Coding standards document

July 21, 2018

NAMES:	**STUDENT NUMBER:**
Xolo K Dandashe	14245681
Phuti K Setoaba	13032616
Khodani M Mufamadi	14197520

1 Style,Structure and layout

1.1 Variables, Methods and attributes:

• Should be short and descriptive.

• Should begin with lower-case letter.

• Upper-case letter should be used immediately after the first name description of the variable.

• Should use a single upper-case letter at the start of each new word within the name.

• Should not be a single character e.g. string x

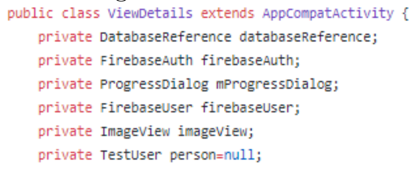


Figure 1: Expected representation for naming variables.

```
public class ViewDetails extends AppCompatActivity {
    private DatabaseReference databaseReference;
    private FirebaseAuth firebaseAuth;
    private ProgressDialog mProgressDialog;
    private FirebaseUser firebaseUser;
    private ImageView imageView;
    private TestUser person=null;
```

1.2 Accessor and Mutators

• Should have names based on the attribute to which they provide access.

• Should begin with the prefix get or set followed by the name of the attribute beginning with a capital letter e.g. getItem

1.3 Indentation and Layout

• Blocks that are nested more should be indented more.

• Lines should be kept to a sensible length to make the code easier to read and print.

• Single blank lines should be used to separate methods and to emphasise blocks of code.

1.4 Bracketing

• Brackets will be used to clearly show the blocks of code they encapsulate.

• Closing curly brackets should be placed on the line after the last line of the code they enclose, at the same level of indentation as the start of the header on which the block begins.

• Curly brackets should always be used for if-, while- and for-, even when not strictly needed such as when the body only contains a single statement.

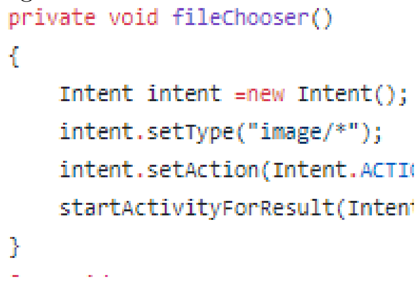


Figure 2: Indentations and Bracketing.

```
private void fileChooser()
{
    Intent intent =new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTI
    startActivityForResult(Inten
}
```

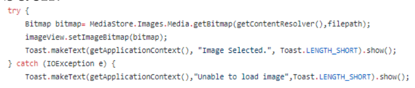
1.5 Exceptions

• Exceptions should be used where necessary.

• Instead of throwing basic Exception classes, sub-classes should be made with more meaningful names.

• Appropriate catch statements should be put in place to allow the program to recover if need be.

Figure 3: Exception handling feedback.



```
try {
    Bitmap bitmap= MediaStore.Images.Media.getBitmap(getContentResolver(),filepath);
    imageView.setImageBitmap(bitmap);
    Toast.makeText(getApplicationContext(), "Image Selected.", Toast.LENGTH_SHORT).show();
} catch (IOException e) {
    Toast.makeText(getApplicationContext(),"Unable to load image",Toast.LENGTH_SHORT).show();
```

1.6 Imports

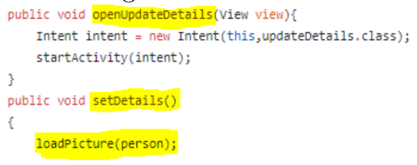
• Should avoid using import *(importing every package), rather each class should be specifically imported as required.

• Any unused imports should be removed if they are no longer needed to make it clear which classes will be used.

2 Naming Conventions

This section will cover examples referring to some of the conventional rules mentioned above.

Figure 4: Methods.



```
public void openUpdateDetails(View view){
    Intent intent = new Intent(this,updateDetails.class);
    startActivity(intent);
}
public void setDetails()
{
    loadPicture(person);
```

2.1 Classes

• Class names should begin with a capital letter and each new word within the name should begin with a capital letter e.g ”ClassName”.

2.2 Exception Classes

• Exception classes should follow the same rule as normal classes, but should end with the word Exception e.g ”ClassNameException ”.

2.3 Methods

• Methods should begin with a lower case letter and each new word within the name should begin with a capital letter e.g ”methodName()”.

2.4 Variables and Attributes

• Both variables and attributes should being with a lower case letter and each new word within the name should begin with a capital letter exactly the same as methods e.g "variableName".

2.5 Packages

• Package names should all be in lower case e.g "demo.package".

3 XML

3.1 Tags

• XML tags should be ordered as follows: "xmlns" first, then id,then `layout_width` and `layout_height` alphabetically.

• Add a space between the closing slash and the final attribute.

E.g. `android:textSize="10dp" />`

• Self closing tags should be used when an XML element doesn't have any contents, you should use self closing tags.

3.2 Layout XML ID Naming / Java Class Widget Declaration Variable

This section provides a guide on how to link xml elements with java variables.

