Mobile App Development

The following is a portfolio of lab exercises designed to demonstrate the base classes commonly used in Android mobile Apps.

Building Your First Android App: Todo

(The code for this exercises can be found at: `https://github.com/ebbi/android`)

Nearly every concept in this exercise will be revisited; the aim here is to introduce the basics of an Android App.

The particular objectives to achieve the aim for this exercise include:

1. Navigating Android Studio
2. MVC architectural pattern, Activity class, XML Layouts and View objects
3. References to View Widgets, Listeners, and Anonymous inner classes
4. Debug

Navigating Android Studio

Android Studio is the official IDE for Android App Development. The following options are accessible through the welcome page and the top level menu bar in Android Studio.

▶ Create a new Android project

Git

Git is a distributed version control system and indispensable for any serious development. Here is a overview video guide. The exercises include the commands needed in typical practical usage.

▶ Setup a Github or Bitbucket repository

▼ Create a git branch to try the initial code for a Todo app.

```
git status
git branch todo
git checkout todo
git status
git log
```

MVC architectural pattern, Activity class, XML Layouts and View objects

▶ Directory and file structure of an Android project

View objects know how to draw themselves on the screen and respond to user input. `activity_todo.xml` has the definition of the layout and the View object TextView. Open the `app > res`

`> layout > activity_todo.xml` file; this is clearly the View definition and the V in the MVC architectural pattern.

A *model object* has the application logic and the data the logic is applied to. In this example, the Todo app would typically have the logic for creating and updating todo lists with the data being a particular list. We shall create this as an exercise. The model for the default Hello world is minimal as the logic is to display the "Hello World" constant string as the data.

Controller objects connect the view and the model objects together. Typically, controller objects respond to events triggered by View objects and manage the flow of data between the model and the view. A click on a Button View object is a common example of an event handled by a controller.

Android has the MVC architecture and it follows naturally particularly with the view objects and layout being abstracted and separately defined in XML. The separation of the Model and the Controller has to follow good programming practice. A controller as its name suggest should only have control logic and delegate all else to model classes.

▶ View for Todo

References to View Widgets, Callbacks, Listeners and Anonymous inner classes

The question is how does the XML view widget definition become a View object?

The entry point to the App is in: `java > com.example.todo.TodoActivity`. The class has one Activity method, onCreate(Bundle) which is called when an instance of the activity subclass is created.

To set the UI, `setContentView(R.layout.activity_todo)` is called. This method *inflates* each widget in the layout XML resources file and instantiate it as defined by its attributes, hence the equivalent object.

To get a reference to the inflated widget, the class Activity has a method `findViewById(int id)` which returns a View object for the widget id passed to it.

Lets practice with a button object to incrementally display each todo task in a TextView object.

▶ Lets practice with a button object to incrementally display each todo task in a TextView object.

Debug

▼ There are compile and runtime errors

To clarify, consider the following code for the `onClick(View v)` for the `buttonNext.setOnClickListener` method. The code intentionally has both type of compile and run time errors and the correct code is included in the comment.

```
buttonNext.setOnClickListener(new View.OnClickListener(){
  @Override
  public void onClick(View v) {

    /*
    BUG!
```

```
    Compile time error: mTodoIndexx is misspelled
    Runtime Error: no check for maximum number of items in the todos array

    // Bug fix compile error, Correct spelling mTodoIndex and not mTodoIndexx
    // Bug fix run time error, use the remainder as index to the array, i.e.
    //      mTodoIndex = (mTodoIndex + 1) % todos.length;
    */

    mTodoIndexx += 1;
    mTextView.setText(todos[mTodoIndex]);
  }
});
```

Compile time errors are generally to do with the syntactical rules. In this case, the editor in Android Studio would not recognise `mTodoIndexx` as the integer variable defined was `mTodoIndex`. It is highlighted in red and if you hover the mouse over it, it will display a message: *Can not resolve symbol 'mTodoIndexx'*. Note, the messages tab (bottom tool bar) also has the same compiler error message.

If the error was not clear, then the next step is to search for the error message. To avoid deprecated old code, set the search date to past year. Also, from the search result, use known sites such as the Android Developer API Guides (note, the site has a powerful search) and Stack overflow

Run time errors are due to inconsistency in the program logic or algorithm. In this example, the program runs fine until the last element of the array and then crashes. There is a failure in the logic in that the index for the array is incremented without checking for the end of the array. This leads to an attempt to access a non existent element of the array. See the comment in the code for correcting the error. This is the correct code: `mTodoIndex = (mTodoIndex + 1) % todos.length;` Note, % in Java returns the remainder, hence the index will never exceed the array size (an alternative less efficient solution would be to test for the size of the array).

