Android Service

Dac Hoang Nguyen

Introduction

- Your role
- Your background and experience in the subject:
  - Java.
  - Basic Android.
- What do you want from this course

Course Objectives

- At the end of the course, you will have acquired sufficient knowledge to:
  - **Android Service Overview**.
  - **Started Service.**
  - **Bound Service.**

Agenda

Set Up Environment

- To complete the course, your PC must install:
  - Eclipse with Android plugins
  - Android SDK

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Provide your feedback in the End of Course Evaluation

Assessment Disciplines

- Class Participation: 40%
- Assignment: 60%
- Final Exam: 0%
- Passing Scores: 70%

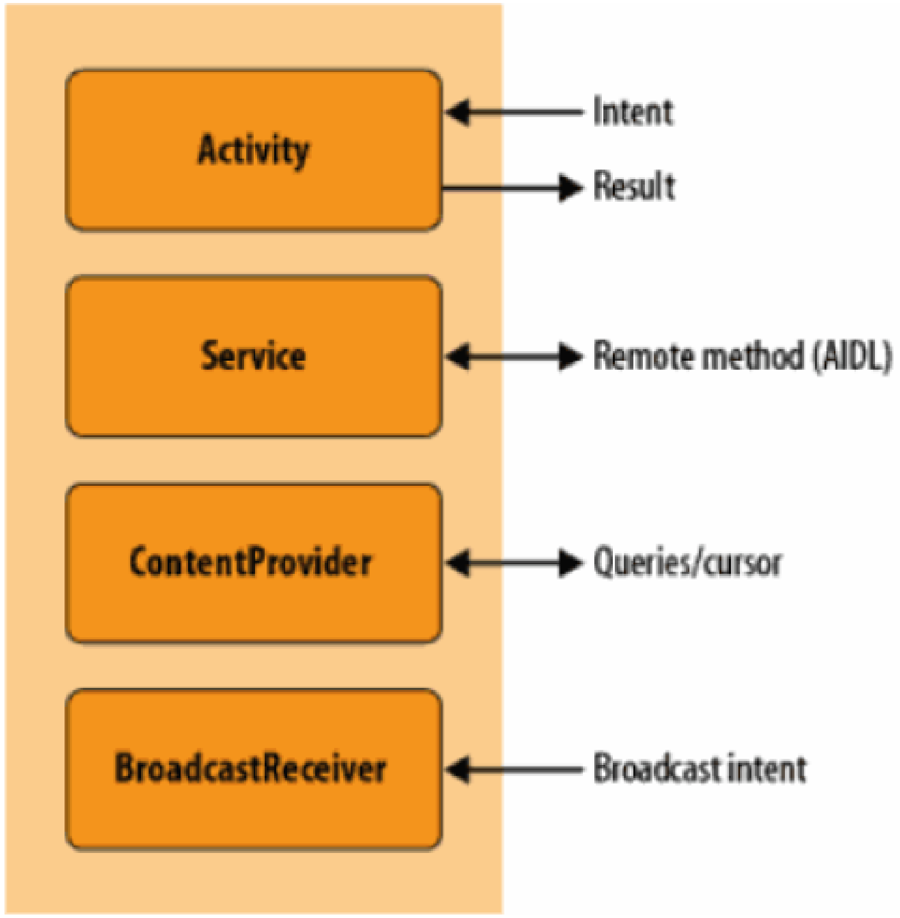
Android Service Overview

Android Service Overview

- What is a Service?
- Forms of services.
- Lifecycle of Services.
- Creating a Service.

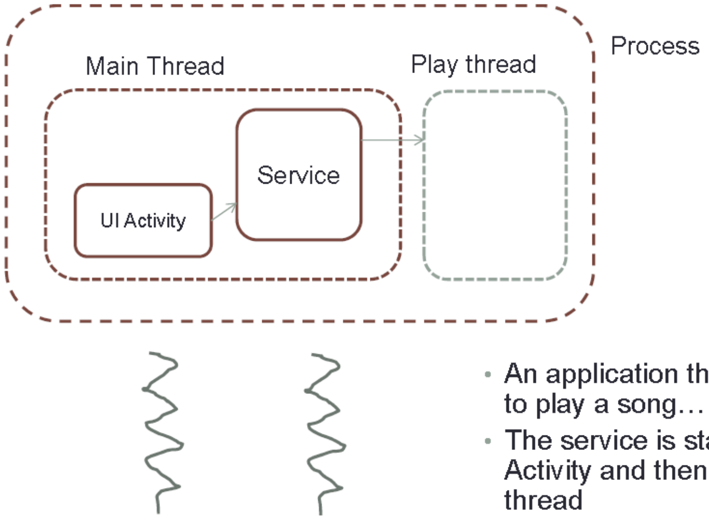
What is an Android Service?

