Decision Maker (Android)

Create an app that will make decisions for you when you are feeling undecisive

Jump To Tutorial Section:

Introduction

App Overview

- [ ] NOTE Your app will have two "screens"

```
Welcome Screen
- Displays the App Name
- Displays a description of the App
- Has a button to get started

Decision Screen
- Poses a Question on something that needs to be decided on
- Shows a list of possible choices
- Allows user to enter new choices
- Allows user to edit existing choices
- Allows user to delete existing choices
- Has a button that will decide on a choice
```

Tutorial Part 1

Create New Android Project

- [ ] ACTION Create New Android Application

```
If you're using Eclipse, at the welcome screen, click on the button "New
Android Application"

If you're using Android Studio, at the welcome screen, click on "New Proj
ect..."
```

- [ ] ACTION Project Configurations

```
Application Name: Decision Maker
(This can be anything you want.)

Project Name: DecisionMaker
(Typically, this is one word and doesn't contain any spaces or underscore
s.)

Package Name: com.example.decisionmaker
(Typically, reverse domain syntax to prevent conflicts with others.  For
the purposes of this tutorial, com.example.decisionmaker is good, but whe
n you're pushing the the app store, this needs to be changed)

Minimum Required SDK: API 19: Android 4.4 (KitKat)

Target SDK: API 19: Android 4.4 (KitKat)

Compile With: API 19: Android 4.4 (KitKat)

Theme: Holo Light with Dark Action Bar

Click "Next"
```

- [ ] ACTION Project Configurations (continued...)

```
Uncheck "Create custom launcher icon"

Uncheck "Create activity"

Check "Create Project in Workspace" (this is checked by default)

Click "Finish"
```

Create Welcome Screen

- [ ] NOTE Steps necessary to create a new screen

```
In order to create a new screen, we need the following:
1) Create a Layout file (XML file that defines elements on the page)
2) Create an Activity file (Java class)
3) Let the app know there is a new "activity" by declaring it in the Andr
oidManifest.xml
```

- [ ] CODE Create the Layout file for the Welcome Activity

New File: res/layout/activity_welcome.xml

Select LinearLayout for the Root Element option when you create this new Android XM L Layout file. When you first open it, you may see the Graphical Layout view in Ecl ipse. Switch to the XML view by clicking on tab 'activity_welcome.xml' at the bott om of the screen. (Click on 'text' in Android Studio.)

This file will define the elements we want to display on the screen. We will later bind events to the button in our Java code.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
        >
    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Stop Wasting Time..."
            android:layout_gravity="center"
            android:padding="10dp"
            android:textSize="20dp"
            />
    <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Use this app for the times you and your friends
 take forever to decide on something"
            android:padding="10dp"
            android:gravity="center"
            />
    <Button
            android:id="@+id/get_started"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Get Started"
            android:padding="10dp"
            android:layout_gravity="center"
            />

</LinearLayout>
```

- [ ] CODE Create a Welcome Activity

New File: src/com/example/decisionmmaker/WelcomeActivity.java

```
package com.example.decisionmaker;

import android.app.Activity;

public class WelcomeActivity extends Activity {

}
```

- [ ] CODE Create an onCreate method inside WelcomeActivity class

File: src/com/example/decisionmmaker/WelcomeActivity.java
LIF: Anywhere inside the WelcomeActivity class.
Imports: import android.os.Bundle;

The onCreate() method is called by the Android framework when the Activity is first created. This is when the user goes to the Welcome screen.

setContentView() sets the view elements on the screen using the layout passed in.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_welcome);
}
```

- [ ] CODE Declare Activity in AndroidManifest.xml

File: /AndroidManifest.xml
LIF: Inside the "application" node

This lets the system know there is a Welcome Activity

Same as before, if you see a GUI editor, switch the the XML view by clicking on the bottom tab, AndroidManifest.xml

```
<activity android:name=".WelcomeActivity"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

- [ ] ACTION Launch Your Application

```
Run your application.  If your emulator isn't setup yet, follow the instr
uctions in section "Running on the emulator" from http://developer.androi
d.com/tools/building/building-eclipse.html

This will launch your emulator as well as your app.

