
  public void run() {
    ... // Things to do in a new thread!
  }                                          Worker Thread

  public static void main(String[] args) {
    Worker t = new Worker();
    t.start();
    ... // Things to do in main thread!
  }                                          Main Thread
}
```

```
public class Worker implements Runnable {
  ...
  public void run() {
    ... // Things to do in a new thread!
  }                                          Worker Thread

  public static void main(String[] args) {
    Worker t = new Worker();
    new Thread(t).start();
    ... // Things to do in main thread!
  }                                          Main Thread
}
```


Using Thread in Java

- Using anonymous inner-class
  - Corresponds to the method of using the Runnable interface for thread.
    - Passing an anonymous object that implements Runnable to a Thread object.
  - Don't need to give it a name. used for a one-time task.
    - There's no need to reuse it because its closely connected to the UI.
    - It can access all the members of the enclosing outside class.

No need for external class to inherit(implement) Thread(Runnable).

```
public class Worker implements Runnable {
  ...
  public void run() {
    ... // Things to do in a new thread!
  }                                          Worker Thread

  public static void main(String[] args) {
    Worker t = new Worker();
    new Thread(t).start();
    ... // Things to do in main thread!
  }                                          Main Thread
}
```

```
public class Worker {
  ...
  public static void main(String[] args) {
    Worker t = new Worker();
    new Thread( new Runnable() {
      public void run() {
        ... // Things to do in a new thread!
      }                                      Worker Thread
    } ).start();
    ... // Things to do in main thread!
  }                                          Main Thread
}
```

Using Thread in Java

Use a separate worker thread (background thread) in Activity

1. Create 'Thread Test Project'

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  ...
  tools:context=".MainActivity">

  <Button
    android:id="@+id/test"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Test Thread"
    android:onClick="processClick"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    for (int i = 0; i < 5; i++) {
      try {
        Thread.sleep(6000);
        Log.d("THREAD", "delays return: " + i);
      } catch (Exception e) {}
    }
    Log.d("THREAD", "all done.");
  }

  public void processClick(View v) {
    ...
  }
}
```

