ES3 Lecture 9

Android development

Android

- Google's mobile OS
  - Runs on many hardware platforms from many manufacturers
  - Even an Android microwave!
- Largely open source
  - Based on a Linux kernel with custom drivers
  - Note this means that vendors often customize Android to their own ends...
- Development is in Java
  - Applications run in the Dalvik Virtual Machine, which runs on top of the core OS
  - All user applications run in the VM
    - Although you can implement library code natively and call it from the VM
    - Uses Java Native Interface
- Much more open than iPhone
  - background processes, can access SMS, make phone calls, scan Bluetooth and so on

Java

- One major advantage of Android is that Java is already well-known
  - Android extends the API with a android.* class hierarchy
- Unfortunately, the Java implementation is not (completely) compatible with standard Java
  - Language is basically the same, but the class library is not.
  - Standard Java API is partly there, but not all of it -- and there's no guarantee it will work as you expect
- Standard Java development tools can be used
  - Eclipse is by far the most common tool, and there are specific Android development plugins to make development easy

Development

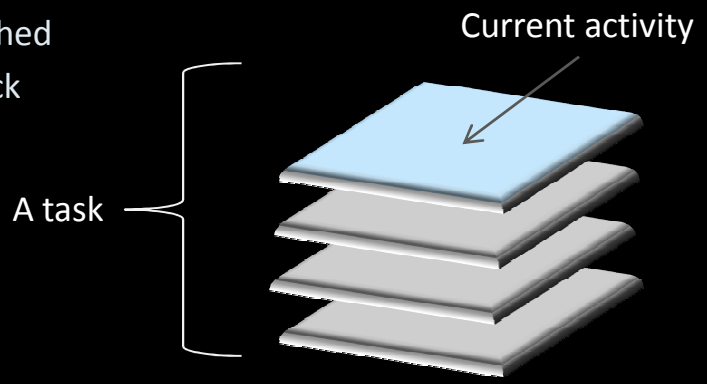
- There is a good emulator available which can support many device variations
  - The emulator and real devices support remote console access, so that debugging, deployment and configuration can be greatly simplified
- Standard IDE and debugging tools are provided by Eclipse, using the ADT plugins for Android extensions
- Signing is required for all apps, but is lightweight
  - Developers can self-sign for any of their own devices (this is automatic when debugging)
  - For release, self-signing is still possible (no authority is required)
    - Release certificates must identify the creating organiziation

Types

- Android has several types of components
  - Unlike the iPhone, which just has apps
- These are:
  - Activity
    - an application, or rather "part" of an application in a *task*
    - **e.g. a map view**
  - Service
    - a background process that doesn't have continuous user interaction
    - **e.g. a network SSID scanner**
  - BroadcastReceiver
    - a handler for incoming events (such as phone calls, SMS, battery warning)
    - **e.g. an SMS autoresponder**
  - ContentProvider
    - a service which provides data to multiple other applications
    - **e.g. offers an API to an SQLite database**

Applications, Tasks and Stacks

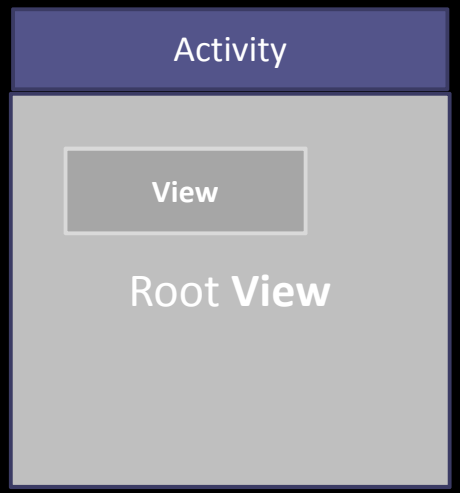
- Android can run multiple applications simultaneously
- From the point of the view of the user, each application is a **task**
  - A **task** consists of a **stack** of **Activity** elements
  - **Activities** are not one to one mappings to applications!
- **Applications can share components!**
  - For example, you can invoke an **Activity** from another application or library, and it will appear as part of the current application
    - You can then return to the previous **Activity** when finished
    - Activities can be pushed and popped from the task stack
  - **This is an extremely flexible setup!**



