Mobile Application Development
Aileen Pierce

LIST VIEWS AND ADAPTERS

Developing Android Apps

- Android also follows the model view controller (MVC) architecture
  - Model: holds the data and classes
  - View: all items for the user interface
  - Controller: links the model and the view together. The backbone or brain of the app.
- A layout describes the appearance of the screen
- An activity is a single, defined thing a user can do

Layout

- The **`View`** class is the building block for all user interface components
- All layouts and user interface components are subclasses of **`android.view.View`** so they all share common functionality
  - Getting and setting properties
    - **`findViewById()`**

Layout

- Linear Layout
  - Arranges views next to each other either horizontally or vertically
- Relative Layout
  - Arranges views relative to each other as well as to their parent view

Properties

- **`android:id`**
  - Gives the component a unique identifying name
  - Lets you access the widget in your code
  - Lets you refer to the widget in your layout
- **`android:text`**
  - the text displayed in that component

Resources

- A resource is a part of your app that is not code – images, audio, xml, etc
- You should use resources for strings instead of hard coding their values
  - Easier to make changes
  - Localization

`android:text="@string/heading"`

  - **`@string`** indicates it's a string in the strings.xml resource file
  - heading is the name of the string

Activities

- An activity is a single, specific task a user can do
- Each activity has its own window for its view The window typically fills the screen, but may be smaller than the screen and float on top of other windows
- An app can have as many activities as needed
- Each activity is listed in the AndroidManifest.xml file

Intents

- An intent tells the app you're about to do something such as start a new activity
  - An explicit intent tells the app to start a specific activity
  - An implicit intent tells the app to start any activity that can handle the action specified

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

Passing Data

- You can add extra information to your intent to pass data to the new activity

```
intent.putExtra("message", value);
```

  - The `putExtra()` method is overloaded so you can pass many possible types
  - Call `putExtra()` as many times as needed for the data you're passing

Receiving Data

- When a new activity starts it needs to receive any data passed to it in the intent
- Get access to the intent

```
Intent intent = getIntent();
```

- Retrieve the data sent in the intent

```
String msg =
intent.getStringExtra("message");
```

- You aren't limited to strings, there are many other getxxxExtra methods

Java

- Remember your semicolons!
- To log messages to the console in Java use `System.out.println("string");`
- Access levels are
  - public: accessible outside the class
  - private: only accessible inside the class
  - protected: accessible by the class and its subclasses

Java

- Methods

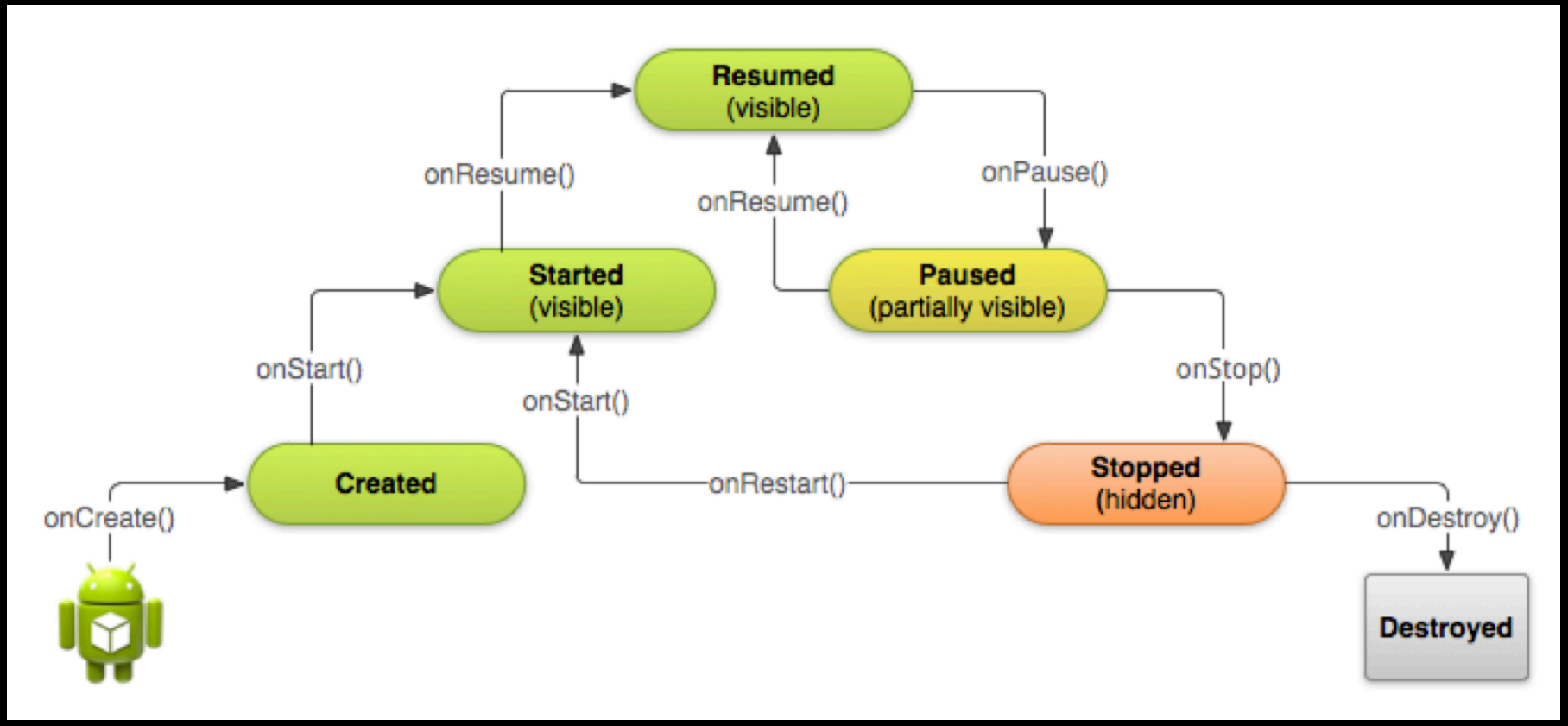
```
access return_type method_name(parameters)
{
    method body
}
```

