Services

Programming the Android Platform

Service Class

- Application component
- No user interface
- Two main uses
  - Performing background processing
  - Supporting remote method execution

Service Class (cont.)

- A Service can be activated by a client component via
  - Context.startService(Intent intent)
- The started Service runs in the background
  - Services often designed to perform a single operation & then terminate themselves
  - Started Services do not return results
- Note: Services do not run in their own threads

Service Class (cont.)

- Client components can bind to a Service when they want to interact with it
  - Context.bindService (Intent service,
                ServiceConnection conn, int flags)
- Service will be started if necessary
- Service remains active as long as at least one client is bound to it

Example Services

- Logging Service
  - Client Activity sends log messages to service
  - Service writes messages to a log console
- Music playing Service
  - Client Activity tells service to play a music file
  - Services plays music in the background (even if Client Activity pauses or terminates)
- ID Service
  - Client Activity requests system-wide unique ID
  - Service returns ID to Client

Logging Service

- Service requests represented as Intents
- Uses a Service subclass called IntentService
- IntentService requests handled sequentially in a single worker thread
- IntentService started and stopped as needed

Logging Service (cont.)

```
public class BGLoggingDemo extends Activity {
  public void onCreate(Bundle savedInstanceState) {
  ...
    buttonStart.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent(BGLoggingDemo.this,
                                        BGLoggingService.class);
            intent.putExtra("course.examples.Services.Logging",
                                "Log this message");
            startService(intent);
        }
    });
  }
}
```

Logging Service (cont.)

```
public class BGLoggingService extends IntentService {
…
  public int onStartCommand(Intent intent, int flags, int startId) {
     super.onStartCommand(intent, flags, startId);
     return START_NOT_STICKY;
  }
  protected void onHandleIntent(Intent intent) {
     // create and start new Thread to handle request
     …
     Log.i(TAG,arg.getCharSequenceExtra
      ("course.examples.Services.Logging").toString());
  }
…
}
```

Logging Service (cont.)

```
<application ... >
   <activity android:name=".BGLoggingDemo"
                                              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:enabled="true" android:name=".BGLoggingService" />
  </application>
```

Notes on Services

- The LoggingService is a simplified example
  - It doesn't need to be implemented as a Service
    - You could simply do the logging in a new Thread
- Use Services when you want to run a component even when a user is not interacting with the Service's hosting application

Music Player Service

- Client Activity can start/stop playing music via a Service
  - If music is playing when client leaves the foreground, music service will continue playing

Music Player Service (cont.)

```
public class MusicService extends Service {
    MediaPlayer player;
     ...
    public void onCreate() {
        player = MediaPlayer.create(this, R.raw.braincandy);
        player.setLooping(false);
    }
    public int onStartCommand (Intent intent, int flags, int startid) {
        player.start();
        return START_NOT_STICKY;
    }
    ...
}
```

Music Player Service (cont.)

```
public class MusicServiceDemo extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    ...
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View src) {
         ...
         startService(
           new Intent(MusicServiceDemo.this,MusicService.class));
      }
     });
   }
}
```

ID Service

- Client uses a Service hosted in another application
- Client needs an ID from service
- Requires inter-process communication (IPC)

Implementing a Service

- Define remote interface in the Android Interface Definition Language (AIDL)
- Implement remote interface
  - Stub & application-specific methods
- Implement Service methods
- Implement Client methods

Define Remote Interface

- Declare interface in a .aidl file

```
package course.examples.Services.KeyCommon;

interface KeyGenerator {
   String getKey();
}
```

AIDL Syntax

- Similar to Java interface definition syntax
  - Can declare methods
  - Cannot declare static fields
- Remote method parameters can be labeled
  - in: (default) transferred to the remote method
  - out: returned to the caller
  - inout: both in and out

AIDL Data Types

- Java primitive types
- StringList
  - List elements must be valid AIDL data types
  - Generic lists supported
- Map
  - Map elements must be valid AIDL data types
  - Generic maps not supported
- CharSequence
- Other AIDL-generated interfaces
- Classes implementing the Parcelable protocol

Compile .aidl File

- Generate a Java interface with same name as .aidl file
  - Eclipse does this automatically
- Generated interface contains:
  - Abstract inner class called Stub
  - Interface & helper methods

Implement Remote Methods

```
public class KeyGeneratorImpl extends Service {
  …
  private final KeyGenerator.Stub binder =
                                        new KeyGenerator.Stub() {
    public String getKey() {
      // generate unique ID **in a thread-safe manner** & return it
    }
  };
…
```

Implement Service Methods

```
…
    public IBinder onBind(Intent intent) {
        return this.binder;
    }
}
```

Implement Client

```
public class KeyUser extends Activity {
  private KeyGenerator service; // handle to Remote Service
  private boolean bound;
// Remote Service callback methods
  private ServiceConnection connection =
                                      new ServiceConnection() {
    public void onServiceConnected(
                    ComponentName className, IBinder iservice) {
      service = KeyGenerator.Stub.asInterface(iservice);
      bound = true;
    }
...
```

Implement Client (cont.)

```
…
public void onServiceDisconnected(
                                        ComponentName className) {
      service = null; bound = false;
   }
 };
…
```

