Android Handbook

This document outlines resources and best practices to use when creating your Android application.

General Java Style

Refer to the CS 61B style guide for the best Java programming practices.

Logging

Sometimes as a developer it is important to know when an event occurs or what data is present, similar to when System.out.println is used when developing in Java. With Android, there are a few ways to do this, some of which display the information on the device and some in the console in Android Studio.

Console

System.out.println still works just as you'd expect. The Log class allows you to add a "tag" to what is printed in the console so that you can filter more easily. There are different types of logs for even further filtering.

Device

Toasts allow you to create a small popup with text at the bottom of the device's screen.

Utils

It is good to have a Utils class to handle functionality that is used over and over. Functions in this class will be public static functions so that they can be called anywhere. For example, I have a function that takes a title and message as parameters and creates a dialog with it.

Strings.xml

Something that hasn't really been stressed in the training program is using the strings.xml folder to store strings that are used throughout your application.

Bind Variable to View in a Layout

```
ListView listView = (ListView) findViewById(R.id.listView);
```

Get Text from EditText

```
EditText editText = (EditText) findViewById(R.id.editText);
String message = editText.getText().toString();
```

Set an onClickListener for a Button (or any View)

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
            //Whatever you want to happen
      }
});
```

Open a New Activity Using an Intent (and Adding an Extra)

```
public static final String EXTRA_MESSAGE = "extraMessage";
Intent intent = new Intent(getApplicationContext(),
                           NewActivity.class);
intent.putExtra(EXTRA_MESSAGE, message);
startActivity(intent);
```

TextView Attributes

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Text Here"
android:id="@+id/textView"
android:textColor="@android:color/black"
android:textStyle="bold|italic"
```

`"wrap-content"` makes the view as big as whatever it contains
`"match-parent"` makes the view as big as the parent (usually the size of the screen)

Additional Resources:
http://developer.android.com/reference/android/widget/TextView.html

This site states every attribute and explains the possible values.

Layouts

RelativeLayout: Places elements “relative” to one another
LinearLayout: Aligns views in columns or rows

Making a Toast Message

```
Toast.makeText(getApplicationContext(), "Toast message",
                Toast.LENGTH_SHORT).show();
```

`getApplicationContext()` might have to be changed to reflect however the context can be attained. However, most of the time this will work.

`Toast.LENGTH_LONG` can be used as well to display the toast for a longer time.

Additional Resources:
http://developer.android.com/guide/topics/ui/notifiers/toasts.html

SharedPreferences (Saves Key-Value Pairs on the Device)

```
public static final String DAY_KEY = "dayKey";
SharedPreferences sharedPreferences =
     PreferenceManager.getDefaultSharedPreferences(
     getApplicationContext());
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString(DAY_KEY, "Friday"); //Saves value "Friday" with that
                                        key
editor.apply(); //Nothing is saved until apply() or commit() are
                  called
sharedPreferences.getString(DAY_KEY, ""); //Gets the value associated
     with that key, returns an empty string ("") as the default
value.
```

Additional resources:
http://developer.android.com/reference/android/content/SharedPreferences.html
http://developer.android.com/reference/android/content/SharedPreferences.Editor.html

