Mobile Apps Development

COMP-304

Fall 2018

Review of Lecture 10

- SMS Messaging
- To send an SMS message programmatically, you use the **SmsManager** class:

SmsManager sms = **SmsManager**.*getDefault();*

sms.**sendTextMessage**(phoneNumber, **null, message, null, null);**

- Create two **PendingIntent** objects to **monitor the status of the SMS message-sending process**

- Create and register two BroadcastReceivers in the **onResume()** method:

registerReceiver(smsDeliveredReceiver, **new IntentFilter(DELIVERED));**
registerReceiver(smsSentReceiver, **new IntentFilter(SENT));**

The two PendingIntent objects are passed into the last two arguments of the **sendTextMessage()** method:

SmsManager sms = SmsManager.*getDefault();*
sms.**sendTextMessage**(phoneNumber, **null, message, sentPI, deliveredPI);**

Review of Lecture 10

- To listen for incoming SMS messages, you create a **BroadcastReceiver** subclass
  - Override the **onReceive()** method
  - To **extract the content of each message**, you use the static **createFromPdu()** method of **SmsMessage** class.

- Sending the message back:
  - In the **onReceive** method:

```
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
```

Review of Lecture 10

- Using the **Android Web API**
  - use the **WebView** control to **display web content** to the screen.
  - WebView control uses the **WebKit** rendering engine to draw HTML content on the screen
  - WebView control requires the internet **permission**.
  - use **loadUrl** method to load content into a WebView control
- To render raw HTML, you can use the **loadData()** method
- Change the scale of the web content to fit the page within the WebView control:

  wv.**setInitialScale**(30);
- Modify the behavior of the control: the **WebSettings** class, the **WebViewClient** class, and the **WebChromeClient** class:

  WebSettings settings = wv.**getSettings()**;

  settings.**setJavaScriptEnabled(true);**

Review of Lecture 10

- **WebViewClient** class enables the application to listen for certain WebView events:
  - use **WebViewClient** to handle the **onPageFinished()** method to draw the title of the page on the screen, etc
  - implement the WebViewClient class and override the **shouldOverrideUrlLoading()** to prevent device's browser from loading the page

Developing Android Services

Objectives:

- ❑ **Explain Android Service API.**
- ❑ **Create custom services by performing long running tasks in a Service.**
- ❑ **Execute asynchronous tasks on separate threads using IntentService, and stablish communication between a Service and an Activity.**

Android Services

- ❑ Android offers the **Service** class to create application components that handle **long-lived operations** and include functionality that **doesn’t require a user interface**.
- ❑ Services are **started**, **stopped**, and controlled from other application components, including Activities, Broadcast Receivers, and other Services
- ❑ Running Services have a **higher priority than inactive or invisible (stopped) Activities**, making them less likely to be terminated by the run time’s resource management

Creating your own Services

- Steps to create a simple service:
  - Define a class that extends the **Service** base class:

```
public class MyService extends Service {
}
```

  - Within the MyService class, **override** the following methods

```
@Override
public IBinder onBind(Intent arg0) { ... }
@Override
public int onStartCommand(Intent intent, int flags, int
    startId) { ... }
@Override
public void onDestroy() { ... }
```

Creating your own Services

- The **onBind()** method enables you to **bind an activity to a service**.
  - This in turn enables an activity to **directly access members and methods inside a service**.
- The **onStartCommand()** method is **called when you start the service** explicitly using the **startService()** method.
  - This method signifies the start of the service, and you code it to do the things you need to do for your service.
- The **onDestroy()** method is **called when the service is stopped** using the **stopService()** method.
  - This is where you clean up the resources used by your service.

