EE 579: Wireless and Mobile Networks Design & Laboratory

Android Classes

Amitabha Ghosh
Department of Electrical Engineering
USC, Spring 2014

Lecture notes and course design based upon prior semesters taught by Bhaskar Krishnamachari and Murali Annavaram.

Outline

- Administrative Stuff
- Intent Class
- Android Permissions
- Fragment Class
- User Interface Classes
- Android Networking

Intent Class

Activity Creating Intent Object

- One activity can programmatically start another activity by
  - Creating an Intent object
  - Passing that Intent object to a method
    - e.g., startActivity(), startActivityForResolve())

- Intent class – how they are created, what fields they have, and what information the fields contain

- Two ways Android decides which activity to be started when a method such as startActivity() is called
  - Explicit activation
  - Implicit activation via intent resolution

The Intent Class

- A data structure that serves two purposes
  - To specify an operation to be performed
  - To notify events to other components
- Intents provide a flexible language (easy way) for specifying operations to be performed
  - Pick a contact, take a photo, dial a phone number, display a map
- In practice, intents are constructed by one activity that wants some work to be done
- Android uses the intent to start another activity that actually performs the desired work

Intent Class Fields

- Action
- Data
- Category
- MIME Type
- Target component
- Extra
- Flag

Intent Field: Action

- A string that represents or names the desired operation

- Built-in examples
  - ACTION_DIAL – dial a number
  - ACTION_EDIT – display data to edit
  - ACTION_SYNC – synchronize device data with server
  - ACTION_MAIN – start as initial activity of an application

- Setting the action field in several ways
  - Pass the action string as a parameter to the intent constructor
    - Intent newInt = new Intent(Intent.ACTION_DIAL);
  - Create an empty intent, then call setAction()
    - Intent newInt = new Intent()
    - newInt.setAction(Intent.ACTION_DIAL)

Intent Field: Data

- Represent data associated with the intent
  - Formatted as a Uniform Resource Identifier (URI)

- Example 1: Data to view on a map
  - Uri.parse(“geo:0,0?q=1600+Pennsylvania+Ave+Washington+DC”)

- Example 2: Number to dial in the phone dialer
  - Uri.parse(“tel:+155555555”)

- The parse() method takes the string and returns a URI object

Intent Field: Data

- Setting the data field in two ways

- Pass it to the constructor when creating the intent
  - Intent newInt = newIntent(INTENT.ACTION_DIAL, Uri.parse(“tel:+1555555”))

- Using the setData() method
  - Intent newInt = new Intent(INTENT.ACTION_DIAL)
  - newInt.setData(Uri.parse(“tel:+1555555”))

Intent Field: Category

- Category provides additional information about the components that can handle the intent

- Example1
  - CATEGORY_BROWSABLE – activity can be invoked by a browser to display data by a URI link

- Example 2
  - CATEGORY_LAUNCHER – the target activity can be the initial activity of a task, and is listed in top-level app launcher

Intent Field: Type

- Specifies the MIME type of the intent data
  - MIME: Multipurpose Internet Mail Extensions

- Examples
  - image/png, image/jpg
  - text/html, text/plain

- In the type is not specified, Android will try to infer one

- Setting the type or both the data and type fields
  - Intent.setType(String type)
  - Intent.setDataAndType(Uri data, String type)

Intent Field: Component

- Identifies the intent’s target component

- Can set this field when there is exactly one component that should receive this intent

- Setting the component field
  - By passing a context object and a class object to the intent constructor, representing the target component that should perform the desired operation
    - Intent newInt = Intent(Context packageContent, Class<?> cls)
  - Create an empty intent and use one of the methods:
    - setComponent()
    - setClass()
    - setClassName()

Intent Field: Extra

- Extra contains additional info associated with the intent
  - Treated as a map (key-value pairs)
  - Target activity should know the name and type it intends to use