You should be able to see the Welcome Screen you just created.

The "Get Started" Button currently does not do anything since we have not
 bound any events to it.
```

Create Choices Screen

- [ ] ACTION Try Creating Another Screen on Your Own...

```
Use the Welcome Screen steps as an example.

Create another screen that has text that poses a question of something th
at needs to be decided on.

Example questions:
- "What's for lunch?"
- "What tech should I learn next?"
- "Where should I go for vacation?"

Name the Activity file DecisionActivity.java.

Steps for how to do this are below...
```

- [ ] CODE Create a Layout File for the Decision Activity

```
New File: res/layout/activity_decision.xml

This file will define the text heading with the question you want to pose, and a pl
ace holder for the list of possible questions.
We will populate the list with our Java code later.
Note that we are reusing our header and text styles.
```

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

    <TextView
            android:text="What's for lunch today?"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />
    <ListView
            android:id="@android:id/list"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />
</LinearLayout>
```

☐ CODE Create a Decision Activity

New File: src/com/example/decisionmmaker/DecisionActivity.java

Instead of extending from Activity, inherit from ListActivity (a subclass of Activi
ty).
Read more: http://developer.android.com/reference/android/app/ListActivity.html

```
package com.example.decisionmaker;

import android.app.ListActivity;
import android.os.Bundle;

public class DecisionActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_decision);
    }
}
```

☐ CODE Declare the Activity in the AndroidManifest.xml

File: /AndroidManifest.xml
LIF: Inside "application" node

```
<activity android:name=".DecisionActivity" android:label="Decision Day"/>
```

- [ ] ACTION Compile Your App

```
There isn't anything we can preview yet, since there is no way to get to
this new "screen".

However, it would be good to make sure your app compiles before adding mo
re code.
```

Navigate from Welcome Screen to Decision Screen

- [ ] CODE Add functionality to the button on Welcome Activity to navigate to Decision Activity

```
File: src/com/example/decisionmaker/WelcomeActivity.java
LIF: Inside onCreate method

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
```

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_welcome);

        Button getStartedButton = (Button) findViewById(R.id.get_started)
;
        getStartedButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent myIntent = new Intent(WelcomeActivity.this, Decisi
onActivity.class);
                startActivity(myIntent);
            }
        });
    }
```

- [ ] ACTION Preview the "Get Started" Button

```
Launch your app and try clicking on the "Get Started" button.

You should be able to navigate from the Welcome Screen to the Decision Sc
reen.
```

Part 1 Exercises

- [ ] ACTION Complete Exercises

```
If we are still in the 1st Code Block, try applying what you have learned
 with the exercises below.

If we're in the 2nd Code Block, go ahead and jump to "Tutorial Part 2".
```

- [ ] ACTION Style Your App

```
Check out the Bonus section at the bottom of the tutorial called 'Style Y
our View'
```

- [ ] ACTION Create an Developed By "screen"

```
From the Welcome Screen, add some text that says who the app was develope
d by.

Make this text clickable.  When the text is clicked, it should take you t
o another screen that has "About the Developer" information.

To close the new screen and get back to the welcome screen, use the finis
h() method.
```

Tutorial Part 2

Create a Data Source to Save Choices

- [ ] NOTE Learn about Storage Options

```
We are going to allow the user possible choices to the question that was
posed.  It would be a bad user experience if the user can't save these ch
oices, so we need a way to save whatever was entered.

There are 5 different ways to save data in an Android app.

Learn about the different ways here:  http://developer.android.com/guide/
topics/data/data-storage.html

We will be using one of the simplest ways, Shared Preferences. This store
s primitive data in key-value pairs.
```

- [ ] NOTE Learn about files needed for storing data

```
We will be creating 2 Java classes (Choice and ChoiceDataStore) for savin
g data into SharedPreferences.

Choice represents a model of a choice object that is used to store inform
ation about a choice.

ChoiceDataStore is used to interface with the data store.  This class wil
l be used for reading, writing, and deleting choices from SharedPreferenc
es.
```

- [ ] ACTION Create new package for data related classes

```
Create a new package named 'com.example.decisionmaker.data'.

