EMBEDDED SYSTEMS PROGRAMMING 2016-17

Android Services

APP COMPONENTS

- **Activity**: a single screen with a user interface
- **Broadcast receiver**: responds to system-wide broadcast events. No user interface
- **Service**: performs (in the background) long-running operations (e.g., music playback). No user interface
- **Content provider**

SERVICES: SUMMARY (1/3)

- Run in the background (indefinitely, if necessary) to perform
  - long-running operations
  - work for client processes
- Handled via the **`Service`** and **`IntentService`** abstract classes, plus the `Intent` class
- Declared in `AndroidManifest.xml`

SERVICES: SUMMARY (2/3)

- A service can operate
  - so that it can be stopped when not needed
  - so that it is never stopped, and even relaunched after a forced stop (that freed up resources)

- A service can be
  - private to the application that defines it,
  - available to other applications as well

SERVICES: SUMMARY (3/3)

- By default, an instance of `Service` runs in the main thread of its hosting process.
  It is possible — indeed, it is advisable — to create worker threads; instances of `IntentService` do so automatically

- It is possible to run the service in a separate process by specifying so in the manifest

SERVICES: KEY MANIFEST ATTRIBUTES

- **`android:enabled`** (boolean)
  Whether or not the service can be instantiated by the system

- **`android:exported`** (boolean)
  Whether or not components of other apps can invoke the service or interact with it

- **`android:permission`** (string)
  Name of a permission (e.g., a `signature` permission) that an entity must have in order to launch the service or bind to it

- **`android:process`** (string)
  Name of the process where the service is to run

USING A SERVICE

1. Implement the service
   as a subclass of `Service` or `IntentService`

2. Declare the service in the application's manifest

3. Start the service by invoking the **`startService`**
   and/or **`bindService`** methods of your component
   class

4. To communicate with the service: use intents,
   or bind to the service (**`bindService`** method)

COMMUNICATING WITH A SERVICE

- **Use intents**
- **Bind with the service and implement a `Binder`**
  Works only if the client and service are in the same application and process
- **Bind with the service and use a `Messenger`**
  Works even when the client and service are not in the same process, but the service can not multithread
- **Bind with the service and use IPC via AIDL**

SERVICE CLASS

- Base class for all Android services
- No UI; use **notifications** to interact with the user
- Remember: by default, no threads

SERVICE: STARTED, BOUND

- **Started service**
  Created by invoking the **`Context.startService(Intent service)`** method. The service performs the job specified by the intent and does not return a result to the caller. When the job is over, the service may 1) terminate itself or 2) wait for further jobs

- **Bound service**
  Created by invoking the **`Context.bindService(Intent service, ServiceConnection conn, int flags)`** method. Runs as long as at least one app component is bound to it. Offers a client-server, bidirectional communication interface that allows components to interact with the service, even across processes (IPC)

- A service can be started, bound, or both

SERVICE: KEY METHODS (1/3)

- **`void onCreate()`**
  Called by the system when the service is first created

- **`int onStartCommand(Intent intent, int flags, int startId)`**
  Called after another component has started the service by invoking `Context.startService(...)`. The integer `startId` is a unique token representing the start request. Must return a value that describes how the OS should continue the service after a kill (more about it later)

- **`IBinder onBind(Intent intent)`**
  Called after another component has requested to bind with the service by invoking `Context.bindService(...)`. Must return an interface that the component can use to communicate with the service

ONSTARTCOMMAND METHOD

- Must return an `int` that describes how the OS should continue the service in the event that it has been killed

- **`START_NOT_STICKY`**
  The OS does not recreate the service, unless there are pending intents to deliver. Any unfinished or pending jobs are lost: the app must manually restart them

- **`START_STICKY`**
  The OS recreates the service and calls `onStartCommand(...)` with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. Unfinished jobs are lost, pending jobs are not

- **`START_REDELIVER_INTENT`**
  The OS recreates the service and calls `onStartCommand(...)` with the last intent that was delivered to the service. Pending intents are delivered in turn

SERVICE: KEY METHODS (2/3)

