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

- 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 ratio 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/training/basics/firstapp/starting-activity.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
    - Usually used to start an activity in your own app
    - Provide the class name when creating the Intent
    - Call startActivity(Intent)
- An implicit intent does not name a specific activity. Instead you declare what type of action you want to perform which allows a component from another app to handle it
    - Usually used to start an activity in a different app
    - Provide the type of action when creating the Intent
    - Call startActivity(Intent)
    - Android uses intent resolution to see what apps can handle the intent
        - Compares the information in the intent to the intent filters of other apps
            - Each activity has intent filters defined in its app's AndroidManifest.xml file
            - An intent filter specifies what types and categories of intents each component can receive
            - 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 there is only one match between the intent and intent filters then Android starts that activity and passes it the Intent object
  - If there is more than one match between the intent and intent filters, Android presents the user a list of apps to pick from

Creating Intents
https://developer.android.com/reference/android/content/Intent.html
An Intent object includes all the information needed to determine what activity to start

- Context for the intent which is usually "this"
- Class name for an explicit intent
- Action for an implicit intent
- Data
  - A URI object references the data, usually dictated by the intent's action
  - set the data on the intent using the URI object
- Category description of the intent
  - Most intents don't need a category
- Extras enable intents to carry additional information
  - 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.

Receiving Intents
When a new activity starts it has access to the Intent passed to it

- Can access any data passed 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.

Returning Data
If you want the second activity to be able to send data back to the first activity:
In the first activity:

- Call startActivityForResult (intent, requestCode) instead of startActivity(intent)
  - requestCode is an integer that will be returned from the second activity when it finishes
- Implement onActivityResult(requestCode, resultCode, data) which gets called when the second activity finishes and control returns to the first activity
  - requestCode is the integer passed to the second activity
  - resultCode is a code sent back from the second activity

    - data is the intent

In the second activity to send data back

- create an intent and use the put methods to add data to it
- call setResult(resultCode)
    - resultCode should be Activity.RESULT_OK or Activity.RESULT_CANCELLED
- call finish() to exit the activity
    - you can use onBackPressed() to implement this when the user taps the back button

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

Taco Intents

We're going to add on to our Taco app to include a model class and implement both explicit and implicit intents.

Kotlin class

We're going to create a custom Kotlin class for taco shop info.
In the app/java folder select the taco folder (not androidTest or test)
File | New | Kotlin file/class (or right click)
Name: TacoShop
Kind: Class

We're going to create a TacoShop class with two data members to store the taco shop name and URL.

Since this class will be handling data I've made it a data class so the Kotlin compiler will automatically override the toString(), equals(), hashCode(), and copy() methods from the Any class and provide implementations for this data class.

The class also has methods that sets the taco shop name and location based on the location provided. Two of the methods are private because they are only called within the class an not from outside it.

```
data class TacoShop(var name: String="", var url: String=""){
    fun suggestTacoShop(position:Int){
        setTacoShopName(position)
        setTacoShopUrl(position)
    }

    private fun setTacoShopName(position:Int){
        when (position) {
            0 -> name="Illegal Petes"
            1 -> name="Chipotle "
            2 -> name="Bartaco"
```

```
            else -> name="taco shop of your choice"
        }
    }

    private fun setTacoShopUrl(position:Int){
        when (position) {
            0-> url="https://www.illegalpetes.com"
            1 -> url="https://locations.chipotle.com/co/boulder/1650-
28th-st"
            2 -> url="https://bartaco.com/"
            else ->
url="https://www.google.com/search?q=boulder+taco+shop"
        }
    }
}
```

It's not great that my code is dependent on the order of the items in the spinner. My other option would be to use the spinner item's text but then it would be dependent on the text so if it changed, or was translated to another language, the logic would break. Neither is ideal.

Button

In the layout file let's add another button that we'll use to start a new activity that will suggest a place to get tacos.
If you have a ScrollView you might need to temporarily change layout_height to match_parent so you have room to add a button.
Make the button's id locationButton.
Add a string resource called locationButton with the value Find tacos for the text.
Add needed constraints.