This is where we will store classes related to saving data into the Share
dPreferences.
```

- [ ] CODE Create a Choice Model

```
New File: src/com/example/decisionmaker/data/Choice.java

This is a POJO (plain ol' java object) that is used to save information into the Da
ta Source.

The id field will be used as the key for the SharedPreferences.
The name field will be the name of the choice (e.g. If your app decides where to go
 for lunch, a choice name will be "Giordano's".
```

```
package com.example.decisionmaker.data;

public class Choice {

    private String id;
    private String name;

}
```

- [ ] CODE Add accessors for the id and name fields of the Choice class

File: src/com/example/decisionmaker/data/Choice.java

We will need to read these fields in the app, so create some accessors.

```
public String getId() {
    return id;
}
public String getName() {
    return name;
}
```

- [ ] CODE Create a constructor for the Choice class that accepts id and name

File: src/com/example/decisionmaker/data/Choice.java

This will allow us to instantiate choices with values from our data source.

```
public Choice (String id, String name) {
        this.id = id;
        this.name = name;
    }
```

- [ ] CODE Create a Choice Data Source

New File: src/com/example/decisionmaker/data/ChoiceDataSource.java

This class will be responsible for creating, getting, updating, and removing items from the data storage.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.SortedSet;
import java.util.TreeSet;
```

```
package com.example.decisionmaker.data;

public class ChoiceDataSource {
    private static final String PREFKEY = "choices";
    private SharedPreferences choicePrefs;

    public ChoiceDataSource(Context context) {
        choicePrefs = context.getSharedPreferences(PREFKEY, Context.MODE_
PRIVATE);
    }
}
```

☐ CODE Create a method inside ChoiceDataStore that will retrieve all the values stored in SharedPreferences

File: src/com/example/decisionmaker/data/ChoiceDataSource.java

```
public List<Choice> findAll() {
        List<Choice> choiceList = new ArrayList<Choice>();

        for (Map.Entry<String, ?> entry : choicePrefs.getAll().entrySet()
) {
                String key = entry.getKey();
                String name = (String) entry.getValue();
                Choice choice = new Choice(key, name);
                choiceList.add(choice);
        }

        return choiceList;
     }
```

☐ CODE When the Decision Activity starts, set the dataSource for use throughout the Activity

File: /src/com/example/decisionmaker/DecisionActivity.java

```
import com.example.decisionmaker.data.ChoiceDataSource;
```

```
public class DecisionActivity extends ListActivity {

    private ChoiceDataSource dataSource;

        @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_decision);
        dataSource = new ChoiceDataSource(this);
    }

}
```

- [ ] CODE Create a custom method to display all Choices

File: /src/com/example/decisionmaker/DecisionActivity.java

The setListAdapter() method in this code populates the ListView of id "list" with t
he array that is passed to it.
We defined the ListView earlier in our layout.

```
import android.widget.ArrayAdapter;
```

```
private List<Choice> choices;

    private void displayAllChoices() {
        choices = dataSource.findAll();
        ArrayAdapter<Choice> adapter =
                new ArrayAdapter<Choice>(this, android.R.layout.simple_li
st_item_1, choices);
        setListAdapter(adapter);
    }
```

- [ ] CODE Call the method to display choices in the list view

File: /src/com/example/decisionmaker/DecisionActivity.java
LIF: Inside onCreate() method

```
displayAllChoices();
```

- [ ] ACTION Preview Your Changes

```
You should be able to launch your app with no errors.

You won't see any choices displayed yet because we haven't stored any dat
a in the SharedPreferences yet.
```

Save Choices into Data Store

- [ ] CODE Create a method in ChoiceDataSource for saving Choice information into SharedPreferences

File: /src/com/example/decisionmaker/data/ChoiceDataSource.java

```
public boolean save(Choice choice) {
        SharedPreferences.Editor editor = choicePrefs.edit();
        editor.putString(choice.getId(), choice.getName());
        editor.commit();
        return true;
    }
```

