# Services and Threads

Dominic Duggan
Stevens Institute of Technology

1

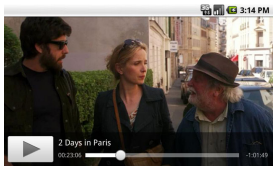
## SERVICES

2

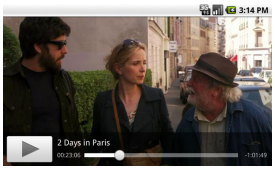
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android ...>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WIFI" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity ...> ... </activity>
        <provider ...> ... </provider>

        <service
            android:name=".ChatService"
            android:exported="false">
        </service>

        <receiver android:name=".ChatApp.Receiver">
            <intent-filter>
                <action android:name="android.action.intent.PROVIDER_CHANGED”/>
            </intent-filter>
        </receiver>

    </application>
</manifest>
```



## Activity



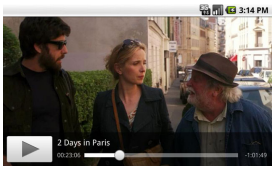
## Activity



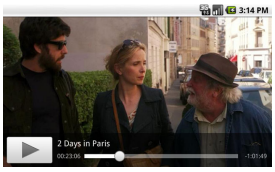


5

## Activity

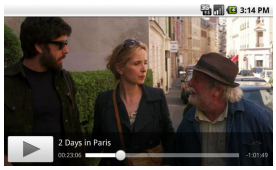


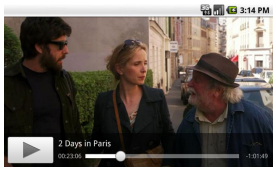




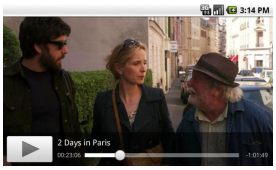

6

## Activity



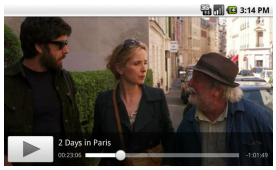



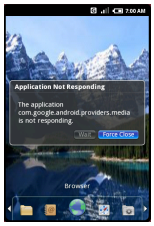

## Service



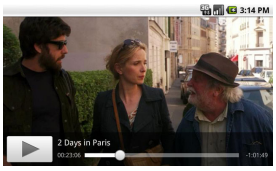

7

## Activity

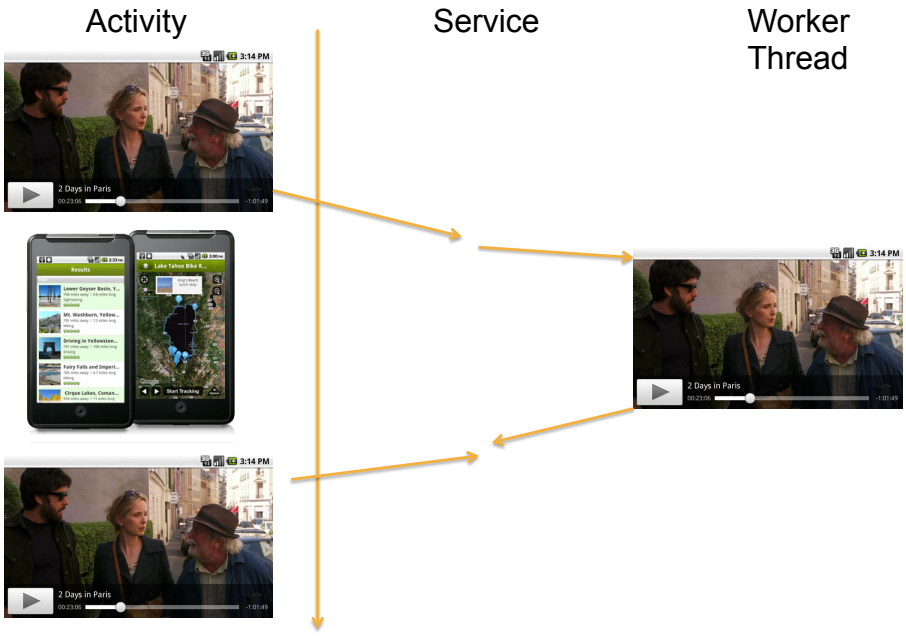



## Service




8




9

# Services

- Motivation: Manage background work
  - I/O (databases, files)
  - Network
  - Music player
  - Image processing (camera)
- Service: framework manages task resources
  - Life cycle: background thread termination

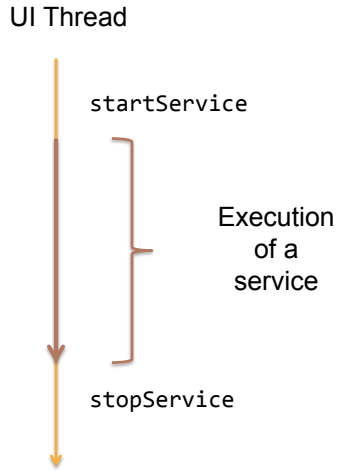
10

# Service Design Issues

- Starting/stopping vs binding/unbinding
  - Perform background request
  - Maintain background activity
  - Communicate with UI
- Local vs remote
- Communication with UI client
  - Passing arguments
  - Receiving intermediate results
  - Receiving final results

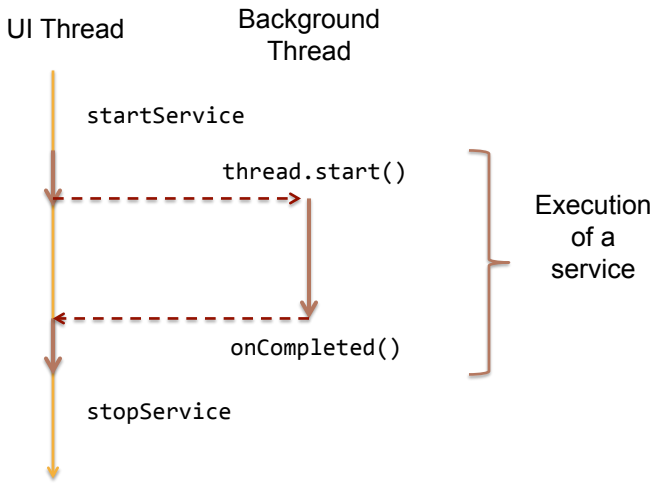
11

# Services and Threads



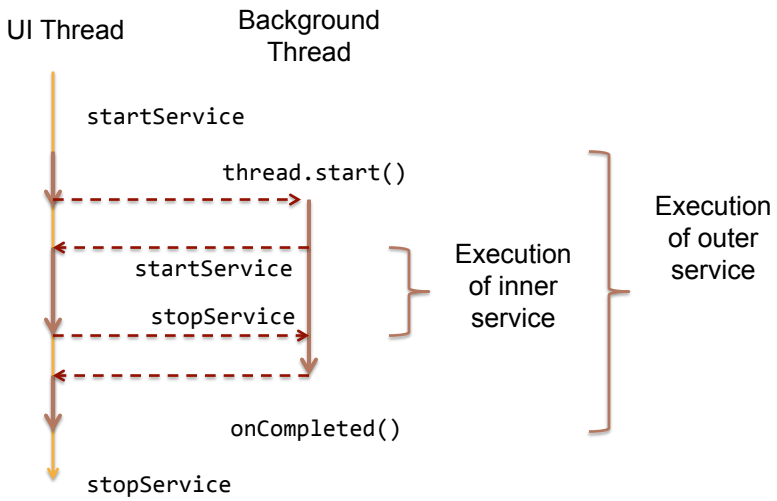

12

# Services and Threads




13

# Services and Threads




14

## INTENT SERVICE

15

## Intent Service

- Simple request-driven service
- Request = intent to start service
- Performed on background thread
- All requests serialized on one thread

16

# Intent Service

- Perform request on background thread
- Client: send service request

