Mobile Software Development for Android - I397

IT COLLEGE, ANDRES KÄVER, 2015-2016

EMAIL: AKAVER@ITCOLLEGE.EE

WEB: HTTP://ENOS.ITCOLLEGE.EE/~AKAVER/2015-2016/DISTANCE/ANDROID

SKYPE: AKAVER

Android app components

- Logging
- Activity
- Intent
- Fragment
- Service
- ContentProvider
- BroadcastReceiver

Android - logging

- Full support in ide
- Use android.util.Log
- Verbose Log.v(), debug Log.d(), info Log.i(), warn Log.w(), error Log.e(), or "what a terrible Failure" Log.wtf()
- Deployed application should not contain logging code
- Use BuildConfig.DEBUG flag for checking state (deployed or not)
- if (BuildConfig.DEBUG) { Log.e(Constants.TAG, "onCreate called"); }
- TAG – string, usually statically fixed

- public static int w (String tag, Throwable tr)
- public static int w (String tag, String msg, Throwable tr)

Android - activity

- Activity – one screen (UI and code)
- User interface – 1..n activities
- Every activity is separate component
- Activity can start other activities
- Back stack (lifo)

Android - activity

- Activity
- FragmentActivity
- ListActivity
- PreferenceActivity
- TabActivity

Android - activity

- One activity is designated as “Main”
  - Launched on first app activation
- Every time new activity is started, previous one is stopped
- Previous activity is stored in the back stack
- When activity is stopped/paused, callback methods are called
- Callbacks – create, resume, stop, destroy, etc…

Android – new activity

- Create subclass of Activity (or subclass of subclass of Activity)
- Implement callbacks
  - OnCreate()
  - OnPause()
- Implement user interface
  - XML layout file
  - Or programmatically

Android – new activity, manifest

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

- Declaration in AndroidManifest is mandatory
- Specify intent filters
  - Intent filter declares, how **other** system components may use this activity
- Auto-created stub for main activity
  - Action action.MAIN – activity responds to the “main” action
  - Category category.LAUNCHER – actitvity is placed into launcher category

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Android – new activity, starting

- startActivity(intent)
- Starting your own activity – specify class name

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

- Calling other activities

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

- Intent.EXTRA_EMAIL – stores list of email recipients

Android – new activity, starting for result

- StartActivityForResult()
- Implement onActivityResult() callback method

```
private void pickContact() {
    // Create an intent to "pick" a contact, as defined by the content provider URI
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
    startActivityForResult(intent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // If the request went well (OK) and the request was PICK_CONTACT_REQUEST
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
        // Perform a query to the contact's content provider for the contact's name
        Cursor cursor = getContentResolver().query(data.getData(),
        new String[] {Contacts.DISPLAY_NAME}, null, null, null);
        if (cursor.moveToFirst()) { // True if the cursor is not empty
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
            String name = cursor.getString(columnIndex);
            // Do something with the selected contact's name...
        }
    }
}
```

Android – shut down activity

- Finish()
- shut down previously started activity – finishActivity()