- [ ] CODE Create sample choices programatically when DecisionActivity starts

File: src/com/example/decisionmaker/DecisionActivity.java
LIF: Inside onCreate() method, just before displayAllChoices()

This is temporary code so that we can get some data to view in our app quickly. Later, we will remove this code and add functionality to create choices through the app interface.

```
Choice choice1 = new Choice("1", "Chipotle");
        dataSource.save(choice1);

        Choice choice2 = new Choice("2", "Jamba Juice");
        dataSource.save(choice2);
```

- [ ] ACTION Preview Your Changes

```
On the Decision Screen, you should be able to see the sample choices you
created.

But wait... instead of seeing the choice name you see the choice address
in memory.  We'll fix this in the next step.

Another note: Since we are temporarily creating sample data in the onCrea
te method, duplicate data will get created every time you start the app.
```

- [ ] CODE Override the toString method in your Choice model

File: src/com/example/decisionmaker/data/Choice.java

The ListAdapter by default calls the toString method for display. We need to overr
ide this method to return the name.

```
@Override
    public String toString() {
        return this.getName();
    }
```

- [ ] ACTION Preview Your App

```
You should now see the Choice names show up in the list on your Decision
screen.
```

- [ ] ACTION Delete sample data creation

```
Now that we know the list view is properly displaying Choices, delete the
 code where you created sample data.

This won't delete all the sample data already created, but at least it wi
ll stop more from being created.

In the next section, we will learn how to delete a choice from the app.
```

Delete Existing Choice

NOTE Delete Intro

```
In Android, when you hold down on an item, a context menu can pop up.  We
 will use this context menu and have a "Delete" menu option for deleting
choices from the list.
```

CODE Create a method that will remove Choice from the Data Store

File: src/com/example/decisionmaker/data/ChoiceDataSource.java

```
public boolean remove(Choice choice) {
        if (choicePrefs.contains(choice.getId())) {
            SharedPreferences.Editor editor = choicePrefs.edit();
            editor.remove(choice.getId());
            editor.commit();
        }
        return true;
    }
```

CODE Register a Context Menu for the list of Choices

File: src/com/example/decisionmaker/DecisionActivity.java

Let the Android framework know you want a context menu on your list view. The regi
sterForContextMenu is provided by the Android framework.

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_decision);
...


        registerForContextMenu(getListView());

...

    }
```

CODE Add Delete Element to the Context Menu

File: src/com/example/decisionmaker/DecisionActivity.java

The onCreateContextMenu is another method provided by the Android Framework, which is called whenever a context menu is created (which we did in the previous step).

```
    private int currentChoiceId;


    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu
.ContextMenuInfo menuInfo) {
        AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterCon
textMenuInfo) menuInfo;
        currentChoiceId = (int) info.id;
        menu.add(0, 9999, 0, "Delete Choice");
    }
```

- [ ] CODE React to Delete Element Being Clicked

File: src/com/example/decisionmaker/DecisionActivity.java

Lastly, onContextItemSelected is called when an item that is registered is clicked.

We registered "Delete Choice" menu item with an id of 9999, so we need to react acc ordingly to that item click.

```
@Override
    public boolean onContextItemSelected(MenuItem item) {
        if (item.getItemId() == 9999) {
            Choice choice = choices.get(currentChoiceId);
            dataSource.remove(choice);
            displayAllChoices();
        }

        return super.onContextItemSelected(item);
    }
```

- [ ] ACTION Preview Your App

```
You should be able to delete your choices by holding down on them and sel
ecting the "Delete Choice" item.
```

Create Choices from App

- [ ] CODE Add a button on your Decision Screen

```
New File: res/layout/activity_decision.xml

This button will eventually allow a user to add a new choice
If you have completed the bonus section to style your app, you can apply the button
 style and background here (optional).
```

```
<Button
            android:id="@+id/new_choice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Add Choice"
            android:layout_gravity="center"/>
```

- [ ] CODE React to the New Choice Button Being Clicked

```
File: src/com/example/decisionmaker/DecisionActivity.java

import android.view.View;
import android.widget.Button;

Since we haven't created a method for creating a new choice yet, for now, let's dis
play a message when the new button is clicked.
```

```
private void bindNewChoiceButton() {
        Button newChoiceButton = (Button) findViewById(R.id.new_choice);
        newChoiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "New Button Has B
een Clicked",
                Toast.LENGTH_SHORT).show();
            }
        });
    }
```

- [ ] CODE Call the code to Bind the New Choice Button

File: src/com/example/decisionmaker/DecisionActivity.java
LIF: Inside onCreate() method

```
bindNewChoiceButton();
```

- [ ] ACTION Preview Your App

```
You should be able to see a message pop up when you click the "Add Choice
" button on the Decision screen.