- One of the four primary application components.
- No user interface.
- Two main uses
  - Performing long-running operations in the background
  - Supporting remote method execution
- Managed by android and has its own lifecycle.
- Running in the main thread of its hosting process



Forms of Android Services

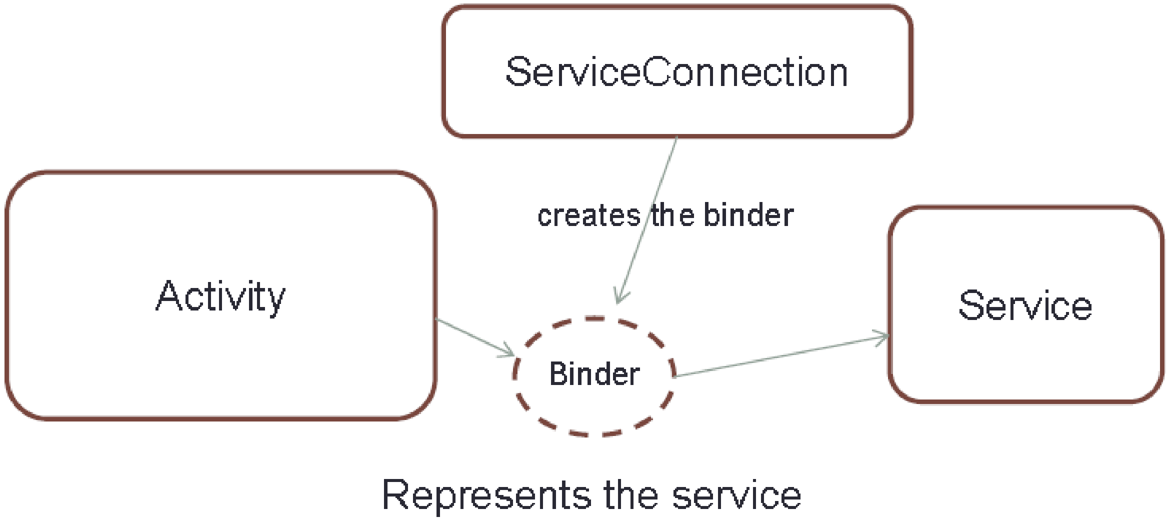
- Stated:
  - Application component, such as an Activity, starts the service with the method call **startService()**.
  - Once started service can run in background indefinitely.
  - Generally services do not return a result (see bound service).
  - Service should stop itself when done.
  - For example: playing music



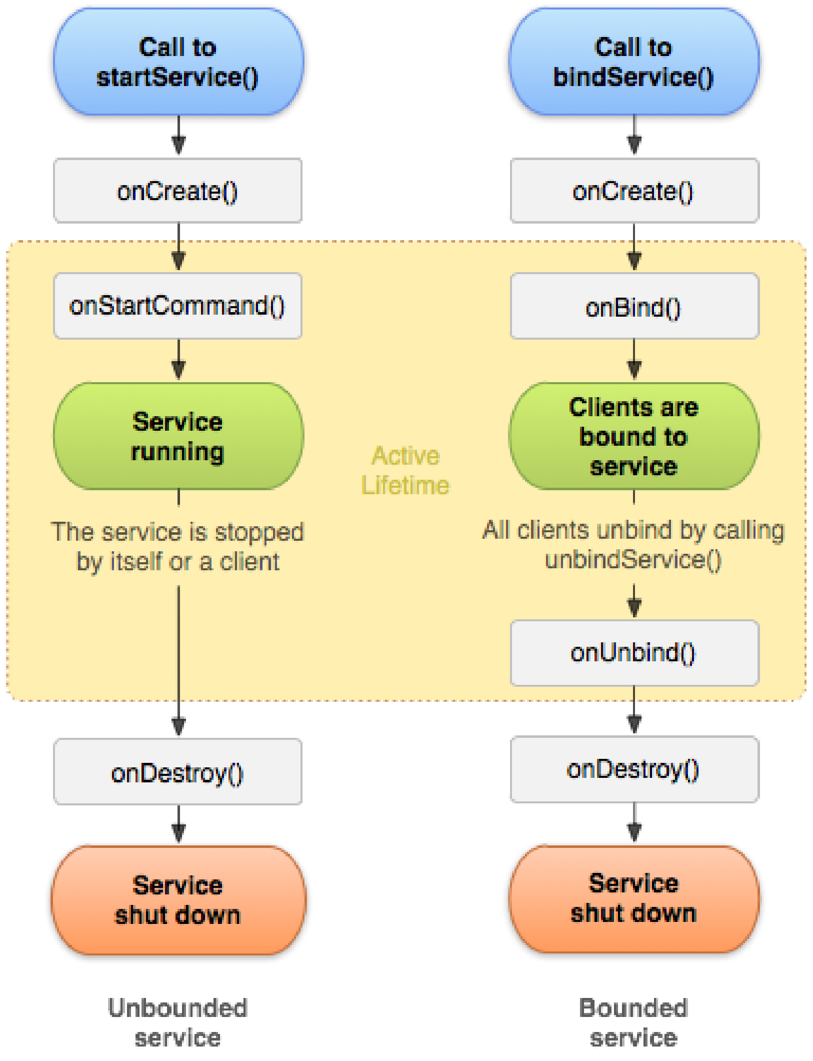
- An application that runs a player to play a song…
- The service is started from the Activity and then it spawns a thread

Forms of Android Services

- Bound
  - Application component binds itself to existing service via the **bindService()** method.
  - Bound service provides client-server interface that allows application component to interact with service.
  - Interact with service, send requests, get result via IPC (inter process communication.
  - Service runs as long as one or more applications bound to it.
  - Destroyed when no applications bound.