```
Intent request = new Intent(this, SendService.class);

request.setAction(SendService.SEND_ACTION);

request.putExtra(...DESTINATION, destinationAddress);
request.putExtra(...SOURCE, mySenderId);
request.putExtra(...MESSAGE, “Hello!”);

startService(request);
```

17

# Intent Service

- Service: receive and perform request

```
public class SendService extends IntentService {

  public SendService() { super(“SendService”); }

  public static void SEND_ACTION =
      “org.example.messages.SEND”;
  public static void MESSAGE = “message”;
  public static void DESTINATION = “destination”;
  public static void SOURCE = “source”;

  ...
}
```

18

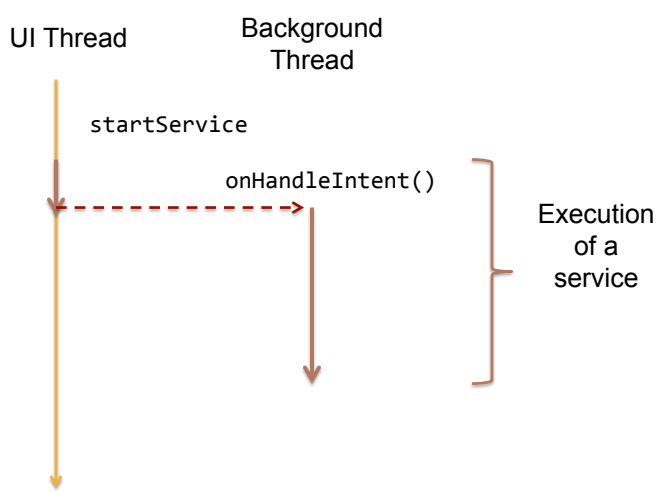
# Intent Service

- Service: receive and perform request

```
public class SendService extends IntentService {
  public void onHandleIntent(Intent request) {
    if (request.getAction().equals(SEND_ACTION) {
      String dest =
        request.getStringExtra(DESTINATION);
      String source = request.getStringExtra(SOURCE);
      String message = request.getStringExtra(MESSAGE);
      Socket socket;
      ...
      socket.send(packet);
    }
  }
}
```

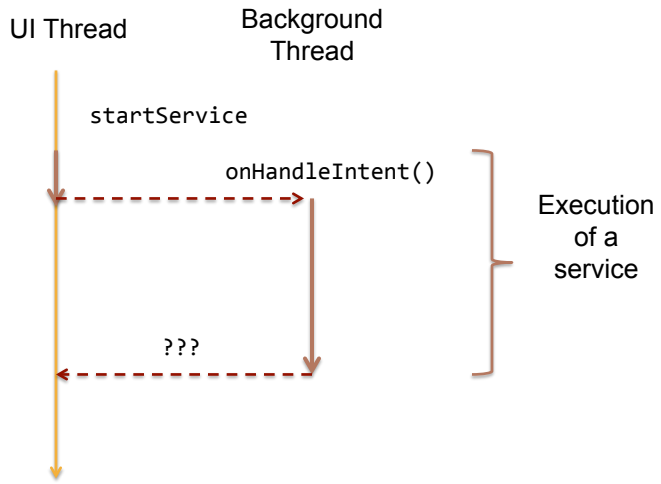
19

# Intent Service




20

# Intent Service with Result




21

# RESULT RECEIVER

22

# Result Receiver

- Client-side receiver
  - Define callback via inheritance:

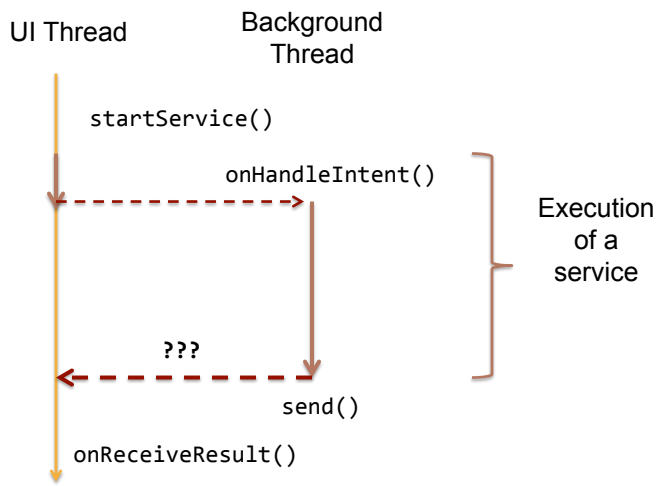
```
public AckReceiver extends ResultReceiver {
  void onReceiveResult(int resultCode,
                       Bundle result) { ... }
}
request.putExtra(RECEIVER, new AckReceiver(...));
```

- Server-side receiver
  - Proxy for sending reply

