ATLS 4120: Mobile Application Development
Week 12: Activities and Intents

Android 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.
- These categories should never overlap.

Activities

https://developer.android.com/guide/components/activities/intro-activities.html
https://developer.android.com/reference/android/app/Activity.html

- 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
- There is typically a 1:1 ration for activity and layout

To start an activity you need to define an intent and then use the startActivity(Intent) method to start a new activity.

Intents

https://developer.android.com/guide/components/intents-filters.html
https://developer.android.com/reference/android/content/Intent.html
Intents request an action such as starting a new activity.

- Provides the binding between two activities

You build an Intent with two key pieces of information

- Action – what action should be performed
- Data – what data is involved

There are two types of intents

- An explicit intent tells the app to start a specific activity
  - Provide the class name
- An implicit intent declares what type of action you want to perform which allows a component from another app to handle it
  - the app will start any activity that can handle the action specified
  - Android uses intent resolution to see what activities can handle the intent
  - An intent filter specifies what types and categories of intents each component can receive
  - Each activity has intent filters defined in the AndroidManifest.xml file
  - Android compares the information given in the intent with the information given in the intent filters specified in the AndroidManifest.xml file.
  - An intent filter must include a category of android.intent.category.DEFAULT in order to receive implicit intents
  - If an activity has no intent filter, or it doesn't include a category name of android.intent.category.DEFAULT, it means that the activity can't be started with an implicit intent. It can only be started with an explicit intent using the fully qualified component name.

- Android first considers intent filters that include a category of android.intent.category.DEFAULT
- Android then matches the action and mime type with the intent filters
- If just one activity can handle the intent, that activity is chosen
- If there is more than one activity that can handle the intent, the user is presented a list to chose from
- Android tells the activity to start even though it's in another app and passes it the intent

Data
https://developer.android.com/training/basics/firstapp/starting-activity.html
You can add extra information to your intent to pass data to the new activity using the putExtra(String, value) method

- The putExtra(String, value) method is overloaded so you can pass many possible types
- Call putExtra(String, value) as many times as needed for the data you're passing
- Each call to putExtra(String, value) is setting up a key/value pair and you will use that key to access that value in the intent you're starting.

When a new activity starts it needs to receive any data passed to it in the intent using the getExtra() methods.

Using intents Android knows the sequence in which activities are started. This means that when you click on the Back button on your device, Android knows exactly where to take you back to.

Events
https://developer.android.com/guide/topics/ui/ui-events.html
Android enables you to easily respond to common events through event listeners on the View class.
An event listener is an interface in the View class that contains a single callback method. These methods will be called automatically by the Android framework when the View to which the listener has been registered is triggered by user interaction with the UI control.

- Implement the listener
- Implement the callback method
- Assign the interface to a UI control

Coffee
Create a new project called Coffee
Minimum SDK: API 21
Empty Activity template
Activity name: FindCoffeeActivity
Check Generate Layout File
Layout Name: activity_find_coffee
Don't check Backwards Compatibility

User Interface
Use the textview provided as a heading that says "Coffee Shop Finder".
Change text appearance to be Material.Title
Also change its text property to use a string resource.
Make sure AutoConnect is on.
Add a spinner and a textview above it that will describe the spinner with text "Choose your crowd"
Add a button with the text "Find Coffee".

Add an imageView below the button. Copy an image into the res/drawable folder and use that as the src in the xml. You should also add a contentDescription using a string resource.
The spinner, button, and image all should have an id since we'll be referring to these from our code.
If you add these from top to bottom they will be added to the layout below each other.
Make sure you use string resources for all text.

Strings.xml

```
<resources>
   <string name="app_name">Coffee</string>
   <string name="title">Coffee Shop Finder</string>
   <string name="coffee_type">Choose your crowd</string>
   <string name="button">Find Coffee</string>
   <string-array name="crowd">
      <item>popluar</item>
      <item>cycling</item>
      <item>hipster</item>
      <item>tea</item>
      <item>hippie</item>
   </string-array>
   <string name="coffee_image">coffee cup</string>
</resources>
```

Java class

We're going to create a custom Java class for coffee shop info.
In the app/java folder select the coffee folder (not androidTest or test)
File | New | Java class (or right click)
Select …/app/src/main/java
Name: CoffeeShop
Kind: Class

We're going to create a CoffeeShop class with two data members to store the coffee shop name and URL.
Create getter and setter methods for both. We'll also create a private utility method that chooses the coffee shop so both setter methods can call this method instead of duplicating the functionality.

```
public class CoffeeShop {
   private String coffeeShop;
   private String coffeeShopURL;

private void setCoffeeInfo(Integer coffeeCrowd){
   switch (coffeeCrowd){
      case 0: //popular
         coffeeShop="Starbucks";
         coffeeShopURL="https://www.starbucks.com";
         break;
      case 1: //cycling
         coffeeShop="Amante";
         coffeeShopURL="http://www.amantecoffee.com/";
         break;
```

```
        case 2: //hipster
            coffeeShop="Ozo";
            coffeeShopURL="https://ozocoffee.com";
            break;
        case 3: //tea
            coffeeShop="Pekoe";
            coffeeShopURL="http://www.pekoesiphouse.com";
            break;
        case 4: //hippie
            coffeeShop="Trident";
            coffeeShopURL="http://www.tridentcafe.com";
            break;
        default:
            coffeeShop="none";
            coffeeShopURL="https://www.google.com/search?q=boulder+coffee+shops&ie=utf-
8&oe=utf-8";
    }
}

    public void setCoffeeShop(Integer coffeeCrowd){
        setCoffeeInfo(coffeeCrowd);
    }

    public void setCoffeeShopURL(Integer coffeeCrowd){
        setCoffeeInfo(coffeeCrowd);
    }

    public String getCoffeeShop(){
        return coffeeShop;
    }

    public String getCoffeeShopURL(){
        return coffeeShopURL;
    }
}
```