- Example: Intent.EXTRA_EMAIL: email recipients
  - Intent newInt = new Intent(Intent.ACTION_SEND)
  - newInt.putExtra(android.content.Intent.EXTRA_EMAIL, new String[] {“aporter@cs.umd.edu”, “ceo@microsoft.com”})

- Setting the extra field
  - Several forms depending on the data type
  - Storing a string – putExtra(String name, String value)
  - Storing an array of floats – putExtra(String name, float[] value)

Intent Field: Flags

- Flags specify how an intent should be handled

- Example 1
  - FLAG_ACTIVITY_NO_HISTORY
    - Do not put this activity in the history stack

- Example 2
  - FLAG_DEBUG_LOG_RESOLUTION
    - Print extra logging information when this intent is process

Using Intent to Start Activities

- Programmatically start activities by using methods such as
  - startActivity(Intent intent, …)
  - startActivityForResult(Intent intent, …)
- Target activity – Android has two ways to figure out which single activity it will start
- Can be named explicitly by setting the intent’s component
- Can be determined implicitly based on
  - Intent
  - Properties of activities installed on device

Explicit Activation

- Example of an application starting another activity
  - HelloWorldWithLogin

- Comprises two activities
  - LoginActivity – checks username and password
  - HelloAndroidActivity – shows “Hello Android” message

- Look into the code to understand better

Implicit Activation

- Intent resolution process – when the activity to be started is not explicitly named, Android tries to find activities that match the intent

- Intent resolution depends on two types of information
  - An intent describing a desired operation
  - Intent filters, describing which operations an activity can handle
    - Specified either in AndroidManifest.xml or programmatically

- Intent resolution looks specifically at three fields
  - Action field
  - Data field (both URI and MIME type)
  - Category

Specifying Intent Filter

- Using the intent-filter tag
  - For example, if an activity can dial phone numbers, it should use intent filters with “android.intent.action.DIAL” as the actionName

```
<activity …>
    <intent-filter …>
        …
        <action android:name=“actionName” />
        …
    <intent-filter>
…
</activity>
```

Specifying Intent Filter

- Adding data to intent-filter tag

```
<activity …>
    <intent-filter …>
        …
        <data
            android:mimeType=“string”
            android:scheme=“string”
            android:host=“string”
            android:port=“string”
            android:path=“string”
            android:pathPattern=“string”
            android:pathPrefix=“string”
        />
    <intent-filter>
…
</activity>
```

Specifying Intent Filters

- Handling geo: scheme intent – if an activity wants to publish that it can show maps

```
<activity …>
   <intent-filter …>
      …
      <data android:scheme="geo" />
      …
   <intent-filter>
…
</activity>
```

```
<intent-filter …>
   …
   <category android:name="string" />
   …
<intent-filter>
```

- It can also specify category in intent filters as above

Example: Maps Application

- Google Maps can handle intents that have an action of intent.action.VIEW and a data field with a geo scheme

```
<intent-filter …>
  <action android:name=“android.intent.action.VIEW” />
  <category android:name=“android.intent.category.DEFAULT” />
  <category android:name=“android.intent.category.BROWSABLE” />
  <data android:scheme=“geo” />
<intent-filter>
```

- To receive implicit intents an activity should specify an intent-filter with the DAFAULT category as
  - Android.intent.category.DEFAULT

Priority

- When more than one activity can accept a particular intent, Android needs to break tie
  - Ask user
  - Causes Android to prefer one activity over another
  - Value should be greater than -1000 and less than 1000

- More about intent-filters
  - % adb shell dumbsys package

Android Permissions

Permissions

- How Android can define and use permissions to control access to
  - Important data
  - Resources
  - Operations

- Things to cover
  - Android permissions architecture
  - Defining and using application permissions
  - Component permissions and permission-related APIs