Element	**Example**
TextView	TextView textView
ImageView	ImageView imageView
Button	Button button
EditText	EditText editText

4 Documentation

4.1 Comments

It is expected that the code written should be structured in a way that all members can understand it without comments but here are some guides to placing comments:
• Use in-line commenting to help the next developer who might be editing your code.

• Inline comments should appear on the line above the code you are commenting.

• Comments should be added within the body of a method if they are used within that method.

4.2 Resource Files

• Resources file names should be written in `lowercase_underscore` e.g `ic_star.png`

4.3 Layout Files

• Layout files should match the name of the Android components that they are intended for but moving the top level component name to the beginning.

• Example, if you are creating a layout for the SignInActivity, the name of the layout file should be `activity_sign_in.xml`

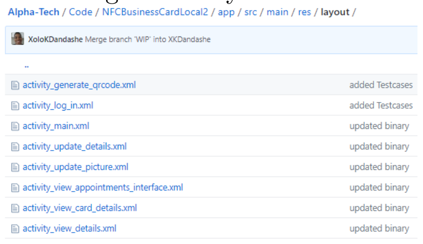


Figure 5: Layout files.

Alpha-Tech / Code / NFCBusinessCardLocal2 / app / src / main / res / **layout** /

XoloKDandashe Merge branch 'WIP' into XKDandashe

..

activity_generate_qrcode.xml	added Testcases
activity_log_in.xml	added Testcases
activity_main.xml	updated binary
activity_update_details.xml	updated binary
activity_update_picture.xml	updated binary
activity_view_appointments_interface.xml	updated binary
activity_view_card_details.xml	updated binary
activity_view_details.xml	updated binary

4.4 Version control

• No commented out code must be committed unless you have a very good reason that is clearly described in a comment by the code you are committing.

4.5 Repository definitions

• All the pictures should be in a folder called images.

• All the files should belong to a specific folder.

• Folder names should be descriptive enough to show what they contain.

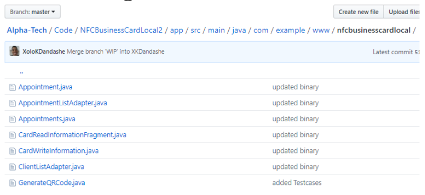


Figure 6: Java files.

Alpha-Tech / Code / NFCBusinessCardLocal2 / app / src / main / java / com / example / www / **nfcbusinesscardlocal** /

XoloKDandashe Merge branch 'WIP' into XKDandashe Latest commit 51

..

Appointment.java	updated binary
AppointmentListAdapter.java	updated binary
Appointments.java	updated binary
CardReadInformationFragment.java	updated binary
CardWriteInformation.java	updated binary
ClientListAdapter.java	updated binary
GenerateQRCode.java	added Testcases

5 Code quality

• We need to ensure that we keep at least a 70% overall correspondence with the coding standards guidelines to ensure that we have a high level of code quality.

• These guidelines are for us to follow as much as possible to ensure production time is minimized and that we do not waste resources trying to explain each other when integration is needed.

• A high level of code quality will enable us to produce better components, improve testing and minimize confusion.

• Reviews will be simpler to perform and improvements will be simpler to implement.

• Android Studio will be used for development which uses Java for backend functions and XML for layout presentation.

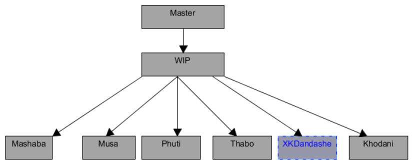
• Coding standard rules mentioned above will be used for resources naming conventions, formatting and also to ensure version control.

• Commit frequently to ensure no important information is lost.

6 Repository

Our Github repository is Alpha-Tech. We use the team name so that members can identify the repository easier in the case of a member having multiple repositories that they are working on.

Figure 7: Branching Structure.



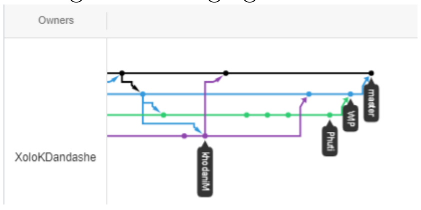
6.1 Branching and Merging

Our repository has a tree like logic-representation where each member is expected to have a branch of their own, a work-in-progress (WIP) branch and the master at the top:
• The structure of how the work will be presented is created in the master branch.

• Once we have set up the master branch, a work-in-progress (WIP) branch is created then clones the master to have the same file structure and layout.

• The WIP branch is the hub where all members can pull and push their work that has not been reviewed but has been completed.

Figure 8: Merging branches.



• Once the work completed, in WIP, is reviewed by other members, then it is pushed to the master.

• No member may commit directly into the master branch or are allowed to push and/or pull from the master due to how it contains ready to be deployed work.

• A member may merge their branch with WIP only once they have completed all the work they have been assigned.

• If a member has merged their branch with WIP then they may delete their branch as for the work will be in the WIP branch which will be later merged with master once all the work is reviewed.