- Can confgiure whether apps uses stacks or single activities

Activities

- An **Activity** is an application component with a user interface
  - Usually a full screen UI
- **Activities** have a window, which contains **Views** (and **ViewGroups**)
  - A **View** is just a UI component, like a button, slider, canvas and so on
  - **Views** respond to events and call handlers in your code
- Activities can start other activities, which will replace them at the top of the task's **stack**
- On the iPhone, you effectively have one **Activity** all the time



Services

- A **Service** just runs in the background
  - Applications (tasks) can start and stop services, and bind to them to communicate
  - Services can't have user interfaces (though they can start a new Activity)
- **Services** persist beyond tasks
  - For example, you could start downloading something in the background in a **Service** subclass and then close the application
    - The download will continue in the background Service
- Obviously this brings risks...
  - can easily clog up the system with hidden services
  - performance and security issues become important

Intents

- **Intent**s are a vital part of the Android API
- An **Intent** is a notification message handled by the OS
  - Intents start Activities, Services and BroadcastReceiver

- User interfaces are launched, for example, by sending an **Intent** to the **Context** (a global context object) to start the **Activity**
  - Context.startActivity() launches an activity
  - Context.startService() launches a service

- Has at least an **action** (specifying what to do) and usually **data (**specified as a URI, specifying data to act on)
  - Can also specify type of data, type of action and arbitrary user data

BroadcastReceiver

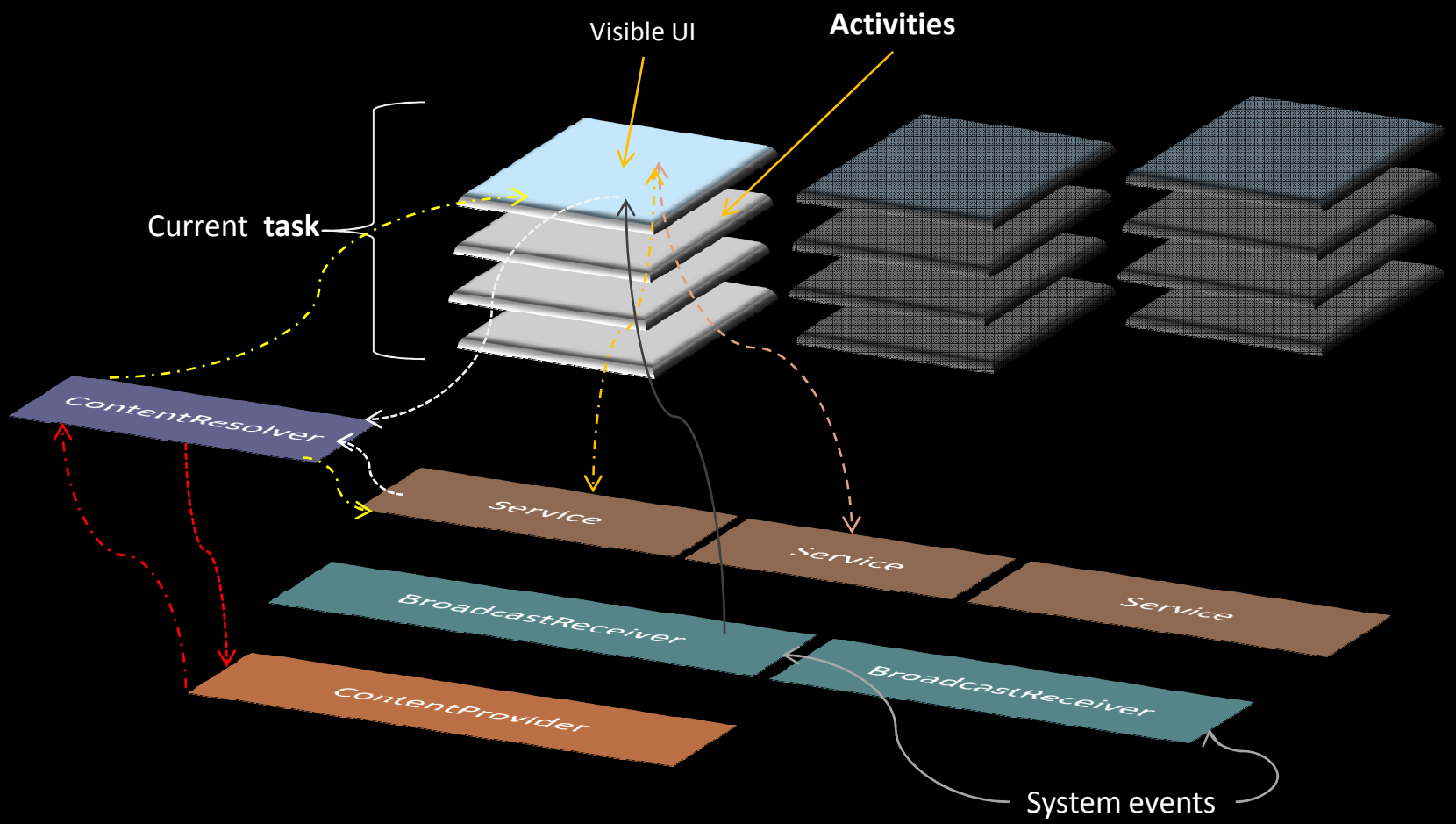
- **BroadcastReceiver** is a bit like a Service
  - No UI
  - But doesn't run continually in the background