Android – activity lifecycle

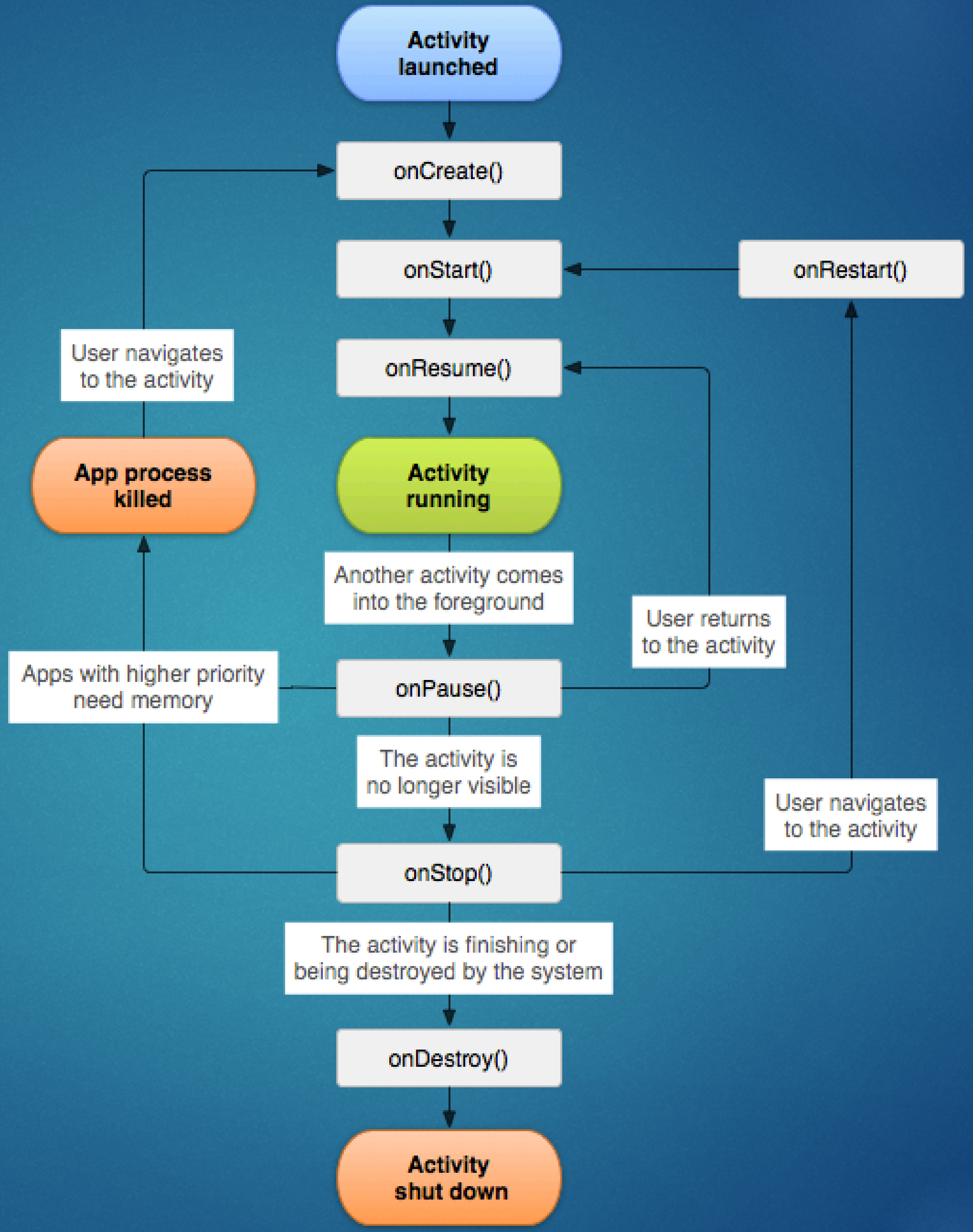
- Three essential states
  - Resumed
    - In foreground, has user focus. "running"
  - Paused
    - Activity is partially visible, and is "alive". Can be killed by system in low memory situation
  - Stopped
    - Activity is 100% obscured by another activity. It is alive, but is not attached to the window manager. Can be killed by system, when memory is needed.
- Paused or Stopped – system calls finish() method on activity. On kills its process. When activity is reopened, it must be created again

Android – Lifecycle callbacks

- Fundamental callbacks
- Must always call the superclass implementation before doing any work

