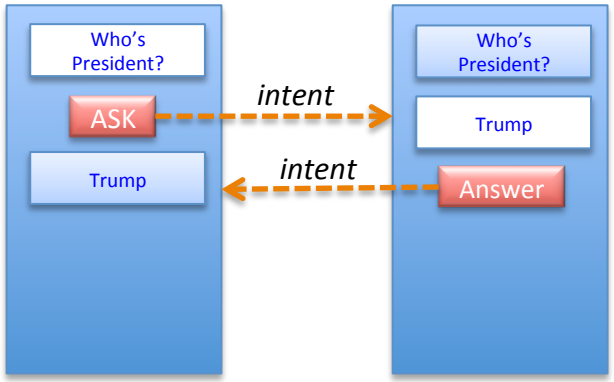
Android

Widgets

- TextView
  - getText(), setText()
- Button
  - Event handling
- EditText
  - getText(), setText()
- Seekbar
  - setOnSeekBarChangeListener()
- Progressbar
  - setMax(), setProgress()
- ToggleButton
  - setOnCheckedChangeListener()
- WebView
  - loadUrl(), reload(), goBack(), goForward(), canGoBack(), canGoForward()
  - <use-permission android:name="android.permission.INTERNET"/>
  - setWebViewClient(new WebViewClient);

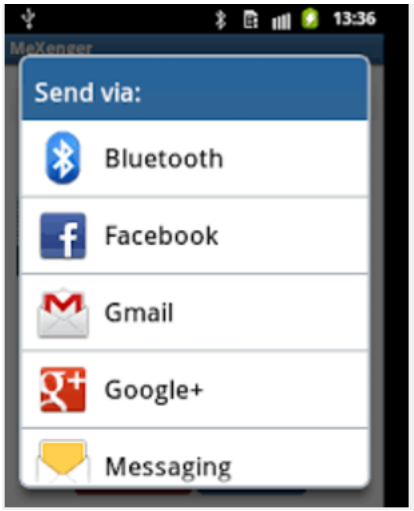
Starting Activity with Data Passing

- Start activity without data passing
  - startActivity(new Intent(this, <class>));
- Start activity with data sent
  - Caller
    - Intent intent = new Intent(this, <class>);
    - Intent.putExtra(<key>, <data>)
    - startActivity(intent);
  - Callee
    - getIntent().getStringExtra(<key>);
- Start activity with data back
  - Caller
    - startActivityForResult(<intent>, <request code>)
    - onActivityResult(<request code>, <result-code>, <intent>)
      - <intent>.getStringExtra(<key>)
  - Callee
    - Intent.putExtra(<key>, <data>)
    - setResult(<result-code>, <intent>)
    - Finish()



Sending Implicit Intents

- Sending implicit intents
  - <intent>=new Intent(Intent.<action>);
    <intent>.setData(Uri.parse(<uri>);
    <intent>.putExtra(...)
    <intent>.setType(…)
    startActivity(<intent>)
  - Chooser
    - startActivity(Intent.createChooser(<int>, <title>));



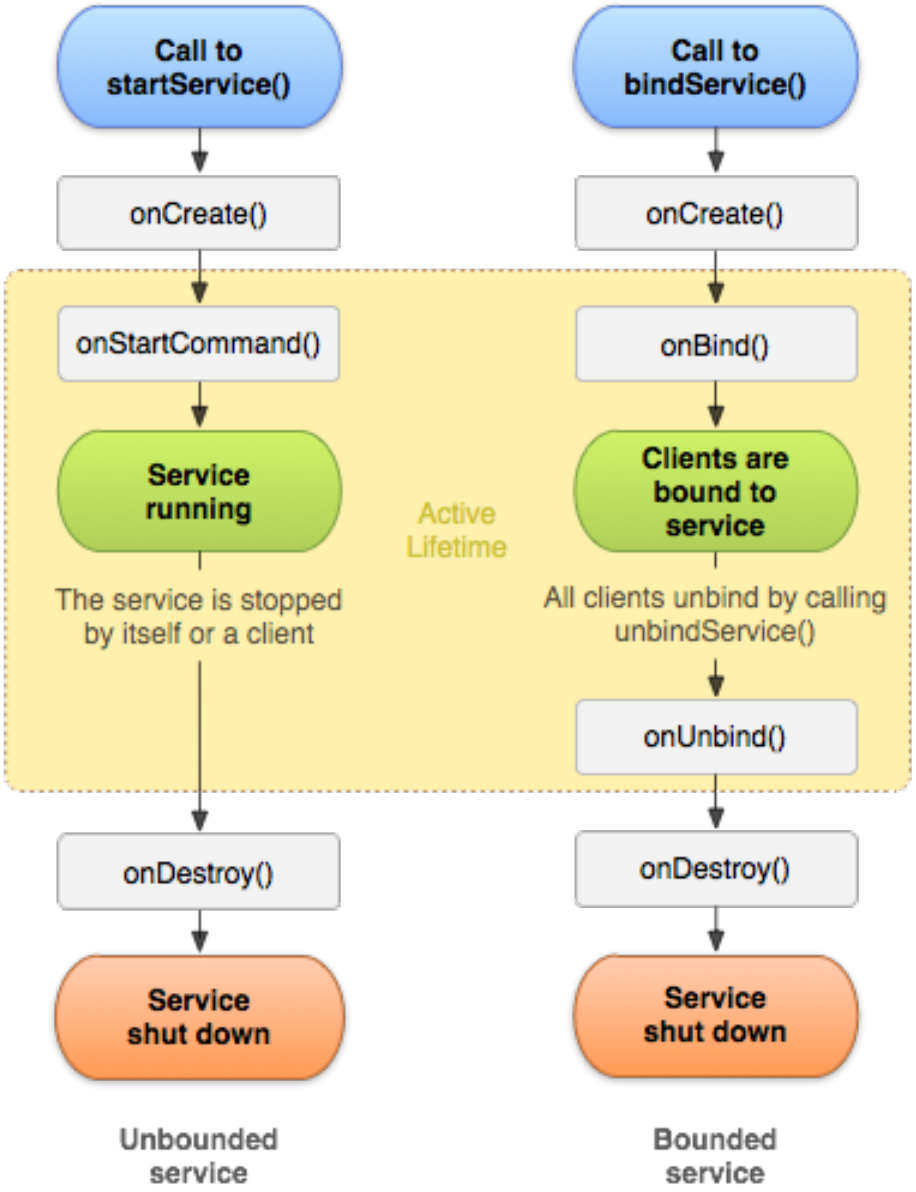
- Open a map
  - <action>=ACTION_VIEW
  - <uri>="geo:<lat>,<lon>"
- Open a URL in Webrowser
  - <action>=ACTION_VIEW
  - <uri>="http://<url>"
- Open an App Store
  - <uri>="market://details?id=<id>"
  - If App store is not available, try to catch ActivityNotFoundException, and do in browser
- Send an email
  - <action>=ACTION_SENDTO
  - <uri>="mailto:"
  - <intent>.putExtra(Intent.EXTRA_{EMAIL|SUBJECT|TEXT}, <data>)
  - <intent>.setType("text/plain")
- Sending a text
  - Use Intent.ACTION_SEND, Intent.EXTRA_TEXT

