Software Practice 3

- Today’s lecture
- Today’s Task

Prof. Hwansoo Han
T.A. Jeonghwan Park 43
MULTITHREAD
IN ANDROID
Activity and Service

before midterm after midterm

Dalvik VM

Main Thread

Looper

Message Queue

Broadcast Receiver

Activity

Activity

UI Events

System Events

Poll Threads

External Service Call

Local Service Call

Process

ARCS
Java Thread

- Thread is an execution stream of a process
  - Sharing a core between multiple threads
- Most of modern OS support multithread
- Android support multithread with some reasons
  - Each thread is dedicated for its own job
  - Separate foreground(UI) and background(other) thread
Multithread in Android

- Android manages UI with Main Thread and others with threads in Poll Threads
  - The job for other threads is called as external Service
  - The job for main thread which is not UI related work is called as local Service
Main Thread

- UI jobs are executed by special module for thread management, Looper
  - Looper manages not only UI related jobs, but also all foreground jobs
ASYNCTASK
AsyncTask

- a class that supports main thread work and background thread work at the same time

- Executed `AsyncTask` is an object
  - Cannot be reused
  - executed only once

- Does not be cancelled when an activity is quitted
  - Should be manually cancelled when an activity finished

- Only one `AsyncTask` can be run at a time
AsyncTask Workflow

1. aTask.execute()
2. onPreExecuted()
3. doInBackground()
4. publishProgress()
5. onProgressUpdate() : UI refresh
6. publishProgress()
7. onProgressUpdate() : UI refresh
8. return(result)
9. onPostExecuted()
Callback Methods of AsyncTask

- **onPreExecute()**
  - invoked on the **main thread** before the task is executed

- **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 result of computation must be returned by this step and will be passed back to the last step
Callback Methods of AsyncTask

- **onProgressUpdate(Progress...)**
  - invoked on the main 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 UI while the background computation is still executing

- **onPostExecute(Result)**
  - invoked on the main thread after the background computation finishes
Methods of AsyncTask

- `execute(Params...)`
  - Executes the task with the specified parameters
- `cancel(boolean)`
  - Attempts to cancel execution of this task
- `getStatus()`
  - Returns the current status of this task
- `isCancelled()`
  - Returns true if this task was cancelled before it completed normally, else false
Special Method of AsyncTask

- `publishProgress(Progress...)`
  - Trigger the execution of `onProgressUpdate(Progress...)` on the main thread
  - This method can be invoked from `doInBackground(Params...)` to publish updates on the main thread, especially UI, while the background computation is still running
Generic types in AsyncTask

- **AsyncTask** has three generic types which declare the data type of each parameter

```java
public class HelloAsyncTask extends AsyncTask<String, Void, String> {
    public String result;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected String doInBackground(String... params) {
        return result;
    }

    @Override
    protected void onPostExecute(String s) {
        super.onPostExecute(s);
    }
}
```
Example of AsyncTask

```java
/**
 * Param1: param of doInBackground(), UI thread -> background
 * Param2: param of onProgressUpdate(), background -> UI thread
 * Param3: return of doInBackground(), param of onPostExecute(),
 * background -> UI thread
 */

AsyncTask<Integer, Double, Strings> sleepTask = new AsyncTask<Integer, Double, Strings>() {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // Run in UI Thread
        // Something before background work
    }

    @Override
    protected String doInBackground(Integer... params) {
        // Run in Background
        return null;
    }

    @Override
    protected void onProgressUpdate(Double... values) {
        super.onProgressUpdate(values);
        // Run in UI Thread
        // Something while background work
        // Executed when we call "publishProgress(Param...)
    }

    @Override
    protected void onPostExecute(String aDouble) {
        super.onPostExecute(aDouble);
        // Run in UI Thread
        // Something after background work
    }
};
```
Example of AsyncTask

```java
final TextView tv = (TextView)findViewById(R.id.textview);
final Context mContext = this;

AsyncTask<Integer, Double, String> sleepTask = new AsyncTask<Integer, Double, String>() {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // Run in UI Thread
        Toast.makeText(mContext, "Start AsyncTask", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected String doInBackground(Integer... params) {
        // Run in Background
        for (int i=0; i<params[0]; i++) {
            try {
                Thread.sleep(1000); // sleep, need try-catch clause
                tv.setText(String.valueOf(values[i]));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    @Override
    protected void onPostExecute(String eDouble) {
        super.onPostExecute(eDouble);
        // Run in UI Thread
        Toast.makeText(mContext, "End AsyncTask", Toast.LENGTH_SHORT).show();
    }
};

sleepTask.execute(10);
```