```
ResultReceiver resultReceiver =
      intent.getExtraParcelable(RECEIVER);
Bundle result = new Bundle();
result.put(ACK, ...);
resultReceiver.send(resultCode, result);
```

23

# Intent Service with Result Receiver




24

# Result Receiver

- Client-side receiver
  - Define callback via inheritance:

```
public AckReceiver extends ResultReceiver {
  void onReceiveResult(int resultCode,
                       Bundle result) { ... }
}
request.putExtra(RECEIVER, new AckReceiver(...));
```

- Server-side receiver
  - Proxy for sending reply

```
ResultReceiver resultReceiver;
Bundle result = new Bundle();
result.put(ACK, ...);
resultReceiver.send(resultCode, result);
```

25

# Result Receiver

- Client-side receiver
  - Define callback via inheritance:

```
Public AckReceiver extends ResultReceiver {
  void onReceiveResult(int resultCode,
                       Bundle result) { ... }
}
request.putExtra(RECEIVER,
                 new AckReceiver(new Handler()));
```

- Server-side receiver
  - Proxy for sending reply

```
ResultReceiver resultReceiver;
Bundle result = new Bundle();
Result.put(ACK, ...);
resultReceiver.send(resultCode, result);
```

26

# Handlers and ResultReceiver

- Handler
  - Per-thread message queue
  - Post messages
  - Post tasks (`Runnable` objects)
- Bind receiver to **current** thread handler:

```
request.putExtra(RECEIVER,
                 new AckReceiver(new Handler()));
```

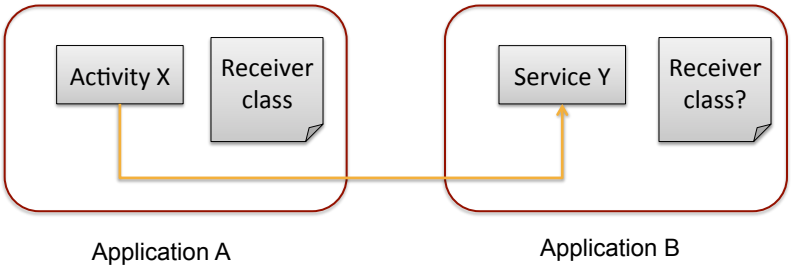
- Post reply to **requester** handler:

```
resultReceiver.send(resultCode, result);
```

27

# Receiver Wrapper

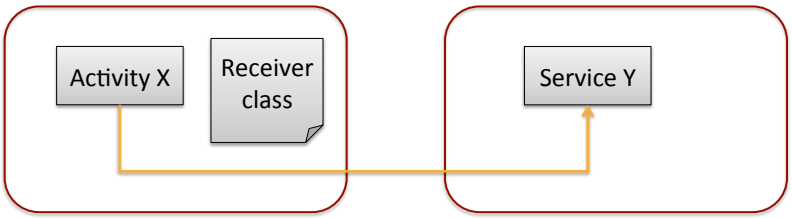
- Issues with `ResultReceiver`
- What if client activity is destroyed?
- Client-specific `ResultReceiver` class in service process?




28

# Receiver Wrapper

```
public class ResultReceiverWrapper
       extends ResultReceiver {

  public ResultReceiverWrapper(Handler handler) {
    super(handler);
  }
```



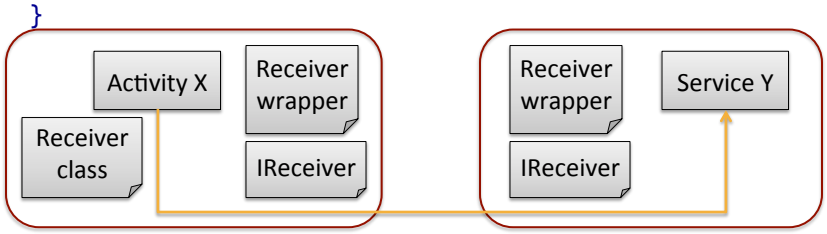

Application A

Application B

29

# Receiver Wrapper

```
public class ResultReceiverWrapper
       extends ResultReceiver {

  private IReceiver receiver;

  public interface IReceiver {
    public void onReceiveResult(int resultCode,
                                Bundle resultData);
  }
```



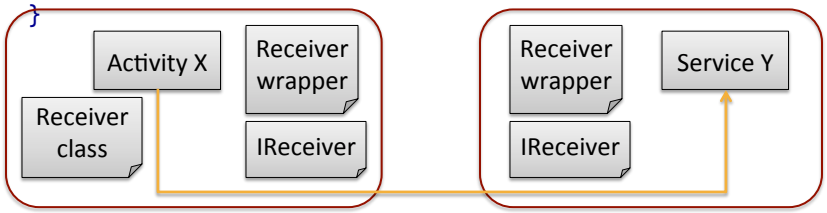

Application A

Application B

30

# Receiver Wrapper

```
public class ResultReceiverWrapper
       extends ResultReceiver {
  public void setReceiver(Ireceiver receiver) {
    this.receiver = receiver;
  }
  protected void onReceiveResult(int resultCode, Bundle data) {
    if (receiver != null) {
      receiver.onReceiveResult(resultCode, resultData);
    }
  }
```




Application A

Application B

31

# Activity

- Declarations

```
IReceiver receiver;
ResultReceiverWrapper wrapper;
```

- Enable receiver

```
public void onResume() {
  super.onResume();
  wrapper.setReceiver(receiver);
}
```

- Disable receiver:

```
public void onDestroy() {
  super.onDestroy();
  wrapper.setReceiver(null);
}
```

32

# BINDING SERVICE

33

# Service

```
public class MyService extends Service {

  @Override
  public void onCreate() {
    // Actions to perform when service is created.
  }

