Android - Fragments, Background processing, Services, Broadcasts, etc..

Lukas Prokop

Context

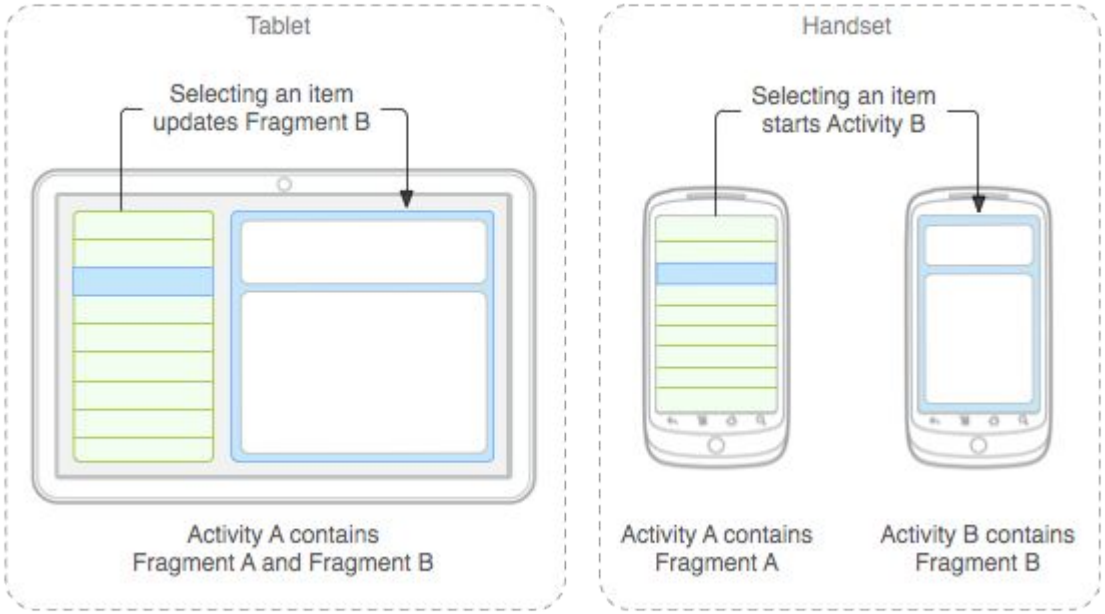
- Abstract class implemented by components
- Provides functionality required by all components
  - Access to resources
  - BroadcastReceiver registration/unregistration
  - Run Activity, Services, BroadcastReceivers
  - Binds Services

Context - types

- Application
  - Single instance
  - Instance of Context
- Activity/Service
  - Multiple instances
  - Instance of Context
  - Take care about of leaking Activity context
- BroadcastReceiver
  - Receive new instance of Context in `onReceive()`
  - `registerReceiver()` and `bindService()` doesn't work
- ContentProvider
  - Not instance of Context
  - `getContext()` returns Context of application which is running in

Fragment

- Simplify creating UI for phones and tablets
- Added dynamically or statically
- Since API 11, backported in support library



Fragment

- One activity can holds multiple fragments
- Fragment can be in multiple activities (different instances)
- Can be retained to preserve its state during configuration change
- Fragment can be added/removed from activity during lifetime
- Fragment and activity lifecycle is strongly tight together

Fragment - states

- Same state as host activity
- Resumed
  - Fragment is visible in the running activity
- Paused
  - Another activity in foreground, but hosting activity is still visible
- Stopped
  - Fragment is not visible. Hosting activity is stopped or fragment is removed from the activity, but added to backstack. Still alive, but can be killed with the hosting activity.