Delay the execution of a thread

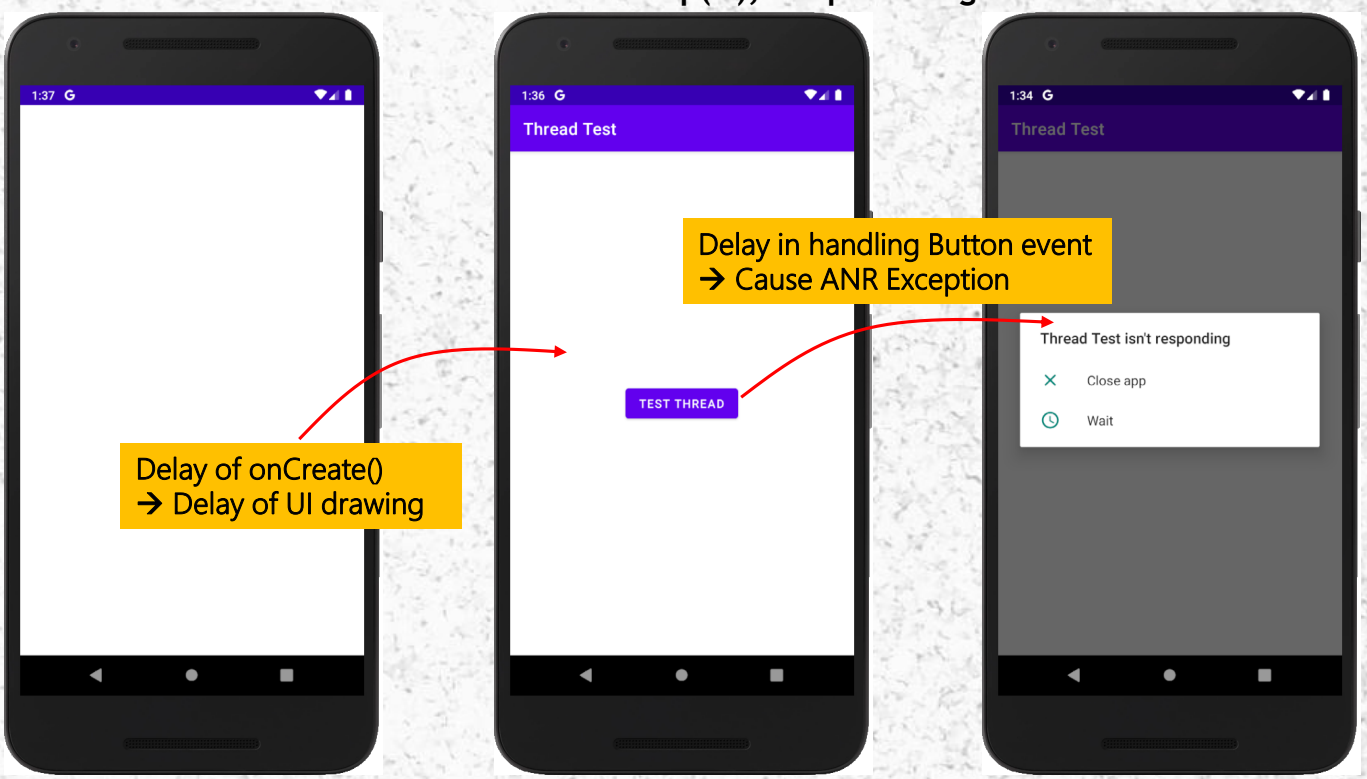
Copy & Paste

Using Thread in Java

(Use a separate worker thread (background thread) in Activity)

2. Run and Test the 'Thread Test Project'

- Increase the time in Thread.sleep(…), keep clicking the Button!



Using Thread in Java

- (Use a separate worker thread (background thread) in Activity)

3. Modify and Run the 'Thread Test Project'

- Modify the code to use a separate Thread

```
public class MainActivity extends AppCompatActivity {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    for (int i = 0; i < 5; i++) {
      try {
        Thread.sleep(6000);
        Log.d("THREAD", "delays return: " + i);
      } catch (Exception e) {}
    }
    Log.d("THREAD", "all done.");
  }

  public void processClick(View v) {
    ...
  }
}
```

MainActivity.java

Copy & Paste

```
public class MainActivity extends AppCompatActivity
implements Runnable {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(this).start();
  }
  public void processClick(View v) {
    new Thread(this).start();
  }
  public void run() {
    for (int i = 0; i < 5; i++) {
      try {
        Thread.sleep(6000);
        Log.d("THREAD", "delays return: " + i);
      } catch (Exception e) {}
    }
    Log.d("THREAD", "all done.");
  }
}
```

MainActivity.java

Using Thread in Android

- (Use a separate worker thread (background thread) in Activity)

4. Avoid creating Threads on every Button clicks → use run flag!

```
public class MainActivity extends AppCompatActivity
implements Runnable {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new Thread(this).start();
  }
  public void processClick(View v) {
    new Thread(this).start();
  }
  public void run() {
    for (int i = 0; i < 5; i++) {
      try {
        Thread.sleep(6000);
        Log.d("THREAD", "delays return: " + i);
      } catch (Exception e) {}
    }
    Log.d("THREAD", "all done.");
  }
}
```

MainActivity.java

Add flag

```
public class MainActivity ... {
  private boolean running = false;
  protected void onCreate(Bundle savedInstanceState) {
    ...
    if (!running) new Thread(this).start();
  }
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;
    for (int i = 0; i < 5 && running; i++) {
      try {
        Thread.sleep(6000);
        Log.d("THREAD", "delays return: " + i);
      } catch (Exception e) {}
    }
    Log.d("THREAD", "all done.");
    running = false;
  }
}
```

MainActivity.java

Using Thread in Android

Manipulating UI from external Thread

- Manipulate UI from another Thread: Modify 'Thread Test Project'

```
public class MainActivity ... {
  private boolean running = false;
  protected void onCreate(Bundle savedInstanceState) {
    ...
    if (!running) new Thread(this).start();
  }
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;
    for (int i = 0; i < 5 && running; i++) {
      try {
        Thread.sleep(6000);
        Log.d("THREAD", "delays return: " + i);
      } catch (Exception e) {}
    }
    Log.d("THREAD", "all done.");
    running = false;
  }
}
```