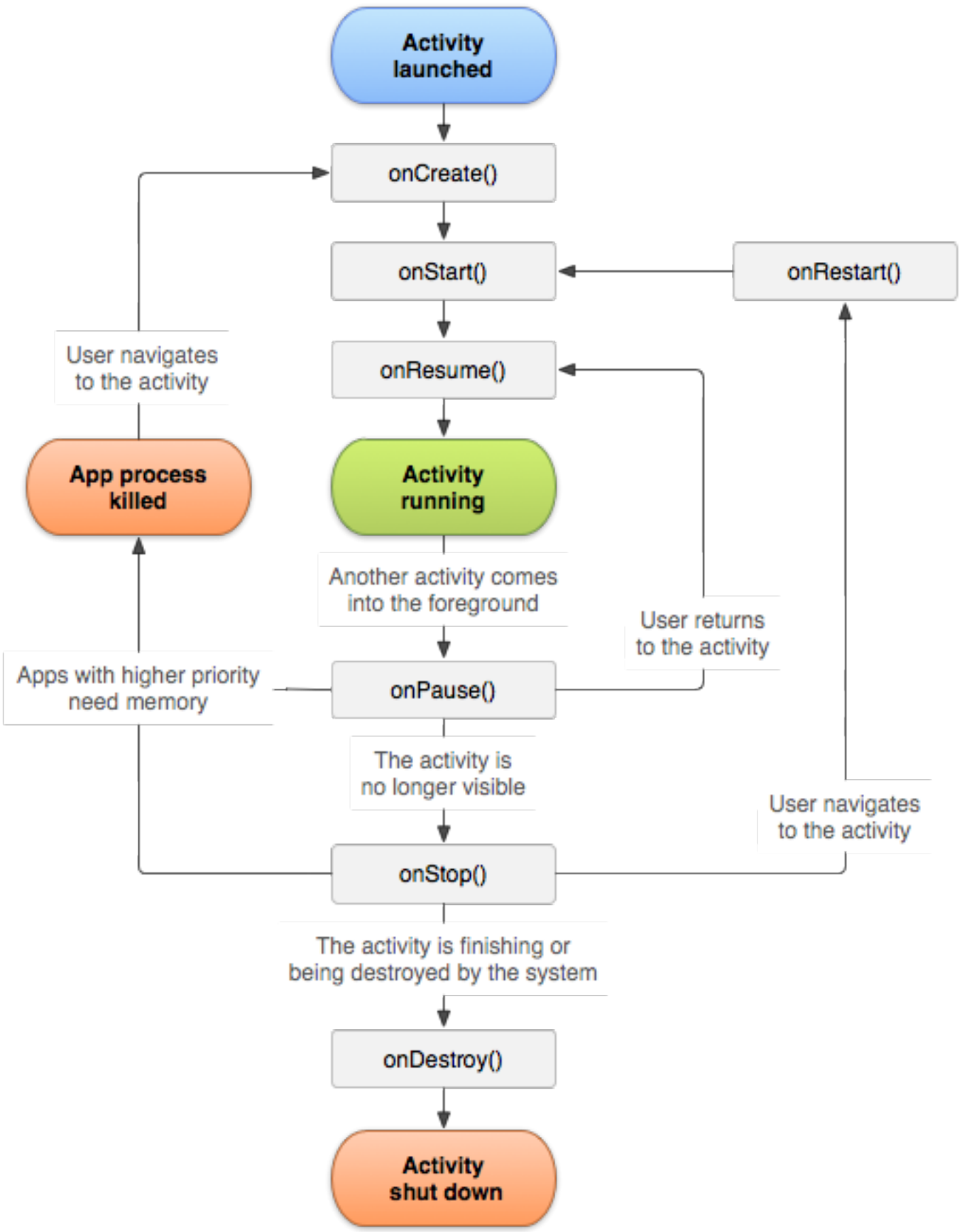
If the run time error message was not immediately clear then the next step is to see the stack trace in the **Android Monitor** tab (bottom tool bar). There is generally a link with the class name and line number that you could click. This is the last statement that could not be executed.

If examining the stack trace did not resolve the run time error; it is useful to set a debugging breakpoint at the line number the execution stopped. Click on the left margin of the line of code, `mTodoIndex += 1;` and notice the red circle. Now try, `Run > Debug 'app'`. Click on Next and note how the program stops at the breakpoint just set. Note the values of the variables. Hovering over any object or variable will reveal their current value. Try stepping through the code and see the `mTodoIndex` values until it crashes. Stepping through the code between breakpoints and examining expected values provide further information for resolving runtime error messages.

