MT/Final Study Guide

Saturday, February 24, 2018 2:43 PM

Activities

- An activity in Android is responsible for managing user interaction with a screen of information.
- A layout defines a set of UI objects and their position on the screen, written in XML.
- Widgets like buttons, layouts, and text inputs exist as objects with type View in Android. They have attributes that can be modified to change their appearance. Layouts inherit from the ViewGroup class, as it can contain multiple other Views and ViewGroups, called children.
- To set the XML layout file, you call setContentView(int layoutResID) in the onCreate lifecycle method of the Activity it belongs to.
- To access anything in the resources folder in your android project, you have to use the R class along with one of its subclasses, like R.layout or R.id. To add ids to your XML files, use android:id="@+id/idname" And then you can use R.id.idname to access that element. The R classis automatically generated every time you build your app or add in widgets to the screen.
- To find widgets in your activity so you can change them around, create a object like so:
  - Button buttonName = (Button) findViewById(R.id.buttonID);
- You have to create the button using the library code in Android, and then use the findViewByID method to find the view objectfrom your layout file.
- To set an onClickListener, either have your activity implement View.OnClickListener and create an onClick method that takes in a View object, or you can run the setOnClickListener method and pass in an anonymous class like so:

```
mFalseButton = (Button) findViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.incorrect_toast,
                       Toast.LENGTH_SHORT).show();
        // Does nothing yet, but soon!
    }
});
```

- (A toast is just one of those little Android grey popups that tells the user info. Here, it takes in the activity it will popup in, the string ID from R of the text it will show, and the style of the toast, and calls a method to show it.)
- Android follows a design pattern that is pretty clear. The design rationale behind Activities, Fragments, and Service controller for your view, which is clearly the layouts you display to the user. You can then construct the model in M like in a SQLITE or Firebase DB, or just using prebaked Java POJOs with getters and setters.
- You have to specify in the Android Manifest whenever you create an activity. You also have to designate which act

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() called");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause() called");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() called");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() called");
    }
    ...
}
```

Next, we'll go over the lifecycle of activities

- Every instance of an Activity class has a lifecycle, that transitions between four states, resumed, paused, stopped a
- OnCreate makes the activity when it does not exist and turns it into a stopped state, and onDestroy destroys a stop stopped activity to paused, and onStop changes a paused activity to stopped. An activity is paused when it is visible changes the state to resumed from paused, and on Pause pauses a resumed activity. This is when the activity is in t interacting with it.
- Only one activity can be resumed on Android at one time.
- OnCreate is typically used to inflate widgets and put them on screen, get references to inflated widgets, set listene to external model data like in Firebase.
- Do NOT call the lifecycle methods yourself. The Android OS will handle that for you.
- You should always override and call the super methods for the lifecycle methods.
- When you exit out of an activity on pressing the back button, it calls onPause, on Stop, and onDestroy, and the me home button to exit just puts it into stopped state. Stopped activities can be destroyed, however, by the OS if it fe
- Your activity can hang in toe onPause state when you use multi-window mode, or you launch a new transparent ac
- Whenever you rotate your device, the entire activity is destroyed, and then created, and then resumed. This is why rotations in the activity, as it will always revert to the original value when it was created. To save data, you have to onSaveInstanceState(Bundle outState) method in the activity to save data from previous activities that have been s things:

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    if (savedInstanceState != null) {
        mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
    }
    ...
}
```

  - The above method will also save your data when the OS decides to kill your activity when it is stopped as well. Remember, use external storage if you need to keep data saved between device reboots or complete kills of the activity.

Intents

- To start a new activity in Android, you have to use the startActivity(Intent intent) method. It is a call sent to the Android OS. A part of the OS called the ActivityManager gets the call, and calls the onCreate method of the activity specified by the Intent parameter.
- An intent is an object that a component can use to communicate with the OS. Components are activities, services, broadcast receivers, and content providers.
- The intent constructor for activities is Intent(Context packageContext, Class<?> class) where the class argument is the Activity subclass you want to start, and the context argument tells the ActivityManager which application package the activity subclass can be found in.

```
mCheatButton = (Button)findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Start CheatActivity
        Intent intent = new Intent(QuizActivity.this, CheatActivity.class);
        startActivity(intent);
    }
});
```