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. Interfaces in Kotlin are abstract classes, similar to protocols in Swift. These methods will be called by the Android framework when the event listener assigned to the View is fired by user interaction with the item in the UI.

TextView

Add a TextView below the button with the id reviewTextView, we'll be using it later.
Add needed constraints.
Remove the default text.

MainActivity.kt

We need to create an object of our new TacoShop class.

```
private var myTacoShop = TacoShop();
```

We'll also create a class variable to hold the selected position of the spinner.

```
private var selectedLocationPosition = 0
```

Now let's create a onClick listener for the button.
The onCreate() method is a good place to set up event listeners.

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //event listener
        locationButton.setOnClickListener {
            selectedLocationPosition = spinner.selectedItemPosition
            myTacoShop.suggestTacoShop(selectedLocationPosition)
            Log.i("shop suggested", myTacoShop.name);
            Log.i("url suggested", myTacoShop.url);
        }
    }
```

For now we're just logging the results for testing. Now run your project and look in the Logcat to see if it's working.

New Activity
File | New |Activity
Either Gallery or Basic Activity
Activity name: TacoActivity
Layout Name: activity_taco
Title: TacoActivity
Do not check Launcher Activity as this is not the launcher activity for our app.
Select your current package name (com.example.taco)
Source language: Kotlin
This creates a new layout xml file and Kotlin file for our new activity.
It also updates the AndroidManifest.xml file with a new activity.

The basic template includes a little more than the empty template.
activity_taco.xml has a Coordinator Layout as its root element.
Coordinator layouts can contain other elements and controls the animation and transactions of various child elements with one another.
The layout also has an AppBarLayout which includes a Toolbar.
Includes the content_taco xml layout where we'll add a textView for our taco shop suggestion.
Includes a floating action button which currently implements a snackbar.
Run it to see the snack bar in action and how the coordinator layout handles the animated moves of the rest of the view when the snack bar is visible.

In content_taco.xml add a textView where we'll suggest a taco shop.
Add a text view and make sure the text view has an id. I used the descriptive id of tacoShopTextView
I changed the textAppearance to TextAppearance.AppCompat.Medium.
Add missing constraints and then remove the default text.

Also add a multiline EditText with the id feedbackEditText.
Create a string resource for the hint attribute called feedbackHint with the value Feedback.
Add missing constraints.

In activity_coffee.xml change the fab to an info button by changing srcCompat attribute to @android:drawable/ic_dialog_info. You can do this by selecting the fab and in the srcCompat attribute click on the image or the bar to the right of the value and select ic_dialog_info.

Explicit Intent
Now let's get the button in the main activity to call TacoActivity.
In MainActivity.kt update the listener we set up in onCreate() to create and start an intent.

```
//create intent
val intent = Intent(this, TacoActivity::class.java)
intent.putExtra("tacoShopName", myTacoShop.name)
intent.putExtra("tacoShopURL", myTacoShop.url)

startActivity(intent)
```

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

The first parameter of the Intent constructor is a Context object. We pass "this" which refers to our MainActivity class because the activity class is a subclass of Context, so we can use that. Alternatively, we also could have used getApplicationContext(); an application context would have been accepted just as well.
The second parameter is the Class we want to pass the intent to. It requires a Java class reference so we use the :: to create a class reference and the .java property to obtain a Java class reference for our TacoActivity class. (A Kotlin class reference is not the same as a Java class reference.)

And we add data to the intent to pass the taco shop name and URL to the second activity.
Then we start the activity.
[note: AS is going to add "name:" before your String key)

The code to create the intent is equivalent to:

```
val intent = Intent(this, TacoActivity::class.java).apply{
    putExtra("tacoShopName", myTacoShop.name);
    putExtra("tacoShopURL", myTacoShop.url);
}
```

Receiving Data
When you go into TacoActivity.kt you'll see that the floating action button has been set up as well as an onClickListener for it. We'll use that later.

Now let's get the data sent in the intent. Create two private strings in the class