MainActivity.java

delete

replace

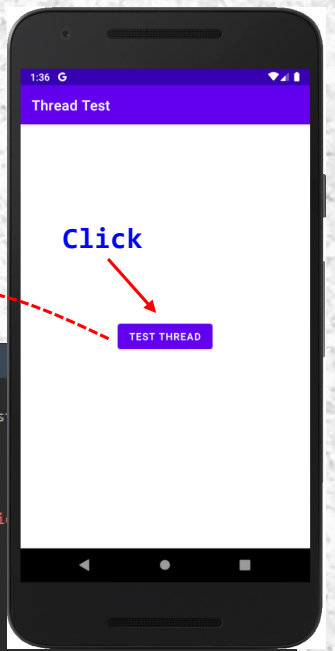
```
public class MainActivity ... {
  private boolean running = false;
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // if (!running) new Thread(this).start();
  }
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;

    Button btn = (Button) findViewById(R.id.test);
    btn.setText("Changed!");

    running = false;
  }
}
```

MainActivity.java

Using Thread in Android

(Manipulating UI from external Thread)

- (Manipulate UI from another Thread: Modify 'Thread Test Project')
  - Check that FATAL EXCEPTION is occuring!
    - Because Android's UI widgets are not Thread-Safe,
    - Blocks other threads from changing the state of UI widget,
    - Only allow manipulations from the UI Thread!



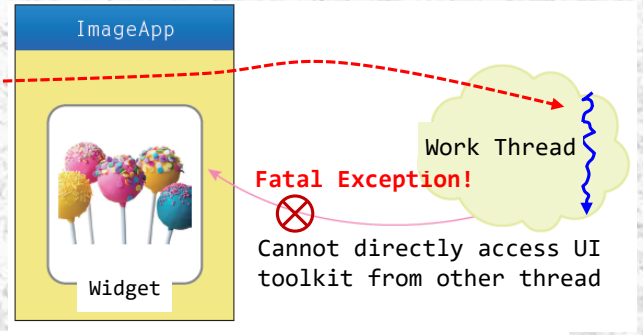
FATAL EXCEPTION!

```
Run: app
D/eglCodecCommon: setVertexArrayObject: set vao to 0 (0) 3 2
I/OpenGLRenderer: Davey! duration=758ms; Flags=0, IntendedVsync=597074731445487, Vsync=597074981445477, Oldes
I/Choreographer: Skipped 33 frames!  The application may be doing too much work on its main thread.
E/AndroidRuntime: FATAL EXCEPTION: Thread-2
    Process: kr.ac.sme.pknu.threadtest, PID: 15044
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hi
        at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:7753)
        at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1225)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at android.view.View.requestLayout(View.java:23093)
        at androidx.constraintlayout.widget.ConstraintLayout.requestLayout(ConstraintLayout.java:3239)
        at android.view.View.requestLayout(View.java:23093)
TODO  Terminal  Build  6: Logcat  Profiler  Database Inspector  4: Run
```

