14.1 Overview of Android

- Blackberry smart phone appeared in 2003
  – First widely used mobile access to the Web

- Smart phone market now dominated by Android, iPhone, and Windows Phone

- Tablets are now popular, using the same operating systems

- Android was initially developed by Android, Inc.

- Acquired by Google in 2005

- In 2007, development was taken over by the Open Handset Alliance, with more than 80 member companies

  - It is an open source product, written in a form of Java, using an XML tag set for the displays

- Mobile devices use two communications methods: WiFi or a cell phone network
14.2 Tools

- Android software is nearly always developed with the help of an IDE

- **Tools we’ll use:**
  - JDK
  - Android Software Development Kit
    - Which includes the Eclipse Android IDE and the Android Development Tools (ADT)

14.3 Architecture of Android Applications

- All Android applications use the MVC architecture

- **Activities**
  - Related to servlets
  - An activity is a Java class instance with an associated view markup file
  - Each activity manages one page of display
14.3 Architecture of Android Applications
(continued)

- **View Files**
  - XML documents that display forms

- **Intents**
  - A means of communication within and among applications
  - An intent is a Java class instance, often used to start and stop activities

- **Implementation**
  - An Android application is built as a project, with one or more activities
  - Applications are compiled to an intermediate form similar to byte code, interpreted by the Dalvik VM
  - Each application runs in its own process on its own copy of the Dalvik VM
14.4 Execution Model

- An Android application starts execution when the user starts it.

- It ends when the user starts another application or the operating system stops it to capture its resources.

- Execution is controlled by callback methods that are implicitly called when system- or user-raised events occur.

- Activities are placed on a stack, with the top one currently running with its view displayed.

→ SHOW Figure 14.1

- The `onCreate` method sets the activity’s view and often initializes some class-scope variables.

- The `onStart`, which is implicitly called when `onCreate` is finished, displays the view and calls `onResume`.

- The `onResume` starts the execution of the activity’s code.
14.4 Execution Model (continued)

- The `onPause` method takes an activity from the *resumed* state to the *paused* state

- The `onStop` method takes an activity from the *paused* state to the *stopped* state

- The `onDestroy` method takes an activity to the *destroyed* state

- All of these are called the *life-cycle methods*

14.5 View Groups

- A view is similar to a form, with several subviews, or components

- A view group, derived from the `ViewGroup` class, is a container for views

- A layout manager, called a *layout*, is also derived from the `ViewGroup` class

- *There are four layout managers:*

  - `LinearLayout` – views are arranged in a column or a row
14.5 View Groups (continued)

- Layout managers (continued):
  - FrameLayout – pins its views around its frame
  - RelativeLayout – views are positioned relative to other views in the layout
  - GridLayout – views are arranged in a grid

- Layouts can be nested

- It is preferable to use markup to create view documents

- Attributes of LinearLayout:
  - orientation – vertical or horizontal
  - layout_width and layout_height
    - match_parent or wrap_content
14.6 Simple Views

- **TextView** – used to define fixed text

- Attributes:
  - **id**
    - Specified as "@+id/the_name"
    - Means the XML parser should parse the whole string and identify it as a resource id
    - Also, the + means it is a new resource to be put in R.java

  - **layout_width** and **layout_height**

  - **text** – the text of the element

```xml
<TextView
    android:id = "@+id/label1"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "The temperature is: " />
```

- Normally, the actual text is defined as a resource in the strings.xml file of the res/values directory

```xml
<string
    name = "temp_label"
    "The temperature is: " />
```

- The **text** attribute then would be:

```xml
android:text = "@string/temp_label"
```
14.6 Simple Views (continued)

- EditText – A subclass of TextView
  
  - Allows editing of its contents (like HTML textboxes)
  
  - Same attributes as TextView, plus hint for initial text and inputType
  
  - Tapping in the text field produces a standard keyboard
  
  - To get a special keyboard, include inputType with a value such as:
    textEmailAddress, textUri, number, and phone

  (there are several others)