Creating your own Services

- All services **must be declared in the AndroidManifest.xml** file, like this:

  <service android:name=".MyService" />

- If you want your service to be available to other applications, you can always add an intent filter with an action name, like this:

  <service android:name="***.MyService***">
  <intent-filter>
  <action android:name=*"net.learn2develop.MyService"* />
  </intent-filter>
  </service>

Creating your own Services

- To start a service, you use the **startService()** method, like this:

```
startService(new Intent(getBaseContext(),
    MyService.class));
```

- If you are calling this service from an external application, then the call to the **startService()** method looks like this:

```
startService(new Intent("net.learn2develop.MyService"));
```

- To stop a service, use the **stopService()** method:

```
stopService(new Intent(getBaseContext(),
    MyService.class));
```

Creating your own Services

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service {
        @Override
        public IBinder onBind(Intent arg0) {
                return null;
        }
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
                // We want this service to continue running until it is explicitly
                // stopped, so return sticky.
                Toast.makeText(this, “Service Started”, Toast.LENGTH_LONG).show();
                return START_STICKY;
        }
        @Override
                public void onDestroy() {
                super.onDestroy();
                Toast.makeText(this, “Service Destroyed”, Toast.LENGTH_LONG).show();
        }
}
```

Creating your own Services

- In the AndroidManifest.xml file, add the **service** tag:

```
<?xml version=”1.0” encoding=”utf-8”?>
<manifest xmlns:android=”http://schemas.android.com/apk/res/android”
package=”net.learn2develop.Services”
android:versionCode=”1”
android:versionName=”1.0” >
<uses-sdk android:minSdkVersion=”14” />
<application
android:icon=”@drawable/ic_launcher”
android:label=”@string/app_name” >
<activity
android:label=”@string/app_name”
android:name=”.ServicesActivity” >
<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>
```

Creating your own Services

- In the main.xml file, add two buttons to start and stop the service:

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout xmlns:android=”http://schemas.android.com/apk/res/android”
        android:layout_width=”fill_parent”
        android:layout_height=”fill_parent”
        android:orientation=”vertical” >
        <Button android:id=”@+id/btnStartService”
        android:layout_width=”fill_parent”
        android:layout_height=”wrap_content”
        android:text=”Start Service”
        android:onClick=”startService”/>
        <Button android:id=”@+id/btnStopService”
        android:layout_width=”fill_parent”
        android:layout_height=”wrap_content”
        android:text=”Stop Service”
        android:onClick=”stopService” />
</LinearLayout>
```

Creating your own Services

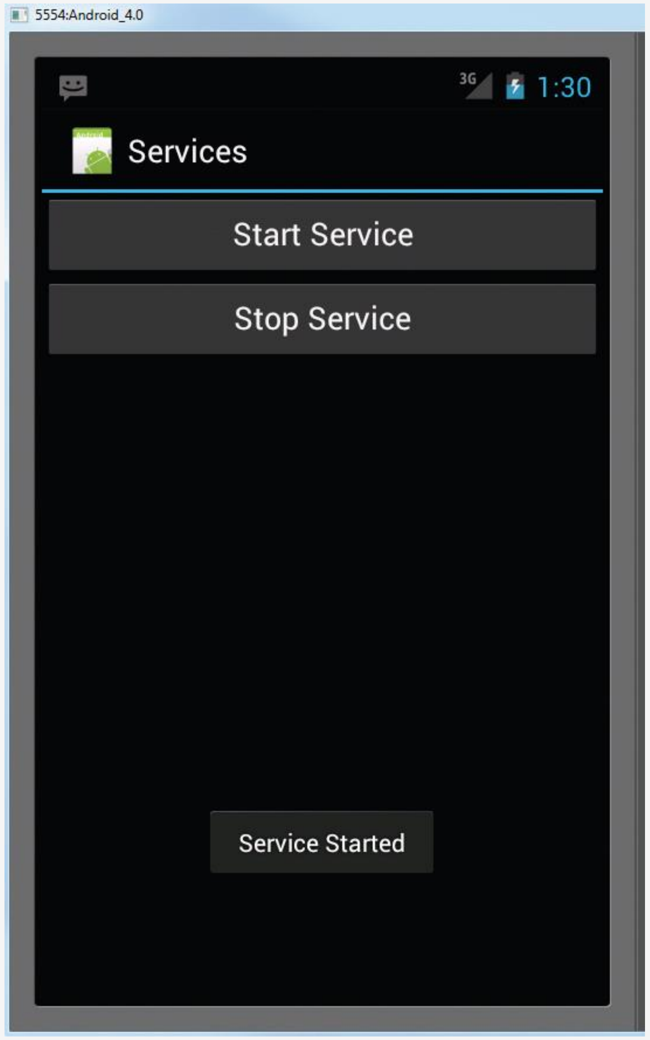
- **Create another activity to test the service:**