- Above is an example of an explicit intent. The Activity Manager will check the QuizActivity Android Manifest for an instanceof CheatActivity. If it finds it, it starts the Activity. If it cannot, it crashes.
- Implicit intents are used to start an activity that exists in another application on the device.
- To send data between activities using intents, you put values as extras onto Intent objects. You can think of them as parameters that you can pass into the new activity. To do this, on the Intent object you pass into startActivity, you call Intent.putExtra(String name, Genericwhatever) where String is the identifier you will use to collect your data in the new activity, and whatever can be whatever data type you really want.
- A better way to do design, however, is to make a static newIntent method in the activity the intent will start. This way, another class does not have to know about the implementation features of the created Activity.
- To get data back from the intent you send, you can use the startActivityForResult(Intent intent, int requestCode) method. Therequest code is used when more than one intent is sent.
- To send a result back to a calling activity, you call setResult(int resultCode, Intent data) in the called activity. The codeis usually either Activity.Result_OK, or Activity.Result_Cancelled. You can put the needed returning data in the Intent extras used in setResult as well.
- To get the data back in the calling activity, you have to implement the onActivityResult(int, int, intent) method which willhandle getting the data. The first two ints are the request code and the result code. These are used to check which intent is returning data if you have multiple, and if the called activity is going to return data or not. Then, the intent can be used to extract data from the called activity.
- Intent filters can be created programmatically, but MUST be specified in the manifest file to resolve matching intents. An intent filter is a way an Android app can register itself as able to respond to a call for a specific action on the system. So if your app can read PDF's, then if another application can launch a generic intent to read a PDF file, your app will come up in the list of apps that will work on the PDF for the calling app. To do this, you can set an intent filter node in the XML tree for an activity in the manifest like so:

```
<activity android:name=".BrowserActivitiy"
          android:label="@string/app_name">
  <intent-filter>
```

```
<activity android:name=".BrowserActivitiy"
          android:label="@string/app_name">
  <intent-filter>
     <action android:name="android.intent.action.VIEW" />
     <category android:name="android.intent.category.DEFAULT" />
     <data android:scheme="http"/>
  </intent-filter>
</activity>
```

- Here, we set our activity as a browser activity that can handle web pages.

Persistence and Content Providers

- If you save files to the internal storage of the application, it is destroyed when the app is uninstalled. If you save it to external memory, like SD cards, then it is saved after application destruction and can be shared among other apps.
- If you do not want to use files but keep persistent data between sessions in your app, you can use SQLite for pure data like variables. Pictures and documents should be used for persisting documents though.
- The Android SQLite library allows you to compile SQL statements by passing Java objects as arguments into things like INSERT and SEARCH WHERE statements. These get returned as statements, which will be searchable with cursors in Java. You can also execute pure SQL statements on it without building the queries out beforehand.

Content Providers

- The standard content providers are contacts, dictionaries, and media. Basically, a content provider allows you to examine saved content on the phone that is not initially in your application, like contacts.
- You'll need to provide specific permissions in the manifest if you use specific content providers.
- Just a basic mechanism for sharing data between apps.

Broadcasts and Broadcast Receivers

- Android apps can send or get broadcast messages from the Android system and other Android apps.
- These are messages that are sent out to each app on the system. If they have the correct intent filter on their manifest, their app can implement functionality to respond to the broadcast message. Alternatively, the app can send out a broadcast with a specific piece of data to interesting candidate apps on the phone so the user can choose which app will process their data collected from the original app.
- If you have an intent filter in your manifest, you have to register a BroadcastReceiver subclass that will respond to it. That subclass calls onReceive which handles the context data and intent data that gets passed in like regular intent data.
- To send a broadcast, use the sendBroadcast(intent) method.
- You can really slow down the phone if you run a lot of broadcast receivers in your app, Android has ways around this now

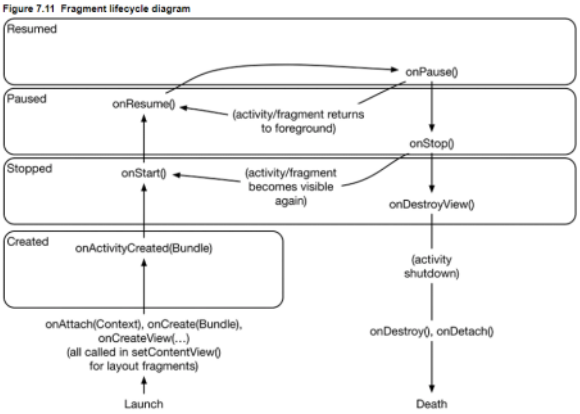
Services (Overview, External, Internal)

