**SWE 6120 – Mobile Applications Development**

**Lab#2 - Android User Interfaces**

We want to create the following user interface for a game called Soduku



Cretae an Android project withy the following values:

Project name: Sudoku

Build Target: Android API - 19

Application name: Sudoku

Create Activity: Sudoku

The Soduku.java program should then be:

package <your package>;

import android.app.Activity;

import android.os.Bundle;

public class Sudoku extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

}

}

Android calls the onCreate( ) method of your activity to initialize it. The

call to setContentView( ) fills in the contents of the activity’s screen with

an Android view widget.

*main.xml* declares the user interface in XML, so that’s the file we need to modify. At runtime, Android parses and instantiates (*inflates*) the resource defined there and sets it as the view for the current activity.

Open tte *main.xml* file.

All Android XML files start with the line

<?xml version="1.0" encoding="utf-8"?>

It just tells the compiler that the file is XML format, in UTF-8 encoding. UTF-8 is almost exactly like regular ASCII text, except it has escape codes for non-ASCII characters

such as Japanese glyphs. Next we see a reference to <LinearLayout>:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height=" match\_parent">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/hello" />

<!-- ... -->

</LinearLayout>

Here is a list of the most common layouts provided by Android:

• FrameLayout: Arranges its children so they all start at the top left of the screen. This is used for tabbed views and image switchers.

• LinearLayout: Arranges its children in a single column or row. This is the most common layout you will use.

• RelativeLayout: Arranges its children in relation to each other or to the parent. This is often used in forms.

• TableLayout: Arranges its children in rows and columns, similar to an HTML table.

Some parameters are common to all layouts:

*xmlns:android=*[*http://schemas.android.com/apk/res/android*](http://schemas.android.com/apk/res/android)

First UI Attempt:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height=" match\_parent">

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/main\_title" />

<Button

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/continue\_label" />

<Button

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/new\_game\_label" />

<Button

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/about\_label" />

<Button

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/exit\_label" />

</LinearLayout>

Instead of hard-coding English text into the *layout* file, we use the *@string/resid* syntax to refer to strings in the *res/values/strings.xml* file. You can have different versions of this and other resource files based on the locale or other parameters such as screen resolution and orientation.

Open that file now, switch to the *strings.xml* tab at the bottom if necessary, and enter the following:

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Sudoku</string>

<string name="main\_title">Android Sudoku</string>

<string name="continue\_label">Continue</string>

<string name="new\_game\_label">New Game</string>

<string name="about\_label">About</string>

<string name="exit\_label">Exit</string>

</resources>

Save *strings.xml* so Eclipse will rebuild the project. When you run the program now, you should see something like the Figure below. It’s readable, but it could use some cosmetic changes.



Let’s make the title text larger and centered, make the buttons smaller, and use a different background color. Here’s the color definition, which you should put in res/values/colors.xml:

<?xml version="1.0" encoding="utf-8"?>

<resources>

<color name="background">#3500ffff</color>

</resources>

And here’s the new layout (also see next figure):

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:background="@color/background"

android:layout\_height="fill\_parent"

android:layout\_width="fill\_parent"

android:padding="30dip"

android:orientation="horizontal">

<LinearLayout

android:orientation="vertical"

android:layout\_height="wrap\_content"

android:layout\_width="fill\_parent"

android:layout\_gravity="center">

<TextView

android:text="@string/main\_title"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginBottom="25dip"

android:textSize="24.5sp" />

<Button

android:id="@+id/continue\_button"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/continue\_label" />

<Button

android:id="@+id/new\_button"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/new\_game\_label" />

<Button

android:id="@+id/about\_button"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/about\_label" />

<Button

android:id="@+id/exit\_button"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/exit\_label" />

</LinearLayout>



The screen is OK in portrait, but how about landscape mode (wide-screen)? The user can switch modes at any time, for example, by flipping out the keyboard or turning the phone on its side, so you need to handle that.

Try it out. In landscape mode, we can’t see the Exit button.