In the next step, let's swap out the code that shows the message for actu
ally creating a new choice.
```

- [ ] ACTION Create an overloaded constructor for Choice and a setter method for name

File: /src/com/example/decisionmaker/data/Choice.java

When we create a new choice from user input, we always need to generate a unique id
.
Let's encapsulate that behavior in a new constructor does not accept any parameters
.
(Don't delete the other constructor - we still need it in the ChoiceDataSource.)

We'll also create a mutator for setting the name instead of passing it as an argume
nt to the constructor.
This way we'll be able to share code between the create and update functionality in
 later steps.

import java.util.UUID;

```
public Choice() {
        id = UUID.randomUUID().toString();
    }

        public void setName(String name) {
                this.name = name;
        }
```

- [ ] CODE Create method for creating new choice

File: src/com/example/decisionmaker/DecisionActivity.java

To capture the name of the choice, we will use an Alert box where a user can enter text.

```
import android.app.AlertDialog;
import android.app.ListActivity;
import android.widget.EditText;
import android.content.DialogInterface;
```

```
private void createChoice() {
        final Choice choice = new Choice();

        AlertDialog.Builder alert = new AlertDialog.Builder(this);

        final EditText input = new EditText(this);
        input.setText(choice.getName());
        alert.setView(input);

        alert.show();
    }
```

- [ ] CODE Save new choice item

File: /src/com/example/decisionmaker/DecisionActivity.java

The above code just displays an alert box that allows users to enter text.
Let's edit the createChoice() method to set the user input on the new choice when t he 'save' button is clicked.
Then we have to display the updated list of choices.

```
private void createChoice() {
                final Choice choice = new Choice();

                AlertDialog.Builder alert = new AlertDialog.Builder(this)
;


                final EditText input = new EditText(this);
                input.setText(choice.getName());
                alert.setView(input);

                alert.setPositiveButton("Save", new DialogInterface.OnCli
ckListener() {
                        public void onClick(DialogInterface dialog, int w
hichButton) {
                                String updatedName = input.getText().toSt
ring();
                                choice.setName(updatedName);
                                dataSource.save(choice);
                                displayAllChoices();
                        }
                });

                alert.show();
        }
```

- [ ] CODE Call the createChoice() method

File: src/com/example/decisionmaker/DecisionActivity.java

In the onClick() event of the listener for newChoiceButton, replace the Toast code with the createChoice() method.

```
private void bindNewChoiceButton() {
        Button newChoiceButton = (Button) findViewById(R.id.new_choice);
        newChoiceButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                createChoice();
            }
        });
    }
```

- [ ] ACTION Preview Your App

```
You should now be able to click on the "Add Choice" button and add choice
s to your list.
```

- [ ] CODE Extract Logic that Prompts User For Choice Name and Saves it into the DataStore into its own method

File: /src/com/example/decisionmaker/DecisionActivity.java

Move all the code from the createChoice method except the first line to this new me
thod called updateChoiceFromInput().
We'll use this method again when we let the user update existing choices.

```
private void updateChoiceFromInput(final Choice choice) {
        AlertDialog.Builder alert = new AlertDialog.Builder(this);

        final EditText input = new EditText(this);
        input.setText(choice.getName());
        alert.setView(input);

        alert.setPositiveButton("Save", new DialogInterface.OnClickListen
er() {
            public void onClick(DialogInterface dialog, int whichButton)
{
                String updatedName = input.getText().toString();
                choice.setName(updatedName);
                dataSource.save(choice);
                displayAllChoices();
            }
        });

        alert.show();
    }
```

- [ ] CODE Update createChoice method to call the new updateChoiceFromInput() method

File: /src/com/example/decisionmaker/DecisionActivity.java

```
private void createChoice() {
        Choice choice = new Choice();
        updateChoiceFromInput(choice);
    }
```

- [ ] ACTION Preview Your App

```
We just did a refactoring, so the behavior of the app should be the same.