  @Override
  public IBinder onBind(Intent intent) {
    // Replace with service binding implementation.
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int start) {
    // Launch a background thread to do processing.
  }
}
```

34

## Types of Service

- Bound service
  - Defined by connections from clients (activities)
- Started service
  - Explicitly started and stopped
  - Special case: `IntentService`
- Combination

35

## Bound Service

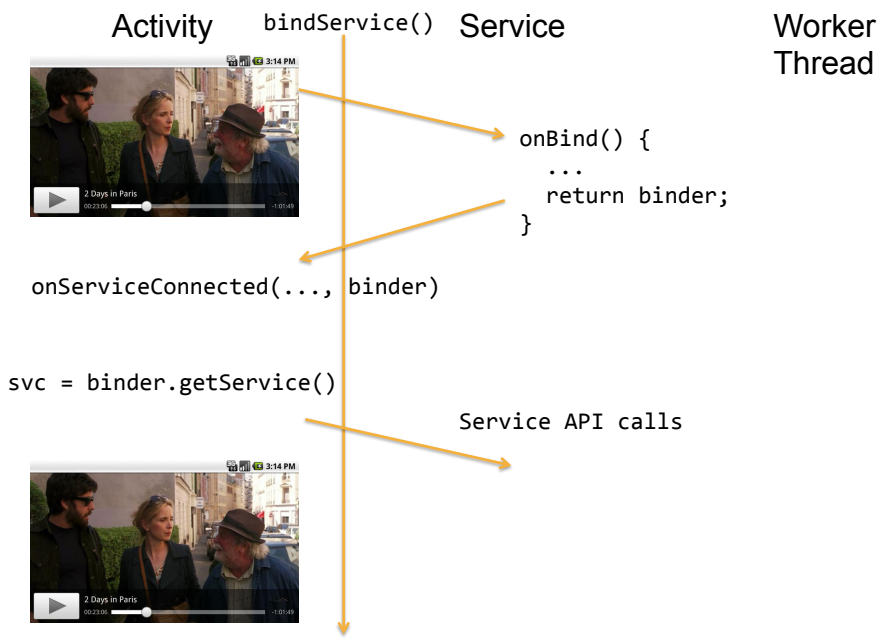
- Started when first client connects
- Ended when last client disconnects
- Binder: client side of connection
  - RPC interface

36

# Bound Service Applications

- Network service
  - Turn on radio when WIFI required
  - Turn off when no clients
- File locking service
  - Start for first lock request
  - Turn off when all locks released

37




38

# Implementing a Binding: Activity

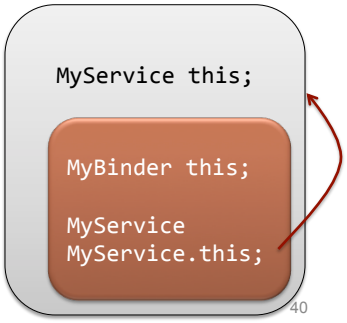
```
private MyService service;

…

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

  // Bind to the service
  Intent bindIntent =
       new Intent(this, MyService.class);
  bindService
       (bindIntent, connection, Context.BIND_AUTO_CREATE);
}
```

39

# Implementing a Binding: Service

```
class MyService extends Service {

  private final IBinder binder = new MyBinder();

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

  public class MyBinder extends Binder {
    MyService getService() {
      return MyService.this;
    }
  }

}
```

```
onBind() {
  ...
  return binder;
}
```




40

# Implementing a Binding: Activity

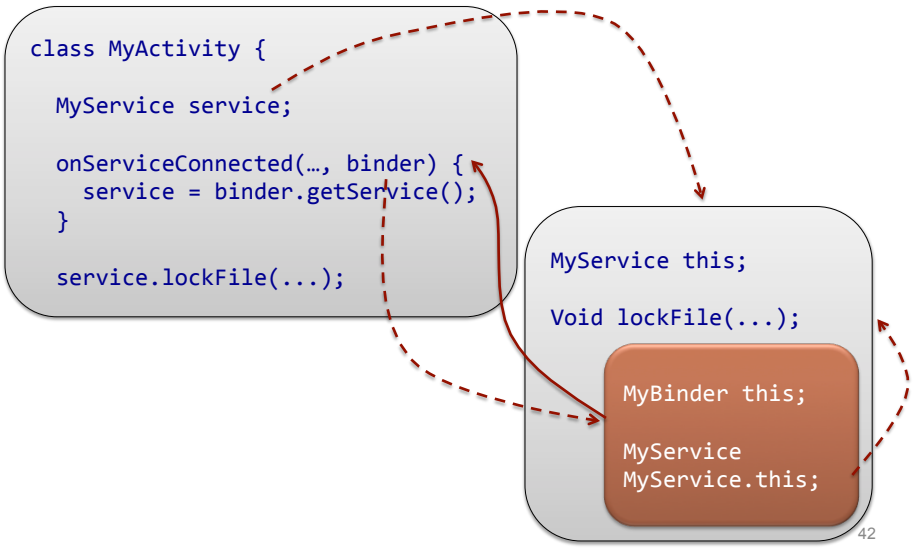
```
private MyService service;

// Handles the connection between the service and activity
private ServiceConnection connection = new ServiceConnection() {
  public void onServiceConnected
              (ComponentName className, IBinder binder) {
    // Called when the connection is made.
    service = ((MyService.MyBinder)binder).getService();
  }

  public void onServiceDisconnected(ComponentName className) {
    // Received when the service unexpectedly disconnects.
    service = null;
  }
};
```

Service API calls

```
onServiceConnected(..., binder)

service =
((MyService.MyBinder)binder).getService()
```

41

# Bound Service




42

# BINDING ACROSS PROCESSES

43

## Binding Across Processes

```
class MyActivity {

  MyService service;

  onServiceConnected(…, binder) {
    service = binder.getService();
  }

  service.lockFile(...);
```

```
MyService this;

void lockFile(...);

onBind()
  return new Binder() {
    getService() {
      return
        MyService.this;
    }
  }
```

# Binding Across Processes

- AIDL interface
  - AIDL language for interfaces
  - Client stubs from interface
- Message channels
  - Less complexity
  - Manual marshalling

45

# Binding Across Processes