```
private var tacoShopName : String? = null
private var tacoShopUrl : String? = null
```

The onCreate() method is called as soon as the activity is created so that's where we'll access the intent, get the data from it, and use it in our text view. Add to onCreate()

```
tacoShopName = intent.getStringExtra("tacoShopName")
tacoShopUrl = intent.getStringExtra("tacoShopURL")

tacoShopName?.let { Log.i("shop received", it) };
tacoShopUrl?.let { Log.i("url received", it) };

tacoShopName?.let {tacoShopTextView.text = "You should get tacos at
$tacoShopName "}
```

tacoShopTextView will likely give you an error if it's the first time you're referencing an id from your layout. Hover over the error to add the import.

```
import kotlinx.android.synthetic.main.content_taco.*
```

We have more log statements here for testing.
Make sure that the string you're using in getStringExtra() is EXACTLY the same as the string you used in putExtra() in MainActivity.kt.

Returning Data

If you start an activity that is going to return data you need to use startActivityForResult() instead of startActivity(). Along with the intent it also takes a request code as the second parameter. This is an Int and it doesn't matter what it is, but you'll use it again later so we'll make it a constant.
Let's update MainActivity to start the activity but in a way that it can receive data back from the TacoActivity class.
Add a class member for the request code. It can be any integer. We use a constant for this as we'll reference it multiple times in the class.

```
private val REQUEST_CODE = 1
```

Replace startActivity() with startActivityForResult()

```
startActivityForResult(intent, REQUEST_CODE)
```

In TacoActivity.kt there's no button we want to call finish() and end the Activity, but when the user taps the back button we want to send the data from the feedback EditText back to MainActivity and show it in the review TextView. We can do this by overriding onBackPressed() as that gets called when the user taps the back button.

```
override fun onBackPressed() {
    val data = Intent()
    data.putExtra("feedback", feedbackEditText.text.toString())
    setResult(Activity.RESULT_OK, data) //must be set before
super.onBackPressed()
    super.onBackPressed()
    finish()
}
```

Back in MainActivity we want to handle the data sent back from TacoActivity.

```
override fun onActivityResult(requestCode: Int, resultCode: Int,
data: Intent?) {
```

```
    super.onActivityResult(requestCode, resultCode, data)
    if((requestCode == REQUEST_CODE) && (resultCode ==
Activity.RESULT_OK)) {

reviewTextView.setText(data?.let{data.getStringExtra("feedback")})
    }
}
```

Implicit Intent

In TacoActivity let's use the FAB to open up the taco shop's web site in an external app.

In TacoActivity.kt implement a method to start an implicit intent to load a web page.

```
    private fun loadWebSite(){
        //create intent

        var intent = Intent()
        intent.action = Intent.ACTION_VIEW
        intent.data = tacoShopUrl?.let{Uri.parse(tacoShopUrl)}

        // start activity
        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.

The code to create the intent is equivalent to:

```
var intent = Intent().apply {
    action = Intent.ACTION_VIEW;
    data = tacoShopUrl?.let{Uri.parse(tacoShopUrl)};
}
```

This can also be shortened to

```
val intent = Intent(Intent.ACTION_VIEW,
tacoShopUrl?.let{Uri.parse(tacoShopUrl)})
```

It's possible that a device won't have any apps that handle the implicit intent you send to startActivity(). Or, an app may be inaccessible because of profile restrictions or settings put into place by an administrator. If that happens, the call would fail and your app would crash. To verify that an activity will receive the intent, we call resolveActivity() on the Intent object. If the result is non-null, there is at least one app that can handle the intent and it's safe to call startActivity(). If the result is null, we don't start the intent.

In onCreate() update the FAB's OnClickListener by replacing the snackbar with a call to our loadWebSite() function.

```
findViewById<FloatingActionButton>(R.id.fab).setOnClickListener {
    loadWebSite()
}
```

If using the Kotlin extension plug-in this would be shortened to:

```
fab.setOnClickListener {
    loadWebSite()
}
```

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