- Services are like activities for background, long term processing. Things like email, music, and synchronization are done with services. Local services are only accessible by a single application, while remote/bound services are accessible from all apps on a phone.
- It's better to use a service than a thread, as the OS can work with them better.
- A service class extends Service, and has a couple of methods it has to implement:
  - onCreate - called when service is created
  - onStartCommand - Called once each time a component starts the service with startService(Intent).
  - onDestroy
  - onBind
- Services are launched just like activities, with an intent.
- AsyncTasks are the easiest way to get a working background thread, but it is ill suited for repetitive and long-running work.
- There are different types of services. The book discusses intent services. Intents passed to a service are called commands. Each command is an instruction to the service to do something. It uses a queue to handle multiple intent commands.
- You have to add services to your manifest like activities.
- Services do NOT automatically run any code on a background thread out of the box. IntentService will provide the boilerplate code to actually do this.
- A non-sticky service stops when the service itself says it is done. The service can stop itself with stopSelf().
- A sticky service stays started until something outside the service tells it to stop by calling Context.stopService(intent).
- External services on a phone are things like viewing websites, REST calls, and map based functionality. It's really anything outside of the phone on the internet that you need to grab and pull into the phone.
- You can use Embedded Web Views to display web pages on the internet.
- You can use Google Maps with the appropriate permissions provided in the manifest file for your project. You also need an APIkey for the service. The map will ALWAYS be loaded in a fragment, you need to use a fragment to display the map.
- Maps fragments also need a License Agreement Fragment to load the actual map.
- Make sure to always check for connectivity before running any type of external service that requires downloading. You should also use threading, but many SDK components have threading built into their libraries.
- Examples of internal services include email, SMS, and telephony, audio, video, and photo capture, and sensors like the accelerometer, light, magnetic, and ambient temp sensors.

Fragments

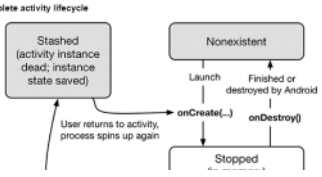
- A fragment is a controller object that an activity can deputize to perform task, the most common being managing a UI.
- This is the fragment lifecycle:

Figure 7.11 Fragment lifecycle diagram



- To achieve real UI flexibility, you have to add fragments to your layout in code. Fragment lifecycle methods are called in code, not managed by the Android OS.
- In a fragment, you use onCreate to call code for before the fragment view is inflated. You use onCreateView to handle code involving the UI. In this method, you have to use View.findViewById instead of just findViewByID, as you are in a fragment, not an activity.
- The fragment manager is what you use to manage fragments in an activity. It has two components, a list of fragments and a back stack of fragment transactions. To get access to the fragment manager, call FragmentManager fs = getSupportFragmentManager() in onCreate in your activity.
- Then, instantiate your UI fragment like CrimeFragment(), and call fs.beginTransaction().add(xml fragment container in activity xml, CrimeFragment()).commit();
- Fragment transactions are used to add, remove, attach, detach, or replace fragments in the fragment list. These are how you recompose screens at runtime.

Figure 3.14 The complete activity lifecycle

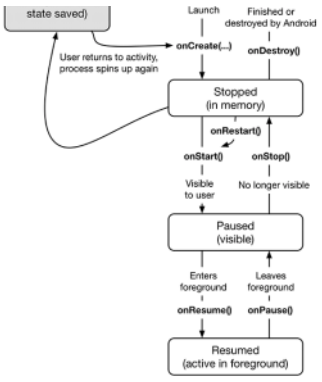


Listing 7.16 Adding a CrimeFragment (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