Permissions

- Applications can define permissions to limit access to
  - User information – e.g., contacts
  - Cost-sensitive APIs – e.g., using SMS/MMS
  - System resources – e.g., using camera
- Permissions are represented as strings
- Applications declare permissions in android.manifest.xml file
  - Permissions that they use themselves
  - Permissions that they require of other components that want to use them

Using Permissions

- Applications specify permissions they use through a <uses-permission> tag
- Users must accept these permissions before an application can be installed
  - Otherwise error or access failure may occur

```
<manifest …>
   <uses-permission android:name=“android.permission.CAMERA” />
   <uses-permission android:name=“android.permission.INTERNET” />
   <uses-permission android:name=
                  “android.permission.ACCESS_FINE_LOCATION” />
…
<manifest>
```

Using Permissions

- MapLocationFromContacts App
  - Select a contact from contacts database
  - Display a map centered on the selected contact’s address

- Since contact list is private, the application must declare that it uses the contact list by setting appropriate permissions in the android.manifest.xml file

Defining Permissions

- Applications can also define and enforce their own permissions to prevent other applications from using them

- Example – an application performs a privileged or dangerous operation, e.g., formatting external memory card
  - Might not want to allow just any application to invoke it
  - “Boom” application

```
<permission>
   android:name=“course.examples.permissionexample.BOOM_PERM”
   android:description=“@string/boom_perm_string”
   android:label=“@string/boom_permission_label_string”
</permission>
```

Component Permissions

- Individual components can set their own permissions, restricting which other components can access them

- Component permissions can take precedence over application-level permissions

- Different types of component permissions
  - Activity permissions
  - Service permissions
  - Broadcast Receiver permissions
  - Content Provider permissions

Activity Permissions

- Restrict which components can start the associated activity

- Checked within the execution of
  - startActivity()
  - startActivityForResult()

- Throws security exception on permissions failure

Service Permissions

- Restrict which components can start or bind to the associated service

- Checked within the execution of
  - Content.startService()
  - Context.stopService()
  - Content.bindService()

- Throws security exception

Broadcast Receiver Permissions

- Restrict which components can send and receive broadcasts

- Permissions checked in multiple places

Content Provider Permissions

- Restrict which components can read and write the data in a content provider

Fragment Class

Fragment

- Fragments were added to Android in version 3.0 to better support user interfaces on large screens (e.g., tablets)

- Because of larger screens, some of the heuristics designed for phones with smaller screens no longer work

- Tablets can support multiple UI panes / user behaviors at the same time

- Example: Quote Viewer application – uses two activities (not user friendly)
  - One shows titles of Shakespeare plays and allows user to select one
  - The other shows a quote from the selected play

Fragment

- Fragment represents a behavior or a portion of the UI within an activity

- Fragment Static Layout application uses a single activity (user friendly for tablets)
  - Has two fragments – one for titles on the left (title fragment), and the other for quotes on the right (quote fragment)

- Fragments are hosted by activities
  - Multiple fragments can be embedded in an activity to create a multi-pane UI
  - A single fragment can be reused across multiple activities

Fragment Lifecycle

- Since fragments are hosted by activities, they have to be loaded into the activities, displayed, removed, etc. as the activity changes its state

- Fragment lifecycle is tied to and coordinated with the lifecycle of its containing activity

- Fragments also have their own lifecycles and receive their own callbacks

- Fragments can be statically or dynamically bound with the hosting activity

Fragment Lifecycle States

- Resumed – fragment is visible in the running activity
- Paused – when the hosting activity is visible, but another activity is in the foreground and has focus
- Stopped – fragment is not visible

Lifecycle Callback Methods

- Fragment receives several callback methods when the hosting activity is created by onCreate(), and the fragment is attached by onAttachFragment()

- onAttach() - Fragment is first attached to its activity

- onCreate() - Initialize the fragment
  - Does not set up the user interface as in Activity.onCreate()

- onCreateView() - Fragment sets up and returns its UI

- onActivityCreated() - Containing activity has completed and the fragment has been installed