- **`void startForeground(int id, Notification n)`**
  Notifies the OS that killing the service would be disruptive to the user. Supplies a `Notification` to be shown to the user while in this state

- **`void stopForeground(boolean removeNotificat)`**
  Removes the service from foreground state, allowing the OS to kill it more freely

STOPPING A SERVICE

- A component requests to stop a given service by invoking **`Context.stopService(Intent service)`**

- A service will not be destroyed as long as there are components bound to it with the `BIND_AUTO_CREATE` flag

- A component requests to unbound from a service by invoking **`Context.unbindService(ServiceConnection conn)`**

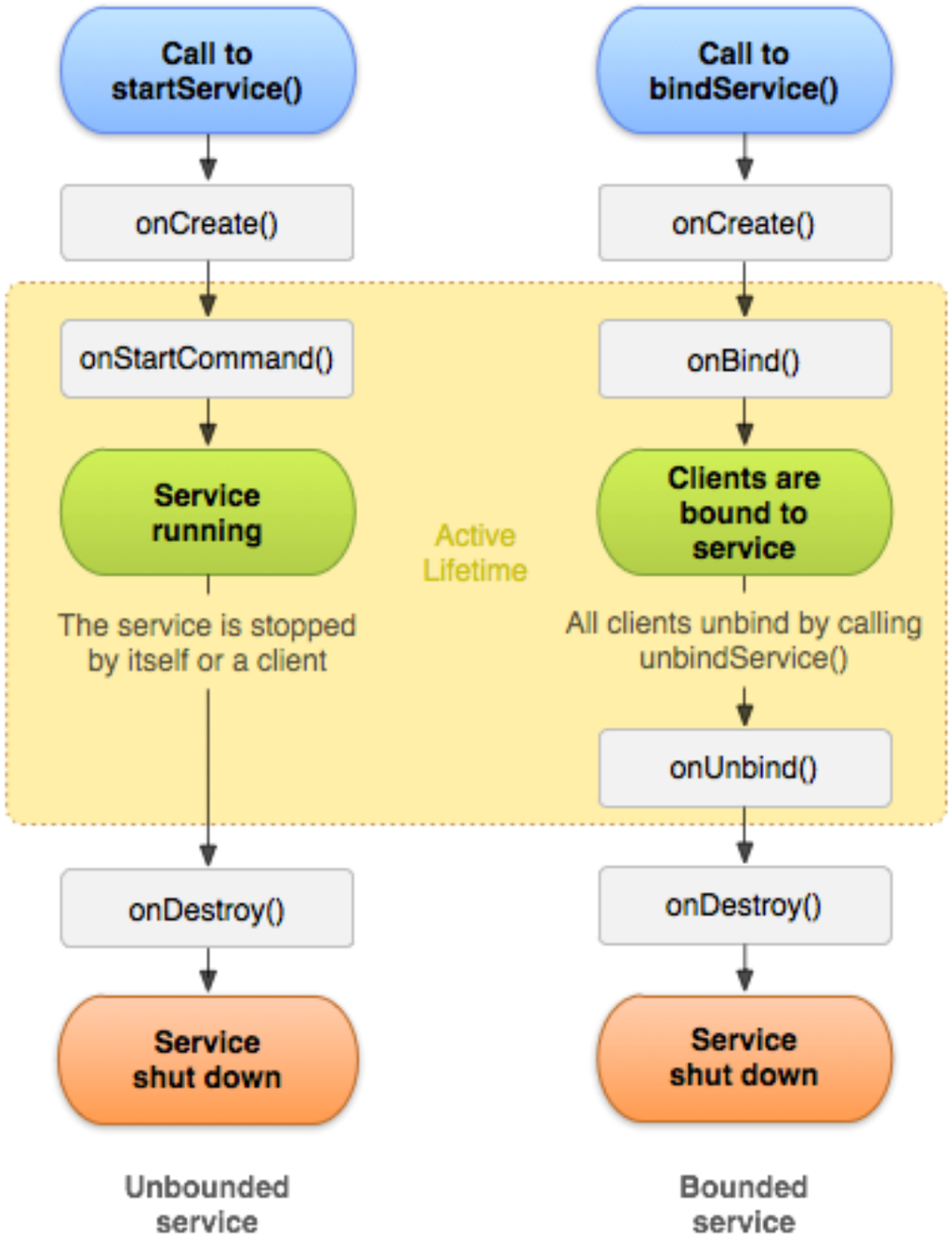
SERVICE: KEY METHODS (3/3)