Fragments - adding to activity

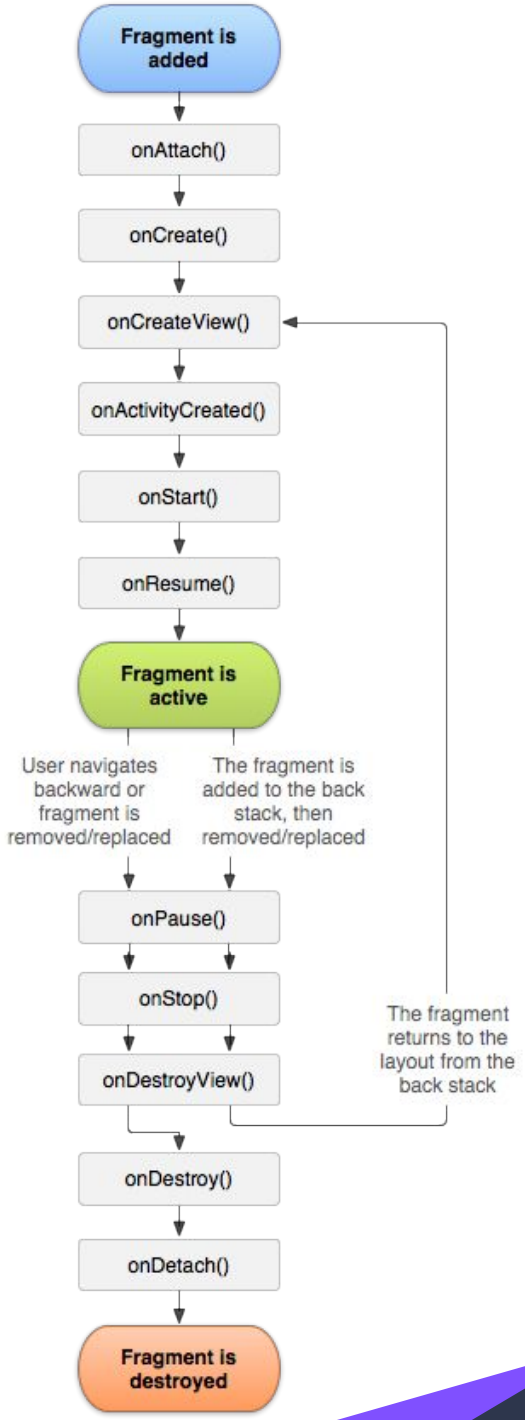
- Statically XML

```
<fragment
   android:id="@+id/fragment_demo"
   class="com.avast.android.helloworld.fragments.DemoFragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent" />
```

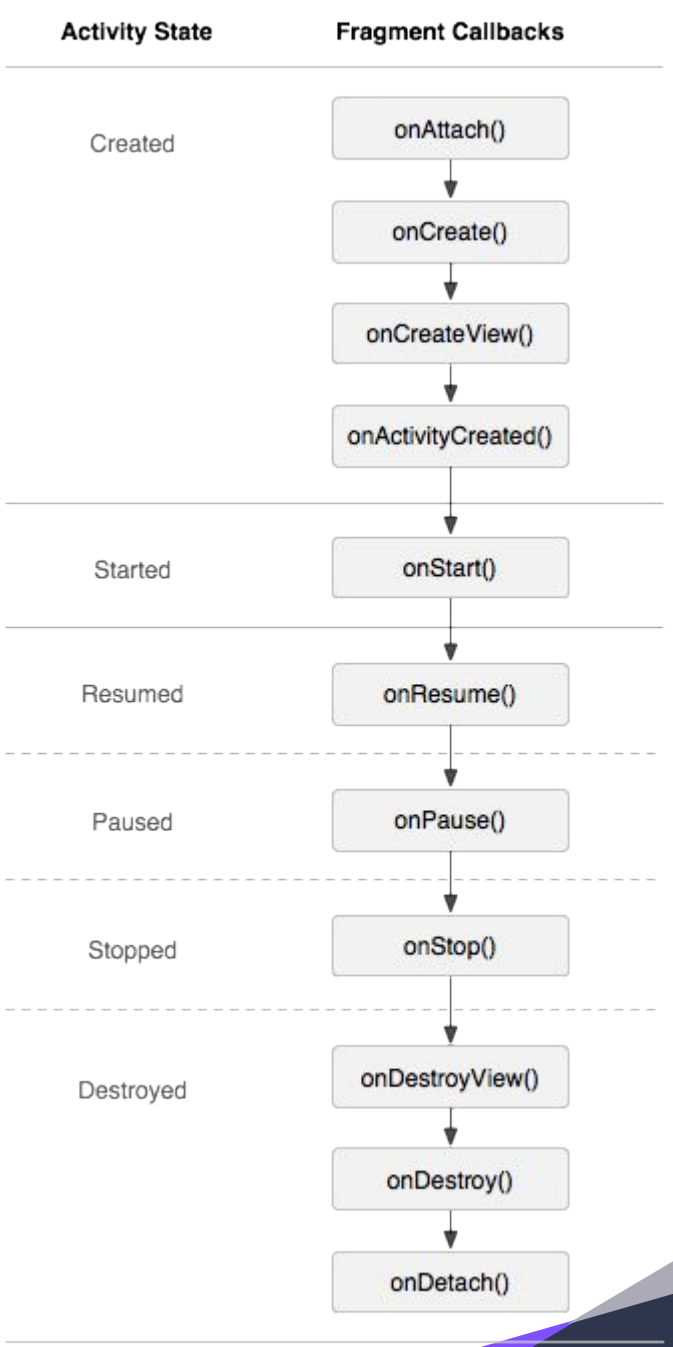
- Dynamically Java

```
FragmentTransaction ft = getFragmentManager().beginTransaction();
ft.add(R.id.fragment_placeholder, new DemoFragment());
ft.commit();
```