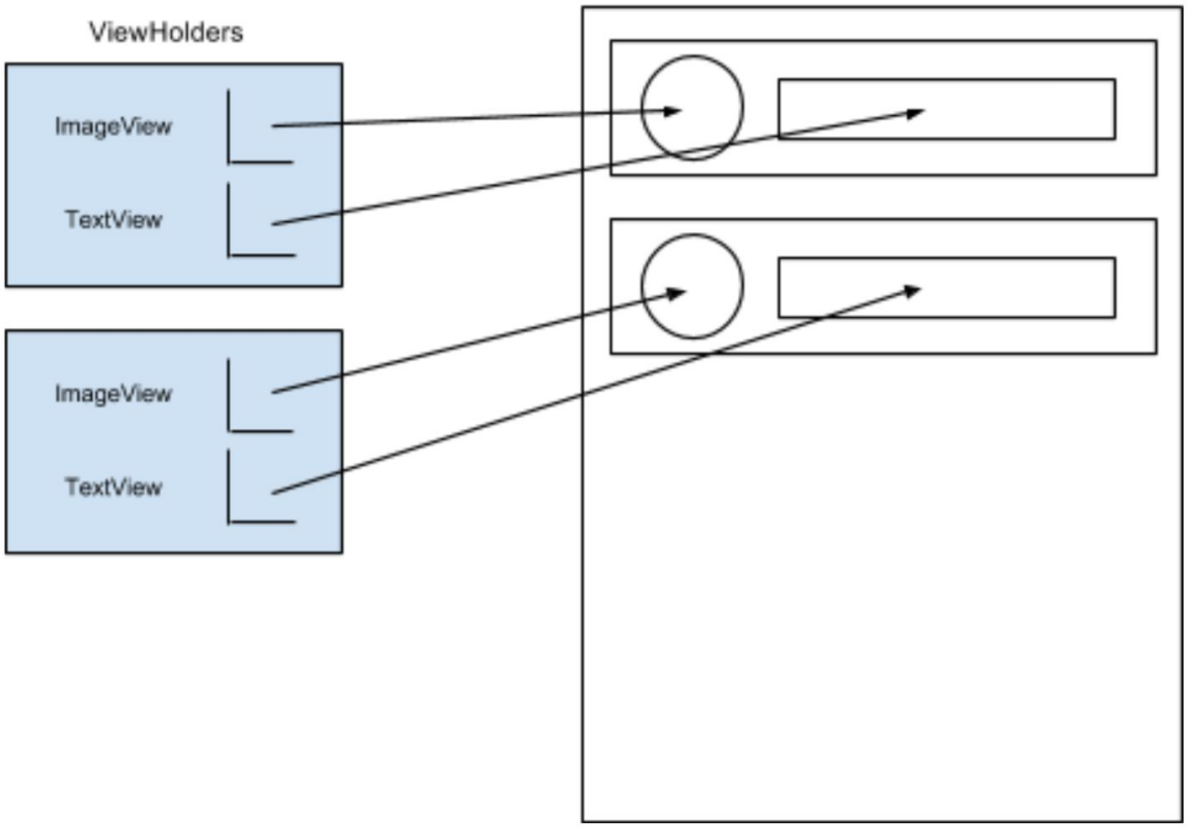
Menus

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
     //Inflate the menu
     //This adds items to the action bar if it’s present.
```

```
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
                case R.id.settings:
                        …
                        return true;
                case R.id.refresh:
                        …
                        return true;
        }
        return super.onOptionsItemSelected(item);
}
```

RecyclerViews



Each row has its own ViewHolder instance, which contains values that point to the views in its respective row (it holds views, hence “View Holder").

```
public CustomViewHolder (View view) {
      super(view);
      this.subjectNameTextView = (TextView)view.findViewById(
            R.id.subjectNameTextView);
      this.imageView = (ImageView) view.findViewById(R.id.imageView);
      this.homeworkDoneCheckBox = (CheckBox)view.findViewById(
            R.id.homeworkDoneCheckBox);
}
```

In simplified terms, a ViewHolder is an object that holds the pointers to the views in each each row. What does that mean? Every row has a TextView, ImageView, and CheckBox. Each row has a ViewHolder, and that ViewHolder holder these 3 views in it (hence "view holder"). This function returns a single ViewHolder; it is called once for every row.

```
@Override
public CustomViewHolder onCreateViewHolder(ViewGroup parent,
                                           int viewType) {
      //This "inflates" the views, using the layout R.layout.row_view
      View view = LayoutInflater.from(parent.getContext()).inflate(
            R.layout.row_view, parent, false);
      return new CustomViewHolder(view);
}
```

This function takes the previously made ViewHolder and uses it to actually display the data on the screen. Remember how the holder contains (pointers to) the 3 views? By doing, for example, "holder.imageView" we are accessing the imageView for that row and setting the ImageResource to be the corresponding image for that subject.

```
@Override
public void onBindViewHolder(CustomViewHolder holder, int position) {
      SchoolSubject schoolSubject =
schoolSubjectsArray.get(position);
      holder.subjectNameTextView.setText(schoolSubject.name);
      holder.imageView.setImageResource(schoolSubject.imageId);
      holder.homeworkDoneCheckBox.setChecked(
            schoolSubject.isHomeworkDone);
}
```

Firebase

Authentication

[Basic password-email account creation]

Sign in

Database

Basic write operations
Reading data once
Listening for events

Miscellaneous

One weird error to watch out for:

When setting the text of a text view to an integer, say `int niceInt`, then make sure you do it this way:

```
textView.setText(niceInt + "");
```

Instead of just:

```
textView.setText(niceInt);
```

In other words make sure you're passing in a string to setText, otherwise it thinks the integer is a string resource id which it isn't!