- **`void stopSelf()`**
  Stops the service. The effect is the same as when a component invokes `Context.stopService(...)`

- **`boolean onUnbind(Intent intent)`**
  Called when all clients have disconnected from a particular interface published by the service. The default implementation does nothing

- **`void onDestroy()`**
  Called by the OS to notify the service that it is being removed. The service should clean up any resources it holds

SERVICE: LIFECYCLE

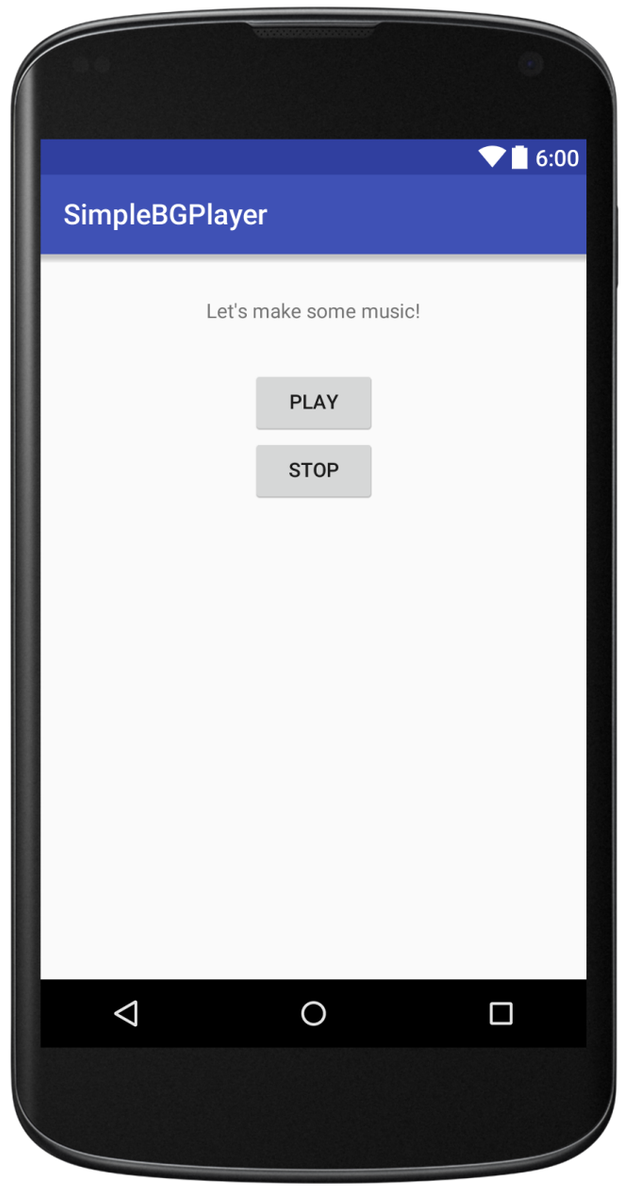


Picture: Google

SERVICE CLASS: EXAMPLE

- A started service that plays music in the background

- Main files:
  - `layout/activity_main.xml`
  - `MainActivity.java`
  - `PlayerService.java`
  - `AndroidManifest.xml`
  - `values/strings.xml`
  - `raw/doowackadoo.mp3`



SIMPLEBGPLAYER CODE (1/6)

- **layout/activity_main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="it.unipd.dei.esp1516.simplebgplayer.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/hello_world"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp" />
    <Button
        android:id="@+id/PlayButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/button_play"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp" />
    <Button
        android:id="@+id/StopButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="@string/button_stop"
        android:layout_below="@+id/PlayButton"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

SIMPLEBGPLAYER CODE (2/6)