- Button Elements

  <Button android:id = "@+id/button_send"
          android:layout_width = "wrap_content"
          android:layout_height = "wrap_content"
          android:text = "@string/button_send" />

With the following in res/values/strings.xml:

  <string name = "button_send" > Send </string>
14.6 Simple Views (continued)

- Events and Handlers

- Button click events are Click

- Can be registered in code, but we’ll do it in markup

    <Button android:id = "..."
           ...
           android:onClick = "the_handler" />

14.7 An Example Application

1. Start Eclipse and select
   
   File/ New/ Android Application Project

   This produces the New Android Application screen

2. Type the application’s name; we use Simple1

3. Change the package name; we chose
   
   com.myexample.simple1

4. Click Next, which produces the Configure Project screen
14.7 An Example Application (continued)

5. Defaults are ok, so click Next to get the Configure Launcher Icon screen, which is ok as is, so click Next again, which produces the Create Activity screen.

6. The Create Activity screen has Blank Activity preselected, which is ok, so click Next to get the Blank Activity screen.

7. Click Finish; this displays the workspace view of the project.

Project workspace:
14.7 An Example Application (continued)

The workspace has much information:

- The left window has *Package Explorer*, which lists all of the projects in the workspace, including our example, *Simple1*
  - The projects are partially elided; clicking the small triangle left of a directory’s name expands the directory (empty triangles indicate elided names)

- Clicking the *activity_main.xml* tab at the top of the central window, we get the workspace showing the graphical layout of *activity_main.xml*
  - The central window shows an emulated display
14.7 An Example Application (continued)

Contents of `activity_main.xml`

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"/>

</RelativeLayout>
```
14.7 An Example Application (continued)

- To run the project on an emulator:

1. Create a specific device emulator

   - Select Window/Android Virtual Device Manager

   - Click New

   - Type in avd as the name for our AVD, choose 5.1 inch WVGA device and the Android 4.3 target

   - Unselect the checkbox for keyboard

   - Select ARM processor, though it is irrelevant for simple projects

2. Back in the Android Virtual Device Manager screen, select avd and click Start; this opens the Launch Options window, on which we click Scale display to real size and Launch

3. After a while, the emulated device screen will appear. Select Run/Run, which may open the Run As window

4. If Run As is opened, select Android Application and click OK (You may need to click the MENU button to see the display)
14.7 An Example Application (continued)

14.8 Running on an Android device

- Requires a driver running on the computer that has Android/ADT

- Connect the device to the computer with the cable used to charge the device

- USB debugging must be turned on
14.9 Using the Intent Class

- We’ll use it to call other activities

- First, create an Intent object

```
Intent intent = new Intent(this, Second.class);
```

- To call the activity:

```
startActivity(intent);
```

- This is ok, but can’t pass data to the second activity, which is no simple process

- To pass the content of an EditText component with the id editText1:

```
EditText editText =
    (EditText) findViewById(R.id.editText1);
```

- First get the address of the code of the component

```
String message = editText.getText().toString();
```

(toString is required because getText returns an Editable type)
14.9 Using the **Intent Class** (continued)

- Add *message* to the Intent object

- So the called activity can identify the message, precede it with a constant string, **EXTRA_MESSAGE**, in the caller

  ```java
  public final static String EXTRA_MESSAGE =
  "com.myexample.interact.MESSAGE";
  
  where interact is the name of the application
  ```

- Attach the text box content (*message*) and the constant string with:

  ```java
  intent.putExtra(EXTRA_MESSAGE, message);
  ```

- To retrieve the message in the called activity:

  - Create an Intent object and use `getStringExtra`:

  ```java
  Intent intent = getIntent();
  String message = intent.getStringExtra(
      MainActivity.EXTRA_MESSAGE);
  ```
14.10 A Second Activity

- The first activity asks the user for his or her name. The second is called with the name. It responds with a personalized greeting, including time and date.

→ SHOW activity_main.xml

- Sometimes, an event handler must be registered in code