Test out the app and make sure nothing has changed.
```

Part 2 Exercises

- [ ] ACTION Complete Exercises

```
If we are still in the 2nd Code Block, try applying what you have learned
 with the exercises below or the exercises from Part 1 Exercises.

If we're in the 3rd Code Block, go ahead and jump to "Tutorial Part 3".
```

- [ ] ACTION Allow Add Choice to be Cancelled

```
Right now, if the user clicks on "Add Choice", there is no Cancel button.

Add a cancel option for the popup.

Resources:
Android documentation on AlertDialog.Builder - http://developer.android.c
om/reference/android/app/AlertDialog.Builder.html
User created example - http://www.mkyong.com/android/android-alert-dialog
-example/
```

- [ ] ACTION Validate User Input

```
Right now, the user can enter anything they want (including nothing).

It would be good to validate the input before saving.
```

Tutorial Part 3

Update Existing Choice

- [ ] CODE React to a List Item Being Clicked

File: /src/com/example/decisionmaker/DecisionActivity.java

A method called onListItemClick() is a called by the Android Framework when a List Item is clicked.
Let's override it to display the name of the choice that was clicked.

```
@Override
        protected void onListItemClick(ListView l, View v, int position,
long id) {
                Choice choice = choices.get(position);
                String messageForDisplay = choice.getName();
                Toast.makeText(getApplicationContext(), messageForDisplay
,
                                Toast.LENGTH_SHORT).show();
        }
```

- [ ] ACTION Preview Your Changes

```
Launch your application.

Click on a Choice.

You should see a message pop up with the same name as the choice you clic
ked on.

Keep in mind that Toasts will display for the entire duration "LENGTH_SHO
RT" before displaying the next Toast, so you may see messages delayed if
you're clicking very fast.

If that works, that means you have successfully bound an event to a list
item being clicked.  Let's replace the Toast with something useful.
```

- [ ] CODE Change listener to Edit Choice Name

File: /src/com/example/decisionmaker/DecisionActivity.java

Now we will replace that code and reuse the updateChoiceFromInput() method we creat
ed earlier.

```
@Override
    protected void onListItemClick(ListView l, View v, int position, long
 id) {
        Choice choice = choices.get(position);
        updateChoiceFromInput(choice);
    }
```

- [ ] ACTION Preview Your App

```
Click on a choice item on the list.

You should be prompted to edit the existing choice.

If you edit the name and hit save, the changes should be reflected in the
 list view.
```

Pick Choice

- [ ] CODE Create a Method that Will Pick A Random Choice

File: src/com/example/decisionmaker/DecisionActivity.java

```
private void decideOnChoices() {
        int randPosition = new Random().nextInt(choices.size());
        Choice selectedChoice = choices.get(randPosition);

        Toast.makeText(getApplicationContext(),
                "Decision has been made!\nGo with " + selectedChoice.getN
ame(),
                Toast.LENGTH_SHORT).show();
    }
```

- [ ] CODE Create a button that will be used for deciding on a choice

File: res/layout/activity_decision.xml

If you have completed the bonus section to style your app, you can apply the button style and background here (optional).

```
<Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Decide!"
            android:id="@+id/decide"
            android:layout_gravity="center"/>
```

- [ ] CODE React to Decide Button Being Clicked

File: src/com/example/decisionmaker/DecisionActivity.java

```
private void bindDecideButton() {
        Button decideButton = (Button) findViewById(R.id.decide);
        decideButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                decideOnChoices();
            }
        });
    }
```

- [ ] CODE Call the function that binds the decide button

File: src/com/example/decisionmaker/DecisionActivity.java
LIF: Inside onCreate() method

```
bindDecideButton();
```

- [ ] ACTION Preview Your App

```
Try clicking on the Decide button in the Decision Screen.