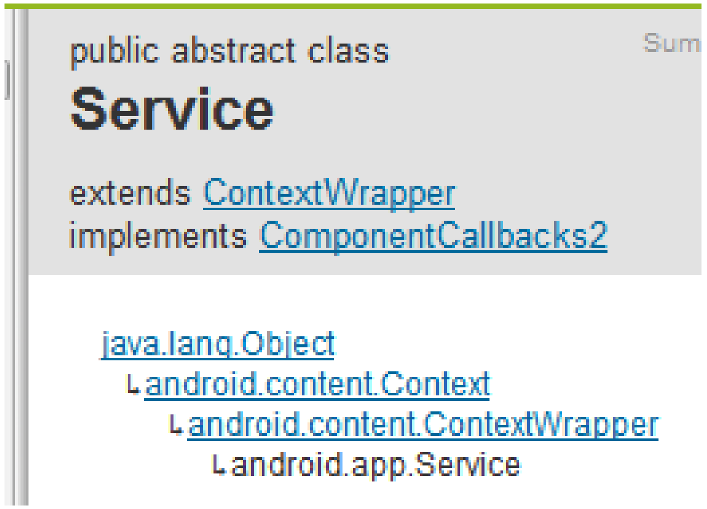


Lifecycle of Android Services



Creating a Service

- Create subclass of Android Service class or one of its existing subclasses.
- Override callback methods that handle important aspects of service lifecycle. Most important of these are:
  - onStartCommand().
  - startService().
  - onBind().
  - onCreate().
  - onDestroy().
  - stopSelf().
  - stopService().



public abstract class
Service
extends ContextWrapper
implements ComponentCallbacks2

```
java.lang.Object
  ↳android.content.Context
    ↳android.content.ContextWrapper
      ↳android.app.Service
```

- Include the Service in an AndroidManifest.xml file

```
<service android:name="com.tutorial.service.services.simple.SimpleService" />
```

Started Service

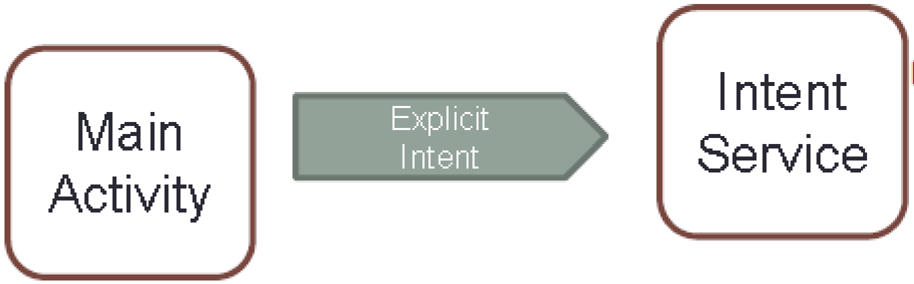
Started Service

- A Started Service is activated via **Context.startService()**.
  - The Intent identifies the service to communicate with & supplies parameters (via Intent extras) to tell the service what to do.
- **startService()** does not block.
  - If the service is not already running it will be started & will receive the Intent via **onStartCommand()**.
  - Services often perform a single operation & terminate themselves.
  - Running Services can also be halted via **stopService()**.
- Started Services don't return results to callers, but do return values via **onStartCommand()**.

Started Services

- Extending the IntentService class
- Extending the Service class

Extending the IntentService class



```
protected void onHandleIntent(Intent intent)
```

- The service needs to be registered in the manifest file.
- The main activity creates an explicit intent pointing to the service.
- The service is started and the ***onHandleIntent()*** *method executed.*
- Intents are queued and served serially.

Extending the IntentService class: example

```java
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;

public class MainActivity extends Activity {

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

        Intent intent = new Intent(this, SimpleIntentService.class);
        startService(intent);
    }
}
```

```java
import android.app.IntentService;
import android.content.Intent;
import android.util.Log;


public class SimpleIntentService extends IntentService{

    public SimpleIntentService() {
        super("SimpleIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i("info", "Intent Service started");
    }
}
```

```xml
<service android:name="com.csc.androidservice.test.SimpleIntentService" />
```

Extending the Service class: example

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

public class SimpleService extends Service {

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

    @Override
    public void onCreate() {
        Log.i("info", "Service onCreate");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        Log.i("info", "Service onStartCommand");

        for (int i = 0; i < 3; i++) {
            long endTime = System.currentTimeMillis() + 10 * 1000;
            while (System.currentTimeMillis() < endTime) {
                synchronized (this) {
                    try {
                        wait(endTime - System.currentTimeMillis());
                    }
                    catch (Exception e) {
                    }
                }
            }
            Log.i("info", "Service running");
        }
        return Service.START_STICKY;
    }

