ATLS 4120/5120: Mobile Application Development
Week 12: Android Lifecycle

In Android an activity is a single, specific task a user can do. Most apps include several different activities that allow the user to perform different actions. The system creates a new instance of the Activity class for each activity that's started.

When the user selects your app icon from the Home screen, it starts the main activity in the Android manifest file. By default this is the activity you defined when you created your Android project.
Every app must have an activity that is declared as the launcher activity. This is the main entry point to the app

- The main activity for your app must be declared in the `AndroidMainfest.xml file`
- Must have an <intent-filter> that includes the MAIN action and LAUNCHER category
- When you create a new Android project the default project files include an Activity class that's declared in the manifest with this filter.
- If either the MAIN action or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

[show this in the Halloween app]

Activities have a lifecycle of states that are important to understand.

Android Activity States
(slide)

- Created
  - An app's main activity has been launched
- Started
  - Activity is becoming visible
- Resumed
  - App is visible in the foreground and the user can interact with it, the running state
- Paused
  - Activity is partially visible, another activity is in the foreground
  - When paused it does not receive user input and doesn't execute any code
- Stopped
  - Activity is in the background and no longer visible
- Destroyed
  - All app processes have ended

Android has callback methods that correspond to specific stages of an activities lifecycle. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

Android Lifecycle Methods
https://developer.android.com/guide/components/activities/activity-lifecycle.html
The lifecycle methods are all from the Activity class

- **onCreate() –** activity is first created
  - Good place to do setup
  - calls **setContentView()** to declare the layout and configure the UI
  - **setContentView(R.layout.activity_main)** is how the activity knows which layout to load in the view
- **onRestart()** – activity was stopped and is about to restart
  - onRestart() doesn't get called the first time the activity is becoming visible so it's more common to use the **onStart()** method
- **onStart()** – activity is becoming visible
  - Followed by **onResume()** if the activity comes into the foreground
  - Followed by **onStop()** if the activity is made invisible
- **onResume()** – activity is in the foreground
  - the system calls **onResume()** every time your activity comes into the foreground, including when it's created for the first time.
- **onPause() –** activity is no longer in the foreground, another activity is starting
  - Followed by **onResume()** if the activity returns to the foreground
  - Followed by **onStop()** if the activity becomes invisible
- **onStop() –** activity is no longer visible
  - Followed by **onRestart()** if the activity becomes visible again
  - Followed by **onDestroy()** if the activity is going to be destroyed
  - Use **onStop()** to perform large, CPU intensive operations such as writing to a database
  - If the device is low on memory **onStop()** might not be called before the activity is destroyed
- **onDestroy() –** activity is about to be destroyed

When you override these methods to implement them make sure you call their super class method first

<u>Pausing and Resuming</u>

- When an activity in the foreground becomes partially obscured it becomes paused
  - Stop ongoing actions
  - Commit unsaved changes (if expected)
  - Release system resources
- As long as an activity is partially visible but not in focus it remains paused
- When the user resumes the activity you should reinitialize anything you released when it paused

When your activity is paused, the Activity instance is kept resident in memory and is recalled when the activity resumes.

<u>Stopping and Restarting</u>

- If an activity is fully obstructed and not visible it becomes stopped
  - Switches to another app
  - Another activity is started
  - User gets a phone call

- The activity remains in memory while stopped
- If the user goes back to the app, or uses the back button to go back to the activity, it is restarted

When your activity is stopped, the Activity instance is kept resident in memory and is recalled when the activity resumes.
You don't need to re-initialize components that were created during any of the callback methods leading up to the Resumed state.
The system also keeps track of the current state for each View in the layout.

Destroying and Recreating

- An activity is destroyed when the system decides it's no longer needed
  - User presses the back button
  - Hasn't been used in a long time
  - Needs to recover memory
- A change in device orientation results in the activity being destroyed and then recreated so the new device configuration can be loaded
  - Device configuration includes screen size, screen orientation, whether there's a keyboard attached, and also configuration options specified by the user (such as the locale).
- If the system destroys the activity due to system constraints, then although the actual Activity instance is gone, the system saves some state data in a Bundle object
  - If the user navigates back to that activity, a new instance of the activity is recreated using the data saved in the Bundle object
- As an activity begins to stop, the system calls the **onSaveInstanceState(Bundle)** method to save the current state of the activity
  - By default the Bundle instance saves information about each View object in your activity layout
  - To save additional data use the **savedInstanceState.putxxx()** methods
    - putInt()
    - putBoolean()
    - putLong()
    - etc
  - To restore your saved state data when the activity is being recreated you can access the Bundle in the **onCreate()** method using the **savedInstanceState.getxxx()** methods
  - Instead of restoring the state during **onCreate()** you may choose to implement **onRestoreInstanceState()** which the system calls after the **onStart()** method.

Let's update Halloween so the state is saved when the device is rotated and the activity is destroyed. (Halloween3.0 state)
In MainActivity.java we had everything in our sayBoo() method. But now we'll need access to the message, the TextView, and the ImageView in multiple methods, so we'll make them class level variables.

```
private String message;
private TextView booText;
private ImageView ghost;
```

It's important to understand that at this level we can't access anything in the layout file, such as findViewById(), because our view hasn't been loaded yet. That happens in onCreate() when setContentView() is called.

First we'll save our state, our message and image, when the activity is destroyed.

```
// invoked when the activity may be temporarily destroyed, save the instance state here
@Override
protected void onSaveInstanceState(Bundle outState) {
    outState.putString("msg", message);
    outState.putInt("imageid", R.drawable.ghost);

    super.onSaveInstanceState(outState);
}
```

Now we have to update onCreate() so we check to see if there's a savedInstanceState. Notice that onCreate() is passed a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    booText = (TextView)findViewById(R.id.message);
    ghost = (ImageView) findViewById(R.id.imageView);

    // recovering the instance state
    if (savedInstanceState !=null){
        message = savedInstanceState.getString("msg");
        int image_id = savedInstanceState.getInt("imageid");

        booText.setText(message);
        ghost.setImageResource(image_id);
    }
}
```

We must set the content view to our layout before we can access anything in our layout.
We have to call findViewById() to set our TextView and ImageView before we possibly try to set their values.
If savedInstanceState exists, we recover the state using the same keys we used when we saved it. Then we update our layout with these values.