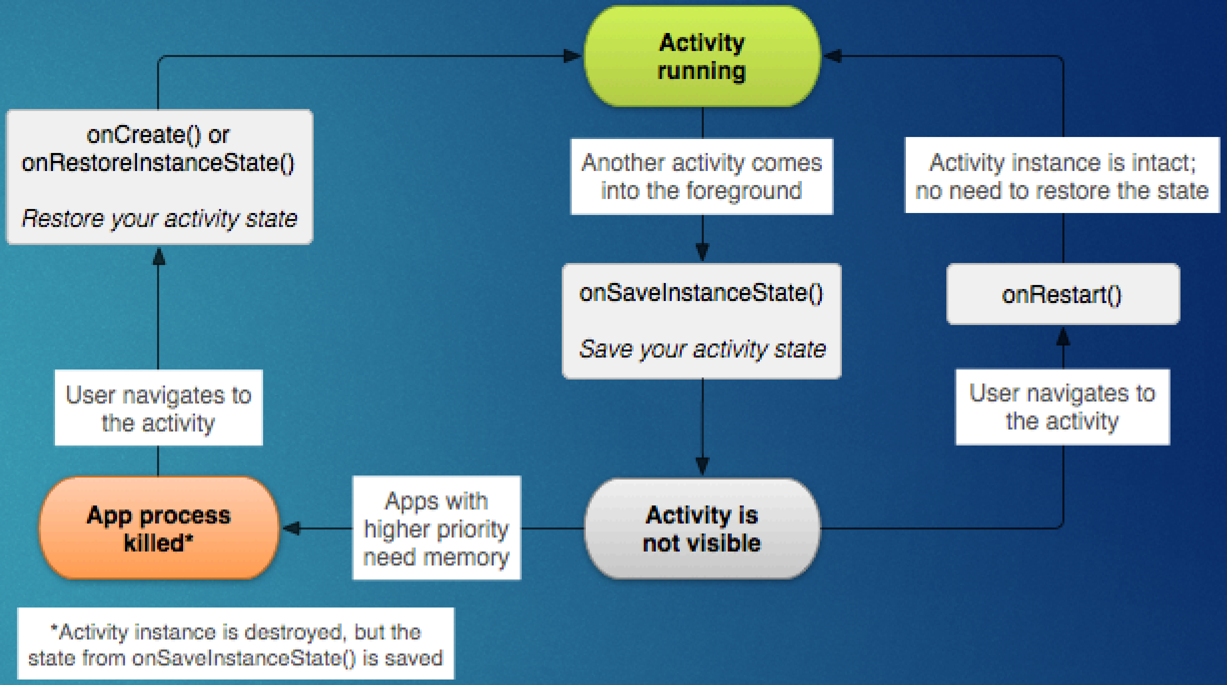
```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Android – Lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Android - SaveInstanceState

- system calls onSaveInstanceState() before making the activity vulnerable to destruction
- Passes Bundle, as name-value pairs
- Save state, using
  - putString() and putInt()
- Bundle is passed back in
  - onCreate() and onRestoreInstanceState()


Activity running
onCreate() or onRestoreInstanceState()
Restore your activity state
Another activity comes into the foreground
Activity instance is intact; no need to restore the state
onSaveInstanceState()
Save your activity state
onRestart()
User navigates to the activity
User navigates to the activity
App process killed*
Apps with higher priority need memory
Activity is not visible
*Activity instance is destroyed, but the state from onSaveInstanceState() is saved

Android - handling conf changes

- Orientation change, physical keyboard, language
- System recreates the running activity
  - calls onDestroy(),
  - then immediately calls onCreate()

Android – Intent and Intent filters

- Messaging object, used for requesting action from another app component
- Fundamental uses
  - Start an activity
    - startActivity or startActivityForResult
  - Start a service
    - startService or bindService
  - Deliver broadcast
    - sendBroadcast, sendOrderedBroadcast, or sendStickyBroadcast

Android – Intent and Intent filters

- Explicit intents
  - Specify component by name (usually in your own app)
- Implicit intents
  - Declare general action to perform
  - System searches in manifests (intent-filter) for suitable activity
  - If several are found, user is presented with dialog for picking

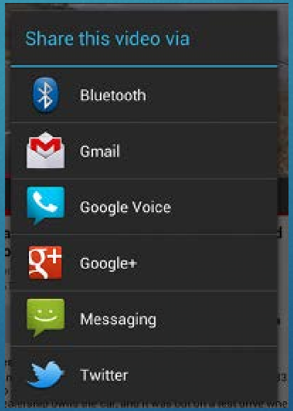
Android - intents

- Explicit intent
- Implicit intent
- Forcing an app chooser
  - To show the chooser, create an Intent using createChooser() and pass it to startActivity()

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
String title = getResources().getString(R.string.chooser_title);
// Create intent to show the chooser dialog
Intent chooser = Intent.createChooser(sendIntent, title);

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```



Android – intent filter

- To advertise which implicit intents your app can receive
- declare one or more intent filters for each of your app components with an <intent-filter> element in your manifest file
- Action
  - intent action accepted, in the name attribute
- Data
  - type of data accepted, using one or more attributes that specify various aspects of the data URI (scheme, host, port, path, etc.) and MIME type
- Category
  - category accepted, in the name attribute.
  - In order to receive implicit intents, you must include the CATEGORY_DEFAULT

Android – intent filter

- Activity declaration with an intent filter to receive an ACTION_SEND intent when the data type is text

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

- An implicit intent is tested against a filter by comparing the intent to each of the three elements.
- To be delivered to the component, the intent must pass all three tests.

Android – intent filter

- Social app

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

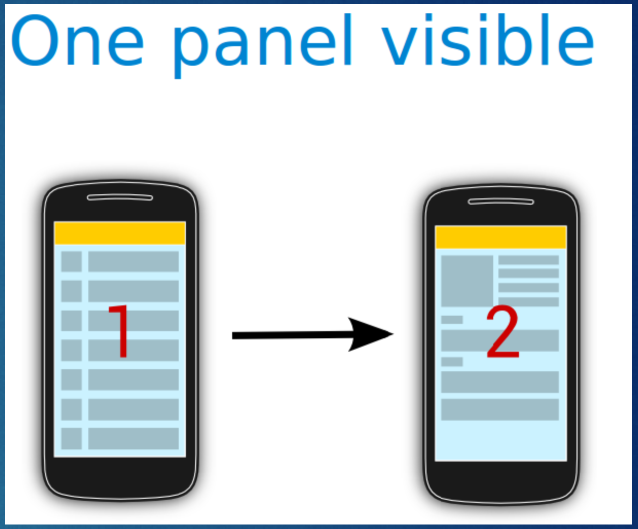
Android – receive intent - code

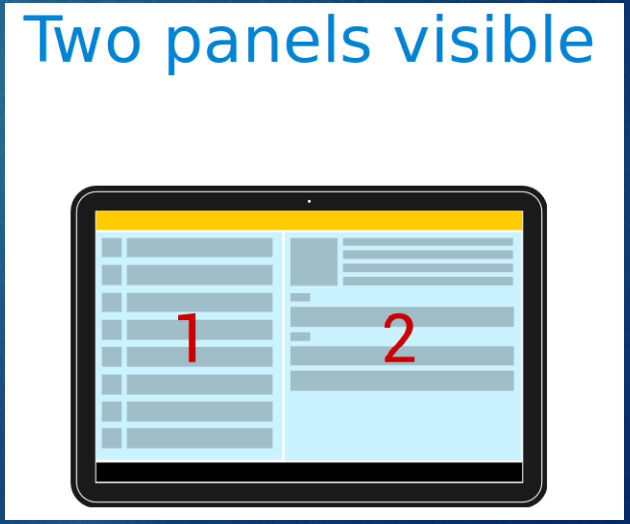
```java
void onCreate (Bundle savedInstanceState) {
    ...
    // Get intent, action and MIME type
    Intent intent = getIntent();
    String action = intent.getAction();
    String type = intent.getType();