Lifecycle Callback Methods

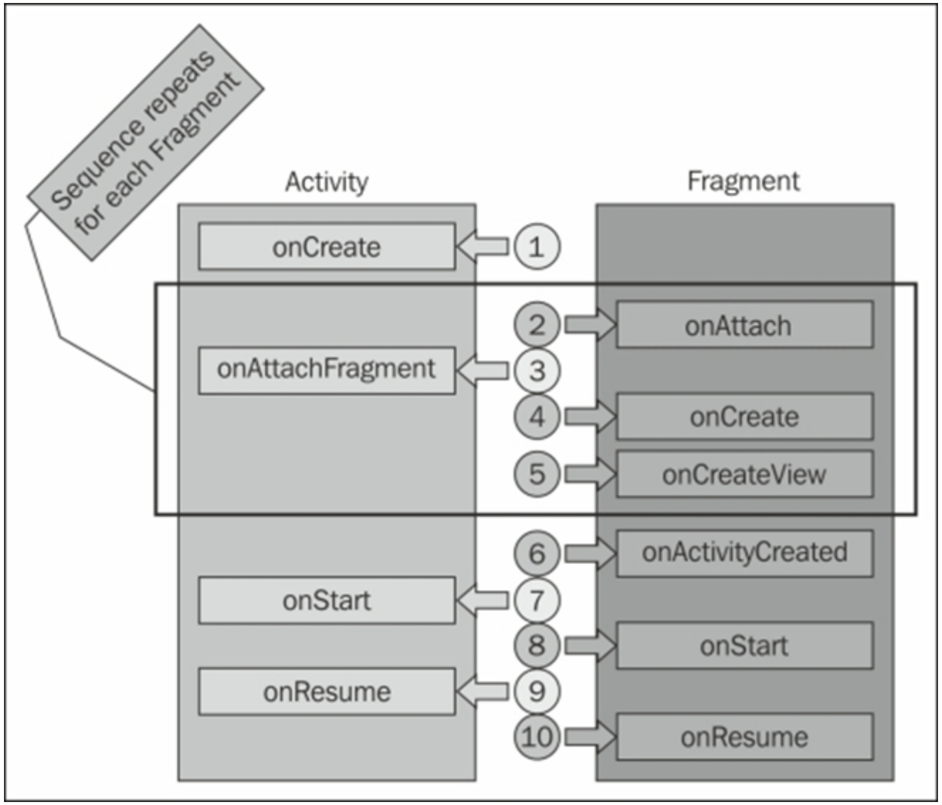
- Android calls the following fragment-specific methods depending on the state of the activity

- Activity started – onStart()
  - Hosting activity about to become visible

- Activity resumed – onResume()
  - Hosting activity about to become visible and ready for user interaction

- Activity paused – onPause()
  - Hosting activity is visible, but does not have focus

