جامعة الأقصى
AL-AQSA UNIVERSITY

Lec. #5: Multithreading

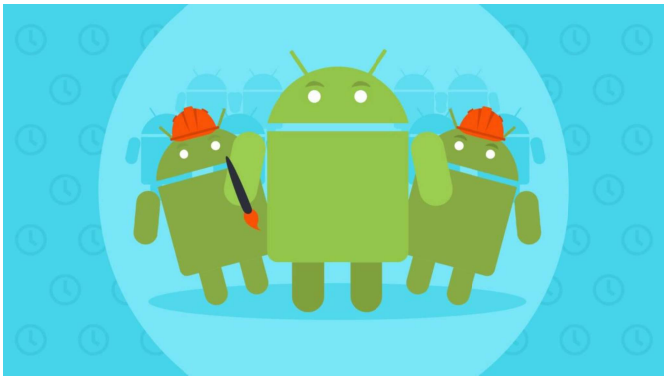
Thread, Handler & AsyncTask

Mobile Applications Development 2

SECOND SEMESTER OF THE ACADEMIC YEAR 2020/2021

What is Multithreading

- A **Thread** is a concurrent unit of execution. 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.



Multithreading

Two techniques to create threads in Android

1) implementing the Runnable interface
- The Runnable interface should be *implemented* by any class whose instances are intended to be executed by a thread. The class must define a method, called *run*, with no arguments.
- invoke Thread constructor with an instance of this Runnable class

2) extending Thread
- Define a subclass of java.lang.Thread
  - Define a *run* method
- In original thread (MainThread, Ui Thread), create an instance of the Thread subclass
  - Then, call *start* method of that instance

Multithreading

```
class MyThreadA extends Thread {
    public void run() { // entry point for thread
      while(true) {
        Log.i("th","ThreadA talking .. ");
      }} }
class MyThreadB extends Thread {
    public void run() { // entry point for thread
      while(true) {
        Log.i("th","ThreadB talking .. ");
      }} }
```

```
public class MainActivity extends
AppCompatActivity {
  @Override
  protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    MyThreadA  t1 = new MyThreadA();
    MyThreadB  t2 = new MyThreadB();
    t1.start();
    t2.start();
  }}
```

Multithreading

```
Thread t1=new Thread(new Runnable() {
    @Override
    public void run() {
    while(true) {
            Log.i("th","t1 is talking .. ");
        }}});
Thread t2=new Thread(new Runnable() {
    @Override
    public void run() {
while(true) {
            Log.i("th","t2 is talking .. ");
        }}});
t1.start();
t2.start();
```


Thread life time

Threads that *are* attached to an activity/fragment: These threads are tied to the lifecycle of the activity/fragment and are terminated as soon as the activity/fragment is destroyed.

Threads that *are not* attached to any activity/fragment: These threads can continue to run beyond the lifetime of the activity/fragment (if any) from which they were spawned.

Thread life time

Threads share the process' **resources** but are able to execute independently.
Applications responsibilities can be separated main thread **runs UI**, and slow tasks are sent to background threads.
Threading provides an useful abstraction of concurrent execution.
A multithreaded program operates faster on computer systems that have multiple CPUs.

Using post, runOnUIThread, Handler

- Android does not permit the user thread to change Ui Components So, user thread should send the change to Ui Thread to do it.
- Ui Thread, Main Thread or Original Thread these names refer to one thing which is the main thread of the application.

```
textView.setText("");
Thread t1=new Thread(new Runnable() {
   @Override
   public void run() {
      for( i=0;i<270;i++){
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace(); } }
      textView.post(new Runnable() {
         @Override
         public void run() {
           textView.append((char) i+"");
         }});
   }});.
```

Using post, runOnUIThread, Handler

- runOnUiThread method can be used to change into more than one Ui components at once.
- Where the method post only used by one view.

```
textView.setText("");
Thread t1=new Thread(new Runnable() {
   @Override
   public void run() {
      for( i=0;i<270;i++){
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();}}
      runOnUiThread(new Runnable() {
         @Override
         public void run() {
            textView.append((char) i+"");}});
}});
```


Using Handler

```
MyHandler handler=new MyHandler();
Message m=handler.obtainMessage();
Bundle b=new Bundle();
b.putChar("ch",(char)i);
m.setData(b);
handler.sendMessage(m);
```

```
class MyHandler extends Handler{
   @Override
   public void handleMessage(@NonNull
Message msg) {
      super.handleMessage(msg);
       char c=msg.getData().getChar("ch");
       textView.append(c+"");}}
```

When posting or sending to **a Handler**, you can either allow the item to be processed as soon as the **message queue** is ready to do so, or specify a delay before it gets processed or absolute time for it to be processed. The latter two allow you to implement timeouts, ticks, and other timing-based behavior.

AsyncTask

```
MyHandler handler=new MyHandler();
Message m=handler.obtainMessage();
Bundle b=new Bundle();
b.putChar("ch",(char)i);
m.setData(b);
handler.sendMessage(m);
```

```
class MyHandler extends Handler{
   @Override
   public void handleMessage(@NonNull
Message msg) {
       super.handleMessage(msg);
        char c=msg.getData().getChar("ch");
        textView.append(c+"");}}
```

When posting or sending to **a Handler**, you can either allow the item to be processed as soon as the **message queue** is ready to do so, or specify a delay before it gets processed or absolute time for it to be processed. The latter two allow you to implement timeouts, ticks, and other timing-based behavior.

AsyncTask

- AsyncTask using three on any kind of generic data type for parameters as in shown example:
- String is the data type of passing data to the AsyncTask while its running via execute method.
- Character is the data type of published data from doInBackground() to onProgressUpdate()
- String is data type of returned value from doInBackground() method. It will be received by argument parameter of the method onPostExecute().

```
new AsyncTask <String,Character,String>(){
   @Override
   protected String doInBackground(String... strings) {
      for( i=0;i<270;i++){
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            e.printStackTrace();}
         publishProgress(new Character[]{(char)i});}
      return "finish"; }
   @Override
   protected void onProgressUpdate(Character... values) {
      super.onProgressUpdate(values);
      textView.append(values[0]+""); }
   @Override
   protected void onPostExecute(String s) {
      super.onPostExecute(s);
      textView.setText(s);
   }}.execute();
```

AsyncTask

- At least doInBackground() method is needed for create AsyncTask which should be implemented.
- Thread embedded in doInBackground() which means the code will run in parallel with others threads including Main Thread.
- Any code were written in others methods such as onPreExecute(), onPostExecute(), onProgressUpdate will be executed by Main Thread.
- By using AsyncTask no need to send data for Main Thread to change Ui, just use onProgressUpdate method.