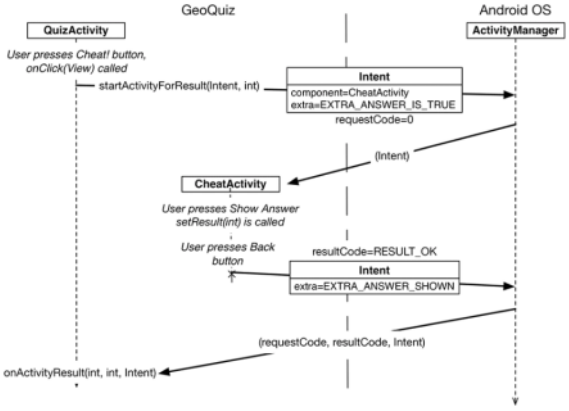
Listing 7.10 Adding a CrimeFragment (CrimeActivity.java)

```
public class CrimeActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```



Non Functional Requirements

- These are typically quality and design requirements, more concerned with building the application in a correct manner
- Some typical NFRs include performance, availability (5 nines uptime), scalability (many devices at once), security, cost, and testability.
- Key NFR metrics for mobile devices are performance, responsiveness, energy conservation and security.
- To meet NFRs, you can quantify what you want for the app. You need to make informed architectural decisions using the architecture of the implementation framework, like Android SDK or iOS SDKs. You can optimize specific things after getting real measurements.
- Tactical optimizations include improving the quality of your code:
  - Using variables to cache data from arrays
  - Avoiding internal getter and setter use
  - Reduced JVM heap access, avoidance of creating variables when the minimum will do along with one time references
  - Using static final for constants (saved on memory)
  - Rely heavily on framework libraries which are optimized to hell and back
- You can use the Profiler in Android Studio to detect how much CPU/RAM/Network your app is using on a live Android device.
- It has a method trace view that allows you to see what methods use up resources. Fixing methods that use a lot of CPU also reduce the amount of battery you use. Minimizing network usage can save app users from data charges.
- To save on network usage, use library specific configurations to change how much you use. LocationRequest in the Google MapsAPI has different priorities you can set when making geolocation calls.
- To optimize for responsiveness, make sure you're not overloading your UI thread on Android. To do this, employ AsyncTasks, Services and Java Threading for long running tasks like I/O for files and network calls, as well as hard computations.
- To use Java Threading, only use the results from the thread to manipulate the UI thread using instructions on the UI thread. The Java Threading SKD is not thread safe with the Android UI thread, so avoid manipulating the UI thread in a separate thread.
- Data is inherently less secure on a mobile app. Its portable with no user login, and weaker passwords used due to the difficulty of data entry. Developers need to share responsibility for security.
- There is a systematic way to do app security correctly:
  - Define a threat model: What are the assets and their value in the app? What are the attacks, and where can they originate?
  - Identify the tactics you will use to protect the app: Detection, Resistance, Mitigation, Recovery
  - Implement tactics using sec tech: Authentication, Acess control, Audit trails like logs, data integrity like encryption, and non-repudiation.
- There is no security through obscurity on Android. APKs can be decompiled, Linux is open source, and Google Play has a limited vetting process. Also, priveliges are determined by the installer of the app, not the app itself.
- Big Android risks:
  - Leaving private files in publicly shared directories
  - SQL injection on the local SQLite DB
  - Leaving API keys and other secrets in decompilable code



Testing

- You need to have Junit installed to run unit tests in Android, as is the Java standard.
- Your Test classes need to extend ActivityInstrumentationTestCase2 <testedActivity>. They need to instantiate both the activities and fragments they will use, as well as the member variables of the activities. You will also need to provide the touch coordinates of what willbe pressed in the test.
- You need to have a setUp method that initiates the needed variables for the test, write your tests, and can use @UIThreadTestas an annotation for a test to force a test to run in the UI thread. Use tearDown() to clean up test case variables from the JVM.
- The constructor of the test needs to make a call like super(TestedActivity.class)
- In your setup method, call the super setup method, then instantiate your activity and fragment objects, and add the fragmentobjects to the activity with a FragmentManager.
- You can use getInstrumentation().waitForIdleSync() to wait for the activity to become idle to avoid starting the test too quick and getting a null fragment.
- In tearDown, call .finish() on the activity you instantiated in the setUp() method. Then call super.tearDown().
- The first test you should run should assert that your activity, fragment, and UI objects are not null.
- Here is how a touch event is done:

```
getActivity().runOnUiThread(new Runnable() { // Run on UI thread
    public void run() {
        System.out.println("Thread ID in TestUI.run:" + Thread.currentThread().getId());
        board.requestFocus();
      // Simulates touch event
      // Hint: Instrumented the onTouchEvent(MotionEvent event) to get good pixel
        values for touch. Why not call onTouchEvent of Board directly?
        MotionEvent newMotionEvent =
                MotionEvent.obtain((long)1, (long)1, MotionEvent.ACTION_DOWN,
                                   (float) 53.0,  (float) 53.0, 0);
        board.dispatchTouchEvent(newMotionEvent); // Dispatches touch event
        mGameSessionFragment.scheduleAndroidsTurn();
        assertEquals(mGameSessionFragment.getPlayCount(), 0); // Assert 0 moves
    }
});
// Assertion does not work outside UI thread
```

- You can see here that we can specify running a test on the UI thread using a method and passing in a thread. You can avoid all this boilerplate by marking the test with the @UIThreadTest annotation!
- The Main UI thread gets subordinated to the unit test thread when these are called, and is terminated when the tests are running. An issue with this is that if you have tasks queued on the main UI thread, they may not launch. To fix this, check to see if you are testing in your fragment/activity, and keep things singularly threaded if you are testing.

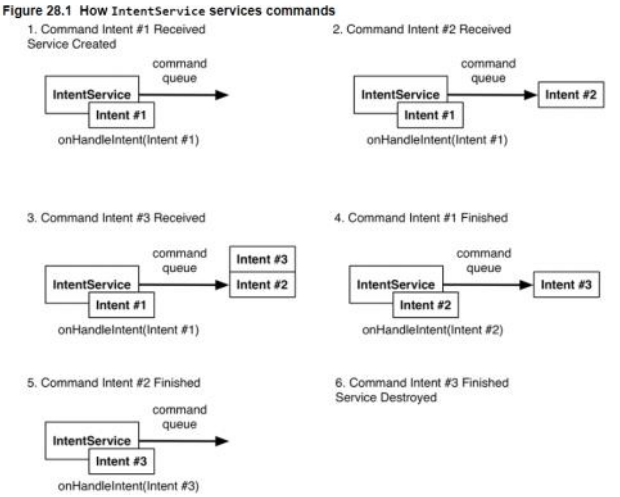
Threading

- Running any network/IO intensive task on the UI thread is a bad idea. The easiest way to run a background thread on Android is to use the AsyncTask utility thread which creates a background thread for you and runs your code in the doInBackground() method.
- To create an AsyncTask:
  - Private class FetchItemsTask extends AsyncTask<Void,Void,Void>{
    @Override
    Protected Void doInBackground(Void… params){}
    }
  - Call new FetchItemsTask.execute to run the doInBackground method on a new thread that won't freeze the UI thread!
- You cannot update the UI from an AsyncTask, as this has the opportunity to corrupt objects in memory due to the read/write problem. You can use the @Overrided onPostExecute() in your AsyncTask class, which runs on the main thread, to update the UI thread.
- To do this, just set the third generic parameter in the class definition to the type of data you want to pass to the UI thread, and do the same for the

return type of doInBackground and the parameter in onPostExecute(). Also: the data your doInBackground method returns is passed to onPostExecute() as a parameter.
- You can call AsyncTask.cancel() to stop a background thread's execution.
- A full overview of the generic parameters of an AsyncTask:
    - The first parameter is the type of parameter passed into the arguments of execute(). These will be given as parameters in doInBackground
    - The second parameter is the type of progress update parameter passed into the onProgressUpdate method of AsyncTasks.
    - The third is the return type of doInBackground and what onPostExecute can work with as a parameter.

Background Services

- A service is a subclass of Context, just like Activity and Fragment, which means that intents can be passed to it. In this example, we work with IntentService, which is the most common kind of service. A good pattern to use is to create a new class that extends IntentService. To send intents to it, make a method called public static Intent newIntent(Context context){ return new Intent(context, YourService.class)}. This way, you can pass an Activity/Fragment to the method, which will then start a new intent to YourService.
- A service's intents are called commands. Here's how they work:



- Commands are handled in the onHandleIntent method of IntentService, which is where the service does its work on a background thread separate from the UI thread. The service is destroyed when its queue is emptied.
- Since this is a context, you have to define it in your manifest file.
- We like these as they can handle long running operations in the background without those pesky ANRs.

OO Design

- Start with a narrative. Write a couple of paragraphs about what your product will do.
- Identify the noun phrases and nouns, and assign them to classes. Adjectives could describe subtypes of classes. Similar categories of nouns could have a superclass. Eliminate redundant classes.
- Look at verbs and verb phrases (Actions) to assign responsibilities to classes. Try to assign these to the appropriate classes, by equally distributing methods without breaking class definitions.
- Map the domain model of classes to the patterns of the core framework you are building for.
- Identify collaborations between classes through typical usage scenarios of the product.
- Make sure each class has a clear name, a long-lived state, collaborators and a cohesive description of responsibility. These should all be captured on CRC cards to keep responsibility hoarding to a minimum.