Lec #6: Services and Broadcast Receivers

Tamer Nadeem
Dept. of Computer Science

Some slides adapted from Victor Matos and Stephen Intille
<table>
<thead>
<tr>
<th></th>
<th>Teams</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Droid Geeks</strong></td>
<td>Robert Lewis</td>
<td>Ashok Jonnalagadda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><strong>Infinite Loop</strong></td>
<td>Elliott Peay</td>
<td>Chris Coykendall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><strong>Energy Sniffer</strong></td>
<td>Mostafa Uddin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>Team Blizzard</strong></td>
<td>Tony Chiou</td>
<td>Timothy Werner</td>
<td>Jeremiah Dunn</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>MinDo</strong></td>
<td>Minhao Dong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>AppSycos</strong></td>
<td>Atsuyuki Harada</td>
<td>Chaintanya Reddy</td>
<td>Sheekha Khetan</td>
<td>Ujwal Manjunath</td>
</tr>
<tr>
<td>7</td>
<td><strong>App Double Screen</strong></td>
<td>Scott Seto</td>
<td>Pierre-Olivier Vanheekhoet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><strong>ODU Droid</strong></td>
<td>Mike Ruffing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td><strong>Jocote</strong></td>
<td>Zhonglu Wang</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td><strong>jhowland</strong></td>
<td>Zac Howland</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Objective

- **Working in Background**
  - Service
  - BroadcastReceiver
  - AsyncTask

- **Two Presentation**
  - Case Studies (7-13) - cont’d
    - Presenter: Ashok Kumar

  - Your Apps Are Watching You
    - Presenter: Elliott Peay
Android Application Components

- Activity
- Views
- Service
- Broadcast Receiver
- Content Provider
- Intents
- Widgets
Services
Services

• A Service is an application component that runs in the background, not interacting with the user, for an indefinite period of time.

• Higher priority than inactive Activities, so less likely to be killed
  • If killed, they can be configured to re-run automatically (when resources available)

• Each service class must have a corresponding `<service>` declaration in its package's `AndroidManifest.xml`
Services

• Services can be started with `Context.startService()` and `Context.bindService()`

• Execute in the main thread of the application’s process.
  • CPU intensive tasks must be offloaded to background threads using `Thread` or `AsyncTask`

• Alarms can be used to fire Intents at set times (by OS, not your app). These can start services, open Activities, or broadcast Intents
Services Lifecycle

- A Service has three lifecycle methods:
  1. void onCreate()
  2. void onStart(Intent intent)
  3. void onDestroy()
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {
    @Override
    public void onCreate() {
        // TODO: Actions to perform when service is created.
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO: Replace with service binding implementation.
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // TODO Launch a background thread to do processing.
        return Service.START_STICKY;
    }

    @Override
    public void onDestroy() {
        // TODO: Actions to perform when service is ended.
    }
}
onStartCommand

- Called whenever the Service is started with call to startService
  - So beware: may be executed several times in Service’s lifetime!
  - Controls how system will respond if Service restarted (START_STICKY)
  - Run from main GUI thread so standard pattern is to create a new Thread from onStartCommand to perform processing and stop Service when complete
Stickyness

• **START_STICKY**
  – onStartCommand called anytime service restarts

• **START_NOT_STICKY**
  – For service started to execute specific action/command
  – Use stopSelf to terminate service when command complete

• **START_RECEIVER_INTENT**
  – Combines both above
Stickyness

• You can use the parameter passed to startService to determine if the service is a system-based restart
  – Null
    • Initial call
  – START_FLAG_REDELIVERY
    • OS terminated the service before it was stopped by stopSelf
  – START_FLAG_RETRY
    • Service restarted after an abnormal termination when service was set to START_STICKY
Determining start condition

```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    if (((flags & START_FLAG_RETRY) == 0) {
        // TODO If it’s a restart, do something.
    } else {
        // TODO Alternative background process.
    }
    return Service.START_STICKY;
}
```
Starting a Service

• Call startService

// Implicitly start a Service
Intent myIntent = new Intent(MyService.ORDER_PIZZA);
myIntent.putExtra("TOPPING", "Margherita");
startService(myIntent);

// Explicitly start a Service
startService(new Intent(this, MyService.class));

(To use this example, would need to include a ORDER_PIZZA constant in MyService class and use an Intent Filter to register the Service as a provider of ORDER_PIZZA)
Stopping a Service