Services

- Service
  - Perform long-running background operations
  - No user interface
  - Other application component can run it
  - Continue to run when Activity stops
  - Component can interact with it via IPC
- Service Types (by starting method)
  - Started
    - A component (eg. Activity) calls startService()
    - No return. When finished, stop itself
  - Bound
    - A component calls bindService()
    - Perform interactions in a sense of client-server concept
    - Exists only if there is a bound component
- Note:
  - A service can be both started (onStartCommand()) and bound (onBind())
  - Whether started or bound, any component can use the service via Intent
    - Unless declared *private*
  - To avoid ANR (App Not Responding) error, use new thread for CPU-intensive work

Service Lifecycle

- No need to call superclass methods
- onCreate
  - initialization. eg, create thread for music play
- onDestroy
  - release resources
- onStartCommand
  - start the job, until stopService()
- onBind
  - return IBinder for interface. Continue until unbindService() from all clients



Declaring a service

- AndroidManifest.xml

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

- Private
  - android;exported=“false”
- Eg
  - <service android:name=“.myService” android:exported=“false” />

Implementing Started Service

- Client
  - Start the service
    - <intent> = new Intent(<context>, <service class>);
      <intent>.putExtra(<key>, <value>);
      startService(<intent>);
  - Stop the service
    - stopService(new Intent(this, <service class>));
- Service
  - Create a service (by Android Studio)
    - Declare in manifest
    - Create a class that extends Service class
  - Implement onStartCommand()
    - <intent>.getStringExtra(<key>)
    - Return <result_code>
      - START_STICKY: when killed, recreate/restart with pending or null intent. Media services
      - START_NOT_STICKY: when killed, recreate/restart with pending intent, otherwise not.
      - START_REDELIVER_INTENT: when killed, recreate/restart with last intent, then pending.

Service with Threads

- Service is done in the same process with hosting App
- CPU-intensive or IO-intensive job can affect the performance of the App
- Therefore, if needed, do the job in a thread
- Solutions
  - Extend IntentService class
    - Creates a thread, and handle requests one after another
  - Create your own threads
    - If need multiple threads to handle requests simultaneously
    - Covered later

Extending IntentService class

- IntentService does
  - Craete a worker thread to handle intents received by onStartCommand()
  - Create a work queue handled by onHandleIntent()
  - Stop service if queue is empty
  - Default onStartCommand() that enqueues intents
  - Default onBind() that returns null
- You do
  - Extend IntentService class
  - Implement constructor, calling super constructor with thread name as a parameter
  - Implement onHandleIntent(<intent>)
  - Optionally, implement onCreate(), onStartCommand(), onDestroy()
    - But call super class methods (eg., return super.onStartCommand())

MediaPlayer

- https://developer.android.com/reference/android/media/MediaPlayer.html
- Play a media file such as mp3
- Methods
  - MediaPlayer.create(<context>, R.raw.<id>)
  - setLoopling(<boolean>)
  - start(), pause(), stop(), release(), isPlaying()
    getDuration(), getCurrentPosition()
- Listener
  - setOnCompletionListener()
    - OnCompletionListener.onCompleteion(MediaPlayer m)