- **values/strings.xml**

```
<resources>
    <string name="app_name">SimpleBGPlayer</string>
    <string name="hello_world">Let\'s make some music!</string>
    <string name="button_play">Play</string>
    <string name="button_stop">Stop</string>
</resources>
```

SIMPLEBGPLAYER CODE (3/6)

- **MainActivity.java**

```
package it.unipd.dei.esp1516.simplebgplayer;

import …

public class MainActivity extends AppCompatActivity {
    private Button bu_play, bu_stop;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Play button: starts the playback music service
        bu_play = (Button)findViewById(R.id.PlayButton);
        bu_play.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent i = new Intent(getApplicationContext(),PlayerService.class);
                i.putExtra(PlayerService.PLAY_START, true);
                startService(i);
            }
        });

        // Stop button: stops the music by stopping the service
        bu_stop = (Button)findViewById(R.id.StopButton);
        bu_stop.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent i = new Intent(getApplicationContext(),PlayerService.class);
                stopService(i);
            }
        });
    }
}
```

SIMPLEBGPLAYER CODE (4/6)

- **PlayerService.java** (1/2)

```
package it.unipd.dei.esp1516.simplebgplayer;

import …

public class PlayerService extends Service
{
    public static String PLAY_START = "BGPlayStart";
    public static String PLAY_STOP  = "BGPlayStop";

    private MediaPlayer myPlayer = null;
    private boolean isPlaying = false;

    @Override
    public IBinder onBind(Intent intent)
    {
        return null;    // Clients can not bind to this service
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        if(intent.getBooleanExtra(PLAY_START, false)) play();
        return Service.START_STICKY;
    }

    private void play()
    {
        if(isPlaying) return;

        isPlaying = true;

…
```

SIMPLEBGPLAYER CODE (5/6)

- **PlayerService.java** (2/2)

```
…
        // Music downloaded from "Public Domain 4U"
        // http://publicdomain4u.com/paul-whiteman-orchestra-doo-wacka-doo-mp3-download
        myPlayer = MediaPlayer.create(this, R.raw.doowackadoo);
        myPlayer.setLooping(true);
        myPlayer.start();

        // Runs this service in the foreground,
        // supplying the ongoing notification to be shown to the user
        Notification notification = new NotificationCompat.Builder(getApplicationContext())
                .setContentTitle("Paul Whiteman Orchestra")
                .setContentText("Doo Wacka Doo")
                .setSmallIcon(R.mipmap.ic_launcher)
                .build();
        final int notificationID = 5786423; // An ID for this notification unique within the app
        startForeground(notificationID, notification);
    }

    private void stop() {
        if (isPlaying)
        {
            isPlaying = false;
            if (myPlayer != null)
            {
                myPlayer.release();
                myPlayer = null;
            }
            stopForeground(true);
        }
    }

    @Override
    public void onDestroy() { stop(); }
}
```

SIMPLEBGPLAYER CODE (6/6)

- **AndroidManifest.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unipd.dei.esp1516.simplebgplayer">

    <application
        android:allowBackup="false"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".PlayerService"
            android:enabled="true"
            android:exported="false" />

    </application>

</manifest>
```

INTENTSERVICE CLASS (1/2)

- Android 1.6+
- Inherits from the `Service` class
- When a request (an `Intent`) arrives, it handles the request in a worker thread, and it automatically stops itself when the work is done
- A developer is basically required to implement only the **`onHandleIntent(Intent intent)`** method
- Requests are queued and processed one at a time

INTENTSERVICE CLASS (2/2)

More precisely, `IntentService` augments `Service`
by implementing the following operations

- It creates a worker thread that executes all intents delivered by invoking `startService(...)`
- It creates a work queue that passes to `onHandleIntent(...)` one intent at a time, so your code does not need to be thread-safe
- It provides a default implementation of `onStartCommand(...)` that sends each received intent to the work queue
- It stops the service after all start requests have been handled, so there is no need to call `stopSelf()`

INTENTSERVICE CLASS: EXAMPLE (1/2)

- Defining a started service

```
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");
    }

    // This method is called from within the 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)
    {
        // Do some work