Fragment lifecycle



Fragment
& Activity lifecycle



Fragment - lifecycle

- onAttach(Activity)
  - Fragment is associated with the activity
  - Set activity as a listener
- onCreate(Bundle)
  - Initial creation of fragment
  - Process fragment extras
  - Not called when fragment is retained across Activity re-creation
- onCreateView(LayoutInflater, ViewGroup, Bundle)
  - Called when view hierarchy needs to be created

Fragment - lifecycle

- onActivityCreated(Bundle)
  - Activity `onCreate()` completed
  - Get references to views
- onViewStateRestored(Bundle)
  - All saved state of the view hierarchy was restored
- onStart()
  - Fragment visible to user (same as `Activity.onStart()`)
- onResume()
  - Fragment interact with user (based on hosting container)
  - Same as `Activity.onResume()`

Fragment - lifecycle

- onPause()
  - Not interact with user anymore
  - Paused activity or fragment manipulation
- onStop()
  - No longer visible
  - Stopped activity or fragment manipulation
- onDestroyView()
  - Disconnect fragment from view hierarchy created in `onCreateView()`
- onDestroy
  - Fragment going to be destroyed
  - Cleanup all resources
  - Not called for retained fragments
- onDetach
  - Detach fragment from activity
  - Remove activity listeners

Retained fragment

- Call `setRetainInstance(true)`
- Survive configuration change, but view needs to be recreated
- `onCreate()` is not called for retained instances
- For background work or data caching

Headless fragment

- Fragment without UI
- In most cases retained fragment

Fragment and Activity

- Fragment is not working without activity
- Activity can call fragment methods directly
- Fragment defines interface to be implemented by Activity to handle fragment requirements

Fragment - passing data

```
public class DemoFragment extends Fragment {
    public static DemoFragment newInstance(int arg1, String arg2) {
        DemoFragment demoFragment = new DemoFragment();
        Bundle data = new Bundle();
        data.putInt("IntKey", arg1);
        data.putString("StringKey", arg2);
        demoFragment.setArguments(data);
        return demoFragment;
    }
}
```

- Android calls non-params constructor when restoring fragments
- Constructor with parameters will not be called

Example

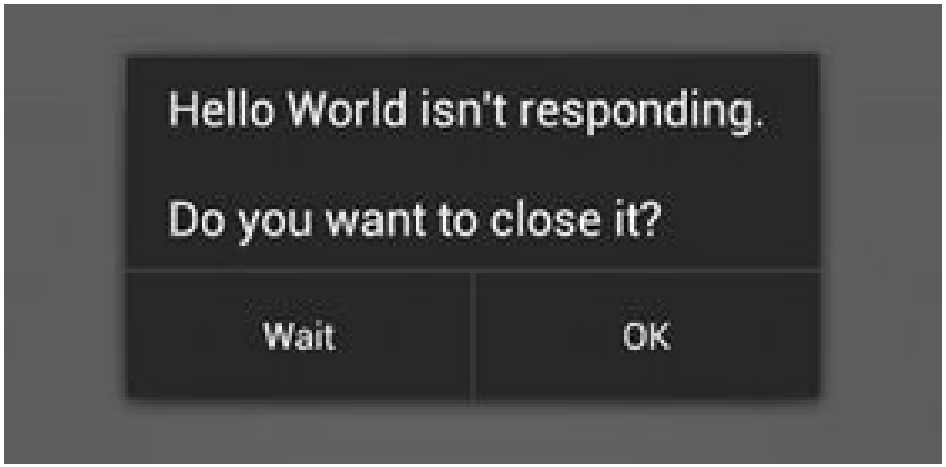
- Updated example from last lecture
- Add fragment with list of user to the main activity
- Open activity/fragment with user detail on phone/tablet
- Add up navigation in detail activity