You should see an pop up message saying which choice you should go with!
```

Exit Choices Screen

- [ ] ACTION Do this one yourself!

```
Let's finish our app by adding a button to close the decision screen and
take us back to the welcome screen.
The way to exit an activity is to call the finish() method.

Instead of providing the code for this one, you should be able to figure
it out based on what you learned so far.
Remember you need to do the following steps:
Add another button to the decision layout.
Bind a listener to the new button.
Override the onClick() method of the listener to call the finish() method
.
```

- [ ] ACTION Preview your app

```
Launch the app again and try out the new button.

If you have trouble, look back through your previous code, check the log
output, or try debugging.
Remember your coach and other attendees are here to help if you get stuck
.
```

Celebrate!!!

- [ ] NOTE Great job! You did it!

```
Show off with your neighbor. If you have time, work on the bonus sections
 below.
```

Bonus

Bonus: Style Your View

- [ ] CODE Define header, text, and button_text styles in a separate resource file

File: res/values/styles.xml
LIF: Inside resources node

We will define styles in this file that can be applied to the text and buttons in o
ur views.
You will not see a change in style until they are applied in a few steps.
This removes the clutter from our xml layout files and allows us to reuse styling c
ode.

Note that we are putting our styles in the main values directory.
If you want styles that are specific to an API version, you can put them in the sty
les.xml files that are inside the values-vXX directories.

```
<style name="header">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#ffffff</item>
    <item name="android:paddingTop">10dp</item>
    <item name="android:paddingBottom">15dp</item>
    <item name="android:paddingLeft">15dp</item>
    <item name="android:textSize">28sp</item>
    <item name="android:shadowColor">#000000</item>
    <item name="android:shadowDx">1</item>
    <item name="android:shadowDy">1</item>
    <item name="android:shadowRadius">2</item>
</style>

<style name="text">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#031c4c</item>
    <item name="android:padding">15dp</item>
</style>

<style name="button_text">
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textColor">#ffffff</item>
    <item name="android:layout_marginLeft">15dp</item>
    <item name="android:layout_marginBottom">15dp</item>
    <item name="android:textSize">16sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:shadowColor">#000000</item>
    <item name="android:shadowDx">1</item>
    <item name="android:shadowDy">1</item>
    <item name="android:shadowRadius">2</item>
    <item name="android:width">100dp</item>
</style>
```

CODE Create a custom button shape

New File: res/drawable/button.xml

Select 'selector' for the Root Element option when you create this new Android XML File.

This file defines styles for two states for the button: one for the default state a nd one for when the button is pressed.
It's got rounded corners and a gradient background.

```
<?xml version="1.0" encoding="utf-8" ?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true" >
    <shape>
      <solid
          android:color="#ff9933" />
      <stroke
          android:width="1dp"
          android:color="#ff6600" />
      <corners
          android:radius="6dp" />
      <padding
          android:left="10dp"
          android:top="10dp"
          android:right="10dp"
          android:bottom="10dp" />
    </shape>
  </item>
  <item>
    <shape>
      <gradient
          android:startColor="#ff9933"
          android:endColor="#ff6600"
          android:angle="270" />
      <stroke
          android:width="1dp"
          android:color="#ff6600" />
      <corners
          android:radius="6dp" />
      <padding
          android:left="10dp"
          android:top="10dp"
          android:right="10dp"
          android:bottom="10dp" />
    </shape>
  </item>
</selector>
```

- [ ] CODE Create the header background

New File: res/drawable/header_background.xml

Select 'layer-list' for the Root Element option when you create this new Android XML File.

This file defines styles for the background of our header text.
Note that we create a layer-list, which will allow us to add multiple shapes on top of each other.
The second two shapes are offset so that it looks like bottom borders.

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape>
            <solid android:color="#544266" />
        </shape>
    </item>
    <item android:top="54dp">
        <shape>
            <solid android:color="#ff9933" />
        </shape>
    </item>
    <item android:top="55dp">
        <shape>
            <solid android:color="#2c0b4d" />
        </shape>
    </item>
</layer-list>
```