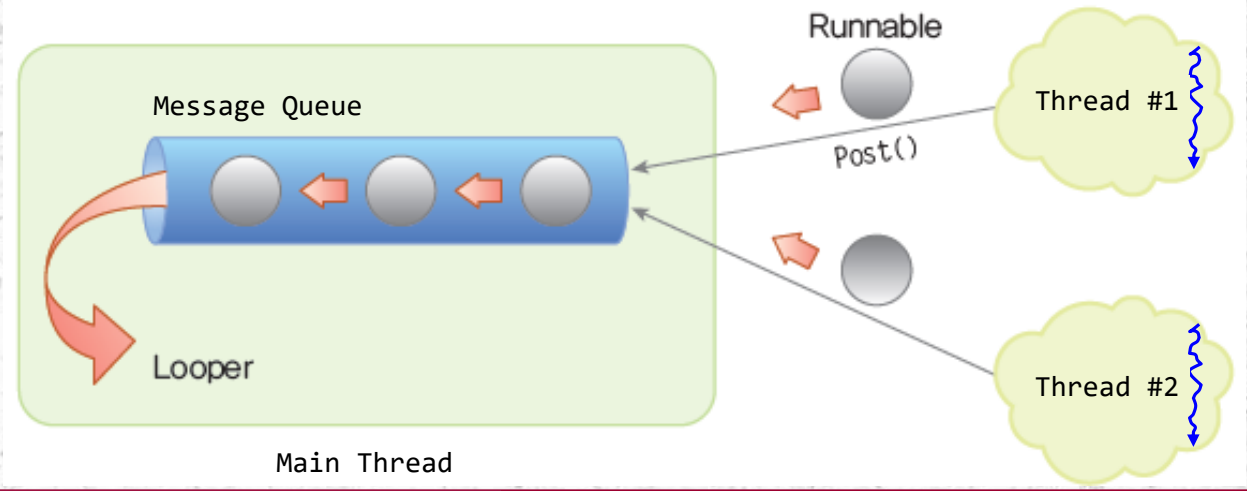
Using Thread in Android

- (Manipulating UI from external Thread)
  - Delegate execution by using Event Queue of the Looper in Main Thread

Be careful not to manipulate the UI from the worker thread that is introduced to run the task that takes long time.



1. For View or Activity
   - View.post(*Runnable*)
   - View.postDelayed(*Runnable*, *long*)
   - Activity.runOnUiThread(*Runnable*)



Using Thread in Android

- (Manipulating UI from external Thread)
  - (1. For View or Activity)
    - e.g.: downloading an image through Network & setting it in ImageView
      - Even if you use a separate thread to download the image, this thread cannot directly manipulate the ImageView. Use View.post(…) instead!

```
public void onClick(View v) {
   new Thread( new Runnable() {
      public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
      }
   } ).start();
}
```

A task that takes long time

A task manipulating UI widget

```
public void onClick(View v) {
   new Thread( new Runnable() {
      public void run() {
         final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
         mImageView.post( new Runnable() {
            public void run() {
               mImageView.setImageBitmap(bitmap);
            }
         } );
      }
   } ).start();
}
```

Delegate to UI Thread

Using Thread in Android

- (Manipulating UI from external Thread)
  - e.g.: Correct the 'Thread Test Project'

```
public class MainActivity ... {
  private boolean running = false;
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // if (!running) new Thread(this).start();
  }
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;

    Button btn = (Button) findViewById(R.id.test);
    btn.setText("Changed!");

    running = false;
  }
}
```

MainActivity.java

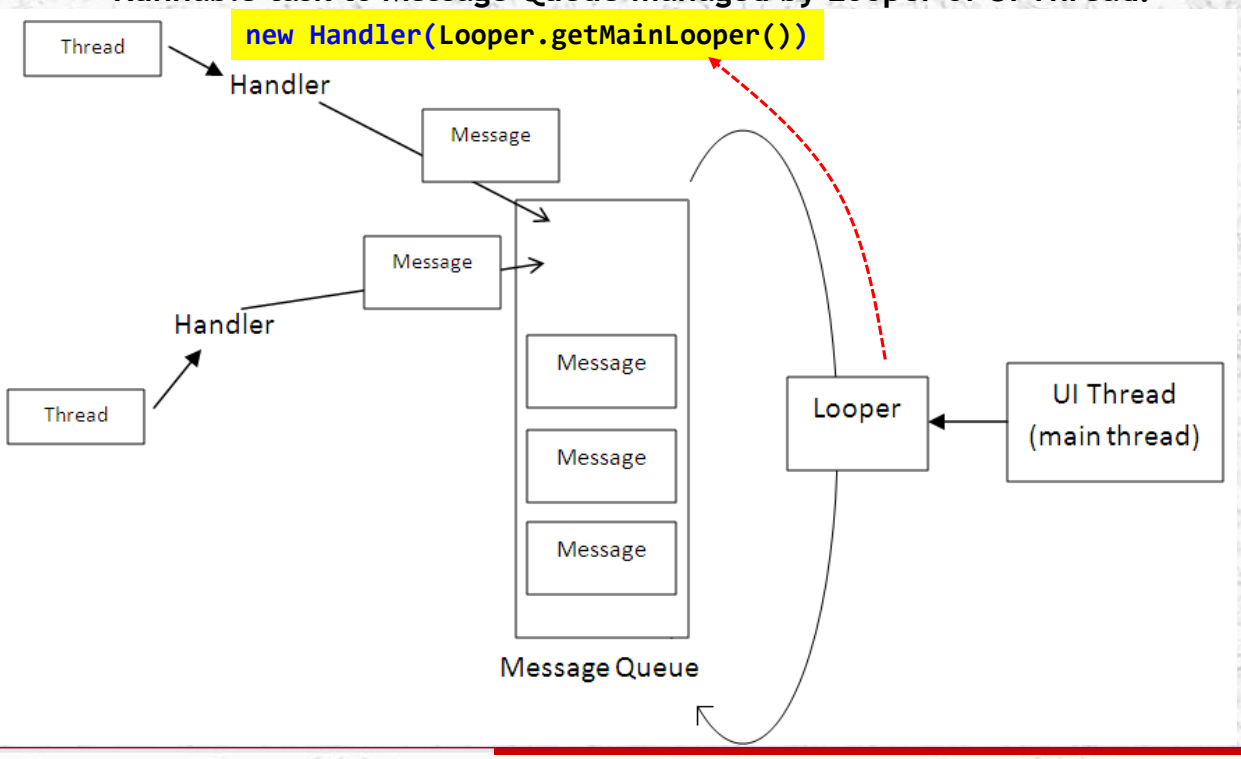
replace

```
public class MainActivity ... {
  private boolean running = false;
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // if (!running) new Thread(this).start();
  }
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;
    Button btn = (Button) findViewById(R.id.test);
    btn.post(new Runnable() {
      public void run() {
        btn.setText("Changed!");
      }
    });
    running = false;
  }
}
```

