Android Basics: Multiscreen Apps

Intents and Activities

Documentation: Intent | Android Developers (https://developer.android.com/reference/android/content/Intent.html)
- ~Intents~ are requests for an action to be performed by another app.
- Apps can specify ~intent filters~ in the `AndroidManifest.xml` file
- An intent filter watches for an intent and when received, it performs a certain action.

```
// Create the intent
Intent numbersIntent = new Intent(MainActivity.this, NumbersActivity.class);

// Start the activity specified in the Intent
startActivity(numbersIntent);
```

Documentation: Intents and Intent Filters | Android Developers (https://developer.android.com/guide/components/intents-filters.html)
- ~Implicit Intents~: use these when you don't care which app component handles the intent, as long as they can handle it.
- Need an Action and Data URI
- Optional: Category, Components, Extras
- Create an object instance of the intent class and in the constructor, pass in an action string (`ACTION_SENDTO`)
- Implicit intents have a resolveActivity block where it handles making sure that the resolution happens

```
// Create the text message with a string
Intent sendIntent = new Intent(Intent.ACTION_SENDTO);
sendIntent.setData(Uri.parse("mailto:"));
sentIntent.putExtra(Intent.EXTRA_SUBJECT, "Just Java order for " + name);
sendIntent.putExtra(Intent.EXTRA_TEXT, priceMessage);

// Verify that the intent will resolve properly
if (sendIntent.resolveActivity(getPackageManager()) !=null) {
startActivity(sendIntent)
```

- ~Explicit Intents~: use these when you know what activity you want to receive your intent (usually used within an application).
    - Context and Component(usually Class/Activity)
        - Optional: Data URI

```
// Executed in an Activity, so 'this' is the Context
Intent intent = new Intent(this, NumbersActivity.class);
startActivity(intent);
```

Editing the AndroidManifest.xml File

Documentation: App Manifest | Android Developers (https://developer.android.com/guide/topics/manifest/manifest-intro.html)

The **label** attribute can be added to an activity in the `AndroidManifest.xml` file. It is what is shown to the user when they visit this activity.

Event Listeners in Android

Documentation: EventListener | Android Developers (https://developer.android.com/reference/java/util/EventListener.html)
~Events~ are things that can occur within an app
- Ex: click event, long click event, drag event, keyboard key event, etc.
~Event listeners~ specify what should happen when a certain event occurs.

1. User interacts with app, clicks on button
2. Click event is sent
3. onClick listener responds with the code contained in its method

Setting Up an Event Listener

1. Define the event listener and the custom behavior for when the event happens.
   In NumbersClickListener.java

```
public class NumbersClickListener implements OnClickListener{
    @Override
    public void onClick(View view) { Toast.makeText(view.getContext(), "Open the list
 of numbers", Toast.LENGTH_SHORT.show();}
}
```

1. Create a new object instance of the event listener (using the constructor)
   In MainActivity.java

```
NumbersClickListener clickListener = new NumbersClickListener();
```

1. Attach the listener to the view

```
buttonView.setOnClickListener(clickListener);
```

Shortcut:

```
buttonView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view){
        Toast.makeText(view.getContext(), "Open the list of numbers", Toast.LENGTH_SH
ORT).show();
    }
});
```

OnClickListener vs onClick

We create an anonymous subclass of OnClickListener and attach it to a view instead of using the onClick XML attribute.

We go from this:

```
android:onClick="myListener" // XML

public void myListener(View view){
    doStuff();
}
```

To this:

```
Button button = (Button) findViewById(R.id.le_button);
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view){
    doStuff();
}
});
```

Why?
- If you ever want to change the behavior of your button while your app is running, you can point your button to another method entirely, or just disable the button by setting an OnClickListener that doesn't do anything.
- When you define a listener using the onClick attribute, the view looks for a method with that name only in its host activity. If you set OnClickListener programmatically, you can control a button's behavior from outside of its host activity.

Interfaces

Interfaces vs Classes

- ::Classes:: are similar to ::Interfaces::.
    - ~Concrete Classes~ contain state and methods which are fully implemented. You can use these right away.

```
public class TextView{

    String mText;
        int mColor;

    void setText(String text) {
        mText = text;
    }


    void setTextColor(int color) {
        mTextColor = color;
    }
    ...
}
```

```
~ ~Interfaces~ have no state, and all of its methods are abstract (not implemented).
You must subclass this, and provide code for the abstract methods.

public interface OnClickListener
    void onClick(View v);
```

- ~Abstract Classes~ are partially implemented. They contain state, with some methods fully implemented, and some methods abstract (not implemented). You must provide code for all of the abstracted methods.

```
public abstract class ViewGroup

    int mChildrenCount;

    void addView(View child)
        addView(child, -1);

    void removeView(View view)
        removeViewInternal(view);

    void onLayout();

    ...
}
```

programming/java/android/beginner/3-multiscreen-apps/1-intents-activities