Android SDK

Lecture 2

Security of Mobile Devices

2023

Outline

Android Manifest

- `AndroidManifest.xml` file
- In the root of an app's directory
- Describes app components and resources
  - App name and package name (unique)
  - Activities, Services, Broadcast Receivers, Content Providers
  - Main(default) activity
  - Permissions
  - Libraries
  - Target/Minimum API level

Permissions

- Request access to resources and APIs for the app
- Provide security through sandboxing
- Declared in the Manifest
- `<uses-permission android:name="android.permission.INTERNET"/>`
- `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>`
- Dangerous permissions need to be requested at runtime when needed

Permissions

- Control who can access your components and resources
  - Start Activity, start/bind Service, send broadcasts, access data in Content Providers
  - ```
    <activity android:name=".ExampleActivity"
        android.permission="com.example.perm.START">
        ...
    </activity>
    ```
  - URI permissions
    ```

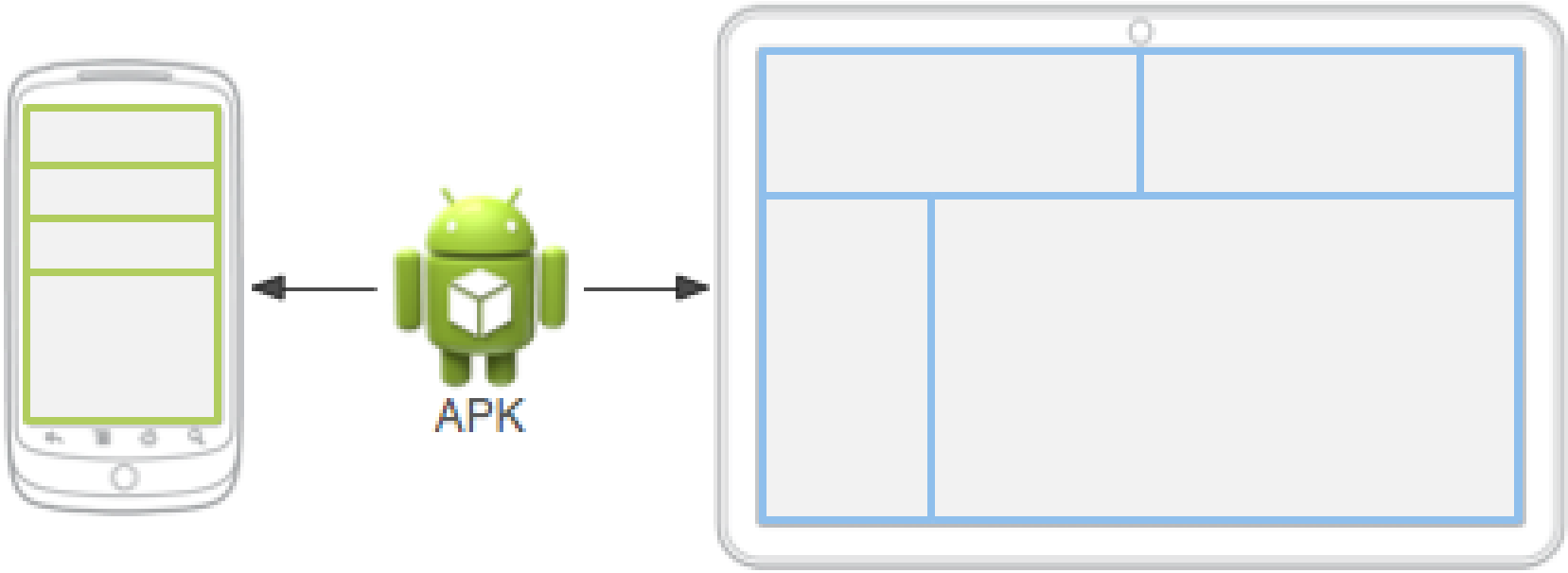
Resources

- `res/` directory
- Each resource type in a different subdirectory
  - Specific name
  - `drawable/`, `layout/`, `values/`, `menu/`, `xml/`, etc.

Resources

- Different configurations may require different resources
  - Bigger screen -> different layout
  - Different language -> different strings
  - Subdirectory for each alternative set of resources
  - `<resources_name>-<config_qualifier>`
  - `drawable-hdpi/` for High Density Screens
  - Resource chosen at runtime based on device configuration
- An ID is generated for each resource name in `gen/`

Resources - Different Configurations



Source: http://developer.android.com

Layouts

- Resources from `res/layouts/`
- Describe the UI of an activity or part of the UI
- UI elements
  - Button, TextView, etc.
- `res/layout/filename.xml`
  - `filename` is used as resource ID
  - `R.layout.filename`
- `R.id.start_button`
- Can be edited as xml or using graphical tools

Layouts - Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
}
```