Background processing

- Threads
- Handler
- AsyncTask
- Loader

Background processing

- Avoid long running operations on Main/UI thread
  - Files, database, network
- Most component runs on Main thread by default
- 5 second to ANR (10s BroadcastReceiver)



Keep your application responsive

Thread

- java.lang.Thread

```
new Thread() {
    public void run() {
        // Long running operation
            . . .
    }
}.start();
```

- Standard java thread
- Simple way how to offload work to the background
- UI can’t be updated from background

Handler

- Sends and processes messages
- Instance is bound to thread/message queue of the thread creating it
  - Scheduling messages and Runnables to be executed at some point in future
  - Enqueue an action to be performed on different thread

Handler

Receiving message on UI thread

- Overriding `handleMessage(Message)`

```
Handler mHandler = new Handler() {
   @Override
   public void handleMessage(Message msg) {
      vEditText

.setText(msg.getData().getString("data_key"));
   }
};
```

Send message from background

- Obtain message is more effective than create new instance
- Requires reference to handler

```
Message message = mHandler.obtainMessage();
message.arg1 = i;
mHandler.sendMessage(message);
```

Async Task

- Simplify running code on background
- `AsyncTask<Params, Progress, Result>`
  - Params - The type of the parameters sent to the task upon execution
  - Progress - type of progress unit published during background operation
  - Result - type of result of background operation

AsyncTask - methods

- `onPreExecute()`
  - UI thread, before executing, show progress bar
- `doInBackground(Params...)`
  - Background thread
  - publishProgress(Progress…)
  - Returns Result
- `onProgressUpdate(Progress...)`
  - UI thread
  - For updating progress, params are values passed in `publishProgress`
- `onPostExecute(Result)`
  - UI thread
  - Returned value from `doInBackground` is passed as parameter

AsyncTask - canceling

- `cancel(boolean)` - Cancel execution of task
- `isCancelled()` - call often in `doInBackground` to stop background processing as quick as possible
- `onCancelled(Result)` - called instead of `onPostExecute()` in case task was cancelled

Memory leaks

- Activity runs AsyncTask which takes long time, meanwhile configuration change happens
- Anonymous or non-static class still keeps reference to Activity => Activity can’t be garbage collected => activity leaks

Memory leaks - Solutions

- Disable configuration changes in manifest
  - Don’t do this, it just hides another bugs
- Retain activity instance
  - Using `onRetainNonConfigurationInstance()` and `getLastNonConfigurationInstance()` deprecated
- WeakReference to activity/fragment or views
- Task as static inner class
- TaskFragment
  - Fragment without UI and called `setRetainInstance(true)`
- AsyncTaskLoader

Loaders

- Async loading data in an activity or fragment

- Available to every Activity and Fragment
- Provide asynchronous loading of data
- Monitor source of data, deliver new results
- Automatically reconnect to last loader’s cursor after a config change

Loader - classes

- LoaderManager
  - Managing loader instances
  - Manage long running operation with conjunction Activity/Fragment lifecycle
- LoaderManager.LoaderCallbacks
  - Callback interface for a client to interact iwth the LoaderManager
- Loader
  - Abstract class that perform async load of data
- AsyncTaskLoader
  - Abstract loader that provides an AsyncTask to do the work
- CursorLoader
  - Subclass of AsyncTaskLoader to query ContentResolver and returns Cursor
  - Best way for async loading data from ContentProvder

Loader

```
// Prepare the loader.  Either re-connect with an existing one,
// or start a new one.
getLoaderManager().initLoader(0, null, this);
getLoaderManager().restartLoader(0, null, this);
```

- Id of loader
- Additional params
- Reference to callbacks

- Creates new loader with given id
  - LoaderCallbacks.onCreateLoader is called
- Reconnect to existing one
  - If data are already loaded, LoaderCallback.onLoadFinished() called immediately