    @Override
    public void onDestroy() {
        Log.i("info", "Service onDestroy");
    }
}
```

Assignment 1

Assignment 1

- Create a Music Player with play and stop buttons.

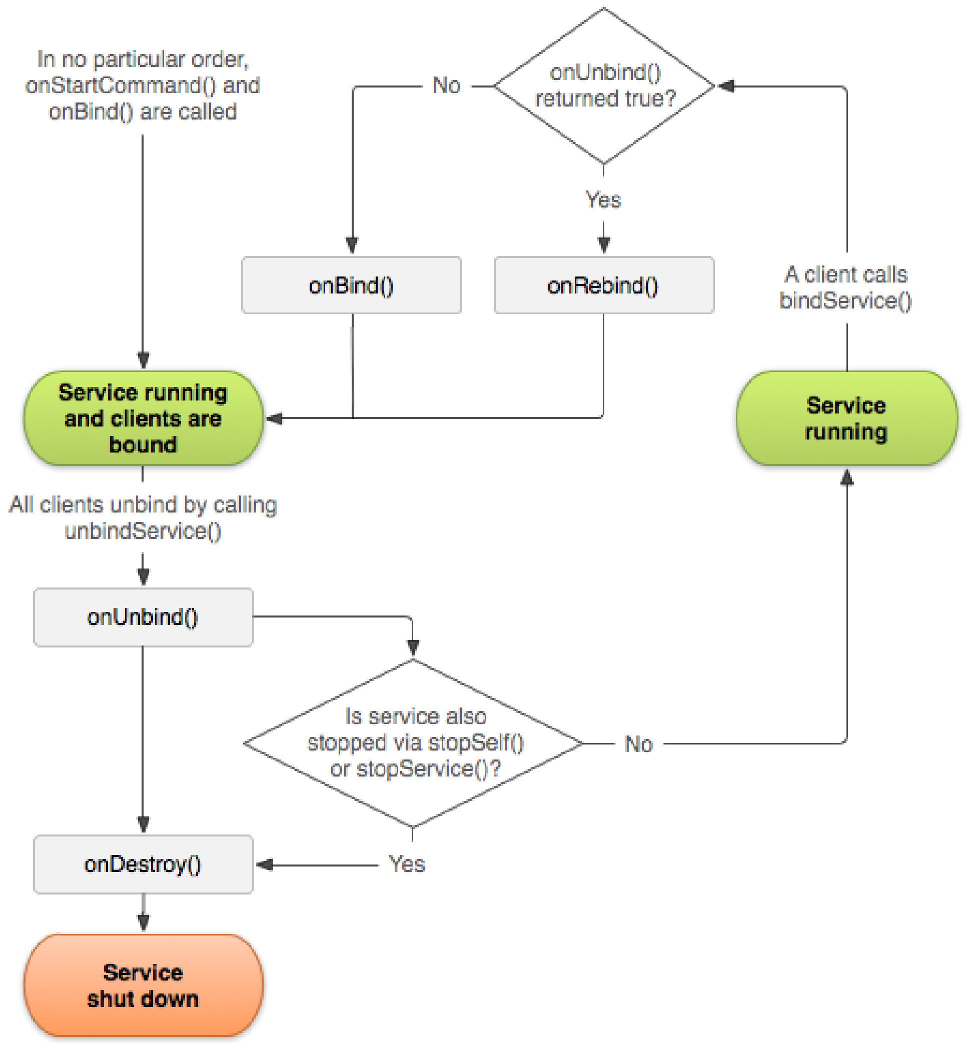
Bound Service

Bound Service

- Lifecycle of a Bound Service
- Bound Service Interactions
- Creating a Bound Service

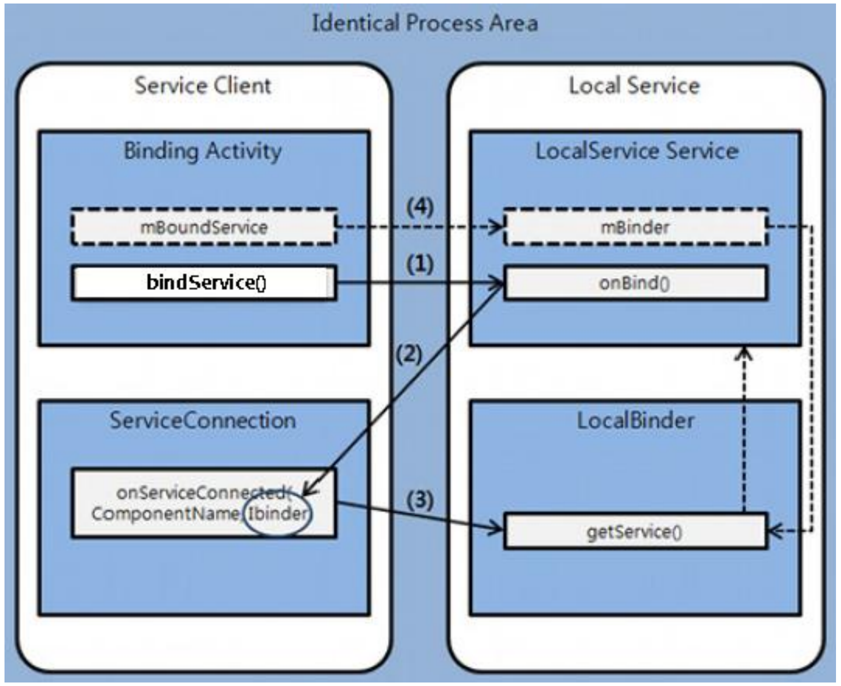
Lifecycle of a Bound Service

- A Bound Service is the server in a client/server interface
  - A Bound Service allows components (such as Activities) to interact with the service
  - send requests
  - get results &
  - converse across processes via IPC
- A Bound Service typically lives only while it serves other application component(s)


In no particular order, onStartCommand() and onBind() are called
onUnbind() returned true?
No
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?
No
Yes
onDestroy()
Service shut down

Bound Service Interactions

- Application components (clients) calls **bindService()** to bind to a service.
- The Android system then calls the service's **onBind()** method, which returns an **IBinder** for interacting with the service.
  - The binding is asynchronous - **bindService()** returns immediately & does not return the **IBinder** to the client.
- To receive the **IBinder**, the client must create an instance of **ServiceConnection** & pass it to **bindService()**.
  - The **ServiceConnection** includes a callback method that the system calls to deliver the **Ibinder**.



