Mobile Programming

Dr. Nader Mahmoud

Lecturer at Computer Science department

Course Content

- **Chapter 1: Getting Started with Android Programming**
- **Chapter 2: Using Android Studio for Android Development**
- **Chapter 3: Activities, Fragments, and Intents**
- **Chapter 4: Getting to know the Android User Interface**
- **Chapter 5: Designing Your User Interface with Views**
- **Chapter 6: Displaying Pictures and Menus with Views**
- **Chapter 7: Data Persistence**
- **Chapter 8: Content Providers**
- **Chapter 9: Messaging**
- **Chapter 10: Location-Based Services**
- **Chapter 11: Networking**
- **Chapter 12: Developing Android Services**

Agenda

- **Chapter 12 - Developing Android Services**
  - How to create a service that runs in the background
  - How to perform long-running tasks in a separate thread
  - How to perform repeated tasks in a service
  - How an activity and a service communicate

Developing Android Services

- A *service* is an application in Android that runs in the background without needing to interact with the user

- For example, while using an application, you might want to play some background music at the same time.

- In this case, the code that is playing the background music has no need to interact with the user; therefore, it can be run as a service.

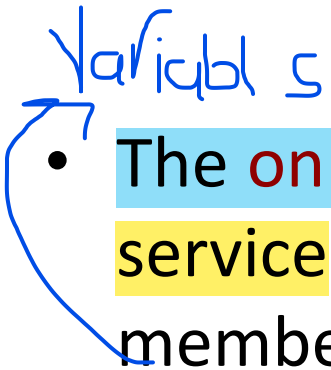
- A good example of this scenario is an application that continually logs the geographical coordinates of the device.

Creating Your Own Services

MyService.java

```
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) {
		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



- 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. For now, you simply return a null for this method.

- The onStartCommand() method is called when you start the service explicitly using the **startService()** method. START_STICKY so that the service continues to run until it is explicitly stopped.

- 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.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.jfdimarzio.services">
<application
      android:allowBackup="true"
      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=".MyService" />
</application>
</manifest>
```

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
android.support.constraint.ConstraintLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/activity_main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.jfdimarzio.services.MainActivity">
    <Button
        android:text="Start Service"
        android:layout_width="90dp"
        android:layout_height="50dp"
        android:id="@+id/btnStartService"
        ……
        android:onClick="startService" />
```

main.xml

```
<Button
    android:text="Stop Service"
    android:layout_width="88dp"
    android:layout_height="48dp"
    android:id="@+id/btnStopService"
    app:layout_constraintLeft_toLeftOf="@+id/activity_main"
    android:layout_marginStart="16dp"
    app:layout_constraintTop_toTopOf="@+id/activity_main"
    app:layout_constraintRight_toRightOf="@+id/activity_main"
    android:layout_marginEnd="16dp"
    app:layout_constraintBottom_toBottomOf="@+id/activity_main"
    android:onClick="stopService" />
</android.support.constraint.ConstraintLayout>
```

MainActivity.java

```
package net.learn2develop.Services;
import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
public class MainActivity extends AppCompatActivity {
    /** 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));
    }
}
```

Creating Your Own Services

- 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>
```

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

- 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"));
```

Service

Performing Long-Running Tasks in a Service

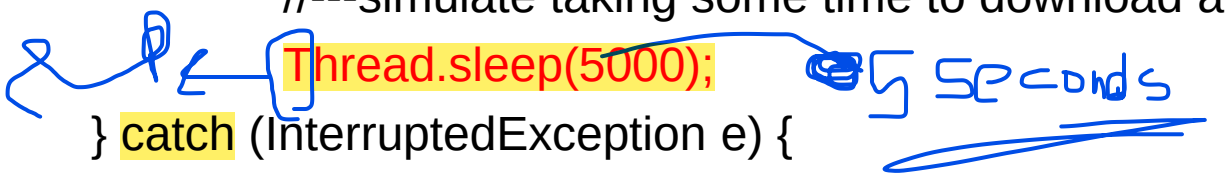
- The service you created in the previous section does not do anything useful
- In this section you modify it so that it simulate downloading a file from a given URL

MyService.java