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class ServicesActivity extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
        }
        public void startService(View view) {
                startService(new Intent(getBaseContext(), MyService.class));
        }
        public void stopService(View view) {
                stopService(new Intent(getBaseContext(), MyService.class));
        }
}
```

Testing a Service

- ❑ Clicking the Start Service button will start the service
- ❑ To stop the service, click the Stop Service button.



Performing Long-Running Tasks in a Service

- For a long-running service, it is important that you **put all long-running code into a separate thread** so that it does not tie up the application that calls it:
- Create an **inner class that extends the AsyncTask** class.
  - The **AsyncTask** class enables you to perform background execution without needing to manually handle threads and handlers.
- The **DoBackgroundTask** class extends the **AsyncTask** class by specifying three generic types:

private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {…}

- The three types specified are URL, Integer and Long.
  - These three types specify the data type used by the following three methods that you implement in an AsyncTask class:

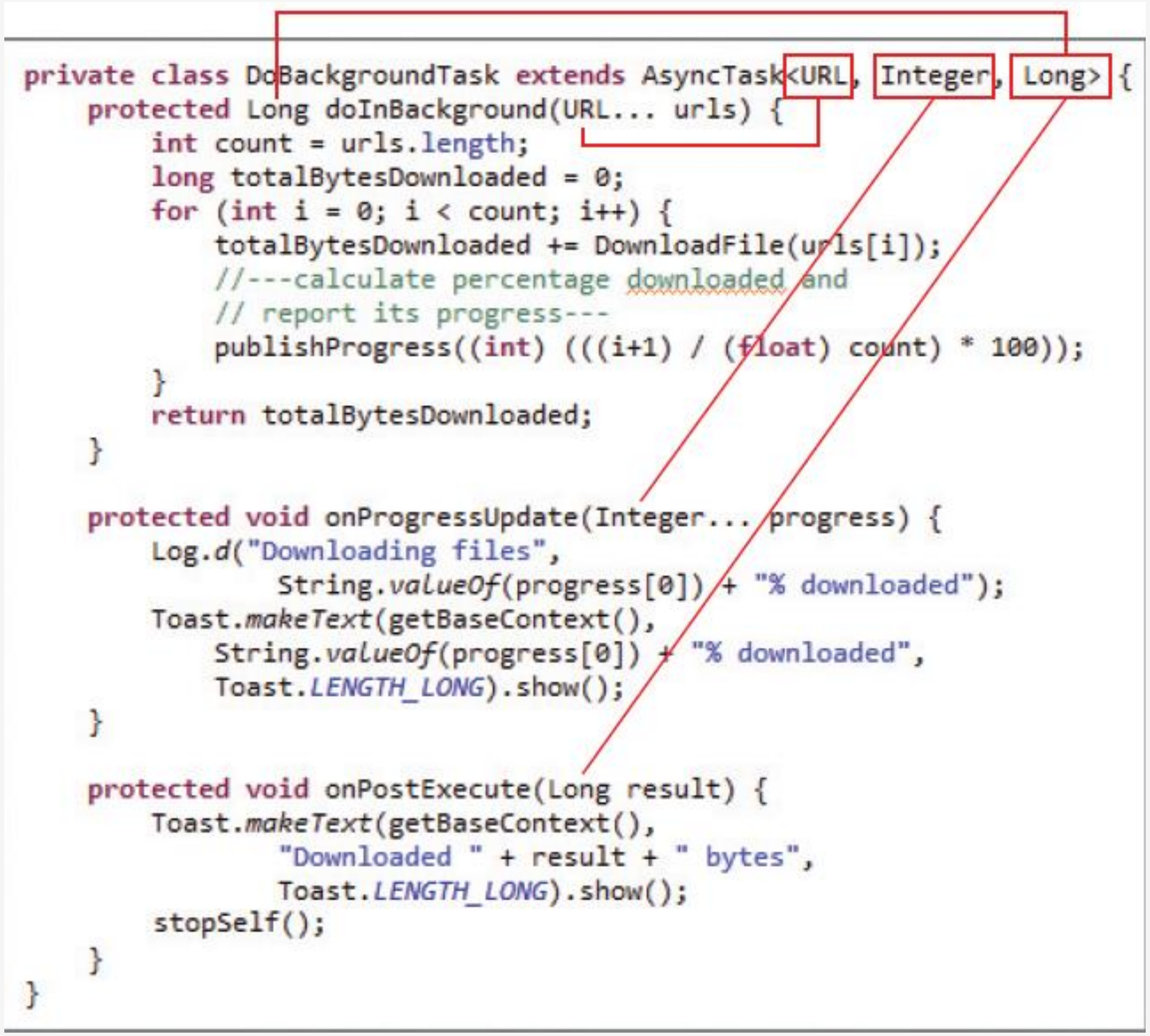
Performing Long-Running Tasks in a Service

- **doInBackground()** — This method **accepts an array of the first generic type** specified earlier.
  - In this case, the type is URL.
  - This method is executed in the background thread and is where you put your long-running code.
- To report the progress of your task, you call the **publishProgress()** method, which invokes the next method, **onProgressUpdate()**, which you implement in an AsyncTask class.
  - The return type of this method takes the third generic type specified earlier, which is Long in this case.

Performing Long-Running Tasks in a Service

- **onProgressUpdate()** — This method is invoked in the UI thread and is called when you call the **publishProgress()** method.
  - It **accepts an array of the second generic type** specified earlier.
    - In this case, the type is Integer.
  - Use this method to report the progress of the background task to the user.
- **onPostExecute()** — This method is invoked in the UI thread and is called when the doInBackground() method has finished execution.
  - This method **accepts an argument of the third generic type** specified earlier, which in this case is a Long.

Performing Long-Running Tasks in a Service