CSC

Bound Service Interactions (cont'd)

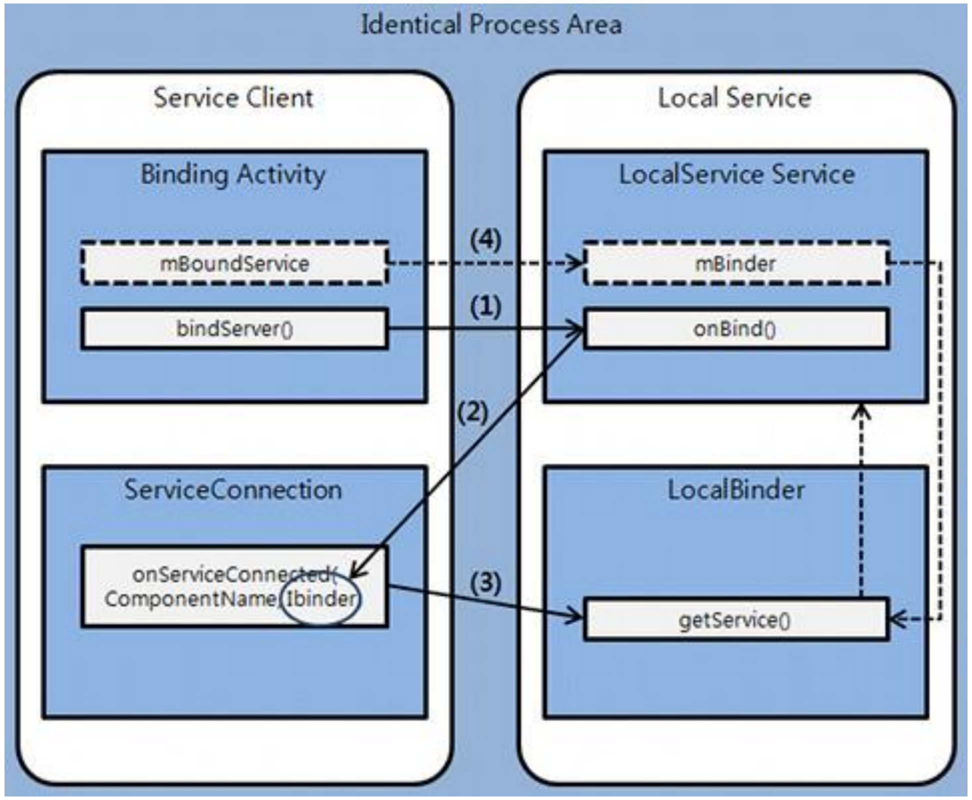
- To bind to a service from your client, you must:
  - Implement **ServiceConnection** & must override two callback methods: **onServiceConnected()** − Android calls this to deliver the **IBinder** returned by the service's **onBind()** method.
  - **onServiceDisconnected()** − Android calls this when the connection to the service is unexpectedly lost, such as when the service has crashed or has been killed (not called with client calls **unbindService()**).
  - Call **bindService()**, passing the **ServiceConnection** implementation.
  - When the system calls your **onServiceConnected()** callback method, you can begin making calls to the service, using the methods defined by the interface
  - To disconnect from the service, call **unbindService()**.
- When your client is destroyed, it will unbind from the service Always unbind when you're done interacting with the service or when your activity pauses so that the service can shutdown while its not being used.

Creating a Bound Service

- When creating a Bound Service, you must provide an **IBinder** via an interface clients can use to interact with the Service via the following:
  - **Extending the Binder class**
    - If your service runs in the same process as the client you can extend the Binder class & return an instance from **onBind()**.
  - **Using a Messenger**