```
package net.learn2develop.Services;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;
import java.net.MalformedURLException;
import java.net.URL;
public class MyService extends Service {
        @Override
        public IBinder onBind(Intent arg0) {
                return null;
        }
        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
            //Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
          try {
               int result =DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
                Toast.makeText(getBaseContext(), "Downloaded " + result + " bytes",
                Toast.LENGTH_LONG).show();
              } catch (MalformedURLException e) {
                      e.printStackTrace();
              }
             return START_STICKY;            }
```

MyService.java

```
private int DownloadFile(URL url) {
        try {
                //---simulate taking some time to download a file---
                Thread.sleep(5000);
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
//---return an arbitrary number representing
// the size of the file downloaded---
        return 100;
    }
@Override
public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
}
}
```



Performing Long-Running Tasks in a Service

- To simulate the delays experienced by the service when downloading the file, you use the Thread. Sleep() method to pause the service for five seconds (5,000 milliseconds).

- As you start the service, note that the **activity** is suspended for about five seconds

- During this time, the entire activity is not responsive, demonstrating a very important point: The service runs on the same thread as your activity

- Two ways:
  1. AsyncTask
  2. IntentService

3- Threads

Performing Long-Running Tasks in a Service

1. AsyncTask

- This example illustrates one way in which you can execute a task **asynchronously** within your service.

- You do so by creating 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> {
```

In this case, the three types specified are URL, Integer, and Long.

Performing Long-Running Tasks in a Service

- **doInBackground()**—This method accepts an array of the first generic type specified earlier.
  - 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().
- **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.

MyService.java

```
public class MyService extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        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) {
          e.printStackTrace();
        }
        return START_STICKY;
    }
```

MyService.java

```
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();
    } }
```

MyService.java

```
private int DownloadFile(URL url) {
    try {
        //---simulate taking some time to download a file---
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
        //---return an arbitrary number representing
        // the size of the file downloaded---
        return 100;
    }
@Override
public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
}
}
```

2- 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.
- The IntentService class is a base class for Service that handles asynchronous requests on demand.
- It is started just like a normal service; and it executes its task within a worker thread and terminates itself when the task is completed

MyIntentService.java

```
import android.app.IntentService;
import android.content.Intent;
public class MyIntentService extends IntentService {
    private Thread thread = new Thread();
    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();
        } }
    private int DownloadFile(URL url) {
        try {
                    //---simulate taking some time to download a file---
                    thread.sleep(5000);
        } catch (InterruptedException e) {
                    e.printStackTrace();
        }
        return 100; } }
```

MainActivity.java

Add the following bolded statement to the MainActivity.java file:

```
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));
}
```

Establishing Communication Between a Service and an Activity

- Often a service simply executes in its own thread, independently of the activity that calls it

- This doesn't pose a problem if you simply want the service to perform some tasks periodically and the activity does not need to be notified about the service's status.

  - For example, you might have a service that periodically logs the geographical location of the device to a database.

- **However, suppose you want to monitor for a particular location.**

service can communicate with an activity using BroadcastReceiver.

MyIntentService.java

```
import android.app.IntentService;
import android.content.Intent;
import android.util.Log;
import java.net.MalformedURLException;
import java.net.URL;
public class MyIntentService extends IntentService {
  public MyIntentService() {
      super("MyIntentServiceName");
  }
@Override
protected void onHandleIntent(Intent intent) {
  try {
      int result = DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
       Log.d("IntentService", "Downloaded " + result + " bytes");
  //---send a broadcast to inform the activity
  // that the file has been downloaded---
   Intent broadcastIntent = new Intent();
   broadcastIntent.setAction("FILE_DOWNLOADED_ACTION");
   getBaseContext().sendBroadcast(broadcastIntent);
 } catch (MalformedURLException e) {
   e.printStackTrace();
 }
}
```

To notify an activity when a service has finished its execution, you broadcast an intent using the **sendBroadcast()** method