MainActivity.java

Using Thread in Android

- (Manipulating UI from external Thread)
  2. Delivering a task to Looper of UI Thread via android.os.Handler
     - A child thread of UI Thread uses Handler.post(*Runnable*) to deliver Runnable task to Message Queue managed by Looper of UI Thread.



Using Thread in Android

- (Manipulating UI from external Thread)
  - (2. Delivering a task to Looper of UI Thread via android.os.Handler)
    - Use the Looper of the UI Thread, not the Looper of the Handler's thread!
      - new Handler(*Looper.getMainLooper()*) ← Runs at the UI Thread's Main Looper
      - new Handler(*Looper.myLooper()*) ← Runs at the current Thread's Looper
        - » If UI Thread and Handler's thread are different, FATAL EXCEPTION occurs!

```
public class MainActivity ... {
  ...
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;
    Button btn = (Button) findViewById(R.id.test);
    btn.post(new Runnable() {
      public void run() {
        btn.setText("Changed!");
      }
    });
    running = false;
  }
}
```

MainActivity.java

replace

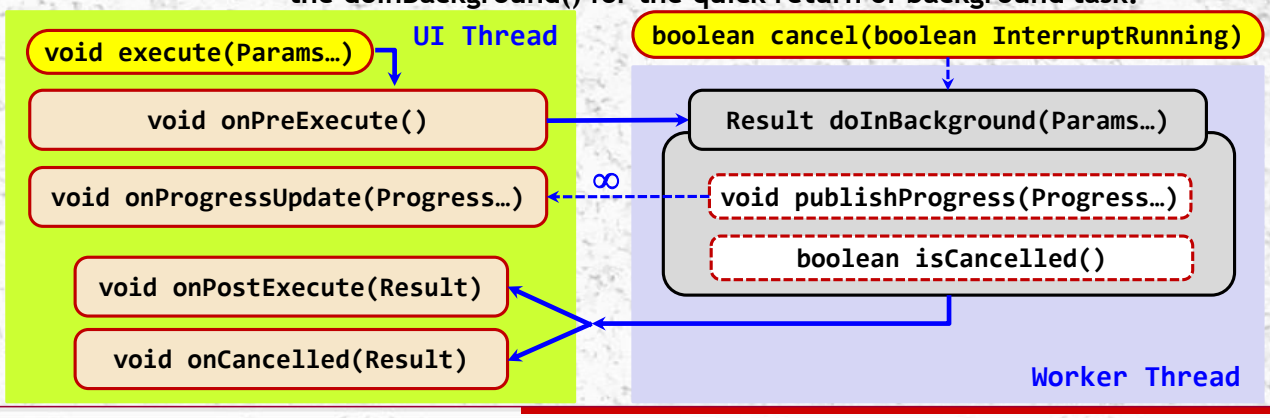
```
public class MainActivity ... {
  ...
  public void processClick(View v) {
    if (!running) new Thread(this).start();
  }
  public void run() {
    running = true;
    Button btn = (Button) findViewById(R.id.test);
    Looper aLooper = Looper.getMainLooper();
    new Handler(aLooper).post(new Runnable() {
      public void run() {
        btn.setText("Changed!");
      }
    });
    running = false;
  }
}
```