    if (Intent.ACTION_SEND.equals(action) && type != null) {
        if ("text/plain".equals(type)) {
            handleSendText(intent); // Handle text being sent
        } else if (type.startsWith("image/")) {
            handleSendImage(intent); // Handle single image being sent
        }
    } else if (Intent.ACTION_SEND_MULTIPLE.equals(action) && type != null) {
        if (type.startsWith("image/")) {
            handleSendMultipleImages(intent); // Handle multiple images being sent
        }
    } else {
        // Handle other intents, such as being started from the home screen
    }
    ...
}
```

```java
void handleSendText(Intent intent) {
    String sharedText = intent.getStringExtra(Intent.EXTRA_TEXT);
    if (sharedText != null) {
        // Update UI to reflect text being shared
    }
}

void handleSendImage(Intent intent) {
    Uri imageUri = (Uri) intent.getParcelableExtra(Intent.EXTRA_STREAM);
    if (imageUri != null) {
        // Update UI to reflect image being shared
    }
}

void handleSendMultipleImages(Intent intent) {
    ArrayList<Uri> imageUris = intent.getParcelableArrayListExtra(Intent.EXTRA_STREAM);
    if (imageUris != null) {
        // Update UI to reflect multiple images being shared
    }
}
```

Android - Fragments

- A panel or pane represents a part of the user interface.
- A *fragment* is an independent Android component which can be used by an activity.
- A fragment runs in the context of an activity, but has its own life cycle and typically its own user interface.
- No Context class, use the getActivity()
-





Android - Fragments

- Extend the android.app.Fragment class or one of its subclasses, for example, ListFragment, DialogFragment, PreferenceFragment or WebViewFragment