Lifecycle Callback Methods

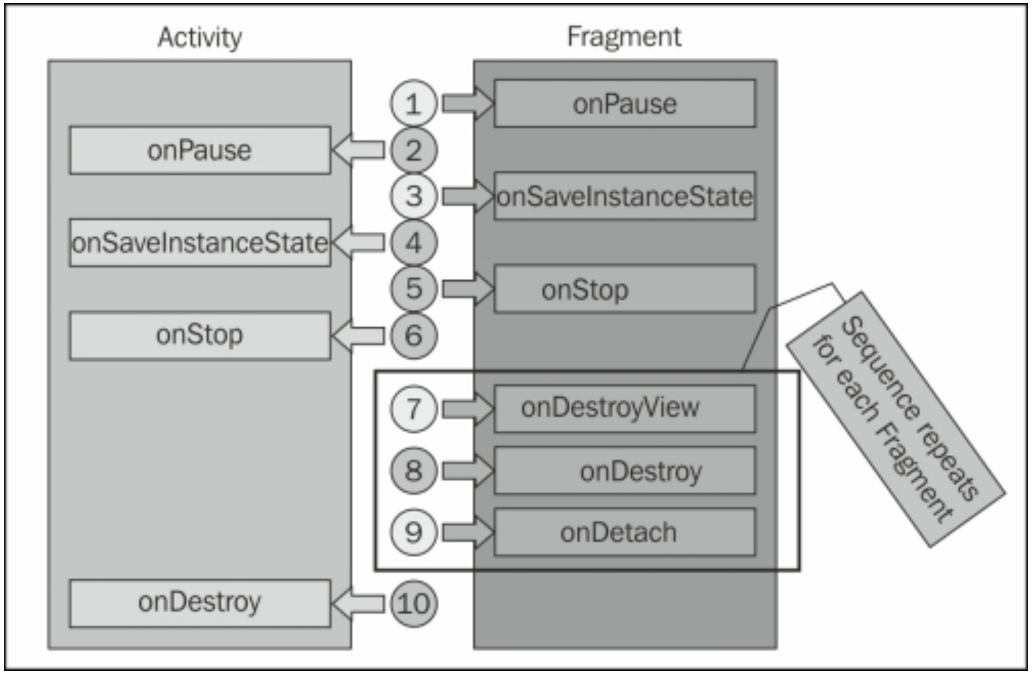


Lifecycle Callback Methods (contd.)

- Activity stopped – onStop()
  - Hosting activity is no longer visible

- When the hosting activity is destroyed, Android calls several methods of the fragment
  - onDestroyView()
    - View previously created by onCreateView() has been detached from the activity
    - Typical actions – clean up resources associated with the view
  - onDestroy()
    - Fragment no longer in use – clean up fragment resources
  - onDetach()
    - Fragment no longer attached to its activity – null out references to the hosting activity

Lifecycle Callback Methods (contd.)



Adding Fragments to Activities

- Two general ways to add fragments to an activity’s layout
  - Declare it statically in the activity’s layout file
  - Add it programmatically using the Fragment Manager

- Once added, fragment layouts can be inflated / implemented in onCreateView()
  - Similar to activities when they call setContentView()
  - Layout can also be created programmatically

- onCreateView() must return the view at the root of the fragment’s layout
  - The view is added to the containing activity

Adding Fragments Dynamically

- Need to do four things to add a fragment to an activity’s layout while it’s running
  - Get reference to the Fragment Manager
  - Begin a fragment transaction
  - Add the fragment
  - Commit the fragment transaction

- Dynamic layout
  - Fragment transactions allow you to dynamically change the application’s user interface
  - Can make the interface more fluid and take better advantage of available screen space

- Example: FragmentDynamicLayout application

Configuration Changes

- Activities can handle configuration changes manually using methods such as
  - onRetainNonConfigurationInstance()
  - getLastNonConfigurationInstance()
  - These two methods are deprecated in the fragment class

- If setRetainInstance(true) is called, Android will kill the activity on configuration changes, but won't destroy the fragment
  - Instead, it will save the fragment state, and detach the fragment from the activity
  - Results in some changes to lifecycle callback sequence
    - onDestroy() will not be called
    - onCreate() will not be called

Configuration Changes Example

- FragmentStaticConfigLayout – landscape mode
  - Both fragments use a large font (32 sp)
  - Title fragment takes more horizontal space (1/3rd)
  - Allows long titles to span multiple lines

- FragmentStaticConfigLayout – portrait mode
  - Both fragments use smaller font (20 sp)
  - Title fragment will use less space (1/4th)
  - Ellipsize (add dots) long titles, limiting them to a single line

User Interface Classes

User Interface Classes

- Views and View Events
- View Groups – Adapter Views and Layouts
- Menus and Action Bar
- Dialogs
- Android provides many classes for constructing user interfaces
- Activities usually display a visual user interface