LoaderManager.LoaderCallbacks

- onCreateLoader()
  - Create loader instance for given ID
- onLoadFinished()
  - Previously created loader finished its load
- onLoaderReset()
  - Previously created loader is being reset, making its data unavailable

Services

- Long running operations in background
- Doesn’t depend on UI
- Can expose API for other applications
- By default runs on UI thread

Started services

- Started by calling `startService()`
- Doesn’t depend on starting component
- Do one thing without return result to the caller
- Override `onStartCommand()` method
- Stop by `stopSelf()` or `stopService()` from outside

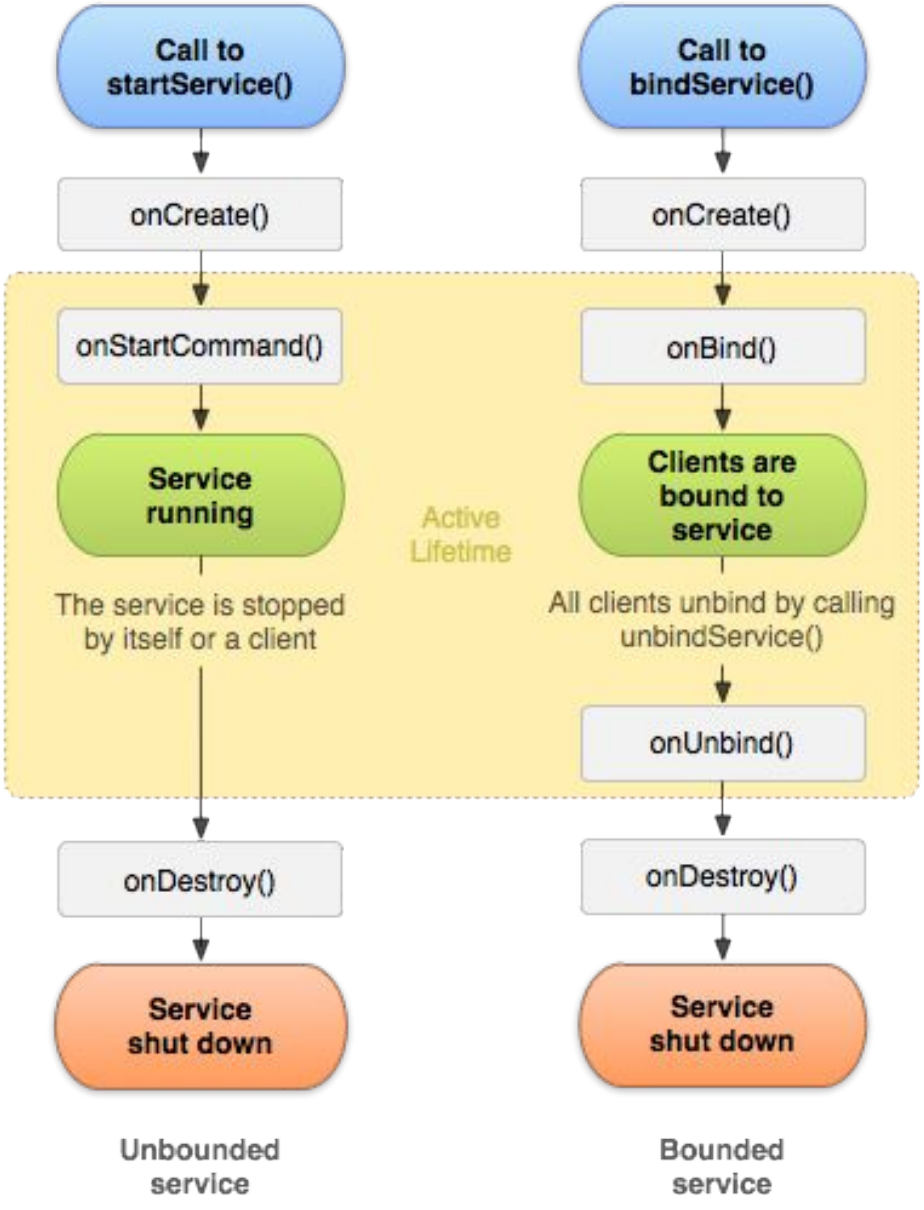
Bound Services

- Component bind to it by calling `bindService()`
- Client server interface for communication
- Service returns IBinder object for communication