    - Create an interface for the service with a Messenger that allows the client to send commands to the service across processes using Message objects
    - Doesn't require thread-safe components
  - **Using Android Interface Definition Language (AIDL)**
    - AIDL performs all the work to decompose objects into primitives that the operating system can understand & marshal them across processes to perform IPC
    - Does require thread-safe components

Extending the Binder class

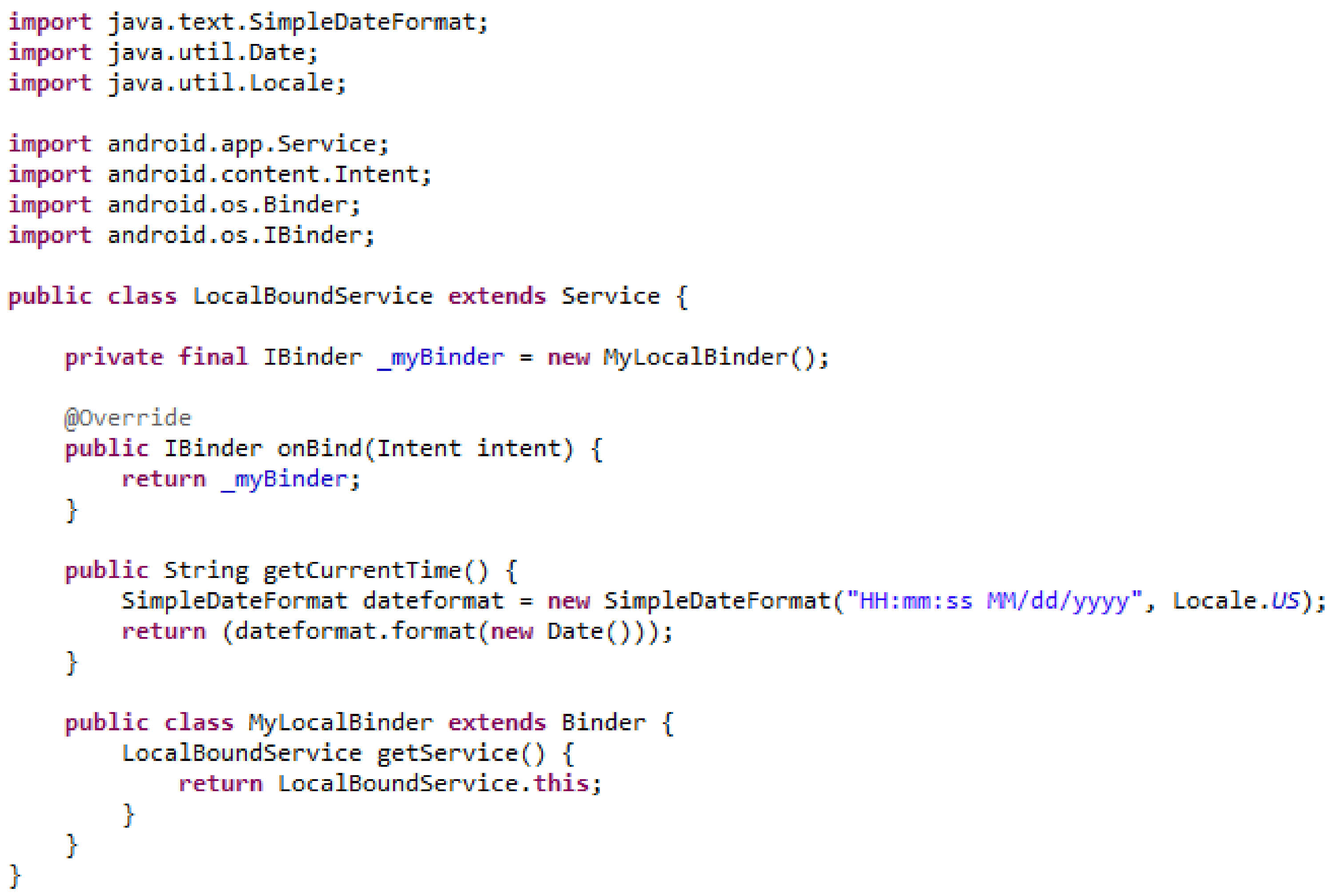
- Sometimes a service is used only by the local application & need not work across processes.
- In this case, you can implement your own Binder subclass that provides your client direct access to public methods in a service.



How to Extend the Binder class

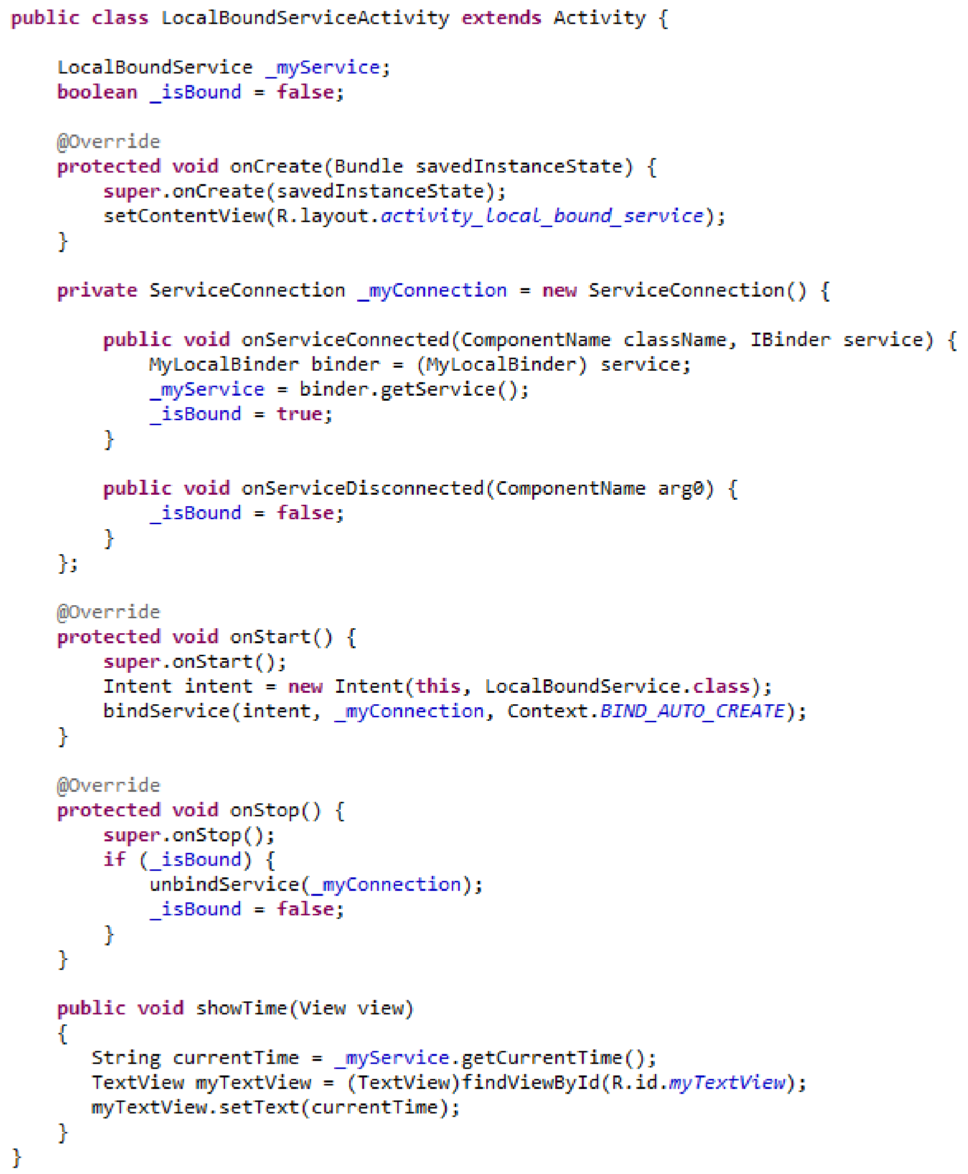
- Here's how to use locally Bound Service:
  - In your service, create an instance of Binder that either:
    - Contains public methods that the client can call.
    - Returns the current Service instance, which has public methods the client can call.
    - Returns an instance of another class hosted by the service with public methods the client can call.
  - Return this instance of Binder from the **onBind()** callback method.
  - In the client, receive the Binder from the **onServiceConnected()** callback method and make calls to the bound service using the methods provided.

Example of Extending the Binder class