• Call stopService

```java
ComponentName service = startService(new Intent(this, BaseballWatch.class));

// Stop a service using the service name.
stopService(new Intent(this, service.getClass()));

// Stop a service explicitly.
try {
    Class serviceClass = Class.forName(service.getClassName());
    stopService(new Intent(this, serviceClass));
} catch (ClassNotFoundException e) {}
```
Binding Activities to Services

• Activity maintains reference to a Service
• Activity can make calls on the Service just as any other instantiated class
• To support this, implement onBind for the Service

```java
private final IBinder binder = new MyBinder();

@Override
public IBinder onBind(Intent intent) {
    return binder;
}

public class MyBinder extends Binder {
    MyService getService() {
        return MyService.this;
    }
}
```
Binding Activities to Services

• Once Service is bound, all public methods and properties are available through the serviceBinder object obtained from the onServiceConnected handler
private MyService serviceBinder; // Reference to the service

// Handles the connection between the service and activity
private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder service) {
        // Called when the connection is made.
        serviceBinder = ((MyService.MyBinder)service).getService();
    }

    public void onServiceDisconnected(ComponentName className) {
        // Received when the service unexpectedly disconnects.
        serviceBinder = null;
    }
};

@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    // Bind to the service
    Intent bindIntent = new Intent(MyActivity.this, MyService.class);
    bindService(bindIntent, mConnection, Context.BIND_AUTO_CREATE);
}
BroadcastReceivers
BroadcastReceiver

• Component that responds to system-wide broadcast announcements.

• Example system broadcasts: screen has turned off, the battery is low, user is present using phone, or a picture was captured.

• Applications can initiate broadcasts—e.g., to let other applications know that some data has been downloaded to the device and is available for them to use.

• Don’t display a UI, but can create a status bar notification to alert the user when a broadcast event occurs.
• Usually, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

  – Important: you must complete tasks in a BroadcastReceiver in <10s. If you have a task that will take longer, you must start a new thread to avoid app assassin OS.
BroadcastReceiver Lifecycle

The system delivers a broadcast Intent to all interested broadcast receivers, which handle the Intent sequentially.
BroadcastReceiver Lifecycle

A broadcast receiver has a single callback method:

```java
void onReceive(Context curContext, Intent broadcastMsg)
```

1. When a broadcast message arrives for the receiver, Android calls its `onReceive()` method and passes it the `Intent` object containing the message.

2. The broadcast receiver is considered to be active only while it is executing this method.

3. When `onReceive()` returns, it is inactive.
Example

• Starting Broadcast

```java
public static final String NEW_LIFEFORM_DETECTED = “com.paad.action.NEW_LIFEFORM”;

Intent intent = new Intent(NEW_LIFEFORM_DETECTED);
intent.putExtra(“lifeformName”, lifeformType);
sendBroadcast(intent);
```

• BroadcastReceiver

```java
import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.Context;

public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //TODO: React to the Intent received.
    }
}
```
The manifest of applications using Android Services must include:

1. A `<service>` entry for each service used in the application.

2. If the application defines a `BroadcastReceiver` as an independent class, it must include a `<receiver>` clause identifying the component. In addition an `<intent-filter>` entry is needed to declare the actual filter the service and the receiver use.
Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cs495.demos" android:versionCode="1" android:versionName="1.0.0">
    <uses-sdk android:minSdkVersion="4"></uses-sdk>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MyServiceDriver2">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MyService2" />
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="com.paad.action.NEW_LIFEFORM" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```
Native Broadcasts

• ACTION_BOOT_COMPLETED
• ACTION_CAMERA_BUTTON
• ACTION_DATE_CHANGED and ACTION_TIME_CHANGED
• ACTION_MEDIA_BUTTON
• ACTION_MEDIA_EJECT
• ACTION_MEDIA_MOUNTED and ACTION_MEDIA_UNMOUNTED
• ACTION_SCREEN_OFF and ACTION_SCREEN_ON
• ACTION_TIMEZONE_CHANGED

For comprehensive list:
http://code.google.com/android/reference/android/content/Intent.html
AsyncTask
Background threads

• To make app responsive, move all time-consuming operations off main app thread to child thread. **Very important!**