Drawables

- Resources from `res/drawables/`
- Element that can be drawn on the screen
- Can be images (.png, .jpg, or .gif) or xmls
- xmls describe how an UI element reacts to input (pressed, focused)
- xmls point to images
- Visual feedback for interaction

Outline

Activity

- App component
- User interface window, provide user interaction
- Require a layout
- Can only draw and change UI from the main UI thread
  - Computationally intensive or wait based tasks on separate threads

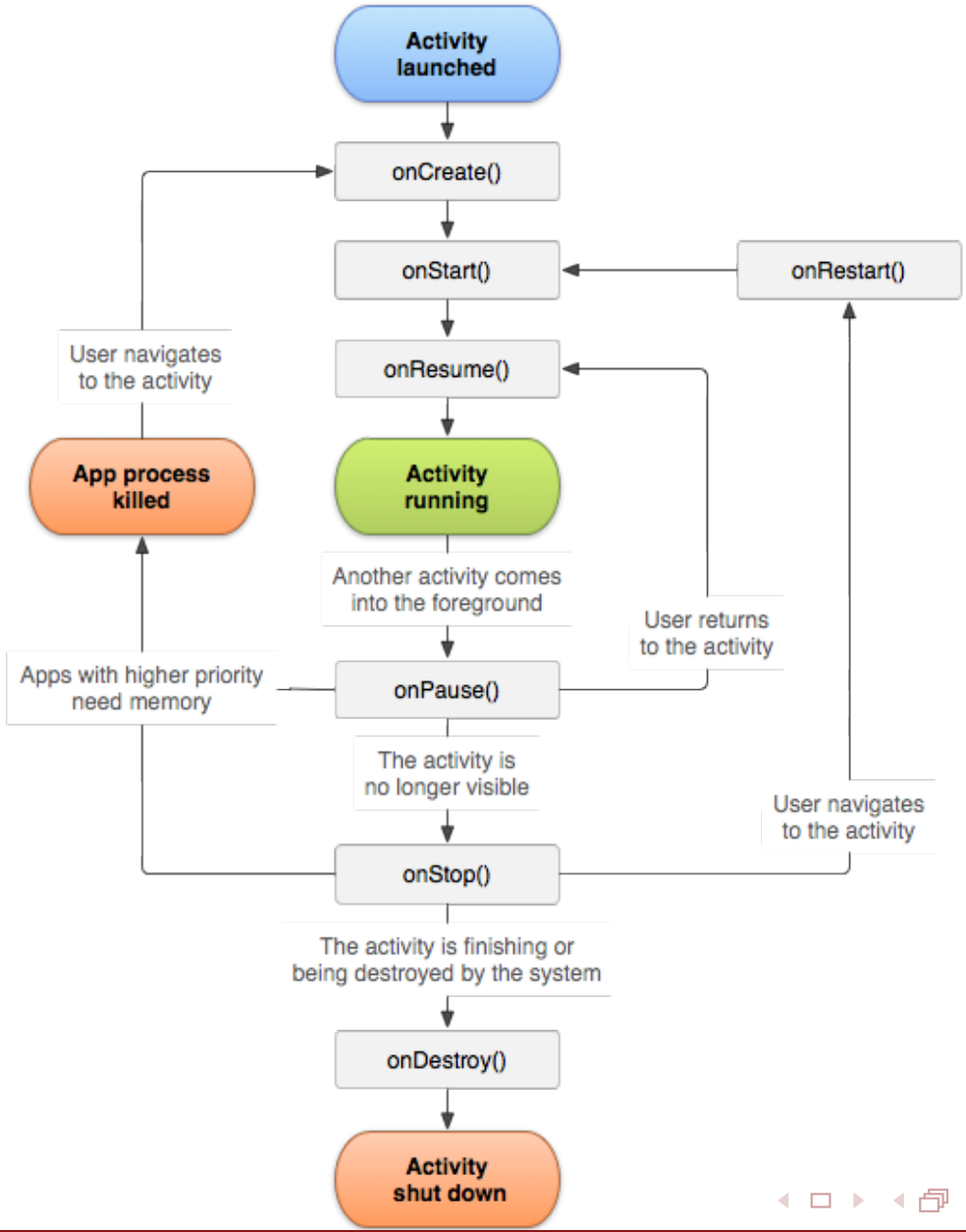
Activity

- An app may include multiple activities
  - Only one is the main activity
  - Activities can start each other -> the previous one is stopped
  - Activity stack ("*back stack*")
  - Back -> activity destroyed and previous one resumed

Activity Declaration in Manifest

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

Activity Lifecycle



Activity Lifecycle Callbacks

```
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").
    }
[...]
```