Implement Client (cont.)

```
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(KeyGenerator.class.getName());
    // bind to Service
    bindService(intent,this.connection,
    Context.BIND_AUTO_CREATE);
 }
 protected void onStop() {
   // unbind from Service
   if (bound) unbindService(this.connection);
   super.onStop();
 }
}
```

Implement Client (cont.)

```
…
public void onCreate(Bundle icicle) {
    …
    goButton.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        try {
          // call remote method
          output.setText(service.getKey());
        } catch (RemoteException e) {}
      }
    });
…
}
```

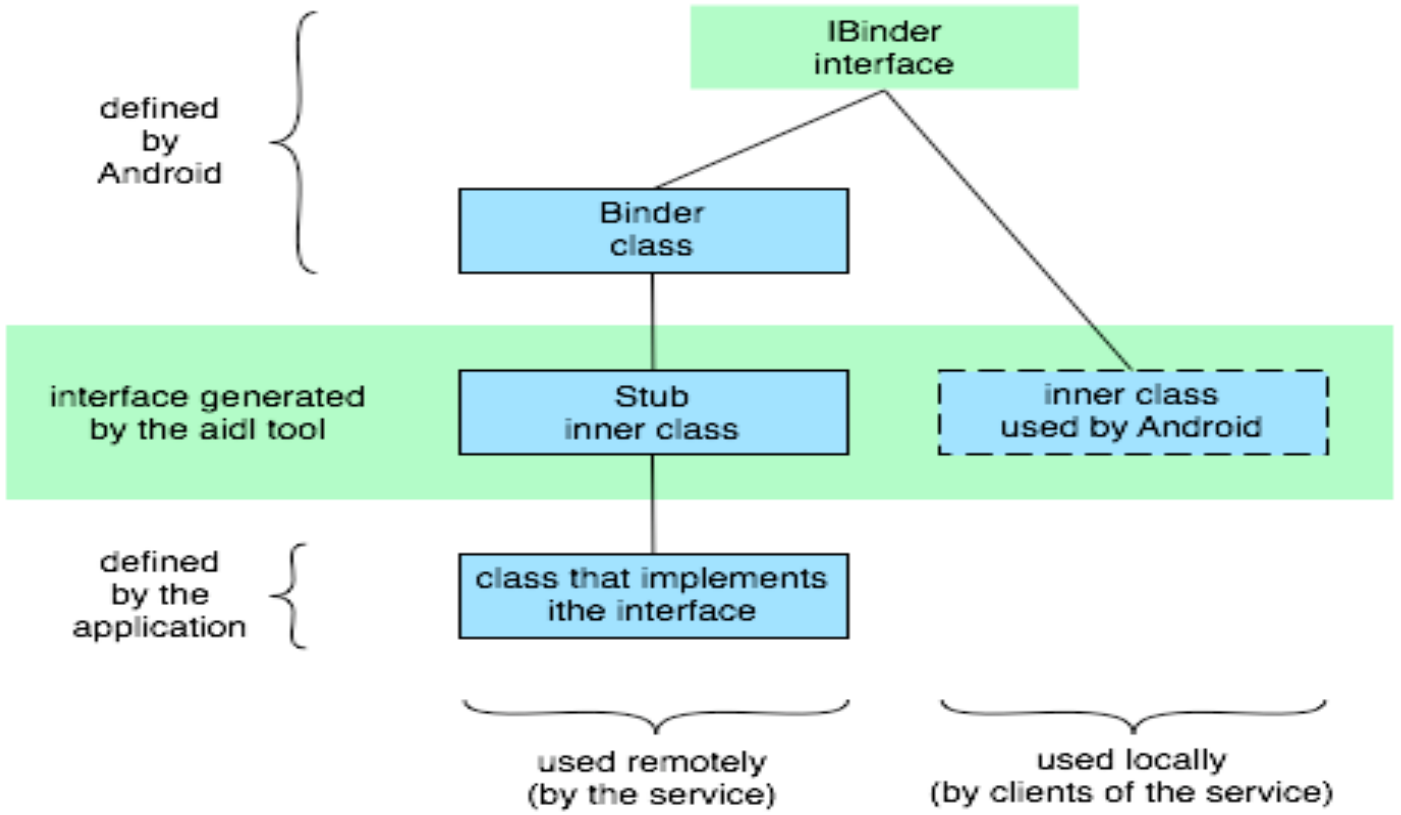
AndroidManifest.xml

```
<manifest … package="course.examples.Services.KeyClient">
   <application …">
      <activity android:name=".KeyUser" …>
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name=
                                    "android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

AndroidManifest.xml

```
<manifest …package="course.examples.Services.KeyService">
  <application …">
    <service android:name=".KeyGeneratorImpl"
                                   android:exported="true">
      <intent-filter>
        <action android:name=
          "course.examples.Services.KeyCommon.KeyGenerator"/>
      </intent-filter>
    </service>
  </application>
</manifest>
```

RPC Interface


IBinder interface
defined by Android
Binder class
interface generated by the aidl tool
Stub inner class
inner class used by Android
defined by the application
class that implements ithe interface
used remotely (by the service)
used locally (by clients of the service)

Lab Assignment

Source Code Examples

- LoggingServiceExample
- MusicPlayingServiceExample
- ServiceWithIPCExampleClient
- ServiceWithIPCExampleService