• Two options:
  – AsyncTask
  – Write own Threads
Creating AsyncTask

```java
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    @Override
    protected Long doInBackground(URL... urls) { //Background thread. Do not interact with UI
        int myProgress = 0; long result=0;
        // [... Perform background processing task, update myProgress ...]
        PublishProgress(myProgress)
        // [... Continue performing background processing task ...]
        // Return the value to be passed to onPostExecute
        return result;
    }

    @Override
    protected void onProgressUpdate(Integer... progress) { //Post interim updates to UI thread; access UI
        // [... Update progress bar, Notification, or other UI element ...]
    }

    @Override
    protected void onPostExecute(Long result) { //Run when doInBackground completed; access UI
        // [... Report results via UI update, Dialog, or notification ...]
        showDialog("Downloaded " + result + " bytes");
    }
}
new DownloadFilesTask().execute(url1, url2, url3);
```
Regular Thread

// This method is called on the main GUI thread.
private void mainProcessing() {
    // This moves the time consuming operation to a child thread.
    Thread thread = new Thread(doBackgroundThreadProcessing);
    thread.start();
}

// Runnable that executes the background processing method.
private Runnable doBackgroundThreadProcessing = new Runnable() {
    public void run() {
        [ ... Time consuming operations ... ]
    }
};
Important: power management

• Just because you have code in a BroadcastReceiver or Service doesn’t mean it will run if the phone goes into a low-power state

• Common problem: create a Broadcast receiver. Create a thread from within it to run code....

• All works fine when phone on and plugged into computer during development

• Fails under normal use because phone shuts down quickly in power management state

• Need to use a WakeLock!
WakeLock

• Control the power state on device (somewhat)

• Used to
  – Keep the CPU running
  – Prevent screen dimming or going off
  – Prevent backlight from turning on

• Only use when necessary and release as quickly as possible
WakeLock

- If you start a service or broadcast an Intent with the onReceive handler of a BroadcastReceiver, it is possible the WakeLock it holds will be released before your service has started! To ensure the service is executed you will need to put a separate WakeLock policy in place.
Creating a WakeLock

PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
WakeLock wakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKELOCK, "MyWakeLock");

wakeLock.acquire();
[ ... Do things requiring the CPU stay active ... ]
wakeLock.release();

PARTIAL_WAKELOCK keeps the CPU running without the screen on
Questions?
Recommended Assignment

• Example 1 (in this slides)

• Example 2 (in this slides)

• Assignment #4: Vision and Scope
Example 1. A very Simple Service

• The main application starts a service. The service prints lines on the DDMS LogCat until the main activity stops the service.

// a simple service is started & stopped
package cs495.demos;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;

public class ServiceDriver1 extends Activity {
    TextView txtMsg;
    Button btnStopService;
    ComponentName service;
    Intent intentMyService;
}
Example 1. cont’d

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

    intentMyService = new Intent(this, MyService1.class);
    service = startService(intentMyService);

    btnStopService = (Button) findViewById(R.id.btnStopService);
    btnStopService.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            try {
                stopService(intentMyService);
                txtMsg.setText("After stopingService: \n" + service.getClassName());
            } catch (Exception e) {
                Toast.makeText(getApplicationContext(), e.getMessage(), 1).show();
            }
        }
    });
}
```
//non CPU intensive service running the main task in its main thread
package cs495.demos;
Import android.app.Service;
Import android.content.Intent;
Import android.os.IBinder;
Import android.util.Log;

public class MyService1 extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i("<<MyService1-onStart>>", "I am alive-1");
    }

    @Override
    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
        Log.i("<<MyService1-onStart>>", "I did something very quickly");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.i("<<MyService1-onDestroy>>", "I am dead-1");
    }
}

//MyService1
Example 1. cont’d - Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cs495.demos"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".ServiceDriver1"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MyService1" />
    </application>
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```
Example 1. cont’d - Layout

```xml
<?xml version="1.0" encoding="utf-8"?>

<AbsoluteLayout
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="fill_parent"
        android:layout_height="120px"
        android:textSize="18sp"
        android:layout_x="0px"
        android:layout_y="57px"/>
    <Button
        android:id="@+id/btnStopService"
        android:layout_width="151px"
        android:layout_height="wrap_content"
        android:text="Stop Service"
        android:layout_x="43px"
        android:layout_y="200px"/>
</AbsoluteLayout>
```
Example 1. cont’d - Running

According to the Log
1. Main Activity is started (no displayed yet)
2. Service is started (onCreate, onStart)
3. Main Activity UI is displayed
4. User stops Service
Example 2. Realistic Activity-Service Interaction

1. The main activity starts the *service* and registers a *receiver*.

2. The service is slow, therefore it runs in a parallel thread its time consuming task.

3. When done with a computing cycle, the service adds a message to an intent.

4. The *intents* broadcasted using the filter: `cs495.action.DEMO2`

5. A *BroadcastReceiver* (defined inside the main Activity) uses the previous filter and catches the message (displays the contents on the main UI).

6. At some point the main activity stops the service and finishes executing.
Example 2. cont’d - Layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget32"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
    <EditText
        android:id="@+id-txtMsg"
        android:layout_width="fill_parent"
        android:layout_height="120px"
        android:textSize="12sp"
    />

    <Button
        android:id="@+id/btnStopService"
        android:layout_width="151px"
        android:layout_height="wrap_content"
        android:text="Stop Service"
    />
</LinearLayout>
```
Example 2. cont’d - Manifest