Scheduled Service

- Started by API like JobScheduler (API >= 21)
- Recommended by Google

Service lifecycle



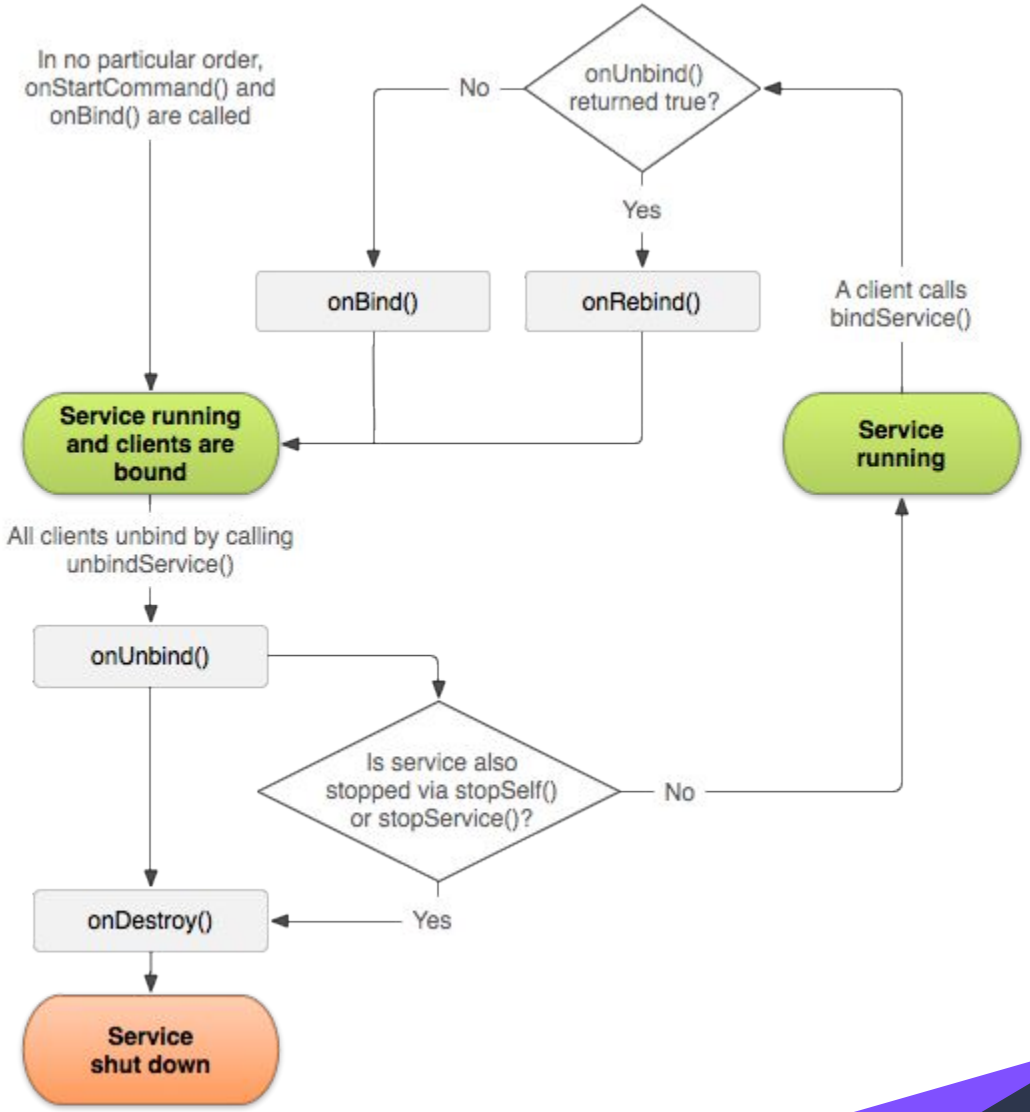
Service lifecycle

- onCreate()
  - Called when the service is being created (after first call of startService() or bindService())
- onStartCommand()
  - Called when startService() is called, delivers starting intent
  - Returned value specify behaviour when it’s killed by system
    - START_STICKY - don’t retain intent, later when system recreate service null intent is delivered (explicitly started/stopped services)
    - START_NOT_STICKY - if there is no start intent, take service out of the started state. Service is not recreated.
    - START_REDELIVER_INTENT - last delivered intent will be redelivered, pending intent delivered at the point of restart

Service - lifecycle

- onBind()
  - When another component binds to service
  - Returns Binder object for communication
- onUnbind()
  - When all clients disconnected from interface published by service
  - Returns true when onRebind should be called when new clients bind to service, otherwise onBind will be called
- onRebind()
  - Called when new clients are connected, after notification about disconnecting all client in its onUnbind
- onDestroy()
  - Called by system to notify a Service that it is no longer used and is being removed.
  - Cleanup receivers, threads..