- [ ] CODE Apply the text, header, and button styles to the elements in the view

File: res/layout/activity_welcome.xml
LIF: TextView and Button elements inside the LinearLayout element

Update the TextView and Button elements to look like the code below.
Notice that we remove all styling attributes and replace them with references to our new style and drawable resources.
Make sure you don't delete the text attributes.

```
<TextView
            android:text="Stop Wasting Time..."
            android:background="@drawable/header_background"
            style="@style/header"
            />
    <TextView
            android:text="Use this app for the times you and your friends
 take forever to decide on something"
            style="@style/text"
            />
    <Button
            android:id="@+id/get_started"
            android:text="Get Started"
            android:background="@drawable/button"
            style="@style/button_text"
            />
```

- [ ] ACTION Preview the new styles

```
Launch your application to see the new styles.
Note that you will need to clean your project or make a small change to a
 Java file for the xml changes to be picked up when launching the emulato
r.
Change the colors or other attributes as desired and preview again.
```

- [ ] ACTION At the end of the tutorial, apply the same styles to the activity view.

```
Once you have completed the tutorial, you can use the styles created abov
e to the activity_decision.xml to style the second screen.
```

- [ ] CODE After you have completed the final section of the tutorial, try moving the two buttons next to each other

```
File: res/layout/activity_decision.xml
LIF: Inside the existing LinearLayout element

In order to place the two buttons on the same line, we'll need to wrap them in a ne
w LinearLayout element.
Note that you should now have two LinearLayout elements in the file - don't delete
the existing one.
The new LinearLayout has a horizontal orientation, while our first LinearLayout in
this view has a vertical orientation.
```

```
<LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        ...(your two existing buttons go here)...
</LinearLayout>
```

- [ ] ACTION Preview Your Style Updates

```
Launch your application to see the new styles.
Note that you will need to clean your project or make a small change to a
 Java file for the xml changes to be picked up when launching the emulato
r.
Change the colors or other attributes as desired and preview again.
```

Bonus: Extract Hardcoded Text To Resources

Having text scattered throughout the layout xml can be hard to maintain, and limit flexibility. Let's move the text into a resource file.

- [ ] CODE Create a string value for the welcome activity heading

```
File: res/values/strings.xml
```

```
<resources>
    <string name="app_name">DecisionMaker</string>
    <string name="app_heading">Stop Wasting Time...</string>
</resources>
```

- [ ] CODE Replace hardcoded layout text with reference to string resource

```
File: res/layout/activity_welcome.xml
LIF: Inside the first TextView element

Replace the value from android:text attribute with a reference to our new string re
source
```

```
<TextView
    .....
    android:text="@string/app_heading"
    .....
    />
```

- [ ] ACTION Preview Your App

```
Launch your app and make sure the heading on the welcome screen still dis
plays correctly.
Try changing the text in the resource file and preview again so that you
can tell the text is coming from the new resource.
```

- [ ] CODE Repeat for all hardcoded text in your layout files. You should end up with a strings.xml file that looks like this:

```
<resources>
    <string name="app_name">DecisionMaker</string>
    <string name="app_heading">Stop Wasting Time...</string>
    <string name="app_description">Use this app for the times you and you
r friends take forever to decide on something</string>
        <string name="get_started">GET STARTED</string>
        <string name="whats_for_lunch">What\'s for lunch today?</string>
        <string name="add_choice">ADD CHOICE</string>
        <string name="decide">DECIDE!</string>
</resources>
```

- [ ] CODE Be sure to replace all instances of android:text in your welcome and decision activity layout files with references to your string resources.

```
File: res/layout/activity_welcome.xml, res/layout/activity_decision.xml

The following lines will go in the TextView and Button elements of the welcome acti
vity view.
Also update the decision view with the appropriate string values.
```

```
<TextView
    ....
    android:text="@string/app_description"
    ..../>

<Button
    ....
    android:text="@string/get_started"
    ..../>
```

- [ ] ACTION Learn about internationalization.

```
By storing your application's text in resource files, you can provide tra
nslations for different languages.
Read more at http://developer.android.com/guide/topics/resources/localiza
tion.html
```