Create a file called *res/layout-land/main.xml* (note the -land suffix) that contains the following layout:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:background="@color/background"

android:layout\_height="fill\_parent"

android:layout\_width="fill\_parent"

android:padding="15dip"

android:orientation="horizontal">

<LinearLayout

android:orientation="vertical"

android:layout\_height="wrap\_content"

android:layout\_width="fill\_parent"

android:layout\_gravity="center"

android:paddingLeft="20dip"

android:paddingRight="20dip">

<TextView

android:text="@string/main\_title"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginBottom="20dip"

android:textSize="24.5sp" />

<TableLayout

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:layout\_gravity="center"

android:stretchColumns="\*">

<TableRow>

<Button

android:id="@+id/continue\_button"

android:text="@string/continue\_label" />

<Button

android:id="@+id/new\_button"

android:text="@string/new\_game\_label" />

</TableRow>

<TableRow>

<Button

android:id="@+id/about\_button"

android:text="@string/about\_label" />

<Button

android:id="@+id/exit\_button"

android:text="@string/exit\_label" />

</TableRow>

</TableLayout>

</LinearLayout>

This uses a TableLayout to create two columns of buttons. Now run the program again (see landscape Figure). Even in landscape mode, all the buttons are visible.



**Implementing an About Box**

When the user selects the About button, meaning that either they touch it (if they have a touch screen) or they navigate to it with the D-pad (directional pad) or trackball and press the selection button, we want to pop up a window with some information about Sudoku. After scrolling through the text, the user can press the Back button to dismiss the window.

We can accomplish this in several ways:

• Define a new Activity and start it.

• Use the **AlertDialog** class and show it.

• Subclass **Android’s Dialog** class, and show that.

For this example, let’s define a new activity. Like the main Sudoku activity, the About activity will need a layout file. We will name it ***res/layout/ about.xml***:

<?xml version="1.0" encoding="utf-8"?>

<ScrollView

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:padding="10dip">

<TextView

android:id="@+id/about\_content"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="@string/about\_text" />

</ScrollView>

Now add strings for the title of the About dialog box and the text it contains to ***res/values/strings.xml****:*

<string name="about\_title">About Android Sudoku</string>

<string name="about\_text">\

Sudoku is a logic-based number placement puzzle.

Starting with a partially completed 9x9 grid, the

objective is to fill the grid so that each

row, each column, and each of the 3x3 boxes

(also called <i>blocks</i>) contains the digits

1 to 9 exactly once.

</string>

Note how a string resource can contain simple HTML formatting and can span multiple lines. The backslash character (\) in about\_text prevents an extra blank from appearing before the first word.

The About activity should be defined in **About.java**. All it needs to do is override onCreate( ) and call setContentView( ):

package org.example.sudoku;

import android.app.Activity;

import android.os.Bundle;

public class About extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.about);

}

}

Next we need to wire all this up to the About button in the Sudoku class. Start by adding a few imports we’ll need to **Sudoku.java**:

import android.content.Intent;

import android.view.View;

import android.view.View.OnClickListener;

In the onCreate( ) method, add code to call findViewById( ) to look up an Android view given its resource ID and setOnClickListener( ) to tell Android which object to tickle when the user touches or clicks the view:

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

// Set up click listeners for all the buttons

View continueButton = findViewById(R.id.continue\_button);

continueButton.setOnClickListener(this);

View newButton = findViewById(R.id.new\_button);

newButton.setOnClickListener(this);

View aboutButton = findViewById(R.id.about\_button);

aboutButton.setOnClickListener(this);

View exitButton = findViewById(R.id.exit\_button);

exitButton.setOnClickListener(this);

}

While we’re in here, we do the same for all the buttons. Recall that constants like .id.about\_button are created by the Eclipse plug-in in R.java when it sees **+id/about\_button** in **res/layout/main.xml.**

The code uses this as the receiver, so the Sudoku class needs to implement the OnClickListener interface and define a method called onClick:3

public class Sudoku extends Activity implements OnClickListener {

// ...

public void onClick(View v) {

switch (v.getId()) {

case R.id.about\_button:

Intent i = new Intent(this, About.class);

startActivity(i);

break;

// More buttons go here (if any) ...

}

}

}

If you run the program and select the About button now, you will get an error (see next Figure). What happened?



We forgot one important step: every activity needs to be declared in AndroidManifest.xml. To do that, double-click the file to open it, switch to XML mode if necessary by selecting the *AndroidManifest.xml* tab at the bottom, and add a new <activity> tag after the closing tag of the first one:

<activity android:name=".About"

android:label="@string/about\_title">

</activity>

Now if you save the manifest, run the program again, and select the About button, you should see something like the next Figure. Press the Back button ( Esc on the emulator) when you’re done. That looks OK, but wouldn’t it be nice if we could see the initial screen behind the About text?