Bound Service lifecycle



Foreground service

- Service process has higher priority
- User is actively aware of it
- System not likely to kill foreground services
- Requires permanent notification (cannot be dismissed), it is under Ongoing heading
- By calling `startForeground(int, Notification)`
- Remove from foreground `stopForeground(int)`

IntentService

- Subclass of Service
- Uses worker thread to handle requests
- Handle only one request at one time
- Creates work queue
- Stops when it run out of work
- Override `onHandleIntent(Intent)` for processing requests, runs on worker thread

Exercises

- Start DownloadService from UserDetailFragment
- Download User in a background thread in the service, use GitHubApi class for that
- Start DownloadIntentService from UserDetailFragment
- Download list of repositories in the IntentService, use GitHubApi class for that

Intent Filter

- Intent contains
  - Component name
    - Explicit intent
  - Action
    - Generic action to perform (send email, open web page, ….)
  - Data
    - Uri object that references MIME type of the data
  - Category
    - String with additionl information about the kind of component that should handle the intent
  - Extras
    - Key-value pairs with additional data
  - Flags
    - Metadata, for example how the activity is lauched

IntentFilter

- Tells the system, which implicit intent is component able to respond
- Based on
  - Intent action
  - Intent category
  - Intent data

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

Intent Filter

- If there is more component which are able respond to the intent, system let user to decide which component/application want to use

BroadcastReceiver

- Receive Intent sent by sendBroadcast()
- Registered in manifest or by a code
- Can be received by another application => Security
- For local broadcasts LocalBroadcastManager
- Global namespace for intents - possible collisions
  - Best practice is to use package name as prefix
- By default runs on main thread in default process

Normal broadcasts

- Sent with Context.sendBroadcast
- Asynchronous delivery (multiple receivers can receive intent at the same time)
- Cannot be aborted due to async behaviour
- More efficient

Ordered broadcasts

- Sent with Context.sendOrderedBroadcast
- Delivered only to one receiver at a time
- Receiver can abort the broadcast, it won’t be passed to another receiver
- Order of receivers is controlled by the priority of the matching intent filter

BroadcastReceiver - AndroidManifest

- If contains intent filter any app can call the receiver
- Receivers are not enabled until first run of app
- Who can send the broadcast can be limited by permissions

```
<receiver android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    . . .
</receiver>
```

BroadcastReceiver - Java

- Without specifying permission any app can send broadcast to you

Register - Activity.onResume()

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction("Action");
registerReceiver(mReceiver, intentFilter);
```

Unregister - Activity.onPause

```
unregisterReceiver(mReceiver);
```

BroadcastReceiver

- Context delivered as parameter
- onReceive must finish in 10 seconds
- For longer tasks run service

```
public class ExampleReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
       // Do some stuff
   }
}
```

Broadcast receiver - permissions

- It is possible to limit who can send broadcast by permissions
- It is possible to protect receiver when it is registered statically and dynamically
- Possible to set permission when sending broadcast

Exercise

- Create BroadcastReceiver handling ACTION_USER_DOWNLOADED action and register/unregister it in onStart()/onStop()
- Notify UserDetailFragment about successful User download from DownloadService via Broadcast

LocalBroadcast

```
LocalBroadcastManager lbManager =
     LocalBroadcastManager.getInstance(context);
lbManager.registerReceiver(mReceiver, intentFilter);
lbManager.unregisterReceiver(mReceiver);
lbManager.sendBroadcast(intent);
lbManager.sendBroadcastSync(intent);
```

Exercise

- Create BroadcastReceiver handling ACTION_REPOS_DOWNLOADED action and register/unregister it in onStart()/onStop() via local broadcast
- Notify UserDetailFragment about successful Repos download from DownloadIntentService via local broadcast


avast
Thanks for attention
prokop@avast.com