```
private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalBytesDownloaded = 0;
        for (int i = 0; i < count; i++) {
            totalBytesDownloaded += DownloadFile(urls[i]);
            //---calculate percentage downloaded and
            // report its progress---
            publishProgress((int) (((i+1) / (float) count) * 100));
        }
        return totalBytesDownloaded;
    }

    protected void onProgressUpdate(Integer... progress) {
        Log.d("Downloading files",
                String.valueOf(progress[0]) + "% downloaded");
        Toast.makeText(getBaseContext(),
            String.valueOf(progress[0]) + "% downloaded",
            Toast.LENGTH_LONG).show();
    }

    protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
                "Downloaded " + result + " bytes",
                Toast.LENGTH_LONG).show();
        stopSelf();
    }
}
```

Performing Long-Running Tasks in a Service

- To download multiple files in the background, create an instance of the DoBackgroundTask class and then called its **execute()** method by passing in an array of URLs:

```
try {
        new DoBackgroundTask().execute(
        new URL(“http://www.amazon.com/somefiles.pdf”),
        new URL(“http://www.wrox.com/somefiles.pdf”),
        new URL(“http://www.google.com/somefiles.pdf”),
        new URL(“http://www.learn2develop.net/somefiles.pdf”));
        } catch (MalformedURLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}
```

Performing Long-Running Tasks in a Service

- When the background thread has finished execution, you can manually call the **stopSelf()** method to stop the service:

```
protected void onPostExecute(Long result) {
    Toast.makeText(getBaseContext(),
    "Downloaded " + result + " bytes",
    Toast.LENGTH_LONG).show();
    stopSelf();
}
```

Performing Long-Running Tasks in a Service