```java
public class DetailFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_rssitem_detail,
                container, false);
        return view;
    }

    public void setText(String url) {
        TextView view = (TextView) getView().findViewById(R.id.detailsText);
        view.setText(url);
    }
}
```

Android – Fragments communication

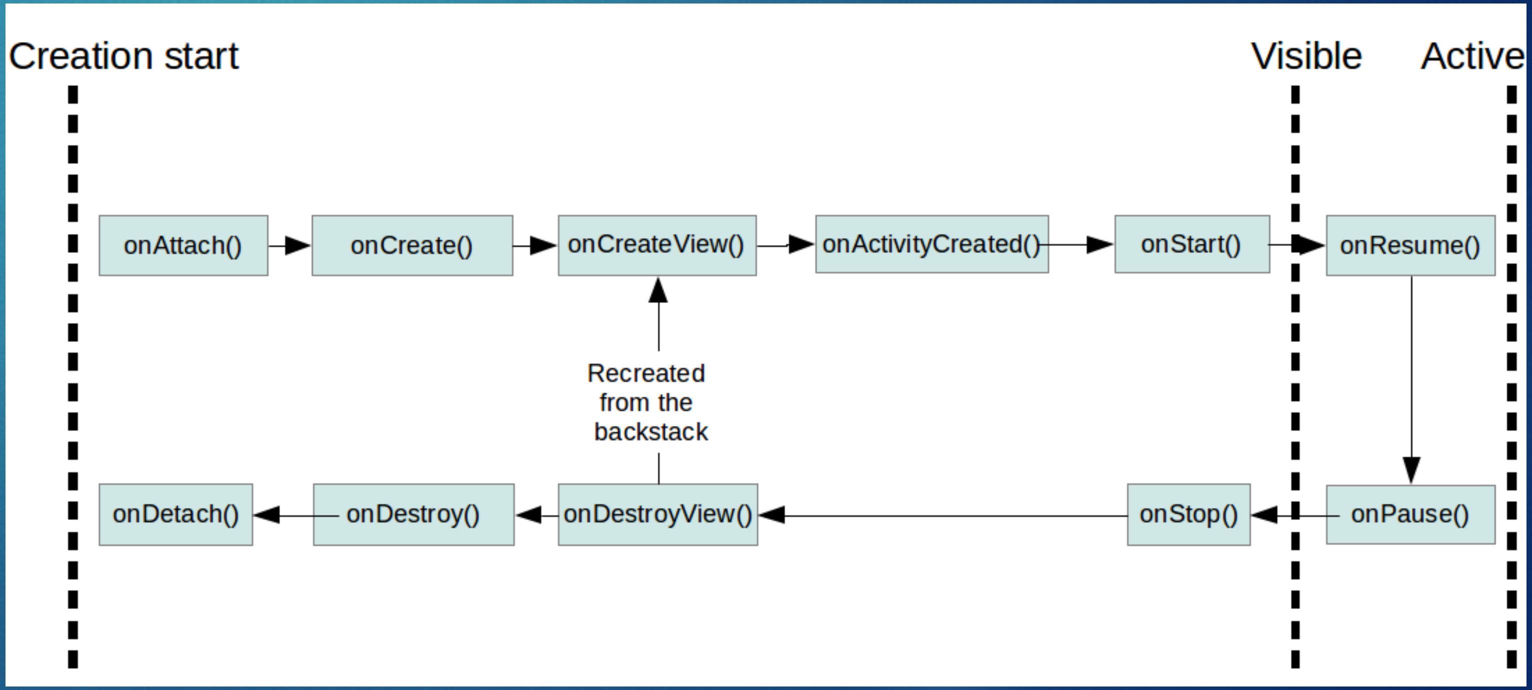
- Should not directly communicate with each other
- Should be done via the host activity
- Define an interface as an inner type
- Require that the activity, which uses it, must implement this interface

Android - Fragments

```
public class MyListFragment extends Fragment {

    private OnItemSelectedListener listener;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_rsslist_overview,
                container, false);
        Button button = (Button) view.findViewById(R.id.button1);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                updateDetail("fake");
            }
        });
        return view;
    }

    public interface OnItemSelectedListener {
        public void onRssItemSelected(String link);
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof OnItemSelectedListener) {
            listener = (OnItemSelectedListener) context;
        } else {
            throw new
                    ClassCastException(context.toString()
                    + " must implemenet "
                    + "MyListFragment.OnItemSelectedListener");
        }
    }

    @Override
    public void onDetach() {
        super.onDetach();
        listener = null;
    }

    // may also be triggered from the Activity
    public void updateDetail(String uri) {
        // create a string just for testing
        String newTime = String.valueOf(System.currentTimeMillis());