```java
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;

public class LocalBoundService extends Service {

    private final IBinder _myBinder = new MyLocalBinder();

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

    public String getCurrentTime() {
        SimpleDateFormat dateformat = new SimpleDateFormat("HH:mm:ss MM/dd/yyyy", Locale.US);
        return (dateformat.format(new Date()));
    }

    public class MyLocalBinder extends Binder {
        LocalBoundService getService() {
            return LocalBoundService.this;
        }
    }
}
```

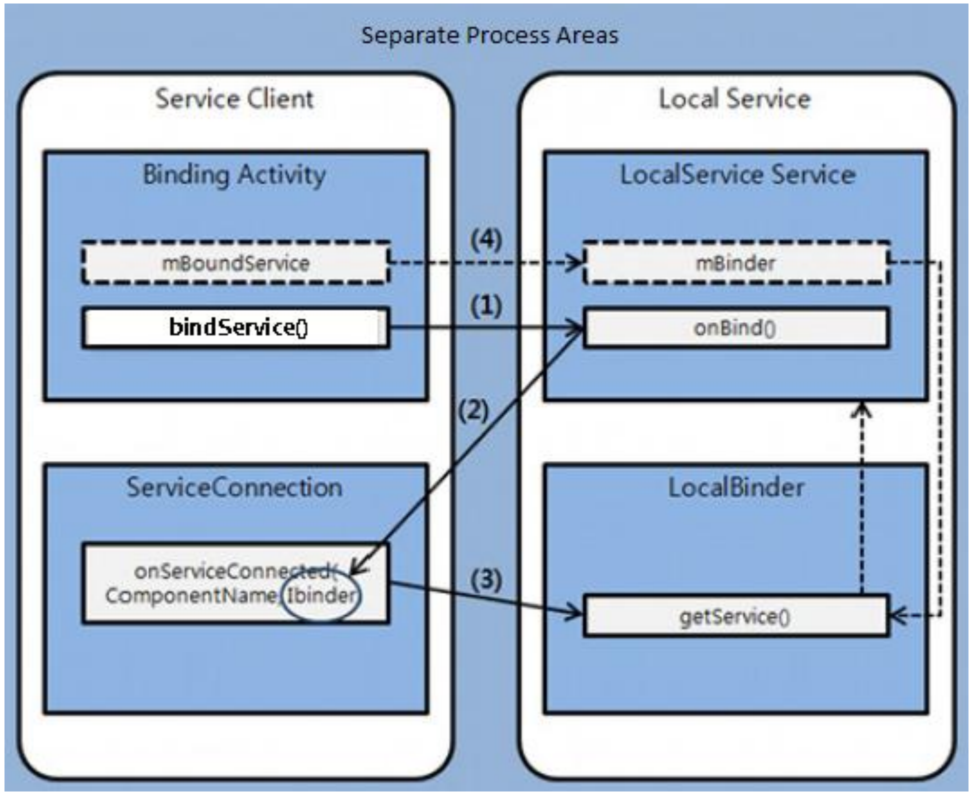
Example of Extending the Binder class (cont'd)