```
public class MyService extends Service {
        @Override
        public IBinder onBind(Intent arg0) {
        }
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
        // We want this service to continue running until it is explicitly
        // stopped, so return sticky.
        //Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        try {
        new DoBackgroundTask().execute(
        new URL("http://www.amazon.com/somefiles.pdf"),
        new URL("http://www.wrox.com/somefiles.pdf"),
        new URL("http://www.google.com/somefiles.pdf"),
        new URL("http://www.learn2develop.net/somefiles.pdf"));
        } catch (MalformedURLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        }
        return START_STICKY; // tells the OS to recreate the service after it has enough
                             // memory and call onStartCommand() again
        }
```

Performing Long-Running Tasks in a Service

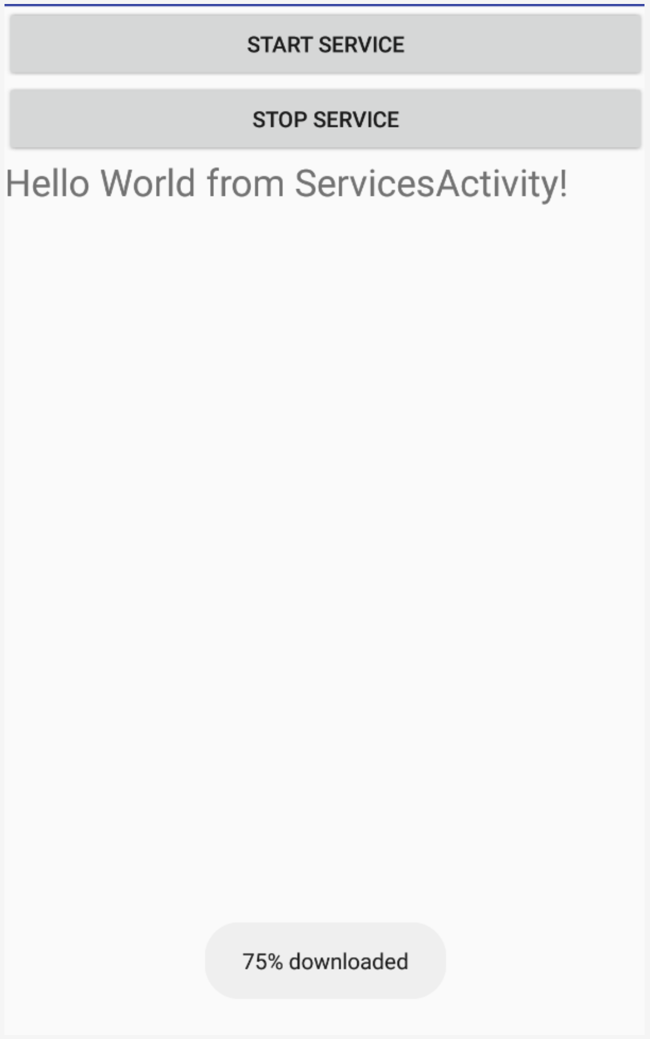
```
private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
        protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalBytesDownloaded = 0;
        for (int i = 0; i < count; i++) {
        totalBytesDownloaded += DownloadFile(urls[i]);
        //---calculate percentage downloaded and
        // report its progress---
        publishProgress((int) (((i+1) / (float) count) * 100));
        }
        return totalBytesDownloaded;
        }
        protected void onProgressUpdate(Integer... progress) {
        Log.d(“Downloading files”,
        String.valueOf(progress[0]) + “% downloaded”);
        Toast.makeText(getBaseContext(),
        String.valueOf(progress[0]) + “% downloaded”,
```

Performing Long-Running Tasks in a Service