```
class MyActivity {

  Messenger messenger;

  onServiceConnected(…, binder) {
    messenger =
      new Messenger(binder);
  }

  messenger.send(message);
```

```
messenger =
  new Messenger(handler);


onBind()
  return
    messenger.getBinder()
```

# Binding Across Processes: Client

```
Messenger messenger;

public void onServiceConnected(ComponentName name,
                               IBinder binder) {
  this.messenger = new Messenger(binder);
}

void sendRequest() {
  Message message = Message.obtain(null, action, 0, 0);
  Bundle args = new Bundle();
  ... Add arguments to args bundle ...
  message.setData(args);
  try {
    messenger.send(message);
  } catch (RemoteException e) {
  }
}
```

47

# Binding Across Processes: Service

```
Messenger messenger;

@Override
public void onCreate() {
  super.onCreate();
  messenger = new Messenger(... messenger handler ...);
}

public IBinder onBind(Intent intent) {
  return messenger.getBinder();
}
```

48

# Binding Across Processes: Service

```
Messenger messenger;

@Override
public void onCreate() {
  super.onCreate();
  messenger = new Messenger(new MessageHandler());
}

class MessageHandler extends Handler {
  public void handleMessage(Message message) {
    Bundle data = message.getData();
    ResultReceiver resultReceiver = data.getParcelable(…);
    switch (message.what) {
      resultReceiver.send(...);
    }
  }
}
```

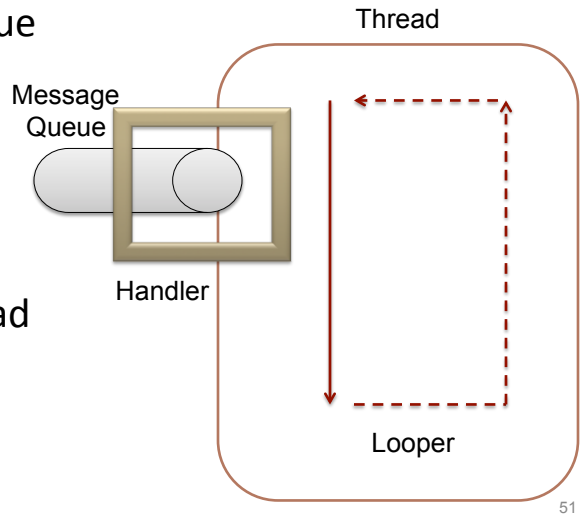
49

# HANDLERS

50

# Handler Abstractions

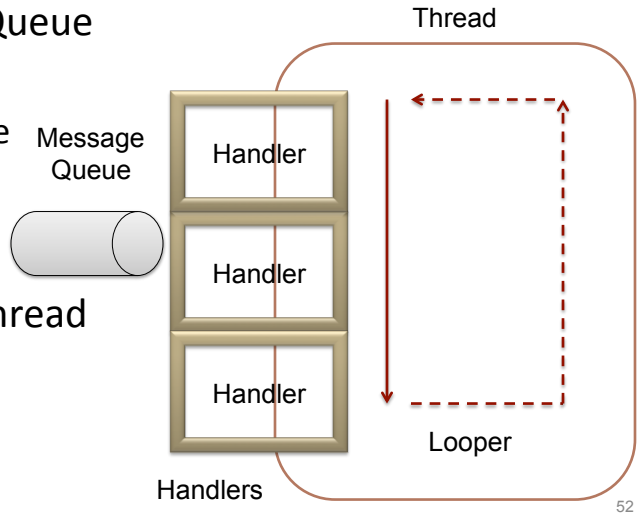
- MessageQueue
  - Message
  - Runnable
- Looper
- Handler
- Handler thread




51

# Handler Abstractions

- MessageQueue
  - Message
  - Runnable
- Looper
- Handler
- Handler thread




52

## Using Handlers

- Define a handler class for processing:

```
class MessageHandler extends Handler {
  void handleMessage(Message message) {
    ...
  }
```

- Instantiate on a looper (thread):

```
Handler handler = new MessageHandler();
```

- Post a message to a handler:

```
handler.sendMessage(…);
```

53

## Using Handlers

- Define a handler class for processing:

```
class MessageHandler extends Handler {
  void handleMessage(Message message) {
    ...
  }
```

- Instantiate on a looper (thread):

```
messenger = new Messenger(new MessageHandler());
```

- Post a message (remotely) to a handler:

```
messenger.send(…);
```

54

## Problem

- Instantiate on a looper (thread):

```
Handler handler = new MessageHandler();
```

- Instantiates on the current thread
- How to instantiate on background thread?
- Define a thread with:
  - Message queue
  - Looper

55

## Handler on Background Thread

- Define a **handler class** for processing:

```
class MessageHandler extends Handler { ... }
```

- Create a background **handler thread**:

```
messengerThread = new HandlerThread(TAG,
          Process.THREAD_PRIORITY_BACKGROUND);
messengerThread.start();
```

- Instantiate handler on background thread **looper**:

```
Looper messengerLooper =
          messengerThread.getLooper();
handler = new MessageHandler(this,
                              messengerLooper);
```

56

## Binding Across Processes: Service

```
Messenger messenger;

public void onCreate() {
  super.onCreate();
  messenger = new Messenger(new MessageHandler());
}

public void onCreate() {
  super.onCreate();
  messengerThread = new HandlerThread(TAG,
                        Process.THREAD_PRIORITY_BACKGROUND);
  messengerThread.start();
  Looper messengerLooper = messengerThread.getLooper();
  handler = new MessageHandler(this, messengerLooper);
  messenger = new Messenger(handler);
}
```

57

## EXPLICITLY STARTED/STOPPED SERVICES

58

# Binding and Disconnecting

- Binding to a Service

```
Intent intent = new Intent(this, MyService.class);
bindService(intent,
            connection,
            flags /* Context.BIND_AUTO_CREATE */ );
```

- Use the Service

```
service.doSomething(...);
/* or messenger.send(...) */
```

- Disconnect:

```
unbindService(connection);
```

59

# Service

```
public class MyService extends Service {

  @Override
  public void onCreate() {
    // Actions to perform when service is created.
  }