View Class

- Key building block for UI components
- Occupy a rectangular space on screen
- Responsible for drawing themselves and for handling events directed to them
- Some predefined views – all use listeners
  - Button
  - ToggleButton
  - CheckBox
  - RatingBar
  - AutoCompleteTextView (e.g., country name match, filters, instead of long scrolling)

Common View Operations

- Set visibility - show or hide view
- Set checked state
- Set listeners – code for specific events
- Set properties – opacity, background, orientation
- Manage input focus – allow view to take / request focus

View Event Sources

- User interaction
  - Touch
  - Keyboard / trackball / d-pad

- System control – Android itself can be a source of events
  - Lifecycle changes e.g., reposition / redraw a view

- Handling view events by listeners
  - onClickListener.onClick() – view has been clicked
  - onLongClickListener.onLongClick() – view has been pressed and held
  - onFocuschangeListener.onFocusChange() – view has received / lost focus
  - onKeyListener.onKey() – view about to receive a hardware key press

Displaying Views

- Views are organized in a tree – outermost view, which holds child views, …

- When Android draws the views on screen, it goes through the view tree multiple times and does different things
  - 1st pass – measure or get dimensions of each view
    - Calls onMeasure()
  - 2nd pass – layout or position each view
    - Calls onLayout()
  - 3rd pass – draw each view
    - Calls onDraw()
  - Other relevant methods
    - onFocusChanged(), onKeyUp(), onKeyDown(), onWindowVisibilityChanged()

ViewGroup

- An invisible view that contains other views and is used for grouping and organizing a set of views

- ViewGroup is a base class for view containers and layouts

- Some predefined ViewGroups
  - RadioGroup – mutually exclusive radio buttons (age groups)
  - TimePicker …
  - DatePicker
  - WebView – displays webpages
  - MapView – displays maps and allows user to interact
  - Gallery
  - Spinner

Adapter & AdapterViews

- For situations where different developers may want to display different kinds of data
  - E.g., ListView – list of songs, images, wallpapers, …
- AdapterViews are view groups whose children are managed not by the view groups themselves, but by an adapter
- Adapter manages the data and provides data views to AdapterView
- AdapterView displays the data views

Examples: ListView, Spinner, Gallery

- ListView – an AdapterView displaying a scrollable list of selectable items
  - Items managed by a ListAdapter
  - ListView can filter the list of items based on text input

- Spinner – an AdapterView providing a scrollable list of items
  - User can select one item from the list
  - Items managed by a SpinnerAdapter

- Gallery – an AdapterView showing a horizontally scrolling list (e.g., swiping images horizontally)
  - Items managed by a SpinnerAdapter

Layouts

- A generic ViewGroup that defines a structure for the views it contains

- Example 1: LinearLayout
  - Child views arranged in a single horizontal or vertical row

- Example 2: RelativeLayout
  - Child views are positioned relative to each other and to the parent view

- Example 3: GridView
  - Child views are arranged in a two-dimensional, scrollable grid

Networking

Networking

- Earlier handheld devices gave us mobility, but with limited connectivity compared to today's devices

- Connect your Android device with another or the Internet using HTTP
  - getRequest()

- Android networking classes

- Processing HTTP responses
  - JavaScript Object Notation Language (JSON)
  - Extensible Markup Language (XML)

Networking Classes

- Android includes multiple networking support classes, e.g.,
  - Java.net package (Socket, URL)
  - Org.apache package (HttpRequest, HttpResponse)
  - Android.net package (URI, AndroidHttpClient, AudioStream)

- Example application
  - Interacts with a server to get earthquake information that has occurred in a particular geographic region
  - Data returned in various formats
    - First we will display just the raw textual data
    - Then how to extract desired information
    - Begs for a map view

Sending HTTP Requests

- We will talk about three classes each one of which will be used to implement the same earthquake application
  - Socket
  - HttpURLConnection
  - AndroidHttpClient