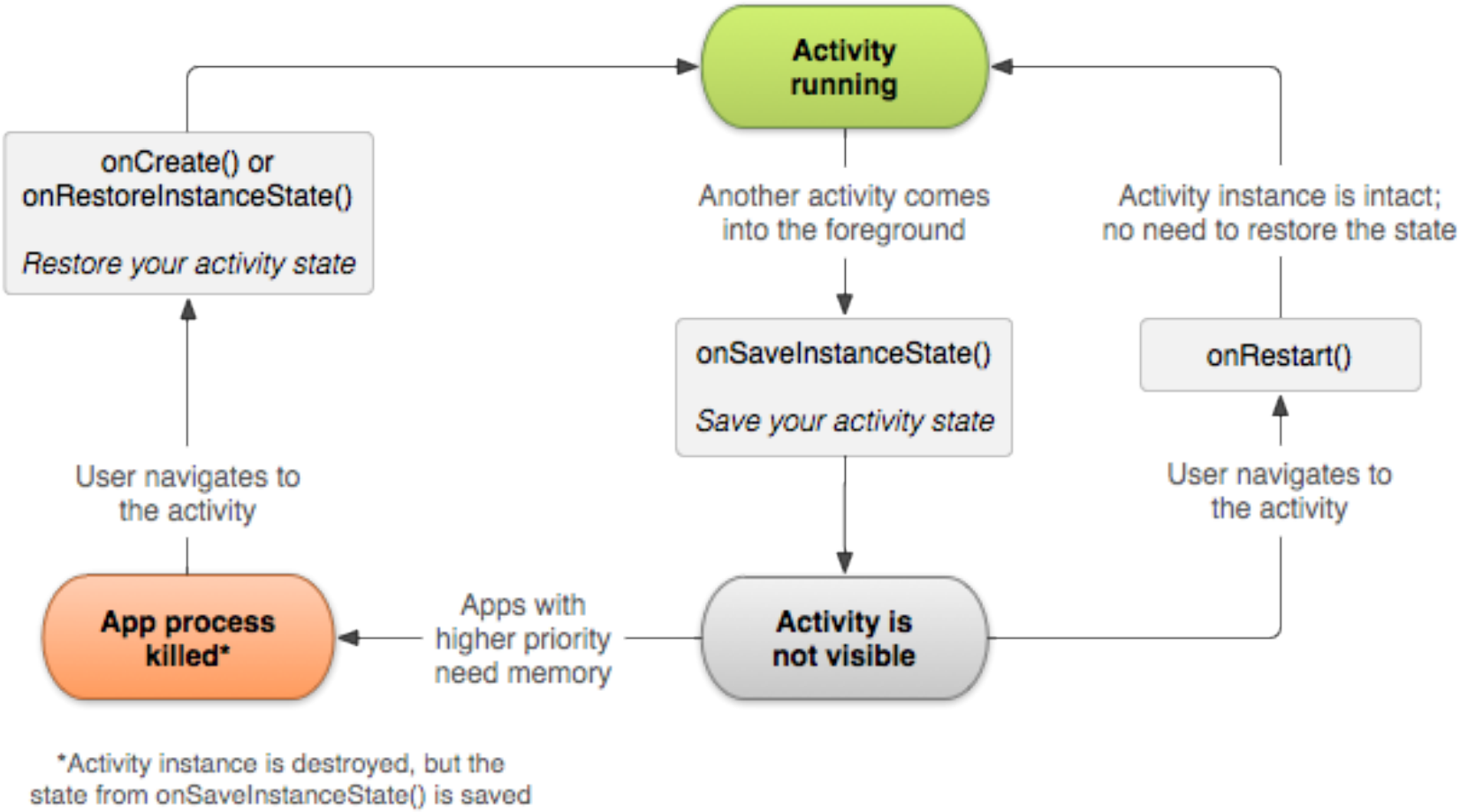
Activity Lifecycle Callbacks (2)

```
[...]
    @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 (is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Activity Lifecycle

- Activities can be killed after `onPause()`, `onStop()` in low memory situations
  - The activity state (objects) are lost
  - Can preserve state by saving objects
  - User interaction can be saved and restored
  - `onSaveInstanceState()` callback
    - Save information in a `Bundle`
  - `onCreate()`, `onRestoreInstanceState()`
    - Restore the activity state
  - Threads can be stopped graciously
    - In `onPause()` threads should be signaled to stop

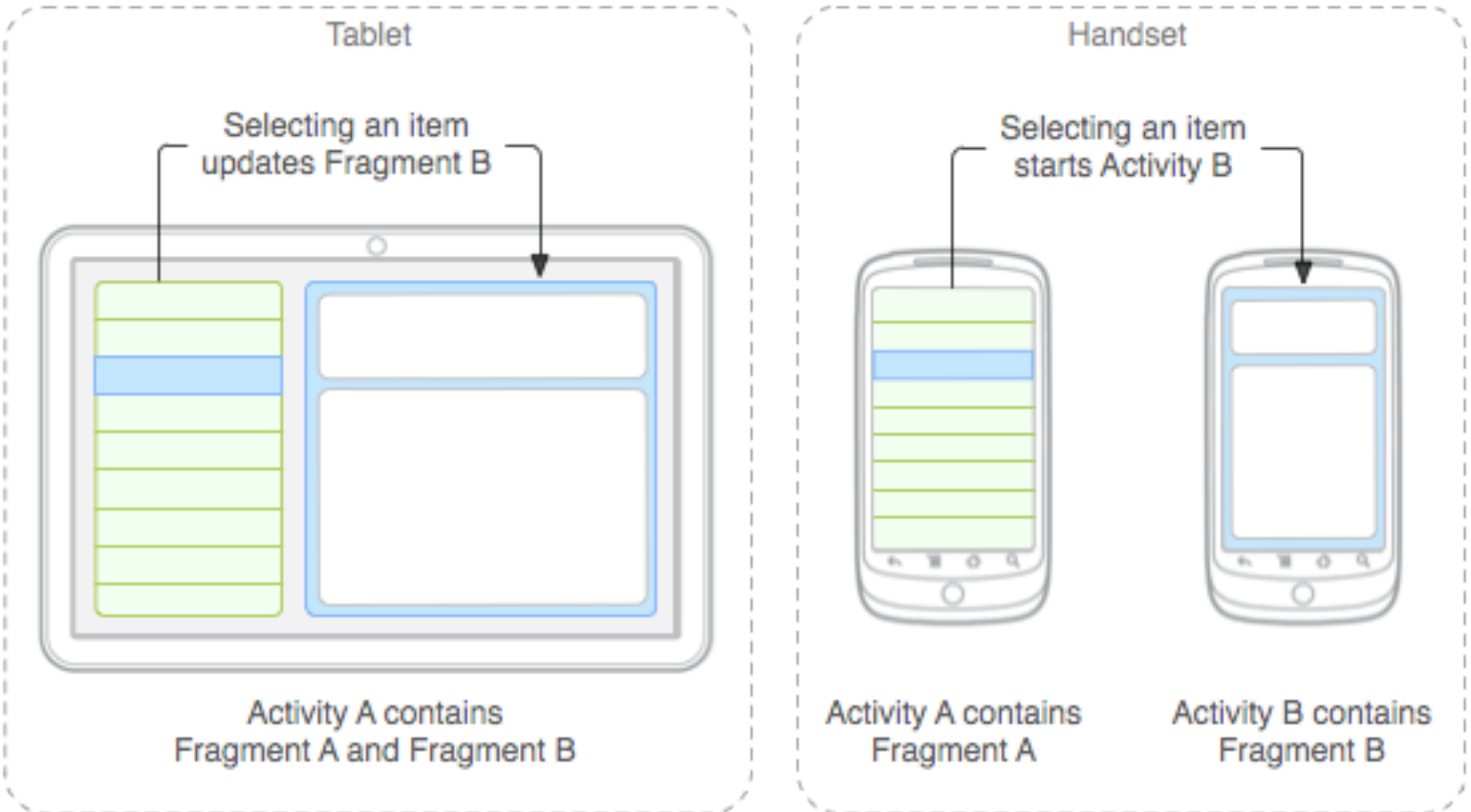
Save Activity State



Fragments

- Represent portions of UI in an Activity
- Can be combined to build a multi-pane UI
  - Same code, different layout for phone / tablet
- Can be reused in multiple Activities

Fragments



Source: http://developer.android.com

UI

- UI is a hierarchy of views
- `View`: rectangular space, provides user interaction
- Buttons, Lists, Images, TextViews, EditTexts
- Callbacks for actions
  - `onTouch()`, `onClick()`, `onLongClick()`
- A `ViewGroup` is a container for other Views or ViewGroups
- View / ViewGroup classes can be extended to create complex views
- Adapters allows for more complex data types to be displayed

Outline

Services

- Perform operations in the background
- Do not provide a UI
- Continue to run even if another app is in foreground
- Able to perform network transactions, file I/O operations, interact with content providers, etc.

Services

- Runs by default in the main thread of the hosting process
- Performing CPU intensive or I/O operations may block the UI
  - Run in a separate thread
  - Run in a separate process
- Start using Intents
- Private service

Service Declaration in Manifest

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService"
               android:exported="false"
                android:process=":exampleservice" />
      ...
  </application>
</manifest>
```