        /*...*/
    }
}
```

Code: Google

INTENTSERVICE CLASS: EXAMPLE (2/2)

- Invoking a service from an activity with an explicit intent

```
Intent i = new Intent(this, HelloIntentService.class);
startService(i);
```

BOUND SERVICE

- Creates a long-standing connection with one or more app components
- A component connects to the service by invoking the **`Context.bindService(Intent service, ServiceConnection conn, int flags)`** method.
  The call returns `true` if the connection was established, `false` otherwise
- Via the **`conn`** object, the service
  - notifies the component when it is started or stopped,
  - provides an interface that specifies how the component can communicate with the service
- Once there are no components bound to the service, the system destroys the service

SERVICECONNECTION INTERFACE

- **`void onServiceConnected(ComponentName name, IBinder servicebinder)`**
  Called when a connection to the service has been established, with the **`servicebinder`** object describing the communication interface with the service

- **`void onServiceDisconnected(ComponentName name)`**
  Called when a connection to the service has been lost

BOUND SERVICE: COMMUNICATION INTERFACE

Several ways to provide an object implementing the `IBinder` interface

- Extend the Android **`Binder`** class
- Use a **Messenger**
  The service defines a `Handler` that is the basis for a `Messenger` that can then share an `IBinder` with the client, allowing the client to send commands to the service using `Message` objects. The client can define an additional `Messenger` of its own so the service can send messages back
- Use **AIDL**
  Create an .aidl file that defines the programming interface. The Android SDK tools use this file to generate an abstract class that implements the interface and handles IPC, which you can then extend within the service

EXTENDING THE BINDER CLASS

The `onBind()` method of the service returns an instance of a `Binder`-derived class that either:

1. contains public methods that the client can call

2. returns the current instance of the class implementing the service, which has public methods the client can call

3. returns an instance of another class hosted by the service with public methods the client can call

BOUND SERVICE: EXAMPLE (1/4)

```
public class LocalService extends Service
{
    // Binder given to clients; LocalBinder is an inner class (see below)
    private final IBinder mBinder = new LocalBinder();

    // Class used for the client Binder
    public class LocalBinder extends Binder
    {
        LocalService getService()
        {
            // Return this instance of LocalService
            // so clients can call public methods
            return LocalService.this;
        }

        // SOLUTION 1: public binder method that clients can call
        public int doSomething_Binder() { /*...*/ }
    }

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

    // SOLUTION 2: public service method that clients can call
    public int doSomething_Service() { /*...*/ }
}
```

BOUND SERVICE: EXAMPLE (2/4)

- A simple activity that binds to `LocalService`

```
public class BindingActivity extends Activity
{
    LocalService mService;
    boolean mBound = false;

    // Callbacks for service binding (ServiceConnection interface)
    private ServiceConnection mConnection = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName className, IBinder service)
        {
            // Cast the IBinder and get LocalService instance
            LocalBinder binder = (LocalBinder) service;
            mService = binder.getService();
            mBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) { mBound = false; }
    };

…
```

Code: Google

BOUND SERVICE: EXAMPLE (3/4)

- A simple activity that binds to `LocalService`

```
…
    @Override
    protected void onStart()
    {
        super.onStart();
        // Bind to the service
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop()
    {
        super.onStop();
        // Unbind from the service
        if (mBound)
        {
            unbindService(mConnection);
            mBound = false;
        }
    }
…
```

Code: Google

BOUND SERVICE: EXAMPLE (4/4)

- A simple activity that binds to `LocalService`

```
…
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    // Called when a button is clicked (the button in the layout file
    // attaches to this method with the android:onClick attribute)
    public void onButtonClick(View v)
    {
        if (mBound)
        {
            // Call a method from the LocalService. If this call were something
            // that might hang, then this request should occur in a separate
            // thread to avoid slowing down the activity performance
            int num = mService.doSomething_Service();
            Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show();
        }
    }
}
```

Code: Google

REFERENCES

- Services
  - Bound Services
  - AIDL
- Running in a Background Service

LAST MODIFIED: MAY 15, 2017