```java
public class LocalBoundServiceActivity extends Activity {

    LocalBoundService _myService;
    boolean _isBound = false;

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

    private ServiceConnection _myConnection = new ServiceConnection() {

        public void onServiceConnected(ComponentName className, IBinder service) {
            MyLocalBinder binder = (MyLocalBinder) service;
            _myService = binder.getService();
            _isBound = true;
        }

        public void onServiceDisconnected(ComponentName arg0) {
            _isBound = false;
        }
    };

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, LocalBoundService.class);
        bindService(intent, _myConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (_isBound) {
            unbindService(_myConnection);
            _isBound = false;
        }
    }

    public void showTime(View view)
    {
       String currentTime = _myService.getCurrentTime();
       TextView myTextView = (TextView)findViewById(R.id.myTextView);
       myTextView.setText(currentTime);
    }
}
```

Using a Messenger

- You can use a Messenger to communicate with a Bound Service in another process.
- This technique allows you to perform IPC between Activities & Services without the need to use AIDL (which is more complicated).



How to Use a Messenger

- Here’s how to use a Messenger:
  - The service implements a **Handler** that receives a callback for each call from a client.
  - The **Handler** is used to create a **Messenger** object (which is a reference to the **Handler**).
  - The Messenger creates an **IBinder** that the service returns to clients from **onBind()**.
  - Clients use the **IBinder** to instantiate the **Messenger** (that references the service's **Handler**), which the client uses to send **Message** objects to the service.
  - The service receives each **Message** in its **Handler**, in the **handMessage()** method.
- Two-way messaging is a slight variation on this.

Example of Using a Messenger

```
public class MessengerService extends Service {

    // Command to service to display msg
    static final int MSG_SAY_HELLO = 1;

    // Handler of incoming client msgs
    class IncomeHandler extends Handler {
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case MSG_SAY_HELLO:
                Toast.makeText(getApplicationContext(), "hello!", Toast.LENGTH_SHORT);
                break;
            default:
                super.handleMessage(msg);
            }
        }
    }

    // Target we publish for clients to send messages to IncomeHandler
    final Messenger mMessenger = new Messenger(new IncomeHandler());

    // When binding to the service, we return an interface to our messenger
    // for sending messages to the service
    public IBinder onBind(Intent intent) {
        return mMessenger.getBinder();
    }
}
```

Example of Using a Messenger (cont’d)

```java
public class MessengerServiceActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_messenger_service);
    }

    /** Messenger for communicating with the service. */
    Messenger mService = null;
    boolean mBound;

    /** Class for interacting with the main interface of the service. */
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            // This is called when the connection with the service has been
            // established, giving us the object we can use to
            // interact with the service.  We are communicating with the
            // service using a Messenger, so here we get a client-side
            // representation of that from the raw IBinder object.
            mService = new Messenger(service);
            mBound = true;
        }
        public void onServiceDisconnected(ComponentName className) {
            mService = null;
            mBound = false;
        }
    };

    public void sayHello(View v) {
        if (!mBound) return;
        // Create and send a message to the service, using a supported 'what' value
        Message msg = Message.obtain(null, MessengerService.MSG_SAY_HELLO, 0, 0);
        try {
            mService.send(msg);
        }
        catch (RemoteException e) { }
    }

    @Override
    protected void onStart() {
        super.onStart();
        bindService(new Intent(this, MessengerService.class), mConnection,
                Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }
}
```

Assignment 2

Assignment 2

- Create a Music Player with play, pause and stop buttons, using Bound Service and Messenger.

Refferences

- Android Service - http://developer.android.com/guide/components/services.html
- Bound Service - http://developer.android.com/guide/components/bound-services.html
- Running in a Background Service - http://developer.android.com/training/run-background-service/index.html

Q&A

Thank You

CSC

BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING