The Service Class

Today's Topics

The Service Class

Implementing started Services

Implementing bound Services

The Service Class

No user interface

Two main uses

Performing background processing

Supporting remote method execution

Starting A Service

Components can start a service by calling

`Context.startService(Intent intent)`

Starting A Service

Once started, the Service can run in the background indefinitely

Started Services usually perform a single operation & then terminate themselves

By default, Services run in the main thread of their hosting application

Binding to a Service

Components can bind to a Service by calling

```
Context.bindService (
        Intent service,
        ServiceConnection conn,
        int flags)
```

Binding to a Service

Binding to a Service allows a component to send requests and receive responses from a local or a remote service

At binding time, the Service will be started, if necessary

Service remains active as long as at least one client is bound to it

ServiceLocalLoggingService

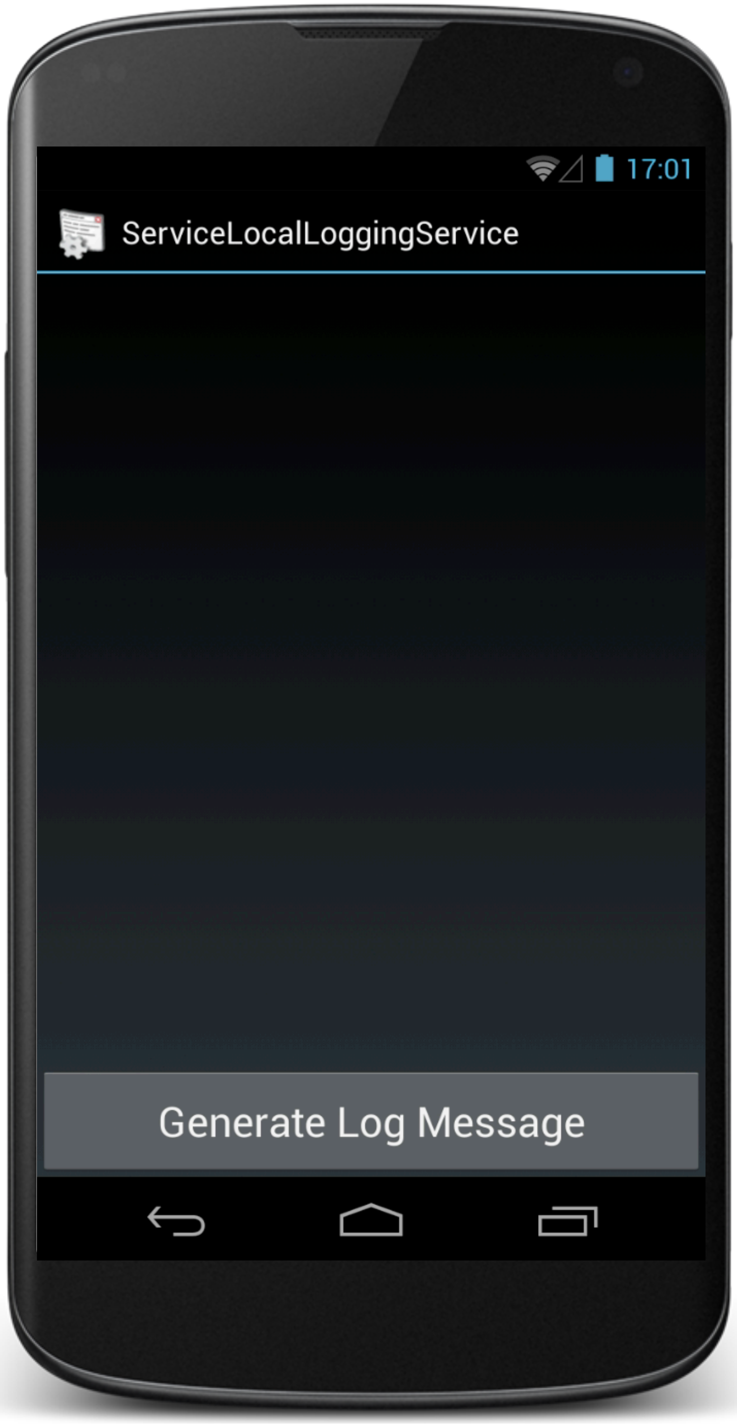
Client sends a log message to a local Service

The Service writes the message to the log console

LoggingService implemented as an IntentService

IntentService

- Subclass of Service
- IntentService requests are handled sequentially in a single worker thread
- IntentService is started and stopped as needed


17:01
ServiceLocalLoggingService
Generate Log Message

ServiceLocalLoggingService

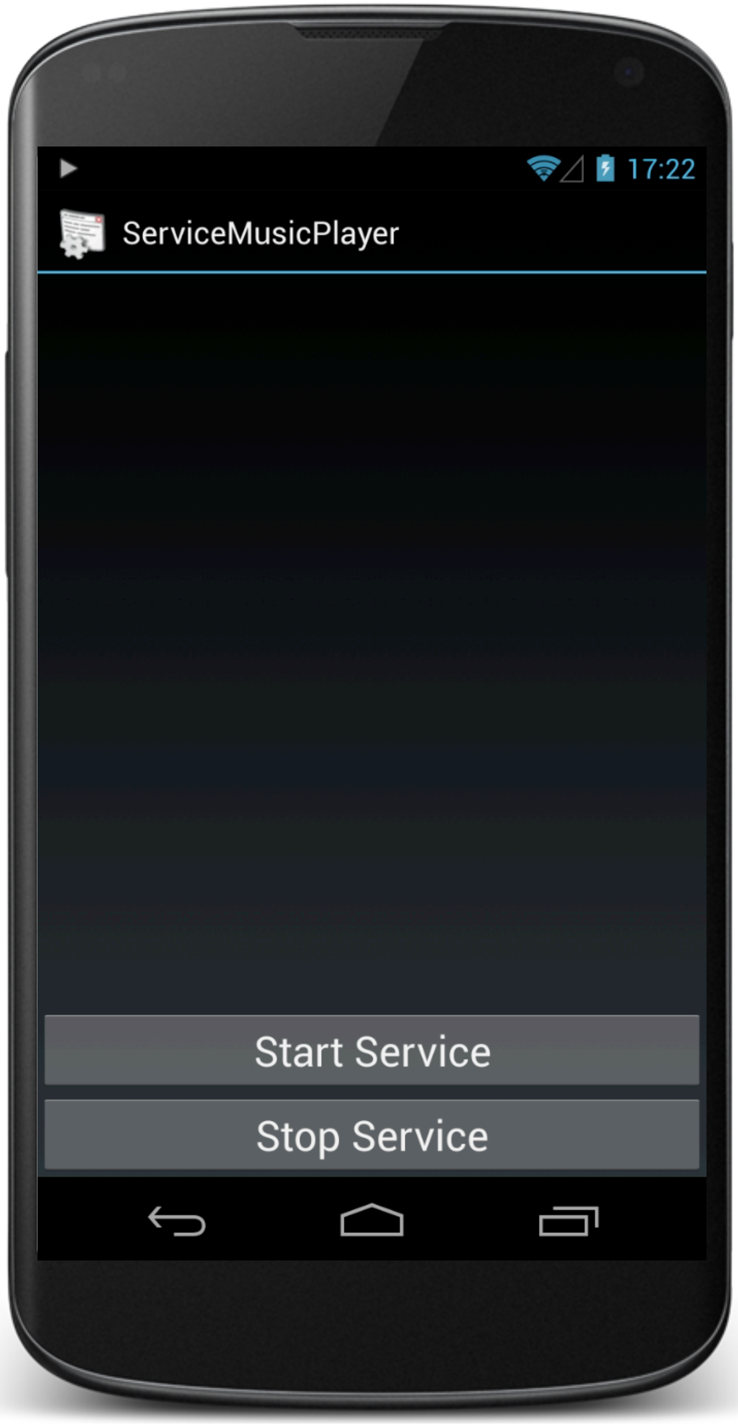
```
final Button messageButton = (Button) findViewById(R.id.message_button);
messageButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {

        // Create an Intent for starting the LoggingService
        Intent startServiceIntent = new Intent(getApplicationContext(),
                LoggingService.class);

        // Put Logging message in intent
        startServiceIntent.putExtra(LoggingService.EXTRA_LOG,
                "Log this message");

        // Start the Service
        startService(startServiceIntent);

    }
});
```

MusicPlayerForegroundService

Client Activity starts service to play a music file

Service plays music as a foreground service

Service continues playing even if Client Activity pauses or terminates


17:22
ServiceMusicPlayer
Start Service
Stop Service

MusicPlayerForegroundService

```java
@Override
public void onCreate() {
    super.onCreate();

    // Set up the Media Player
    mPlayer = MediaPlayer.create(this, R.raw.badnews);

    if (null != mPlayer) {

        mPlayer.setLooping(false);

        // Stop Service when music has finished playing
        mPlayer.setOnCompletionListener(new OnCompletionListener() {

            @Override
            public void onCompletion(MediaPlayer mp) {

                // stop Service if it was started with this ID
                // Otherwise let other start commands proceed
                stopSelf(mStartID);

            }
        });
    }
```

MusicPlayerForegroundService

```
    // Create a notification area notification so the user
    // can get back to the MusicServiceClient

    final Intent notificationIntent = new Intent(getApplicationContext(),
            MusicServiceClient.class);
    final PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
            notificationIntent, 0);

    final Notification notification = new Notification.Builder(
            getApplicationContext())
            .setSmallIcon(android.R.drawable.ic_media_play)
            .setOngoing(true).setContentTitle("Music Playing")
            .setContentText("Click to Access Music Player")
            .setContentIntent(pendingIntent).build();

    // Put this Service in a foreground state, so it won't
    // readily be killed by the system
    startForeground(NOTIFICATION_ID, notification);

}
```

MusicPlayerForegroundService

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

    if (null != mPlayer) {

        // ID for this start command
        mStartID = startid;

        if (mPlayer.isPlaying()) {

            // Rewind to beginning of song
            mPlayer.seekTo(0);

        } else {

            // Start playing song
            mPlayer.start();

        }

    }

    // Don't automatically restart this Service if it is killed
    return START_NOT_STICKY;
}
```

MusicPlayerForegroundService

```java
@Override
public void onDestroy() {

    if (null != mPlayer) {

        mPlayer.stop();
        mPlayer.release();

    }
}
```

Binding to Remote Services

Using the Messenger class

Defining an AIDL interface

Implementing Services with Messengers

Messenger managers a Handler

Allows Messages to be sent from one component to another across process boundaries

Messages are queued and processed sequentially by recipient

Implementing Services with Messengers

Service creates a Handler for processing specific messages

Service creates a Messenger that provides a Binder to a Client

Implementing Services with Messengers

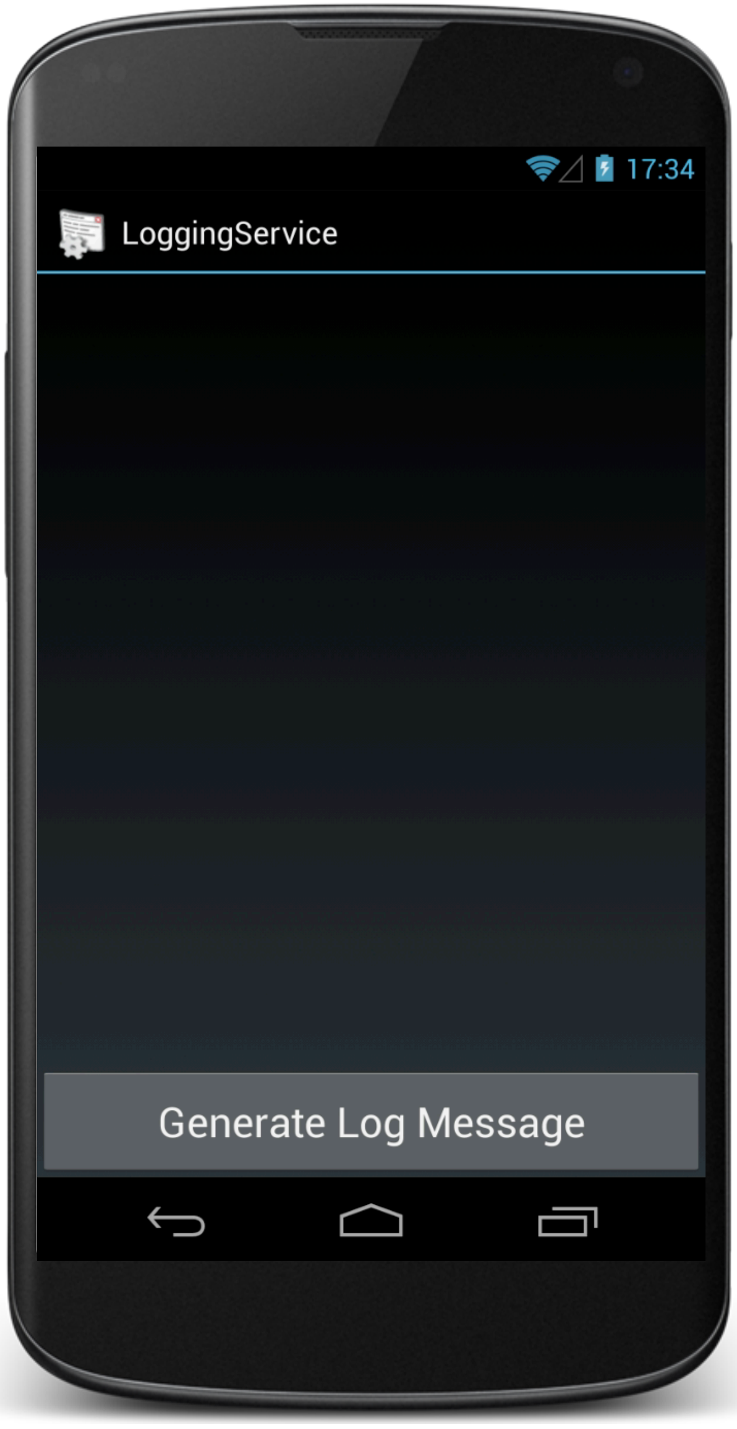
Client uses the Binder to create its own Messenger

Client uses the Messenger to send Messages to the Service

ServiceLoggingWithMessenger ServiceLoggingWithMessengerClient

Client sends log messages to a remote Logging Service

Logging Service writes messages to a log console


17:34
LoggingService
Generate Log Message

ServiceLoggingWithMessenger

```java
// Messenger Object that receives Messages from connected clients
final Messenger mMessenger = new Messenger(new IncomingMessagesHandler());

// Handler for incoming Messages
static class IncomingMessagesHandler extends Handler {

    @Override
    public void handleMessage(Message msg) {

        switch (msg.what) {

        case LOG_OP:

            Log.i(TAG, msg.getData().getString(MESSAGE_KEY));

            break;

        default:

            super.handleMessage(msg);

        }
    }
}

// Returns the Binder for the mMessenger, which allows
// the client to send Messages through the Messenger
@Override
public IBinder onBind(Intent intent) {
    return mMessenger.getBinder();
}
```

ServiceLoggingWithMessengerClient

```
// Intent used for binding to LoggingService
private final static Intent mLoggingServiceIntent = new Intent(
        "course.examples.Services.LoggingServiceWithMessenger.LoggingService");

private Messenger mMessengerToLoggingService;
private boolean mIsBound;

// Object implementing Service Connection callbacks
private ServiceConnection mConnection = new ServiceConnection() {

    public void onServiceConnected(ComponentName className, IBinder service) {

        // Messenger object connected to the LoggingService
        mMessengerToLoggingService = new Messenger(service);

        mIsBound = true;

    }

    public void onServiceDisconnected(ComponentName className) {

        mMessengerToLoggingService = null;

        mIsBound = false;

    }
};
```

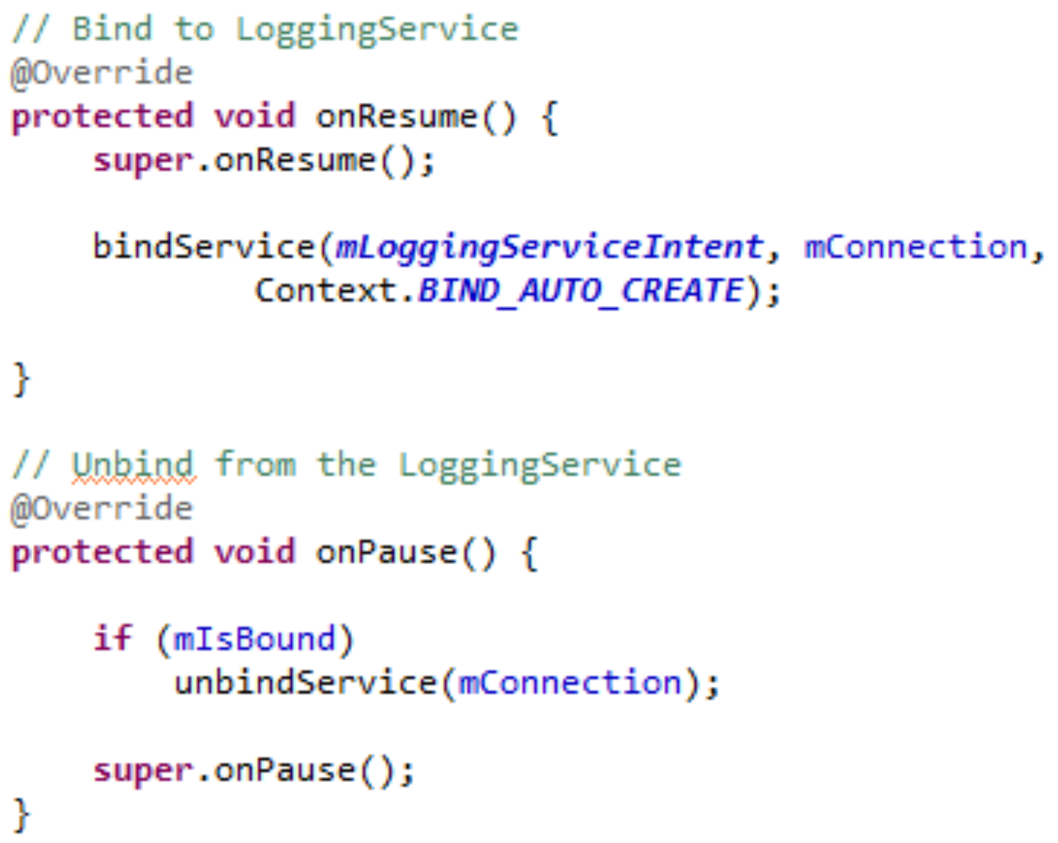
ServiceLoggingWithMessengerClient

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    final Button buttonStart = (Button) findViewById(R.id.buttonStart);
    buttonStart.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {

            if (mIsBound) {

                // Send Message to the Logging Service
                logMessageToService();

            }

        }
    });
}
```

ServiceLoggingWithMessengerClient

```java
// Create a Message and sent it to the LoggingService
// via the mMessenger Object

private void logMessageToService() {

    // Create Message
    Message msg = Message.obtain(null, LOG_OP);
    Bundle bundle = new Bundle();
    bundle.putString(MESSAGE_KEY, "Log This Message");
    msg.setData(bundle);

    try {

        // Send Message to LoggingService using Messenger
        mMessengerToLoggingService.send(msg);

    } catch (RemoteException e) {
        Log.e(TAG, e.toString());
    }
}
```

ServiceLoggingWithMessengerClient



```java
// Bind to LoggingService
@Override
protected void onResume() {
    super.onResume();

    bindService(mLoggingServiceIntent, mConnection,
            Context.BIND_AUTO_CREATE);

}

// Unbind from the LoggingService
@Override
protected void onPause() {

    if (mIsBound)
        unbindService(mConnection);

    super.onPause();
}
```

Implementing Services with AIDL

If a Service must be accessed concurrently, then develop an AIDL interface

Android Interface Definition Language

Implementing Services with AIDL

Define remote interface in the Android Interface Definition Language (AIDL)

Implement remote interface

Implement Service methods

Implement Client methods

Define Remote Interface

Declare interface in a .AIDL file

This defines how components can interact with the Service

AIDL Syntax

Similar to Java interface syntax

Can declare methods to be implemented

Cannot declare static fields

AIDL Syntax

Non-primitive remote method parameters require a directional tag

IN: transferred to the remote method

OUT: returned to the caller

INOUT: both in and out

AIDL Data Types

Java primitive types

String

CharSequence

AIDL Data Types

Other AIDL-generated interfaces

Classes implementing the Parcelable protocol

AIDL Data Types

List

List elements must be valid AIDL data types

Generic lists supported

AIDL Data Types

Map

Map elements must be valid AIDL data types

Generic maps not supported

Example Remote Interface

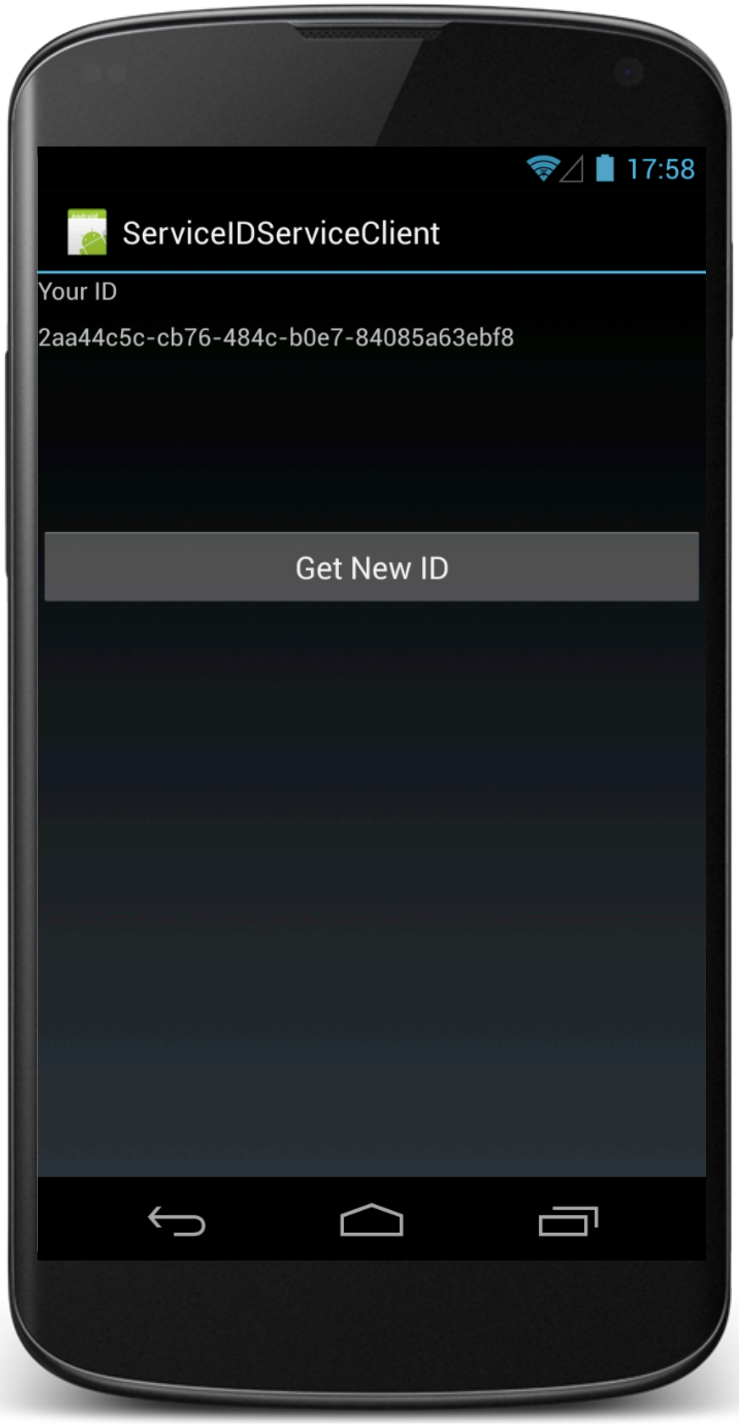
```
interface KeyGenerator {
    String getKey();
}
```

ServiceIDService

ServiceIDServiceClient

Client binds to a Service hosted in another application

Client retrieves an ID from service



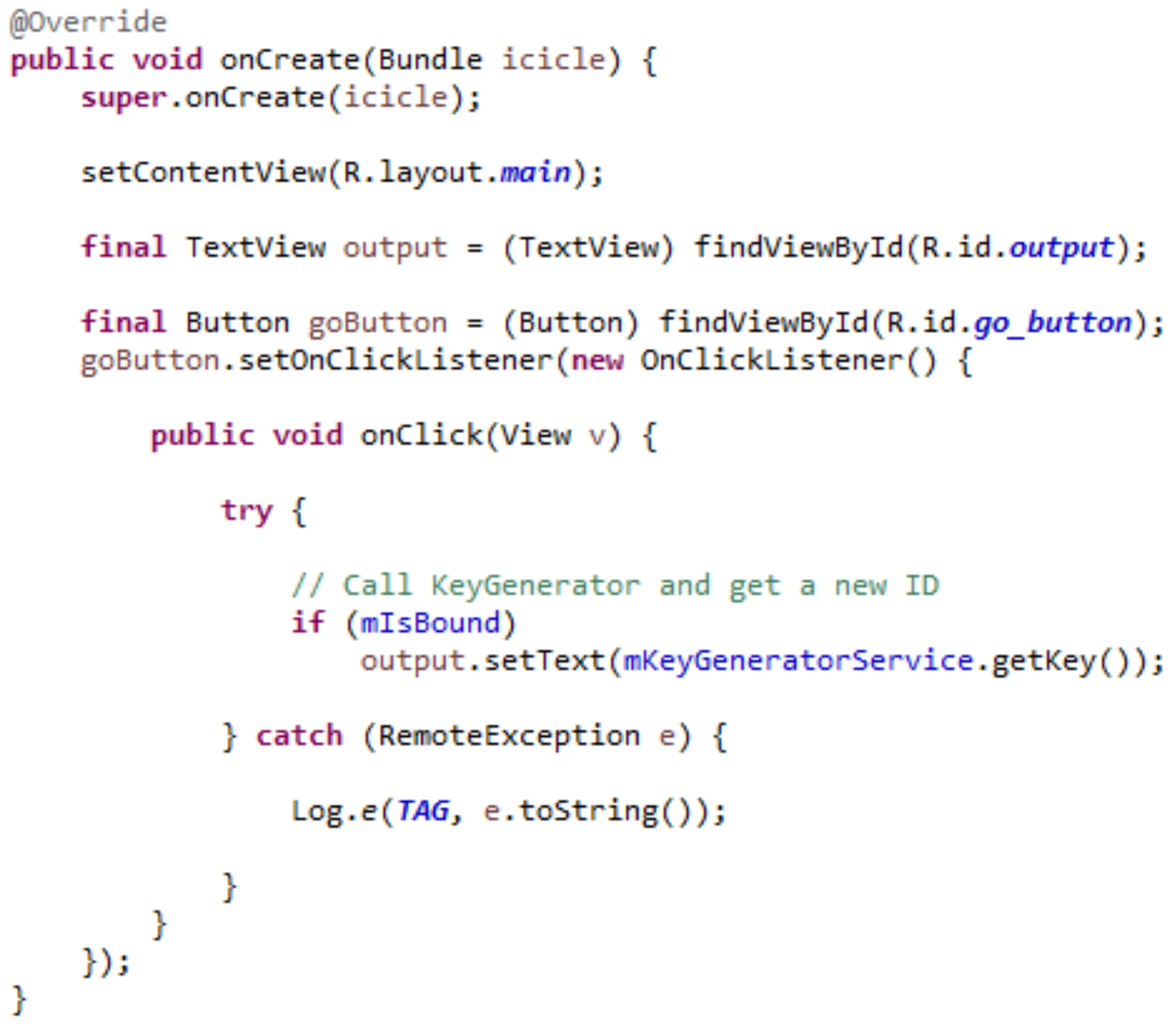
Your ID

2aa44c5c-cb76-484c-b0e7-84085a63ebf8

ServiceIDService

```java
// Set of already assigned IDs
// Note: These keys are not guaranteed to be unique if the Service is killed
// and restarted.

private final static Set<UUID> mIDs = new HashSet<UUID>();

// Implement the Stub for this Object
private final KeyGenerator.Stub mBinder = new KeyGenerator.Stub() {

    // Implement the remote method
    public String getKey() {

        UUID id;

        // Acquire lock to ensure exclusive access to mIDs
        // Then examine and modify mIDs

        synchronized (mIDs) {

            do {

                id = UUID.randomUUID();

            } while (mIDs.contains(id));

            mIDs.add(id);
        }
        return id.toString();
    }
};

// Return the Stub defined above
@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}
```

ServiceIDServiceClient

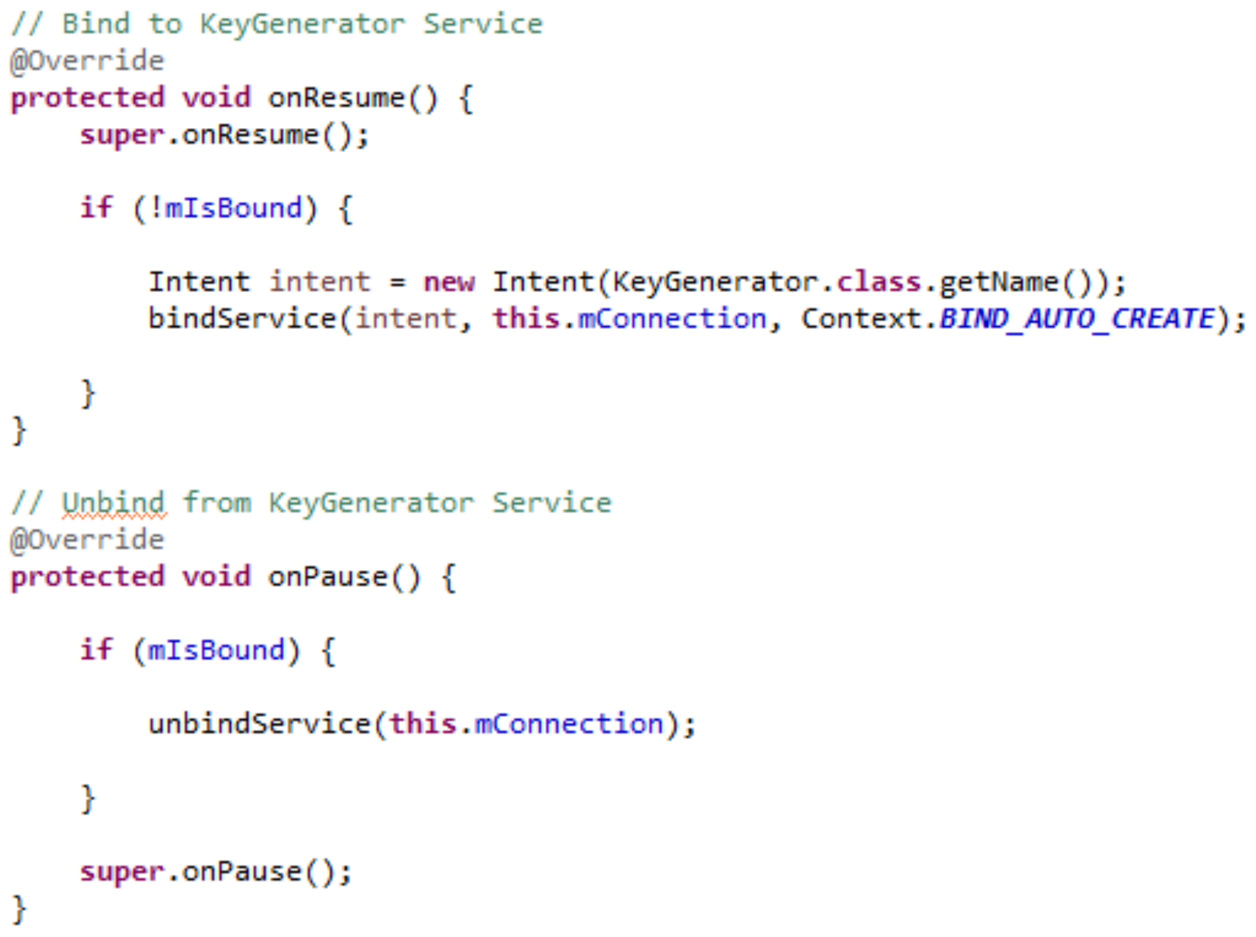


```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    setContentView(R.layout.main);

    final TextView output = (TextView) findViewById(R.id.output);

    final Button goButton = (Button) findViewById(R.id.go_button);
    goButton.setOnClickListener(new OnClickListener() {

        public void onClick(View v) {

            try {

                // Call KeyGenerator and get a new ID
                if (mIsBound)
                    output.setText(mKeyGeneratorService.getKey());

            } catch (RemoteException e) {

                Log.e(TAG, e.toString());

            }
        }
    });
}
```

ServiceIDServiceClient

```
private final ServiceConnection mConnection = new ServiceConnection() {

    public void onServiceConnected(ComponentName className, IBinder iservice) {

        mKeyGeneratorService = KeyGenerator.Stub.asInterface(iservice);

        mIsBound = true;

    }

    public void onServiceDisconnected(ComponentName className) {

        mKeyGeneratorService = null;

        mIsBound = false;

    }
};
```

ServiceIDServiceClient



```java
// Bind to KeyGenerator Service
@Override
protected void onResume() {
    super.onResume();

    if (!mIsBound) {

        Intent intent = new Intent(KeyGenerator.class.getName());
        bindService(intent, this.mConnection, Context.BIND_AUTO_CREATE);

    }
}

// Unbind from KeyGenerator Service
@Override
protected void onPause() {

    if (mIsBound) {

        unbindService(this.mConnection);

    }

    super.onPause();
}
```