Project Tips

Emulation:

- We recommend using Android Studio for the development and emulation environment. If you want to use C++ for the backend, you can integrate C++ to Android Studio or use Eclipse.
- The default Android Studio emulator runs slow on some computers (especially older ones). You can try emulating an older phone which requires less memory space.
- If you have problems with the Android Studio emulator, try using the Genymotion plug-in for Android Studio (https://docs.genymotion.com/desktop/3.0/04_Tools/042_Genymotion_plugin_for_Android_Studio.html). Please note: This plug-in only offers a 30-day free trial period. If you intend to use the software after the trial, you must purchase the full version.
- You're welcome to use any Android version you want for your project, as long as the demo works in the end.

Interfacing:

- There are several options for the *interfacing* portion of the project. All related information is available on the web with a little bit of searching, but it is definitely doable and part of your job to figure it out.
- Interfacing should be one of the sections of the project that gets done early because it is vital to testing the entire project and can be one of the more difficult things to finish and debug.
- If you are using C/C++ for the backend:
  - Explore the Android NDK and the examples.
  - Explore the "jni.h" header for appropriate functions and structures.
    http://xdprof.sourceforge.net/doxygen/jni_8h-source.html
  - You need to include an `extern "C"` block around the appropriate C++ functions otherwise the code may not compile or run properly.
  - You can also compile C++ code into C code using free programs such as LLVM.
    http://stackoverflow.com/questions/1833484/c-frontend-only-compiler-convert-c-to-c/

Sample Code:

- Sample codes on the Android site are important to figuring out how to code your applications.
- To run the NDK demos, you need to add the project to the workspace by importing an Android project from existing code using a new project. Make sure that you point the NDK path in the Eclipse preferences to the appropriate folder (or set up Android Studio correctly).
- Follow the "hello-jni" example here:
  https://github.com/googlesamples/android-ndk/tree/master
- The important part, if you are using a C/C++ backend, is that you need to build the C/C++ code separately from your Java project. It is done via the ndk-build program downloaded with the NDK. The link above contains the details. Mac and Unix users can do this from their terminals; Windows users will need to download Cygwin to achieve the same.

GUI:

- A significant amount of documentation on how to properly use the Android platform can be found here and in associated pages:
  - https://developer.android.com/guide/topics/ui/

Miscellaneous tips:

- If working with Samsung devices, be aware that they call onDestroy() every time orientation changes.
- To enable certain features such as orientation lock, internet use, and saving to memory, you need to activate the corresponding permissions in the manifest file. More information can be found here:
    - http://developer.android.com/guide/topics/security/permissions.html
    - http://developer.android.com/reference/android/Manifest.permission.html
- If your app requires that you need to receive/send data, make sure to use asyncTask to avoid blocking (freezing) the GUI while sending/waiting for a response. AsyncTask task spawns a background thread that will handle transmission independent of the GUI.
    - http://developer.android.com/reference/android/os/AsyncTask.html

Debugging tips:

- Do **not** write the entire app in one go and press play once at the end. It is good to run your app frequently to make sure your logic works before moving on (this extends to all code).
- It is important to pay attention to the errors that the IDE or compiler give you (Java doesn't use a compiler) as they specify *what* the error is and *where* it is located in the code. It is usually beneficial to start with the first error, as fixing it may clear up subsequent errors.
- The barebones method for debugging logic errors is to sprinkle around cout/System.out.println statements to check that you are actually computing what you intend. If you want to be fancy, many IDEs have advanced debuggers that allow you to break at specified points in the program and watch your variables as you step through your code.
- If you experience a nullpointerexception/Segmentation Fault, it usually has to do with accessing "bad" memory! The most common culprit is trying to access indices that do not exist in arrays/other containers. When you see this error, just check your pointers and loop bounds (Java doesn't have pointers, woohoo!).
- Google is your best friend.