        // inform the Activity about the change based
        // interface defintion
        listener.onRssItemSelected(newTime);
    }
}
```

Android – fragments lifecycle

- fragment has its own life cycle
- always connected to the life cycle of the activity
- activity stops, its fragments are stopped
- activity is destroyed, its fragments are also destroyed.



Android – defining fragments

- Adding fragments statically to the layout file
- Different static layout files for different device configurations
- Can be done dynamically

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:baselineAligned="false"
    android:orientation="horizontal">

    <fragment
        android:id="@+id/listFragment"
        class="com.example.android.rssreader.MyListFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"></fragment>

    <fragment
        android:id="@+id/detailFragment"
        class="com.example.android.rssreader.DetailFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"></fragment>

</LinearLayout>
```

Android – fragments demo

- Its demo time (FragmentDemo01)!
- After hours demo - use different number of fragments depending on the portrait/landscape configuration

Android - Service

- Runs in the background without direct interaction with the user
- Not bound to the lifecycle of an activity
- Used for repetitive and potentially long running operations
  - Internet downloads
  - checking for new data
  - Streaming
- Service runs in the same process as the main thread of the app
- Use asynchronous processing in the service

Android – Service (platform)

- Predefined system services
- Application can use them, given the right permissions
  - getSystemService()

Android – Service (custom)

- Declare in manifest
  - Inside <application> tags!
- Extend the Service class or one of its subclasses.
- Start service
- Can also start via bindService(). Allows direct communication with the service
- Use android:exported="false“ for keeping service private

```xml
<service
    android:name="MyService"
    android:icon="@drawable/icon"
    android:label="@string/service_name"></service>
```

```java
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

/**
 * Created by akaver on 07.11.2015.
 */
public class MyService extends Service {

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //TODO do something useful
        return Service.START_NOT_STICKY;
    }

    @Override
    public IBinder onBind(Intent intent) {
        //TODO for communication return IBinder implementation
        return null;
    }
}
```

```java
// use this to start and trigger a service
Intent i= new Intent(this, MyService.class);
// potentially add data to the intent
i.putExtra("KEY1", "Value to be used by the service");
this.startService(i);
```

Android – service restart

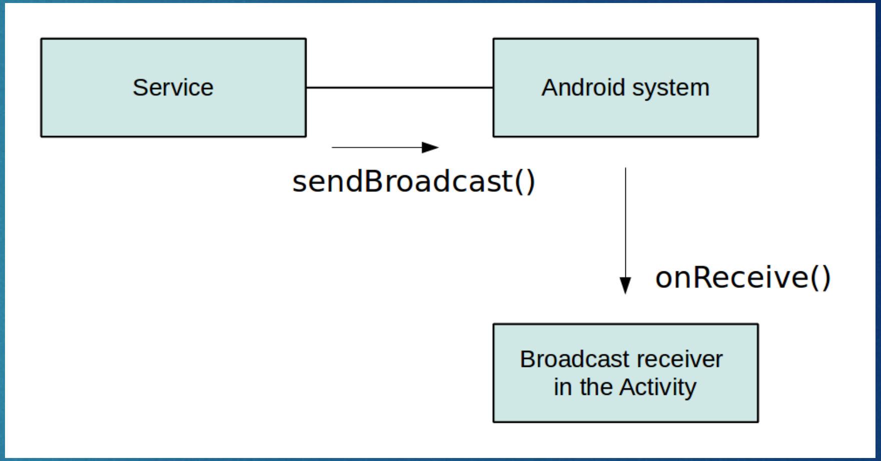
- Service.START_STICKY
  - Service is restarted if it gets terminated. Intent data passed to the onStartCommand method is null. Used for services which manages their own state and do not depend on the Intent data.
- Service.START_NOT_STICKY
  - Service is not restarted. Used for services which are periodically triggered anyway.
- Service.START_REDELIVER_INTENT
  - Similar to Service.START_STICKY but the original Intent is re-delivered to the onStartCommand method.

Android – service stop

- stopService()
  - One call to the stopService() method stops the service.
- stopSelf() – service terminates itself. Used when service finishes its work.

Android – communication with service

- Simple scenario – no direct communication. Service receives intent when starting.
- Using receiver
  - Service broadcasts events
  - Activity registers broadcast receiver and receives events from service
- Activity binds to local service
  - IBinder, onBind()



Android – service

- Service starting, stopping and starting new thread in service
  - ServiceDemo02
- Binder demo
  - ServiceDemo01