  @Override
  public IBinder onBind(Intent intent) {
    // Replace with service binding implementation.
  }

  @Override
  public int onStartCommand(Intent intent, int flags,
                            int startId) {
    // Launch a background thread to do processing.
  }
}
```

60

## Connection-Oriented vs Connectionless

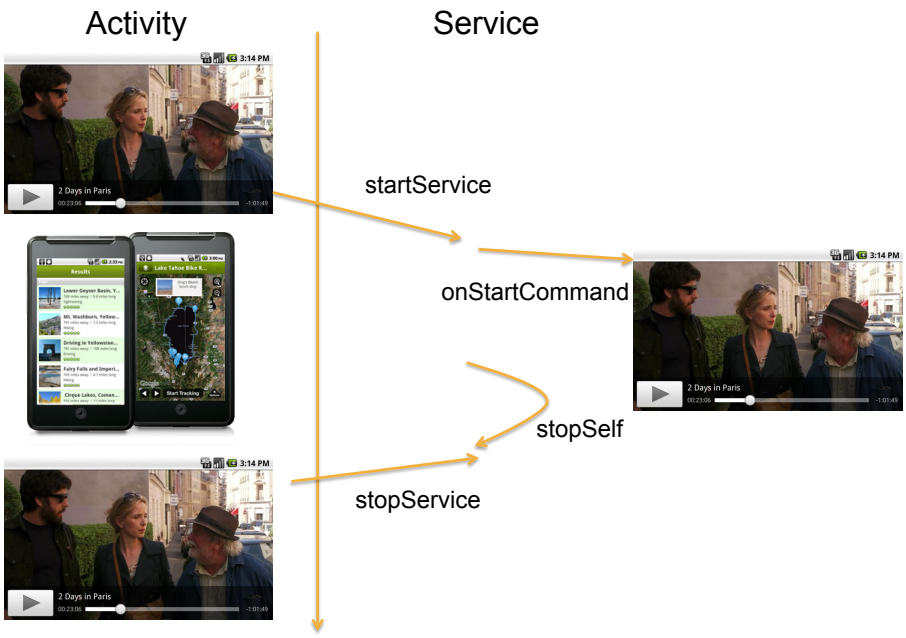
- Connection-oriented
  - Bound service
  - Intent just starts service
  - *Data extras not transmitted to onBind!*
- Connectionless
  - Started service
  - Intent starts service and transmits data

61

## Service

```
public class MyService extends Service {

  @Override
  public void onCreate() {
    // Actions to perform when service is created.
  }

  @Override
  public IBinder onBind(Intent intent) {
    // Replace with service binding implementation.
  }

  @Override
  public int onStartCommand(Intent intent, int flags,
                            int startId) {
    // Launch a background thread to do processing.
  }
}
```

62




63

# Stopping & Starting

- Starting a Service

```
// Explicitly start a Service
startService(new Intent(this, MyService.class)
             flags,
             requestID);
```

- Stopping a Service

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

64

## Service Operations

- On creation
```
void onCreate()
```
- On starting
```
int onStartCommand(Intent intent,
                   int flags,
                   int requestID)
```
- Stopping the service
```
stopSelf()

stopSelfResult(requestID);  /* Maintain order! */
```

65

## Service Operations

- On starting
```
int onStartCommand(Intent intent, int flags,
                   int requestID)
```
- Start Flags
  - `START_FLAG_REDELIVERY`: restarting killed service
  - `START_FLAG_RETRY`: restarting killed during start
- Return flags
  - `START_STICKY`: leave "started" if killed
  - `START_NOT_STICKY`: don't restart without request
  - `START_REDELIVER_INTENT`: restart with intent if killed

66

# Example: IntentLikeService

```
public class IntentLikeService extends Service {

  private Looper looper;
  private ServiceHandler handler;

  private final class ServiceHandler extends Handler {
      public ServiceHandler(Looper looper) {
          super(looper);
      }
      @Override
      public void handleMessage(Message msg) { }
          ... do stuff ...
          // Stop the service using the startId, so that we don't
          // stop the service in the middle of handling another job
          stopSelf(msg.arg1);
      }
  }
```

67

# Example: IntentLikeService

```
public class IntentLikeService extends Service {

  public void onCreate() {
     HandlerThread thread = new HandlerThread(...,
            Process.THREAD_PRIORITY_BACKGROUND);
     thread.start();
     looper = thread.getLooper();
     handler = new ServiceHandler(looper);
  }

  public int onStartCommand(Intent intent, int flags, int startId){
     Message msg = handler.obtainMessage();
     msg.arg1 = startId;
     handler.sendMessage(msg);
     return START_STICKY;
  }
```

68

## BACKGROUND TASKS

69

## APIs for Background Threads

- Java threads
- Handler API
- AsyncTask
- `Activity.runOnUiThread()`

70

## APIs for Background Threads

- Java threads

```
Runnable code = new Runnable() {
  public void run() {
    ... do work ...
  }
}
new Thread(code).start();
```

- Handler API
- AsyncTask
- `Activity.runOnUiThread()`

71

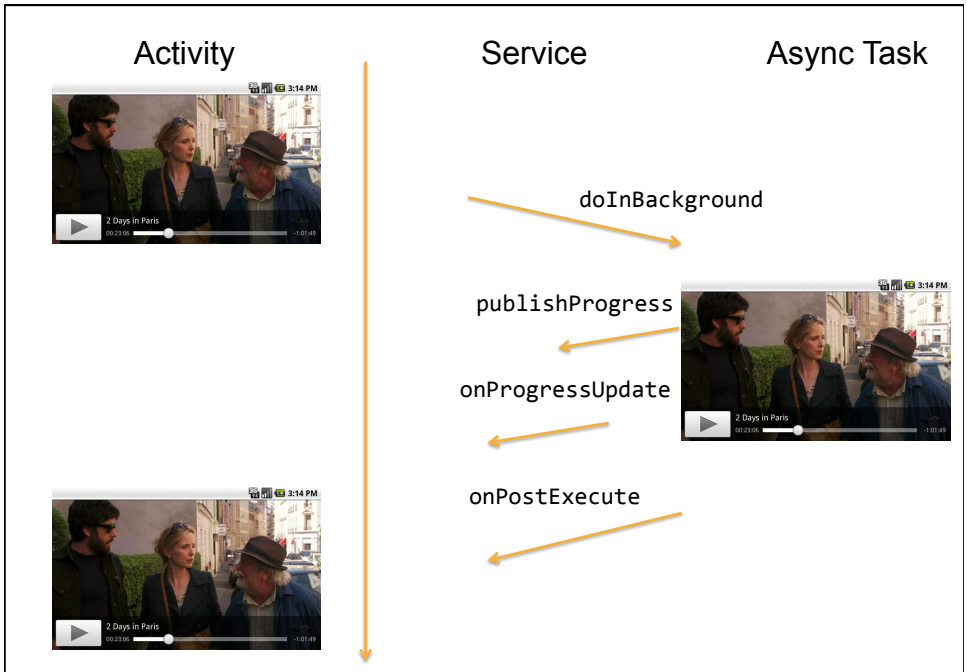
## APIs for Background Threads

- Java threads
- Handler API