```
        Toast.LENGTH_LONG).show();
        }
        protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
        “Downloaded “ + result + “ bytes”,
        Toast.LENGTH_LONG).show();
        stopSelf();
        }
        }
        @Override
        public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, “Service Destroyed”, Toast.LENGTH_LONG).show();
        }
}
```

Testing long-time running service

- MyService Example



Hello World from ServicesActivity!

75% downloaded

Executing Asynchronous Tasks on Separate Threads Using IntentService

- To easily create a service that runs a task asynchronously and **terminates itself** when it is done, you can use the **IntentService** class:

```
public class MyIntentService extends IntentService {
    //private Thread thread = new Thread();
    public MyIntentService() {
    super("MyIntentServiceName");
    }
    @Override
    protected void onHandleIntent(Intent intent) {
    //thread.start();
    try {
    int result = DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
    Log.d("IntentService", "Downloaded " + result + " bytes");
    } catch (MalformedURLException e) {
    e.printStackTrace();
    }
    }
```

Executing Asynchronous Tasks on Separate Threads Using IntentService

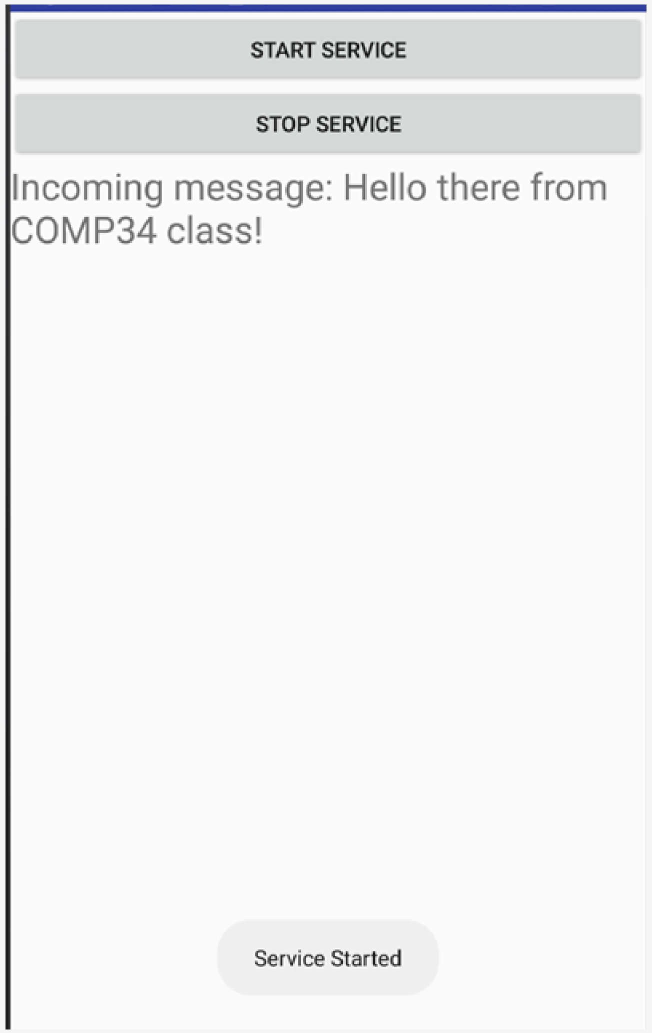
```
    private int DownloadFile(URL url) {
    try {
    //---simulate taking some time to download a file---
    thread.sleep(5000);
    } catch (InterruptedException e) {
    e.printStackTrace();
    }
    return 100;
    }
}
```

Executing Asynchronous Tasks on Separate Threads Using IntentService

```
public void startService(View view) {
//startService(new Intent(getBaseContext(), MyService.class));
//OR
//startService(new Intent("net.learn2develop.MyService"));
startService(new Intent(getBaseContext(),
MyIntentService.class));
}
public void stopService(View view) {
stopService(new Intent(MainActivity.this,
MyIntentService.class));
}
```

Testing MySimpleService

❑ Example



Incoming message: Hello there from COMP34 class!

Service Started

References

- ❑ Textbook (chap. 11)
- ❑ Reference textbook
- ❑ Android Documentation