Looper.myLooper(); try!

MainActivity.java

Using Thread in Android

- (Manipulating UI from external Thread)
  - 3. Asynchronous processing with AsyncTask ← deprecated (API Level 30)
    - Provides a dedicated method for simple background work & UI update
    - Inherit AsyncTask<Params, Progress, Result> and override doInBackground()
      - Start: execute({ Params… | Runnable }), executeOnExecutor(Executor, Params…)
      - Create and execute an AsyncTask within the UI Thread.
    - onPreExecute() → doInBackground() → { onPostExecute() | onCancelled() }
      - Inside of doInBackground(): call publishProgress() to run onProgressUpdate()
      - All the other onXXX() methods except doInBackground() are run by UI Thread.
      - By calling cancel(), we can execute onCancelled() instead of onPostExecute().
        - » isCancelled() returns true for cancelled task → check isCancelled() within the doInBackground() for the quick return of background task.



Using Thread in Android

- (Manipulating UI from external Thread)
  - (3. Asynchronous processing with AsyncTask)
    - Since HoneyComb(v3.0), single thread sequentially handles all AsyncTasks.
      - For parallel execution: use executeOnExecutor(Executor, Params…)
        - » Designate AsyncTask.THREAD_POOL_EXECUTOR as an Executor
    - When no specific type is used for Params, Progress, Result → use Void

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Run (UI Thread)

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Finished AsyncTask object cannot be executed again!

Using Thread in Android

- (Manipulating UI from external Thread)
  - Image File Download using AsyncTask
    - Modify 'Thread Test Project'

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  …
  tools:context=".MainActivity">

  <ImageView
    android:id="@+id/test"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Modify

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest … >
    <application android:allowBackup="true" …
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Add Permission

Using Thread in Android

MainActivity.java

```
...
public class MainActivity extends AppCompatActivity implements Runnable {
  private final String data = "https://isis.pknu.ac.kr/gaon_small.jpg";

  private class ImageDownloadTask extends AsyncTask<String, Integer, Bitmap> {
    protected void onPostExecute(Bitmap bitmap) {
      ImageView iv = (ImageView) findViewById(R.id.test);
      iv.setImageBitmap(bitmap);
      Toast.makeText(getBaseContext(), "Image Downloaded!", Toast.LENGTH_SHORT).show();
    }
    ...
  }

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.id.activity_main);
    this.runOnUiThread(this);
  }
  public void run() {
    new ImageDownloadTask().execute("data");
  }
}
```

gaon_big.jpg

try!

run

```
protected Bitmap doInBackground(String... urls) {
  Bitmap image = null;
  InputStream is = null;
  try {
    URL url = new URL(urls[0]);
    HttpsURLConnection conn =
      (HttpsURLConnection) url.openConnection();
    conn.connect();
    is = conn.getInputStream();
    image = BitmapFactory.decodeStream(is);
  } catch(Exception e) {
    Log.d("THREAD", "Download Error: " + e.toString());
  }
  return image;
}
```


Using Thread in Android

- (Manipulating UI from external Thread)
  - (Image File Download using AsyncTask: showing progress)

MainActivity.java

```
public class MainActivity extends AppCompatActivity implements Runnable {
  private final String data = "https://isis.pknu.ac.kr/gaon_big.jpg";
  private ProgressDialog pDlg;

  private class ImageDownloadTask extends AsyncTask<String, Integer, Bitmap> {
    protected void onCancelled(Bitmap bitmap) {
      Toast.makeText(getBaseContext(), "Download Cancelled!", Toast.LENGTH_LONG).show();
    }
    protected void onProgressUpdate(Integer... values) {
      pDlg.setProgress(values[0]);
    }
    protected void onPostExecute(Bitmap bitmap) {
      ImageView iv = (ImageView) findViewById(R.id.test);
      iv.setImageBitmap(bitmap);
      pDlg.dismiss();
    }
    ...
    ...
  }
  ...
}
```

add

protected Bitmap doInBackground(String... urls) { ... }

```
protected void onPreExecute() {
  pDlg = new ProgressDialog(MainActivity.this);
  pDlg.setTitle("File Download");
  pDlg.setMessage("Downloading Image...");
  pDlg.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
  pDlg.setMax(100);
  pDlg.setProgress(0);
  pDlg.setCancelable(true);
  pDlg.setOnCancelListener(
    new DialogInterface.OnCancelListener() {
      public void onCancel(DialogInterface dialog) {
        ImageDownloadTask.this.cancel(true);
      }
  });
  pDlg.show();
}
```

Using Thread in Android

(Manipulating UI from external Thread)

- (Image File Download using AsyncTask: showing progress)

```
protected Bitmap doInBackground(String... urls) {
  Bitmap image = null;
  InputStream is = null;
  BufferedOutputStream os = null;
  int received = 0;
  try {
    ...
    int fileSize = conn.getContentLength();
    is = new BufferedInputStream(url.openStream(), 512);
    ByteArrayOutputStream dataStream = new ByteArrayOutputStream();
    os = new BufferedOutputStream(dataStream);

    ...
  } catch(Exception e) {
    Log.d("THREAD", "Error: " + e.toString());
  }

  return image;
}
```

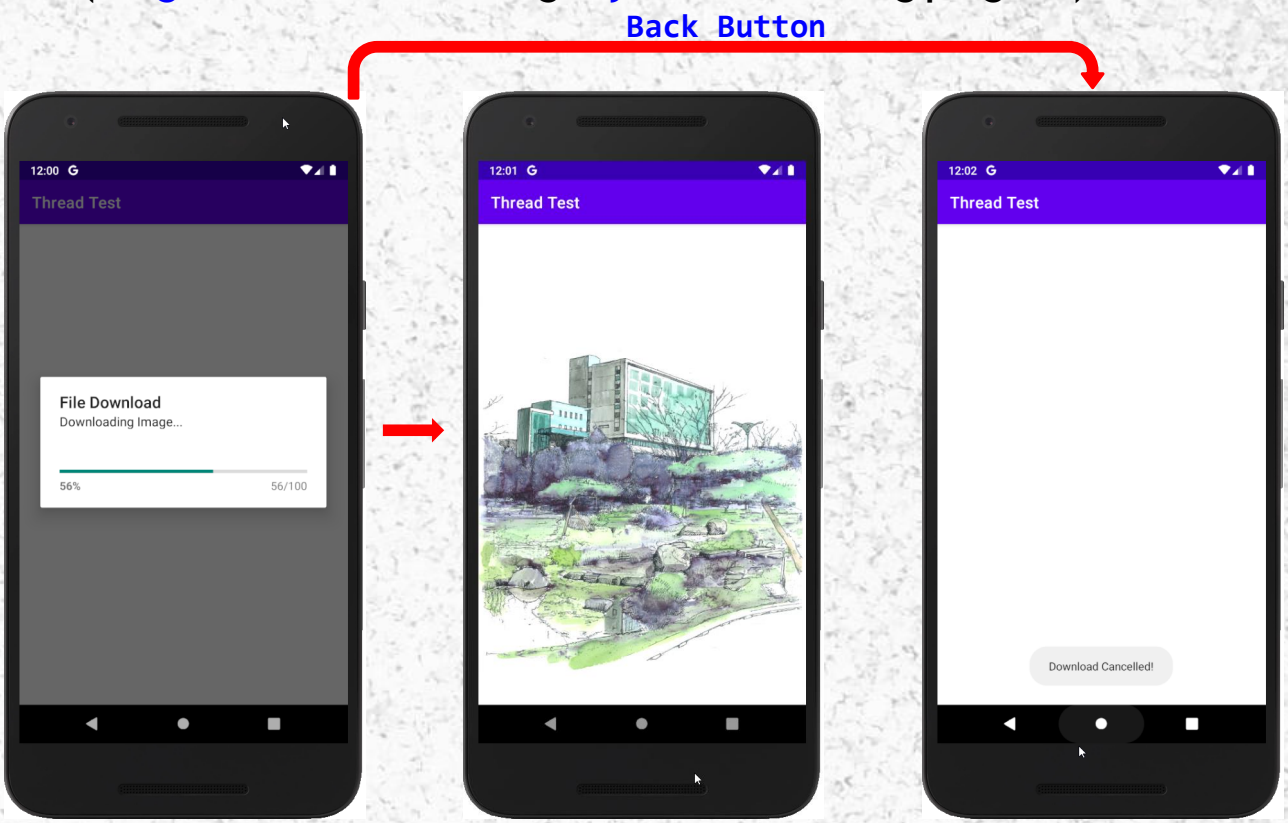
```
URL url = new URL(urls[0]);
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.connect();
```

```
byte buffer[] = new byte[512];
long total = 0;
while ((received = is.read(buffer)) != -1) {
  total += received;
  publishProgress((int) ((total * 100) / fileSize));
  os.write(buffer, 0, received);
  if (isCancelled())
    return null;
}
os.flush();
byte[] data = dataStream.toByteArray();
image = BitmapFactory.decodeByteArray(data, 0, data.length);
```


Using Thread in Android

(Manipulating UI from external Thread)

- (Image File Download using AsyncTask: showing progress)



Using Thread in Android

Async messaging with Handler & Looper

- Use Message Queue and Looper bound to the Handler's thread
  - Can construct a processing mechanism like the Android System's UI Thread
  - Looper: schedule & dispatch Message or Runnable that enters the Queue.
  - Handler: Send Message to Queue, handle callback interface.
- In the Thread: Looper.prepare() → new Handler(…) → Looper.loop()
  - Pass Looper to Handler while creating, explicitly bind to target Thread
    - new Handler(*Looper.myLooper()*) vs. new Handler(*Looper.getMainLooper()*)
  - Call Looper.quit() to finish Looper Thread.
    - Looper handles Message via handleMessage() of bound Handler.

```
public class MyServiceHandler extends Handler {
  @Override
  public void handleMessage(Message msg) {
    // TODO: handle message here!
  }
}
```

```
MyServiceThread server = new MyServiceThread();
server.start();
server.getHandler().postDelayed(myRunnable, 1000);
server.getHandler().sendMessage(myMessage);
server.getHandler().sendEmptyMessage(0);
```

```
public class MyServiceThread extends Thread {
  private Handler mHandler;
  @Override
  public void run() {
    Looper.prepare();
    mHandler = new MyServiceHandler();
    Looper.loop();
  }
  public Handler getHandler() {
    return mHandler;
  }
}
```


Using Thread in Android

(Async messaging with Handler & Looper)

- Thread Test Project: Add inner-class & member to MainActivity class

```
public class MainActivity extends AppCompatActivity implements Runnable {
  private class MyServiceThread extends Thread {
    private Handler myHandler;
    public void run() {
      Looper.prepare();
      myHandler = new Handler(Looper.myLooper()) {
        public void handleMessage(Message msg) {
          switch(msg.what) {
            case 0:
              Log.d("THREAD", "Exit Looper!");
              Looper.myLooper().quit();
              break;
            default:
              Log.d("THREAD", "Message(" + msg.what + ") processed!");
          }
        }
      };
      Looper.loop();
    }
    public Handler getHandler() { return myHandler; }
  }
  private MyServiceThread server = null;
  ...
}
```

```
protected void onResume() {
  super.onResume();
  server = new MyServiceThread();
  server.start();
  new Handler(Looper.getMainLooper()).postDelayed(
    new Runnable() {
      public void run() {
        for (int i = 1; i < 10; i++)
          server.getHandler().sendEmptyMessage(i);
      }
    }, 1000);
}
```

Send delayed message to another Handler considering server's start-up delay.

```
protected void onPause() {
  super.onPause();
  server.getHandler().sendEmptyMessage(0);
}
```

add

Using Thread in Android

- (Async messaging with Handler & Looper)
  - Test: call onPause() & onResume() using Task Manager Button