```
class MyHandler extends Handler { ... }
handlerThread = new HandlerThread(...);
handlerThread.start();
Looper looper = handlerThread.getLooper();
handler = new MyHandler(this, looper);
handler.sendMessage(...);
```

- AsyncTask
- `Activity.runOnUiThread()`

72




# Asynchronous Task

```
private class MyAsyncTask extends AsyncTask<String,Integer,Integer>
{
  @Override
  protected void onProgressUpdate(Integer... progress) {
    // Update progress bar, Notification, or other UI element
  }

  @Override
  protected void onPostExecute(Integer... result) {
    // Report results via UI update, Dialog, or notification
  }

  ...

}
```

Input parameter type

Progress report type

Result type

74

# Asynchronous Task

```
private class MyAsyncTask extends AsyncTask<String,Integer,Integer>
{

  @Override
  protected Integer doInBackground(String... parameter) {
    int myProgress = 0;

    // Perform background processing task, update myProgress

    publishProgress(myProgress)

    // Continue performing background processing task

    // Return the value to be passed to onPostExecute
    return result;
  }
}
```

75

# Asynchronous Task

```
private class MyAsyncTask extends AsyncTask<String,Integer,Integer>
{
  @Override
  protected Integer doInBackground(String...) {
    ...
    publishProgress(myProgress)
    ...
  }
  @Override
  protected void onProgressUpdate(Integer...) { ... }

  @Override
  protected void onPostExecute(Integer...) { ... }
}
```

76

# APIs for Background Threads

- Java threads
- Handler API
- AsyncTask

```
AsyncTask<Void,Void,Integer> task =
  new AsyncTask<Void,Void,Integer>() {
    Integer doInBackground(Void ...params) {
      ... do work on background thread ... return 3;
    }
    void onPostExecute(Integer result) {
      ... post-processing on UI thread ...
    }
  };
task.start();
```

- `Activity.runOnUiThread()`

77

# APIs for Background Threads

- Java threads
- Handler API
- AsyncTask
- `Activity.runOnUiThread()`

```
public class MyActivity extends Activity
      implements LoaderManager.LoaderCallbacks<Cursor>{
  public void onLoadFinished(Loader<Cursor> loader,
                             Cursor cursor) {
    MyActivity.this.runOnUiThread(new Runnable(){
      public void run() {
        ... perform UI updates on the UI thread ...
      }
    }
  }
}
```

78

# BROADCAST RECEIVER

79

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android ...>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WIFI" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /
>

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity android:name=".ChatApp" ...>
        <provider ...>
        <service ...>

        <receiver android:name=".ChatApp.Receiver">
            <intent-filter>
               <action android:name="android.action.intent.PROVIDER_CHANGED”/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

# Background Processing

```
public static final String NEW_MESSAGE_BROADCAST =
      "edu.stevens.cs522.chat.NewMessageBroadcast";

private class ReceiveMessageTask extends
               AsyncTask<Void, MessageInfo, Void> {

   protected Void doInBackground(Void... params) {

      while (true) {
         MessageInfo msg = nextMessage();
         addReceivedMessage(msg);
         ...
      }
   }

}
```

81

# Background Processing

```
private class ReceiveMessageTask extends
               AsyncTask<Void, MessageInfo, Void> {

   private Intent msgUpdateBroadcast =
                  new Intent(NEW_MESSAGE_BROADCAST);

   public void addReceivedMessage(MessageInfo msg) {
      ContentResolver cr = getContentResolver();
      cr.insert(ChatContract.CONTENT_URI, ...);

      /*
       * Logic for updating the Messages cursor
       * is done on the UI thread.
       */
      sendBroadcast(msgUpdateBroadcast);
   }
```

82

# Foreground Update

- Define broadcast receiver
  - Notification for UI updates
- Register when activity becomes active
- Unregister when activity becomes inactive

83

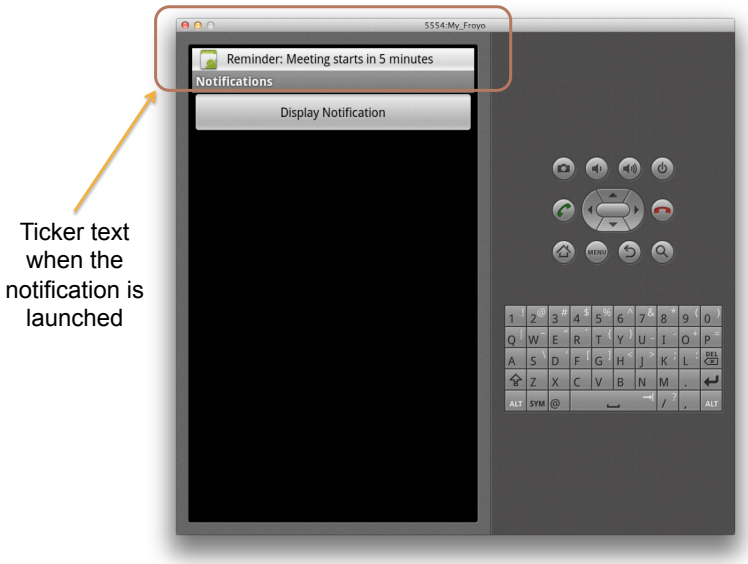
# Broadcast Receiver