An error is issued. How to revise it?
Example of AsyncTask

```java
final TextView tv = (TextView) findViewById(R.id.textView);
final Context aContext = this;

AsyncTask<Integer, Double, String> sleepTask = new AsyncTask<Integer, Double, String>() {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // Run in UI Thread
        Toast.makeText(aContext, "Start AsyncTask", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected String doInBackground(Integer... params) {
        // Run in Background
        for (int i = 1; i < params[0]; i++) {
            try {
                Thread.sleep(1000); // sleep, need try-catch clause
                publishProgress(i); // call onProgressUpdate()
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(Double... values) {
        super.onProgressUpdate(values);
        // Run in UI Thread
        tv.setText(String.valueOf(values[0])); // As it is UI thread, there is no error
    }

    @Override
    protected void onPostExecute(String sDouble) {
        super.onPostExecute(sDouble);
        // Run in UI Thread
        Toast.makeText(aContext, "End AsyncTask", Toast.LENGTH_SHORT).show();
    }

};

sleepTask.execute(10);
```
SERVICE
Service

- A Component that should be after an Activity stops
- Does not provide any UI to user
- Other Component can start a Service, and this Service will run in the background even though the parent Activity is moved onto other Activity
Be careful!!!

- Background does not mean multithreading

- **Service** run in background but is invoked on main thread

- If **Service** execute CPU intensive works, the application may cause “application not response” error
  - i.e., play music, network I/O

- To avoid this situation, you have to create new thread in **Service**
Service Workflow
Manifest of Service

- Like **Activity**, **Service** must be configured in Manifest file as following example
  - `<service android:name=".HelloService" />`
- If you need some intent-filter for your **Service**, you can define it as the case of **Activity**
Types of Service class

- **Service**
  - Base class for all services
  - Uses application’s main thread by default

- **IntentService**
  - Subclass of Service
  - Create a worker thread to handle all of the start request, one at a time
**Callback methods of Service**

- **onStartCommand()**
  - Invoke this method by calling `startService()` when another component requests that the service be started.
  - If you implement this, it is your responsibility to stop the service when its work is complete by calling `stopSelf()` or `stopService()`.

- **onBind()**
  - Invoke this method by calling `bindService()` when another component wants to bind with the service.
  - You must provide an interface that clients use to communicate with the service by returning an `Ibinder`.
  - You must always implement this method; however, if you don’t want to allow binding, you should return `null`.
Callback methods of Service

- **onCreate()**
  - Invokes this method to perform one-time setup procedures when the service is initially created
  - If the service is already running, this method is not called

- **onDestroy()**
  - Invokes this method when the service is no longer used and is being destroyed
  - Your service should implement this to clean up any resources such as threads, registered listeners, or receivers
  - This is the last call that the service receives
Start or Bind Service

- Both `startService()` and `bindService()` methods start the Service, but have different usage.
  - `startService()` creates Service which will not communicate with Activity.
  - `bindService()` creates intercommunicative Service with Activity.
Binder

- An object which enables to communicate between Service and Activity

- Let’s look at the example source code
Tip for Service in Background

- As previously mentioned, base Service is not invoked on background thread
- So, if you want to make your Service run on background thread, use Thread class

```java
Thread t = new Thread() {
    @Override
    public void run() {
        // do what you want
    }
};
t.start();

// other works

// invoke join() method when parent thread finishes it work,
// because child thread will be terminated when parent thread is ended.
t.join();
```
Callback methods of IntentService

- **onHandleIntent()**
  - Invoked on the worker thread with a request to process
  - Only one Intent is processed at a time, but the processing happens on a worker thread that runs independently from other application logic
    - So, if this code takes a long time, it will hold up other requests to the same IntentService, but it will not hold up anything else
  - When all requests have been handled, the IntentService stops itself, so you should not call stopSelf()
Example of IntentService

```java
public class HelloIntentService extends IntentService {

    /**
     * A constructor is required, and must call the super IntentService(String)
     * constructor with a name for the worker thread.
     */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /**
     * The IntentService calls this method from the default worker thread with
     * the intent that started the service. When this method returns, IntentService
     * stops the service, as appropriate.
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a file.
        // For our sample, we just sleep for 5 seconds.
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // Restore interrupt status.
            Thread.currentThread().interrupt();
        }
    }
}
```
[Lab – Practice #5]

- Modify given project to work in following condition
  - Currently application just use the Service as get the data of 777
  - Change this Service to be used for playing music
    - Music player code will be given
    - Music must be played when bind Service
    - Music must be stopped when unbind Service
  - If you add the pause and resume to your application, you will get extra credit
    - Hint) use mService variable!
// Initiate music with data from res
MediaPlayer mp = MediaPlayer.create(this, R.raw.sample);
// Start music
mp.start();
// Pause music
mp.pause();
int length = mp.getCurrentPosition();
// Resume music
// Resume music
mp.seekTo(length);
mp.start();
// Stop music
mp.stop();