- You register events for a **BroadcastReceiver** to respond to
  - When those occur, the OS automatically triggers the appropriate methods of the **BroadcastReceiver** instance registered
  - When events occur, they an **Intent** is passed, which specifies something about the type of action, and any data which may be associated
    - For example, an SMS comes in, and the **Intent** will have the SMS phone number and text in it

ContentProvider

- A **ContentProvider** is another kind of background object
- Instead of responding to events, it responds to requests for data
  - e.g. for queries
- **ContentProviders** register themselves with the runtime
  - When other applications ask for data which the **ContentProvider** can provide, the registered method is called and the data returned
  - Data is always returned as a table with rows of named records (as in a simple database)
  - A **ContentResolver** deals with matching requests to providers who can actually return the data
- Note: data from **ContentProvider** can (optionally) be modified!
- Data is requested using URI's
  - Uniform Resource Identifier of the form *content://com.myapplication.whatever/dataname*

Android Structure



Context

- The **Context** class provides global access to the applications environment
- Only class methods are used
  - e.g. **Context.getResources**() returns all available resources in the application
- The context is used to do things like register handlers for broadcast messages, look up resources, access string tables, access the applications home directory and so on
- Activities are launched with **Context.startActivity,** services with **Context.startService**
- Messages can be sent to all **BroadcastReceivers** with **sendBroadcast**

XML

- Android uses XML extensively for configuration
  - Lots of coding involves writing XML to describe structures rather than coding them in Java
- User interfaces are normally entirely defined in XML
  - No standard UI designer like InterfaceBuilder, although there are some 3rd party tools (of variable quality)
  - The ADT tool in Eclipse does provide a *preview* however
- The application manifest, which describes the basic properties of the application is an XML file

The manifest file

- All Android applications have a *manifest*
  - Similar to an **info.plist** on the iPhone platform
- XML specification which sets important attributes about the application
  - Lists all **Activities**, **BroadcastReceivers**, **Services** and **ContentProviders**
  - Also lists the **Intents** that they can handle
- Application entry point is set here as an **IntentFilter**
  - A filter with **MAIN** action and a **LAUNCHER** category will receive the Intent that launches the app
    - app launching is communicate with an **Intent** like everything else in Android
- Also sets permissions the application requires (e.g. reading personal data)
- Other items like the icon for the application are also set here

Using Intents

- **Intents** represent messages and have several components
  - **Component** (optional): type of receiving object (e.g. com.myapplication.BaseActivity)
  - **Action**: a constant specifying the action this Intent represents
  - **Data:** a URI pointing to the data and its MIME type
  - **Category:** a set of category descriptions of the *receiver* object (e.g. whether it is an initial activity or whether it is visible on the home screen)
  - **Extras:** any additional user data, as a dictionary

- Intents can either be set to a specific class object (the **component**), or Android can resolve the appropriate class
  - An **IntentFilter** is used to represent the **Intents** that an activity/service/broadcastreceiver can handle
  - **IntentFilters** are registered with the runtime (usually by declaring them in the manifest)
  - Android routes appropriate **Intents** to matching **IntentFilters**

Intent Filters

- An **IntentFilter** represents up to three possible tests
  - **Action** test: does the receiver respond to these actions?
  - **Category** test: does the receiver respond to these categories?
  - **Data** test: does the receiver take data with URI's matching a pattern or with given MIME type(s)?