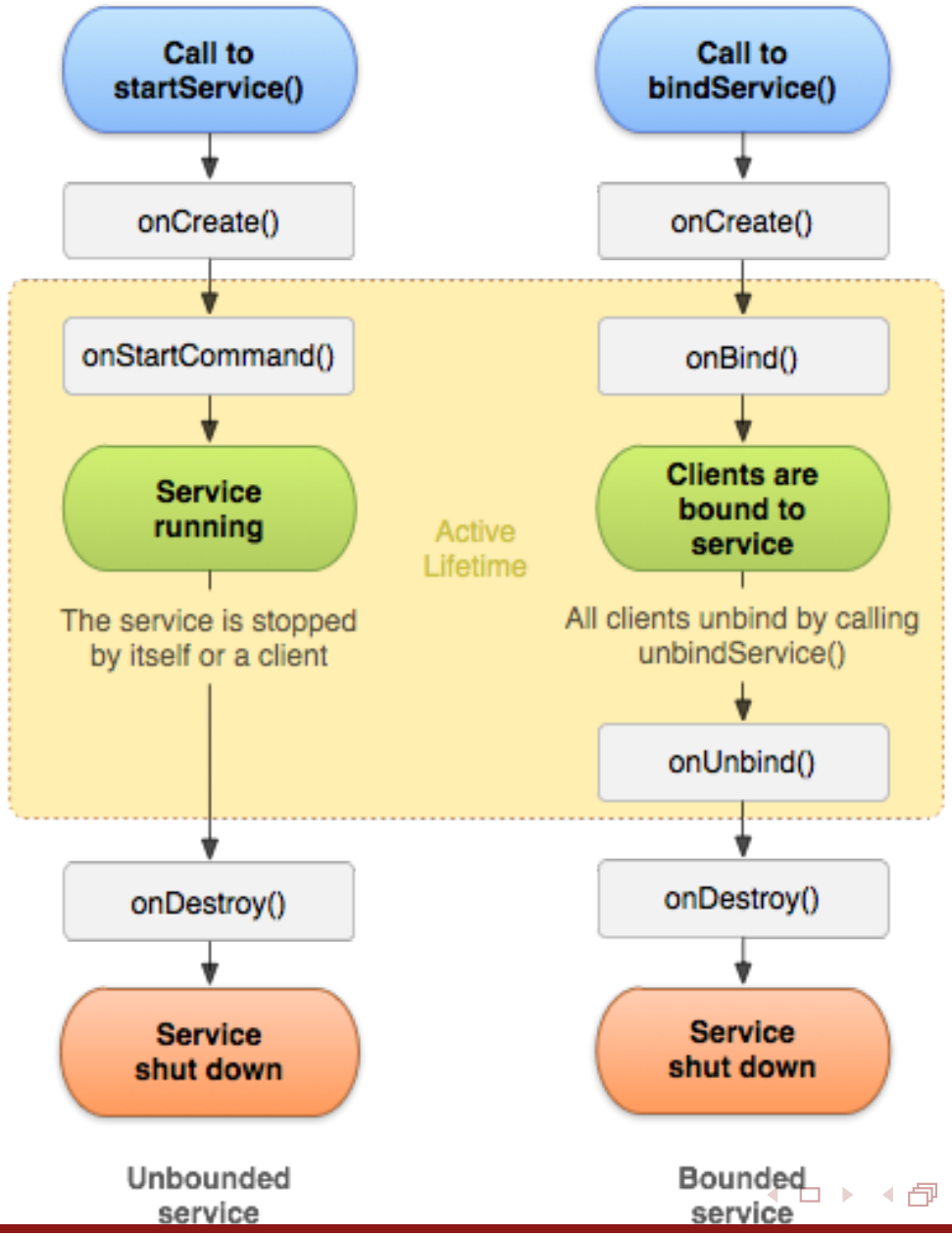
Types of services

- Started
  - An app component calls `startService()`
  - Performs a single operation, then stops itself and does not return a result to the caller
  - Runs even if the caller component is destroyed

Types of services

- Bound
  - An app component binds to it by calling `bindService()`
  - Provides a client-server interface - send requests, return results
  - AIDL - defines the client-server interface
  - Runs as long as the app component is bound to it
  - Check for null service
  - Multiple components can bind to a service at once
  - Service destroyed after all components unbind
- A service can be both started and bound

Service Lifecycle



Service Lifecycle Callbacks

```
public class ExampleService extends Service {
    int mStartMode;       // indicates how to behave
                          // if the service is killed
    IBinder mBinder;      // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind
                          // should be used
    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags,
                                                int startId) {
        // The service is starting,
        // due to a call to startService()
        return mStartMode;
    }
[...]
```

Service Lifecycle Callbacks (2)

```
[...]
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

Foreground Services

- Operation noticeable to the user
- Examples: music player, fitness app
- Must display a notification that can be dismissed by the user
- May configure non-dismissable notification
- App must have `android.permission.FOREGROUND_SERVICE`

Foreground Services

- Starting a foreground service:
  - Start the service with `startForegroundService()`
  - Call `startForeground()` in the callback `onStartCommand()`
- Remove a service from foreground:
  - Call `stopForeground()`
  - The service will keep running in the background

Outline

- http://developer.android.com/guide/topics/manifest/manifest-intro.html
- https://developer.android.com/guide/topics/permissions/overview
- http://developer.android.com/guide/topics/resources/overview.html
- https://developer.android.com/guide/components/activities/intro-activities
- https://developer.android.com/guide/components/activities/activity-lifecycle

- https://developer.android.com/guide/components/services
- https://developer.android.com/guide/components/foreground-services
- https://developer.android.com/guide/components/bound-services

Keywords

- Manifest file
- Permissions
- Resources
- Layouts
- Drawables
- Activity
- Service
- Started Service
- Bound Service
- Foreground Service