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cis493.demos"
    android:versionCode="1"
    android:versionName="1.0.0">
    <uses-sdk android:minSdkVersion="4"/>

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MyServiceDriver"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

        <service android:name="MyService"/>
    </application>
</manifest>
```
Example 2. cont’d – Main Activity

```java
// Application logic and its BroadcastReceiver in the same class
package cs495.demos;
import java.util.Date;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.SystemClock;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;

public class MyServiceDriver extends Activity {
    TextView txtMsg;
    Button btnStopService;
    Component Nameservice;
    Intent intentMyService;
    BroadcastReceiver receiver;
```
Example 2. cont’d – Main Activity

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

    intentMyService = new Intent(this, MyService.class);
    service = startService(intentMyService);

    txtMsg.setText("MyService started -(see DDMS Log)");
    btnStopService = (Button) findViewById(R.id.btnStopService);
    btnStopService.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            try{
                stopService(new Intent(intentMyService) );
                txtMsg.setText("After stopingService: \n"+ service.getClassName());
            } catch(Exception e) {
                e.printStackTrace();
            }
        }
    });

```
Example 2. cont’d – Main Activity

```java
// register & define filter for local listener
IntentFilter mainFilter = new IntentFilter("cs495.action.DEMO2");
receiver = new MyMainLocalReceiver();
registerReceiver(receiver, mainFilter);
} //onCreate

////////////////////////////////////////////////////////////////////////
@Override
protected void onDestroy() {
    super.onDestroy();
    try {
        stopService(intentMyService);
        unregisterReceiver(receiver);
    } catch (Exception e) {
        Log.e("MAIN-DESTROY>>>", e.getMessage());
    }
    Log.e("MAIN-DESTROY>>>", "Adios");
} //onDestroy
```
Example 2. cont’d – Main Activity

/////////////////////////////////////////////////////////////////////
// local (embedded) RECEIVER

public class MyMainLocalReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context localContext, Intent callerIntent) {
        String serviceData= callerIntent.getStringExtra("serviceData");
        Log.e("MAIN>>>", serviceData+ " -receiving data 
            + SystemClock.elapsedRealtime() );
        String now = "\n"+ serviceData+ " ---"
            + new Date().toLocaleString();
        txtMsg.append(now);
    }
} //MyMainLocalReceiver

} //MyServiceDriver
Example 2. cont’d – The Service

```java
// Service uses a thread to run slow operation
package cs495.demos;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {
    boolean isRunning = true;

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```
Example 2. cont’d – The Service

```java
@Override
public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
    Log.e("<<MyService-onStart>>", "I am alive!");
    // we place the slow work of the service in its own thread
    // so the caller is not hung up waiting for us
    Thread triggerService = new Thread (new Runnable() {
        long startingTime = System.currentTimeMillis();
        long tics = 0;
        public void run() {
            for (int i=0; (i< 120) & isRunning; i++) { //at most 10 minutes
                try {
                    //fake that you are very busy here
                    tics = System.currentTimeMillis() - startingTime;
                    Intent myFilteredResponse = new Intent("cs495.action.DEMO2");
                    String msg = i+ " value: "+ tics;
                    myFilteredResponse.putExtra("serviceData", msg);
                    sendBroadcast(myFilteredResponse);
                    Thread.sleep(1000); //five seconds
                } catch(Exception e) { e.printStackTrace(); }
            }
        }
    });
    triggerService.start();
}
```
Example 2. cont’d – The Service

```java
@Override
public void onDestroy() {
    super.onDestroy();
    Log.e("<<MyService-onDestroy>>", "I am dead");

    isRunning = false;
}
```

} //onDestroy

} //MyService

Stop thread