- This is an example filter which shows how the "main" activity in an application is specified

```
<activity ...>
<intent-filter>
  <action android:name="android.intent.action.MAIN"/>
  <category android:name="android.intent.category.LAUNCHER"/>

</intent-filter>
</activity>
```

  - When the application launches, the **MAIN** action **Intent** is sent
  - Android resolves this to this activity and starts the application

Resources

- Applications virtually always have resources
  - Images, text, XML layout files, sound files, databases
  - Android has a **res**/ directory which includes all resource files
- Resources can include
  - XML "Values" files
    - This can specify simple values, like numbers, strings
  - Drawable graphics
    - like jpg or png images
  - Arbitrary XML files
  - Arbitrary binary files
  - Animation data
  - XML layout files
    - for specifying UI layouts
- These can be accessed from your code easily
  - Android compiles the resources and automatically creates a object to access resources
  - This is the globally available **R** instance

The R class

- When Android compiles the resources it generates the **R** class with standard properties for accessing references to data
  - If you define a string resource, you can access with R.string.*name*

```
... in the XML file somewhere in res/ ...
<string name="description"> A test application </string>

... in the code ...

doSomethingWith(R.string.description);
```

  - The same is true for other types (e.g. a layout is found in R.layout.*name*)

- Bitmap images are automatically compiled to Drawable objects
  - e.g. background.png becomes R.drawable.background

```
Drawable background = Context.getResources().getDrawable(R.drawable.background);
```

Programmatic User Interface

- There are two ways of creating user interface components in Android
  - Programmatic user interfaces: generate components by instantiating objects in code
  - XML user interfaces: a UI layout is specified in XML in the resources. References to layout components can be obtained via the **R** class.

- Most user interface creation will use XML (this is strongly encouraged)
  - Sometimes generating code interfaces is essential though

- Procedure:
  - instantiate component objects
  - instantiate a layout and add the components
  - use setContentView to set the root layout

```
TextView text = newTextView(this);
text.setText("Hello, World!");
setContentView(text);
```

View and ViewGroup

- All UI components are subclasses of **View**
- Those that can contain other objects are subclasses of **ViewGroup**
  - **Layouts** are **ViewGroups**
  - Layouts are containers which specify how components will be spatially arranged
  - Examples:
    - **LinearLayout** (vertical/horizontal list of components)
    - **TableLayout** (2D table layout)
    - **RelativeLayout (components laid out by relative edges/centers)**
    - **AbsoluteLayout** (exact pixel positions specified)

XML User interfaces

- Normally, UI will be specified in an XML file in the resources (res/layout/)
  - XML file usually has a **Layout,** with a set of other components inside
    - e.g. **LinearLayout** with **TextViews**
    - Layouts obviously can include other layouts as well...
  - each component can have its properties set (e.g. size, text, color...)
    - Also an ID which is used to get a reference to an object in the code
    - e.g. so that it can be actually added to the screen!
  - Main UI is usually specified in res/layout/main.xml

- Once a layout has been defined, an object can linked with **findViewById**
  - Takes the ID you specified in the XML file and returns the object so that it can be manipulated
  - ID's are always part of the R class, of the form R.id.xxx
    - e.g.R.id.launch_button

```
// this loads the layout specified in res/layout/main.xml
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
}
```

Event Handling

- Event handling is very simple in Android

- Two basic methods:
  - Subclass a UI component and override event methods (**onTouchEvent**() for example)
    - Not recommended for most applications!
  - Use *event listeners*
    - create an event listener class, and attach it to an object
    - listener classes could also be the current Activity (must conform to the appropriate interface for the listener)

```
// Activity definition
// must include the interface for the listener type!
public class LaunchNotifyActivity extends Activity implements OnClickListener {

...

// in onCreate()
// look up the button in the XML file
Button triggerRelease = (Button) findViewById(R.id.trigger);

// make the current object the click listener
triggerRelase.setOnClickListener(this);
```

Summary

- Android applications are highly modular
  - basic components include Activities (with a UI), Services (background), ContentProviders (return information) and BroadcastReceivers (receive system events)
- A **task** is an application from the user point of view
  - can have multiple Activities, all from different applications
  - applications can share UI components
  - can share data access and communicate via broadcast events
- Much of Android involves describing structures in XML rather than implementing them directly
  - e.g. user interface design