MainActivity.java

```
import android.app.IntentService;
public class MainActivity extends AppCompatActivity {
    IntentFilter intentFilter;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    @Override
    public void onResume() {
        super.onResume();
        //---intent to filter for file downloaded intent---
        intentFilter = new IntentFilter();
        intentFilter.addAction("FILE_DOWNLOADED_ACTION");
        //---register the receiver---
        registerReceiver(intentReceiver, intentFilter);
    }
    @Override
    public void onPause() {
        super.onPause();
        //---unregister the receiver---
        unregisterReceiver(intentReceiver);
    }
```

REGISTERING RECIEVER:

The action of this intent that you are broadcasting is set to "FILE_DOWNLOADED_ACTION", which means any **activity that is listening for this intent will be invoked**.

MainActivity.java

```
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(getBaseContext(), MyService.class));
}
private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                        Toast.makeText(getBaseContext(), "File downlo     d!",
                        Toast.LENGTH_LONG).show();
            }
            };
}
```

Reciever to be executed

Binding Activities to Services

- All the services that you have seen are simple—either they start with a counter and increment at regular intervals or they download a fixed set of files from the Internet.

- However, real-world services are usually much more sophisticated, requiring the passing of data so that they can do the job correctly for you.

- Using the service demonstrated earlier that downloads a set of files, suppose you now want to let the calling activity determine what files to download, instead of hardcoding them in the service.

Binding Activities to Services

- There are two ways for doing that:

- **First approach:**

1. First, in the calling activity, you create an Intent object, specifying the service name:

```
public void startService(View view) {
        Intent intent = new Intent(getBaseContext(), MyService.class);
}
```

2. You then create an array of URL objects and assign it to the Intent object through its putExtra() method.

Binding Activities to Services

- There are two ways for doing that:

- **First approach:**

3. You start the service using the Intent object:

```
public void startService(View view) {
        Intent intent = new Intent(getBaseContext(), MyService.class);
        try {
        URL[] urls = new URL[] {
                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")};
        intent.putExtra("URLs", urls);
        } catch (MalformedURLException e) {
        e.printStackTrace();
        }
        startService(intent);
}
```

Binding Activities to Services

- There are two ways for doing that:

- **Second approach:**

  better way to pass data is to bind the activity directly to the service so that the **activity can call any public members and methods on the service directly**.

MyService.java

```
public class MyService extends Service {
    int counter = 0;
    URL[] urls;
    static final int UPDATE_INTERVAL = 1000;
    private Timer timer = new Timer();

    private final IBinder binder = new MyBinder();

    public class MyBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }
    @Override
    public IBinder onBind(Intent arg0) {
        return binder;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        new DoBackgroundTask().execute(urls);
        return START_STICKY;
    }
```

To bind activities to a service, you must **first declare an inner class in your service that extends the Binder class**

Within this class you implement the **getService()** method, which **returns an instance of the service**

You also modify the onBind() method to return the **MyBinder instance**

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    IntentFilter intentFilter;

    MyService serviceBinder;
    Intent i;
    private ServiceConnection connection = new ServiceConnection(){
        public void onServiceConnected(
                ComponentName className, IBinder service){
            //--called when the connection is made--
            serviceBinder = ((MyService.MyBinder)service).getService();
            try {
                URL[] urls = new URL[] {
                        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")};
                //---assign the URLs to the service through the
                // serviceBinder object---
                serviceBinder.urls = urls;
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
            startService(i);
        }
        public void onServiceDisconnected(ComponentName className) {
            //---called when the service disconnects---
            serviceBinder = null;
        }
    };
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {...}
```

you obtain an instance of the service from the onServiceConnected() method by using the getService() method of the service argument

You then start the service using an Intent object

MainActivity.java

Before you can start the service, you must bind the activity to the service. This is done in the startService() method of the Start Service button:

```
public void startService(View view) {
    i = new Intent(MainActivity.this, MyService.class);
    bindService(i, connection, Context.BIND_AUTO_CREATE);
}
```

The bindService() method enables your activity to be connected to the service. It takes three arguments:

1. An Intent object
2. A ServiceConnection object
3. A flag to indicate how the service should be bound

End of Lecture