Activity Lifecycle and the "rotation problem"

The Todo App so far successfully displays the list of todos. There is however the common "rotation problem"; try navigating to the third todo and then rotate the phone until the display is in landscape mode and note the todo is reset to the first item on the list.

Activity has a lifecycle and Android can change the activity's state based on various events.



Every instance of `Activity` and its subclasses has a lifecycle and transitions between 4 states namely, *resumed, paused, stopped* and *nonexistent* with corresponding methods: `onCreate`, `onDestroy`, `onStart`, `onStop`, `onResume` and `onPause` respectively. These methods are called *lifecycle callbacks*. We overide these callbacks and Android calls the lifecycle call backs at the appropriate time such as after rotating the phone from portrait to landscape display.

To complete the exercise we now have a further aim to clarify the Activity lifecycle by resolving the common "rotation problem".

The particular objectives to achieve the new aim include:

1. Use git for sanity!
2. saving data accross rotations by storing the index for the todo array in a `Bundle` object as key value pair by overriding `onSaveInstanceState(Bundle)` method
3. Implement device configuration and resource changes such as alternate layouts due to state transitions.

4. Use a logger class to log messages for change of state for information and debugging.
5. Further use of the Debugger and breakpoints to step through the code.

Lets create a branch to try out ideas for the rotation problem fix and merge or discard the changes accordingly. First, commit the existing changes and merge to the master branch.

```
git status
git add .
git commit -am "Todo activity and initial view"
git checkout master
git merge todo

git branch rotation_fix
git checkout rotation_fix
```

We are now on the *rotation-fix* git branch and ready to try changes to fix the "rotation problem".

The solution to the rotation problem is to store the value of `mTodoIndex` integer accross run time changes like rotation of the phone. Android calls the activity method `onSaveInstanceState(Bundle)` before `onStop()`. We can override this method and add the value `mTodoIndex` to be saved in the `Bundle` object. The bundle object requires a key, value pair.

```
// In case of state change, due to rotating the phone
// store the mTodoIndex to display the same mTodos element post state change
// N.B. small amounts of data, typically IDs can be stored as key, value pairs in a Bundle
// the alternative is to abstract view data to a ViewModel which can be in scope in all
// Activity states and more suitable for larger amounts of data

private static final String TODO_INDEX = "todoIndex";

// override to write the value of mTodoIndex into the Bundle with TODO_INDEX as its key
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
  super.onSaveInstanceState(savedInstanceState);
  savedInstanceState.putInt(TODO_INDEX, mTodoIndex);
}
```

And finally, in the `onCreate(Bundle)` method, restore the index value

```
// check for saved state due to changes such as rotation or back button
// and restore any saved state such as the todo index
if (savedInstanceState != null){
  mTodoIndex = savedInstanceState.getInt(TODO_INDEX, 0);
}
```

Try *Run* and see the problem resolved, the index integer is passed between the state of the activity and is not reset to the first item after rotating the phone.

With the rotation, Android detects the device configuration change and looks for resources that better match the changed configuration. We will illustrate this by creating an alternative landscape view that will be automatically loaded as the state transits from portrait to lanscape and vise versa. This is acheived by a configuration qualifier namely, using -land suffix in the directory name.

▶ View definition for Landscape orientation

Debugging code with change of state can be challenging. The `android.util.log` class sends log

messages to a shared log. There are a variety of option; here is an example of a useful log for debugging while monitoring change of state at run time.

Insert the following after the call to the `super` constructor in the `onCreate` method of the `TodoActivity` class.

```
Log.d(TAG, " *** Just to say the PC is in onCreate!");
```

and define the final variable TAG:

```
public static final String TAG = "TodoActivity";
```

Open the *logcat window* in the *Android Monitor* tab (tool bar at the bottom) and notice the debug message as you rotate the phone and the state changes.

Once, the testing is complete, merge the changes on the current git branch to the master branch.

```
git status
git add *
git status
git commit -am "rotation problem fix"

git branch
git checkout master
git merge rotation-fix
git status
```

As well as `onCreate`, try writting the other Activity lifecycle callbacks with a similar log message.

Reflection and QA

What are the steps coded in a typical `onCreate` method of an Activity?

The term "inflate" in Android referes to instantiating a view object from its XML view definition; can you explain the code and the methods used to acheive this?

Using debugging breakpoints in Android Studio, can you step through the code and explain how the "e;buttonNext"e; is implemented.

What is a `Bundle` object?

What causes the "rotation problem" and how can it be fixed?

Give an example of using the `Log` class and how can it utilised to help correct runtime errors.