Applying a Theme

A theme is a collection of styles that override the look and feel of Android widgets. Themes were inspired by Cascading Style Sheets (CSS) used for web pages—they separate the content of a screen and its presentation or style. Android is packaged with several themes that you can reference by name, or you can make up your own theme by subclassing existing ones and overriding their default values.

We could define our own custom theme in res/values/styles.xml, but for this example we’ll just take advantage of a predefined one. To use it, open the AndroidManifest.xml editor again, and change the definition of the About activity so it has a theme property.

<activity android:name=".About"

android:label="@string/about\_title"

android:theme="@android:style/Theme.Dialog">

</activity>

The @android: prefix in front of the style name means this is a reference to a resource defined by Android, not one that is defined in your program.

Running the program again, the About box now looks like the next Figure. Many programs need menus and options, so the next two sections will show you how to define them.



Adding a Menu

Android supports two kinds of menus. First, there is the menu you get when you press the physical Menu button. Second, there is a context menu that pops up when you press and hold your finger on the screen (or press and hold the trackball or the D-pad center button).

Let’s do the first kind so that when the user presses the Menu key, they’ll open a menu like the one in the next Figure. First we need to define a few strings that we’ll use later:

<string name="settings\_label">Settings...</string>

<string name="settings\_title">Sudoku settings</string>

<string name="settings\_shortcut">s</string>

<string name="music\_title">Music</string>

<string name="music\_summary">Play background music</string>

<string name="hints\_title">Hints</string>

<string name="hints\_summary">Show hints during play</string>

Then we define the menu using XML in res/menu/menu.xml:

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item android:id="@+id/settings"

android:title="@string/settings\_label"

android:alphabeticShortcut="@string/settings\_shortcut" />

</menu>

Next we need to modify the **Sudoku.java** class to bring up the menu we just defined. To do that, we’ll need a few more imports:



import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

Then we override the Sudoku.onCreateOptionsMenu( ) method:

@Override

public boolean onCreateOptionsMenu(Menu menu) {

super.onCreateOptionsMenu(menu);

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.menu, menu);

return true;

}

getMenuInflater( ) returns an instance of MenuInflater that we use to read the menu definition from XML and turns it into a real view.

When the user selects any menu item, onOptionsItemSelected( ) will be called. Here’s the definition for that method:

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.settings:

startActivity(new Intent(this, Settings.class));

return true;

// More items go here (if any) ...

}

return false;

}

Settings is a class that we’re going to define that displays all our preferences

and allows the user to change them.

Adding Settings

Android provides a nice facility for defining what all your program preferences are and how to display them using almost no code. You define the preferences in a resource file called res/xml/settings.xml:

<?xml version="1.0" encoding="utf-8"?>

<PreferenceScreen

xmlns:android="http://schemas.android.com/apk/res/android">

<CheckBoxPreference

android:key="music"

android:title="@string/music\_title"

android:summary="@string/music\_summary"

android:defaultValue="true" />

<CheckBoxPreference

android:key="hints"

android:title="@string/hints\_title"

android:summary="@string/hints\_summary"

android:defaultValue="true" />

</PreferenceScreen>

The Sudoku program has two settings: one for background music and one for displaying hints. The keys are constant strings that will be used under the covers in Android’s preferences database.

Next define the **Settings class**, and make it extend PreferenceActivity:

package org.example.sudoku;

import android.os.Bundle;

import android.preference.PreferenceActivity;

public class Settings extends PreferenceActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

addPreferencesFromResource(R.xml.settings);

}

}

The addPreferencesFromResource( ) method reads the settings definition from XML and inflates it into views in the current activity. All the heavy lifting takes place in the PreferenceActivity class.

Don’t forget to register the Settings activity in AndroidManifest.xml:

<activity android:name=".Settings"

android:label="@string/settings\_title">

</activity>

Now rerun Sudoku, press the Menu key, select the Settings... item, and watch with amazement as the Sudoku settings page appears (see next Figure). Try changing the values there and exiting the program, and then come back in and make sure they’re all still set. Code that reads the settings and does something with them will be discussed in a different Lab (Storing Local Data).