```
public class ChatApp extends Activity {

   CursorAdapter messageAdapter;

   public static class Receiver
                  extends BroadcastReceiver {
      public void onReceive (Context ctxt, Intent intent) {
         messageAdapter.swapCursor(...);
      };
   }

   Receiver receiver;

   public void onStart() { registerReceiver(receiver, filter); }

   public void onStop() { unregisterReceiver(receiver); }
```

84

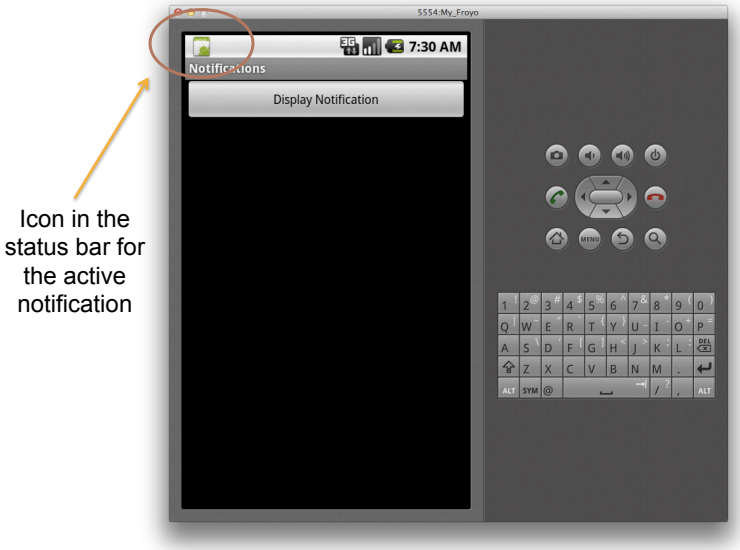
# NOTIFICATIONS

85



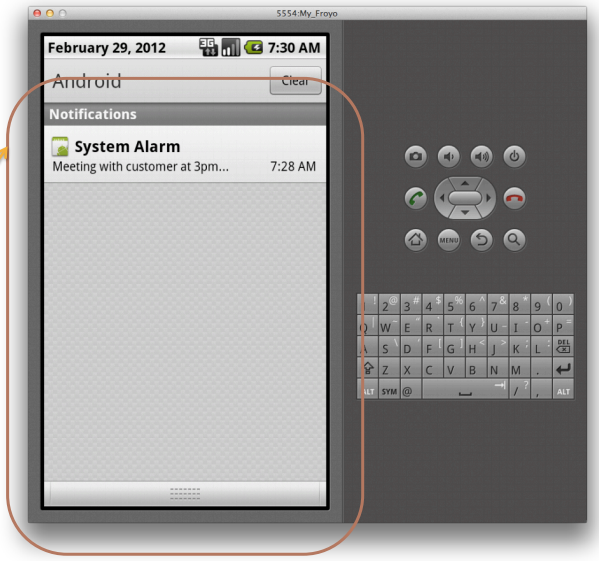

Ticker text when the notification is launched

86



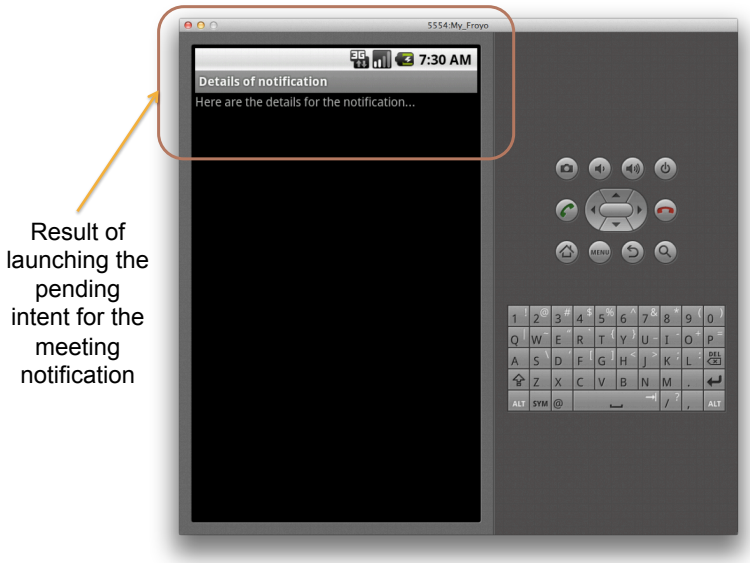

Icon in the status bar for the active notification

87




Extended status bar with further details

88




Result of launching the pending intent for the meeting notification

89

# Notifications

- Creating a Notification

```
// Choose a drawable to display as the status bar icon
int icon = R.drawable.icon;

// Text to display in status bar when notify launched
String tickerText = "Notification";

// The extended status bar orders notification in time order
long when = System.currentTimeMillis();

Notification notification =
      new Notification(icon, tickerText, when);
```

90

# Notifications

- Getting the Notification Manager

```
String svcName = Context.NOTIFICATION_SERVICE;

NotificationManager notificationManager;

notificationManager =
      (NotificationManager)getSystemService(svcName);
```

- Firing the notification

```
int notificationRef = 1;  // App-specific notify identifier
notificationManager.notify
      (notificationRef, notification);
```

91

# Setting Notification Values

```
Context context = getApplicationContext();
// Text to display in the extended status window
String expandedText = "Extended status text";
// Title for the expanded status
String expandedTitle = "Notification Title";

// Intent to launch activity when extended text is clicked
Intent intent = new Intent(this, MyActivity.class);
PendingIntent launchIntent =
      PendingIntent.getActivity(context, 0, intent, 0);

notification.setLatestEventInfo(context,
                                expandedTitle,
                                expandedText,
                                launchIntent);
```

92

## CONCLUSIONS

93

# Conclusions

- Services
  - Started
  - Bound
- Thread management
  - Handlers
  - `AsyncTask` and `runOnUiThread`
- UI updates
  - Loader manager (last week)
  - Broadcast receiver
  - Notification

94