```java
Button button = (Button) findViewById(R.id.mybutton);
button.setOnClickListener(
    new View.OnClickListener() {
        public void onClick(View view) { … }
    });
```

- The main activity defines the event handler for the button, sendMessage, which must call the second activity.

→ SHOW MainActivity.java

- Now build the second activity:

1. Select File/New/Other/Android/Android Activity and click Next
14.10 A Second Activity (continued)

2. Click Next on Blank Activity

3. Type in the activity’s name Response_Activity and its hierarchical parent, com.myexample.Ineract.MainActivity

4. Click Finish

- The framework-produced ResponseActivity.java includes some parts we don’t need

  - Remove onCreateOptionsMenu

  - Remove the block of comments on onOptionsItemSelected

  - Remove setUpActionBar and the call to it

→ SHOW the initial version of ResponseActivity.java (page 625)

- Add the necessary code to it:

  1. Get references to the two TextView boxes

     TextView greetBox =
                   (TextView) findViewById(R.id.greeting);
     TextView dayBox =
                   (TextView) findViewById(R.id.today);
14.10 A Second Activity (continued)

2. Create an Intent object and get the passed data

    Intent intent = getIntent();
    final String message = intent.getStringExtra(
            MainActivity.EXTRA_MESSAGE);

3. Add the code for the first response line and put it in the main activity’s TextView

    String out = "Hello, " + message + ", it’s nice to hear from you.";
    greetBox.setText(out);

4. Add the code for the second response line and put it in the main activity’s TextView

    String date = DateFormat.getDateTimeInstance()
            .format(new Date());
    String day = "Today is " + date;
    dayBox.setText(day);

→ SHOW complete ResponseActivity.java (page 627-628)

- Write the response view file

- Needs two TextView elements for the response with the ids greetBox and dayBox
14.10 A Second Activity (continued)

- If we run this application and enter Alison, we get the response:

![Image of an Android device showing a message from Alison]

Hello, Alison, it's nice to hear from you.
Today is Apr 18, 2013 9:49:58 PM

14.11 More Widgets

- **ToggleButton**

  Traditional (on and off)

  `<ToggleButton`
  android:id = "@+id/togbutton"
  android:layout_width = "wrap_content"
  android:layout_height = "wrap_content"
  android:textOn = "Vibration mode on"
  android:textOff = "Vibration mode off"
  android:onClick = "onToggleClicked" />

14.11 More Widgets (continued)

- **Toggle Buttons** (continued)

  - A skeletal handler:

    public void onToggleClicked(View view) {
      boolean on = ((ToggleButton) view).isChecked();
      if (on) {
        // Actions for when the toggle is turned on
      } else {
        // Actions for when the toggle is turned off
      }
    }

  - The other kind, named switch toggle buttons, are similar, but use a slide button instead of a regular button

- **Checkboxes**
  - Managed individually

  <CheckBox
    android:id = "@+id/checkbox_tomatoes"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "@string/tomatoes"
    android:onClick = "checkbox_handler" />


14.11 More Widgets (continued)

- Checkboxes (continued)

- A handler for two checkboxes:

```java
public void checkbox_handler(View view) {
    boolean checked =
        ((CheckBox) view).isChecked();

    switch(view.getID()) {
        case R.id.checkbox_tomatoes:
            if (checked)
                // Put tomatoes on the pizza
            else
                // Hold the tomatoes
            break;
        case R.id.checkbox_anchovies:
            if (checked)
                // Put anchovies on the pizza
            else
                // Hold the anchovies
            break;
    }
}
```

- Radio Buttons

- Must be nested inside `RadioGroup` elements
14.11 More Widgets (continued)

- Radio Buttons (continued)

→ SHOW XML on bottom p. 630 & top p. 631

- A handler for these three radio buttons is:

```java
public void radiohandler(View view) {
    boolean checked = ((RadioButton) view).isChecked();

    switch(view.getID()) {
        case R.id.ageunder35:
            if (checked)
                // Actions for younger folks
            break;
        case R.id.agebet3560:
            if (checked)
                // Actions for middle age folks
            break;
        case R.id.ageover60:
            if (checked)
                // Actions for older folks
            break;
    }
}
```
14.12 Lists

- **ListView**

  - Elements are similar to TextView elements, with id, layout_width, and layout_height attributes

  - The static part of a ListView can be represented in an array
    - If it is completely static, it can be an Array
    - If the length is dynamic, it needs to be an ArrayList

  - The array is bound to the ListView with an ArrayAdapter object

