Intents and Activities

An activity represents a single task or thing that the user is currently doing. It can have one view, or be split into several fragments in case of a larger display (we will cover fragments later on). Think of it as something between an application and a window.
https://developer.android.com/reference/android/app/Activity.html

Activities have to respond to various OS events or states. ex. user chooses a different app; the system is running low on memory, battery etc. resulting in the “Android Activity Lifecycle”
Make sure you are familiar with the following:
https://wiki.cdot.senecacollege.ca/wiki/File:Android_activity_lifecycle.png

Recall:

Application Manifest: XML based file that the application outlines the activities, services, broadcast receivers, data providers and permissions that make up the complete application. This file used to also hold some information which has now migrated to build.gradle such as the minimum API level.
R.java: A dynamically-generated static class named R containing references to resources (res folder). Once you change any files (ex. XML) in the res folder your R object will be updated (recompiled) automatically. Basically a “Bridge” between xml layouts and Java code.

Intents

Intent: a messaging object which the programmer can use to request an action from another app component. One of the primary ways to do inter-process-communication (IPC) on Android.
https://developer.android.com/reference/android/content/Intent.html

There are two main kinds of intent. Explicit and Implicit.
Explicit Intents specify the component (ex. Activity) to start using its *Java class name*. You will usually use this mode to start Activities in your own application because you know the names of all your clases.

Implicit intents, instead describe a general type of action to be performed. This lets other apps choose whether they want to handle it. For example, if you want to initiate a phone call within your app.

Intents work with the startActivity method available in Activity. StartActivity will start a new Activity from your current Activity.

Implicit Intents

There are many built in Intents to do common system things like initiate a call, sms, email, google search etc. (see constants defined in the link above!)

Constructors:
Intent(String action),
or Intent(String action, Uri uri)

The Android operating system will listen for some events automatically, ex. initiating a phone call or email.

If your app wants to respond to Implicit Intents, it requires a BroadcastReceiver (we will cover BroadcastReciever in more detail after the reading break) or instead an intent-filter to be declared in the AndroidManifest.xml. In your AndroidManifext.xml file you need to provide both an action (up to you) and category (android.intent.category.DEFAULT) in order to launch with StartActivity see:
https://developer.android.com/guide/components/intents-filters.html#Receiving

We can also declare our own implicit intent types. Just make sure the string ex "com.marek.myapplication.WHATEVER" matches the intent-filter used in AndroidManifext.xml

Explicit Intents

if you know the exact Activity in the exact application you want to start you can use an explicit intent.
Explicit Intents invoke a specific class. Use the Intent(Context, Class) constructor to create an Intent for the chosen class explicitly.

Putting extra information into Intents

Before calling StartActivity you can add information to the intent with putExtra(String name, *Type* value);
Where *Type* is one of many possible types (see the API doc, above)
Ex. Types - Bundle (needed for the lab) lets you associate key value pairs much like a Map (data structure)
https://developer.android.com/reference/android/os/Bundle.html
Simple types like int, float (also String) can also be stored as extras without a Bundle

Tutorial: Adding a Button to your activity:

in the project view double click activity_main.xml
drag "Button" from Palette onto your activity_main layout
double click the button to access its text and id properties.
More properties are shown on the right.

To investigate the link between the layout xml and your application code in Java, switch from Design to Text view (tabs at the bottom) to view the xml version of activity_main.xml and you should see something similar to the following
<Button android:text="Send Mail" android:id="@+id/mailButton" />

the id here is how it will be referred to by the R object
and the text is simply what is shown on the button.

to access your button (or any view) from Java you can then use:
Button myButton = (Button) findViewById(R.id.**mailButton**);

Set code to execute on the button (see example):
https://developer.android.com/reference/android/widget/Button.html

Receiving and inspecting an Intent:

Create a second activity by right-clicking the java folder with MainActivity
New->Activity->empty Activity

An activity can get a copy of the intent that started it by calling getIntent() -see if there is any additional information for it…
https://developer.android.com/reference/android/app/Activity.html#getIntent()

From the intent you can get Extra data (the was put there with putExtra) such as intent.getStringExtra("extra_name");
https://developer.android.com/reference/android/content/Intent.html#getStringExtra(java.lang.String)

StartActivityForResult

Starting an Activity in order to get a result back to the calling activity - StartActivityForResult (You'll need this for the lab):
https://developer.android.com/training/basics/intents/result.html

Toast

Sending a toast (also for the lab):
A toast is a popup window that can't be interacted with, and disappears after a short while. Useful for debugging or showing status messages or alerts.
https://developer.android.com/guide/topics/ui/notifiers/toasts.html

Saving and restoring application state when your app is stopped by the OS:

https://developer.android.com/training/basics/activity-lifecycle/recreating.html

Push to github:

VCS->Import Into Version Control->Share Project On Github