FindCoffeeActivity.java

In FindCoffeeActivity.java we first need to create an object of our new CoffeeShop class and then implement a findCoffee() method.

```
private CoffeeShop myCoffeeShop = new CoffeeShop();

private void findCoffee(View view){
//get spinner
Spinner crowdSpinner = findViewById(R.id.spinner);
//get spinner item array postion
Integer crowd = crowdSpinner.getSelectedItemPosition();
//set the coffee shop
myCoffeeShop.setCoffeeShop(crowd);
//get suggested coffee shop
```

```
String suggestedCoffeeShop = myCoffeeShop.getCoffeeShop();
//get URL of suggested coffee shop
String suggestedCoffeeShopURL = myCoffeeShop.getCoffeeShopURL();
Log.i("shop", suggestedCoffeeShop);
Log.i("url", suggestedCoffeeShopURL);
}
```

For now we're just logging the results for testing.

Button

We could add the onClick event to our button like we've been doing, but we're going to use an event listener instead.

You can only use the android:onClick attribute in activity layouts for buttons, or any views that are subclasses of Button such as CheckBoxes and RadioButtons. So it's good to understand how to set up event listeners.

https://developer.android.com/guide/topics/ui/ui-events

An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

The onClick() callback method is called from the View.OnClickListener event listener. This event is fired when the user either touches the item.

To define the method and handle the event, implement the nested interface in your Activity. Then, pass an instance of your implementation to View.setOnClickListener() method.

Update the onCreate() method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_find_coffee);
    //get button
Button button = findViewById(R.id.button);
    //create listener
    View.OnClickListener onclick = new View.OnClickListener(){
        public void onClick(View view){
            findCoffee(view);
        }
    };
    //add listener to the button
    button.setOnClickListener(onclick);
}
```

Now run your project and look in the Logcat to see if it's working.

New Activity

File | New |Activity

Either Gallery or Empty Activity

Activity name: ReceiveCoffeeActivity

Check Generate Layout File

Layout Name: activity_receive_coffee
Do not check Launcher Activity as this is not the launcher activity for our app.
This creates a new layout xml file and java file for our new activity.
It also updates the AndroidManifest.xml file with a new activity.

Our layout will simply consist of a textView where we'll suggest a coffee shop.
Add a text view and give its text a string resource and a descriptive id of coffeeShopTextView
(you can remove the text once your layout is set)
<**string name="suggested_coffee"**>This is your suggested coffee shop</**string**>

Explicit Intent
Now let's get the button in the first activity to call the second activity.
In FindCoffeeActivity.java update findCoffee() to create and start an intent.

```
//create an intent
Intent intent = new Intent(this, ReceiveCoffeeActivity.class);
```

[note: AS is going to add "packageContext:" before this)

Before we start the intent, let's pass data to it.
(press option + return on Mac (Alt + Enter on Windows) to import missing classes.)

Passing Data
Now let's pass the coffee shop name and URL to the second activity.
In FindCoffeeActivity.java BEFORE you start the new activity, add the data to the intent.

```
//pass data
intent.putExtra("coffeeShopName", suggestedCoffeeShop);
intent.putExtra("coffeeShopURL", suggestedCoffeeShopURL);
```

[note: AS is going to add "name:" before your String key)

```
//start the intent
startActivity(intent);
```

Receiving Data
Now let's update ReceiveCoffeeActivity.java to get the data sent in the intent. Create two private strings in the class

```
private String coffeeShop;
private String coffeeShopURL;
```

The onCreate() method is called as soon as the activity is created so that's where we'll get the intent.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_receive_coffee);

    //get intent
```

```
    Intent intent = getIntent();
    coffeeShop = intent.getStringExtra("coffeeShopName");
    coffeeShopURL = intent.getStringExtra("coffeeShopURL");
    Log.i("shop received", coffeeShop);
    Log.i("url received", coffeeShopURL);

    //update text view
    TextView messageView = findViewById(R.id.coffeeShopTextView);
    messageView.setText("You should check out " + coffeeShop);
}
```

Make sure that the string you're using in getStringExtra() is EXACTLY the same as the string you used in putExtra() in FindCoffeeActivity.java.

Implicit Intent

Let's add a button in our second activity to open up the coffee shop's web site in an external app.
Add an image button in the bottom right corner of the layout.
Add an image resource into the drawable folder and use that for the button's src.
By default image buttons have a background. You can change its color or make it transparent.

```
android:background="@android:color/transparent"
```

In ReceiveCoffeeActivity.java implement a method to load a web page.

```
private void loadWebSite(View view){
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse(coffeeShopURL));
    startActivity(intent);
}
```

Uri.parse() parses the string passed to it and creates a Uri object. A Uri object is an immutable reference to a resource or data.

Update onCreate() to set up the event listener and add it to the image button.

```
ImageButton imageButton = findViewById(R.id.imageButton);
//create listener
View.OnClickListener onclick = new View.OnClickListener(){
    public void onClick(View view){
        loadWebSite(view);
    }
};
//add listener to the button
imageButton.setOnClickListener(onclick);
```

Run and use the back arrow to see how it takes you to the last activity you were in.