  - The void return type doesn't return a value

Methods

- Classes have constructor methods to initialize objects of that class
  - Constructors have the same name as the class
  - A class can have multiple constructors that each take a different number and/or type of parameters
  - Constructors don't have a return type
  - If you don't provide a constructor method Java will create one
- Getter methods are used to get data
- Setter methods are used to set data

Android Lifecycle



Finishing Touches

- Launcher(app) icons should be provided
  - mdpi: 48x48 px (baseline)
  - hdpi: 72x72 px
  - xhdpi: 96x96 px
  - xxhdpi: 144x144 px
  - xxxhdpi: 192x192 px
- Launcher icons named ic_launcher.png go into density specific res/mipmap folders (i.e. res/mipmap-mdpi)
- Launcher icons should be designed specifically for Android. Avoid mimicking visual elements and styles from other platforms.

List Views

- List Views are used to display lists of data used to navigate in an app
- You can add a list view to your layout using the **`<ListView>`** element
- A **`ListActivity`** is a special type of activity that only has a list view in it.
  - It has a default layout file so you don't need to create one
  - The **`ListActivity`** class implements its own event listener so you don't need to create one

List Views

- Use the **`android:entries`** property to display static data stored in a string-array in strings.xml
- For nonstatic data stored in a Java array you need to use an adapter

Adapters

- An adapter acts as a bridge between the data source and the view
- Adapter views are view groups that display the data provided by an adapter
- Android has many different types of adapters
- **`ArrayAdapter`** binds arrays to an adapter view
  - can be used with any subclass of the **`AdapterView`** class such as spinners and list views
- You can also define custom adapters

Array Adapters

- To use an array adapter
  - Initialize the array adapter
  - Attach the array adapter to the list view using the **`ListView setAdapter()`** method
- The array adapter does the following:
  - Converts each item in the array to a **`String`** using the **`toString()`** method
  - puts each result in a text view
  - Displays each text view as a single row in the list view

Listeners

- We used the **`android:onClick`** attribute to respond to button clicks but that only works for the **`Button`** class and its subclasses such as **`CheckBox`** and **`RadioButton`**
- **`ListView`** isn't a subclass of button so we can't use **`android:onClick`**
- We have to implement our own event listener for list views

Listeners

- An event listener allows your app to listen for events and respond when an event takes place
  - when views get clicked, receive or lose focus, or a key is pressed
- The **`OnItemClickListener`** is an event listener class in the **`AdapterView`** class
- The **`onItemClick()`** method is called when an item is clicked

Listeners

- To implement an event listener
  - Create the **`OnItemClickListener`**
  - Implement **`onItemClick()`**
  - Attach the **`OnItemClickListener`** to the list view using the **`ListView`** **`setOnItemClickListener()`** method
- Attaching the listener to the list view is what allows the listener to get notified when the user clicks on items in the list view

List Activities

- Because the **`ListActivity`** class already implements an event listener you don't need to create one or bind it to the list view
- Instead you just need to implement the **`ListActivity onListItemClick()`** method

Passing Data

- List activities are often used to display data that when a user clicks will start another activity with detail data
- You do this by creating an intent to start the new activity in **`onListItemClick()`**
- It's common practice to pass the ID of the item clicked in the intent so the detail activity can get the correct detail data to populate its views