```java
final ArrayAdapter<String> adaptr;
adaptr = new ArrayAdapter<String>(
    this,
    android.R.layout.simple_list_item_1,
    listHolidays);

If the ListView element has the id aList, we could bind it with:

    aList.setAdapter(adaptr);
```
14.12 Lists (continued)

- *An example using lists*
  - A grocery list with a button to add an element and one to clear the list

→ SHOW the view document (p. 633)

- The required strings for `strings.xml`

```xml
<string name="new_list_item"> New list item </string>
<string name="add_item"> Add item </string>
<string name="clear_list"> Clear list </string>
```
14.12 Lists (continued)

- Now we change the main activity code file:

1. New class-level declarations so we can reference the ListView and EditText elements:

   ```java
   ListView theListView;
   EditText theEditText;
   ```

2. Declaration for the grocery list array and for a reference to the array adapter

   ```java
   ArrayList<String> groceryList =
       new ArrayList<String>();
   ArrayAdapter<String> adaptr;
   ```

3. Code to get the addresses of the objects associated with the ListView and EditText elements

   ```java
   theListView =
       (ListView) findViewById(R.id.myList);
   theEditText =
       (EditText) findViewById(R.id.newItem);
   ```
14.12 Lists (continued)

4. Instantiate the array adapter

   adaptor = new ArrayAdapter<String>(
            this,
            android.R.Layout.simple_list_item_1,
            grocerList);
   theListView.setAdapter(adaptr);

5. Event handlers for the two buttons

   public void addItem(View, view) {
      groceryList.add(groceryList.size(),
            theEditText.getText().toString());
      adaptor.notifyDataSetChanged();
      theEditText.setText('');
   }

   public void clearList(View view) {
      groceryList.clear();
      adaptor.notifyDataSetChanged();
   }

→SHOW MainActivity.java for the groceryList project (pp. 636-637)
14.13 Data Persistence

- To be useful, the grocery list must be saved between uses of the application

- There are three different ways to persist data in Android: shared preferences (for primitive data), files, and SQLite relational databases

- Shared preferences

- First create a `SharedPreferences` object

```java
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
```

- Then create an `Editor` object

```java
SharedPreferences.Editor edit = prefs.edit();
```

- Data is placed in savings with a `put type` method, such as `putString` or `putFloat`

- Data must be saved and retrieved with the same save identifier
14.13 Data Persistence (continued)

- Code to save data is placed in an overridden version of `onPause` and code to retrieve it is placed in an overridden version of `onResume`

- For our example, we cannot save the array, because it is not a primitive type, so we convert it to a primitive type, a `String`, by first converting it to a `StringBuilder` object

```java
StringBuilder bigString = new StringBuilder();
for (int i = 0; i < groceryList.size(); i++) {
    bigString.append(
        new StringBuilder(groceryList.get(i)).
            append(new StringBuilder(":")));
}
```

Now, `bigString` has the elements of the array with colons between the elements
14.13 Data Persistence (continued)

- To retrieve the data back into the array groceryList

String bigString =
    new String (PreferenceManager.
        getDefaultSharedPreferences(
            getBaseContext()).getString("SAVELIST", 
            ""));
String[] strings =
    bigString.split(":");
for (int i = 0; i < strings.length; 
    i++) {
    groceryList.add(strings[i]);
}

→ SHOW final version of MainActivity (pp. 639-641)

14.14 Debugging Applications

- Eclipse with the Android ADT plug-in includes an integrated debugger

- Right-click the application’s name in the Package Explorer and select Debug As/Android Application
14.14 Debugging Applications (continued)

- Click the Debug button in the upper right corner

- Set a breakpoint on a line of code by putting the cursor to the left of a line and right clicking

- Remove a breakpoint by right-clicking the blue circle at the left end of the line and selecting **Toggle Breakpoint**