CSE 486/586 Distributed Systems
PA Best Practices

Steve Ko
Computer Sciences and Engineering
University at Buffalo

First Things to Keep in Mind

- Your implementation is necessarily going to be full of bugs.
  - Your job is to reduce the number of bugs as many as possible before submitting it.
- Correctness first. Performance later.
  - Don't even think about performance optimization in your first-cut implementation. You will not do it correctly.
- Know how to use sockets *precisely*.
  - What does accept() do? What is a port number? What is socket timeout? What do read(), write(), and flush() do?
  - There is no other way but reading the documentation thoroughly.

First Things to Keep in Mind

- Know the constructs and data structures *precisely*.
  - AsyncTasks, Threads, Cursors, etc.
  - Once again, read the documentation thoroughly.
- Write your code systematically.
  - Write a small chunk, test, and repeat.
  - If you're writing a hundred lines of code before testing it, there's something wrong.
- Structure your code well. Simplify.
  - Use simple, one-shot tasks (e.g., request-response) that work independently, asynchronously, and mostly serially.
  - A rule to compare: if your method has more than 20 lines, think why you're doing it and convince yourself that it's necessary.

First Things to Keep in Mind

- Follow good style conventions
  - Strive to write maintainable and professional-looking code
- Know answers to questions like the following.
  - What is the Java comment style convention?
  - Where are good places to define a field?
  - What are good ways to catch exceptions?
  - What is a good indentation convention?
  - What are good variable/method naming conventions?
  - What is a good brace style?
  - What are good line lengths?
- Good sources
  - https://source.android.com/source/code-style.html
  - https://google.github.io/styleguide/javaguide.html

First Things to Keep in Mind

- Embrace concurrency
  - Dealing with concurrency is your make-or-break point for all later assignments.
  - Keep in mind that your code will behave in a non-deterministic way with multiple threads.
  - We still expect your code to produce deterministic results.
  - You will need to come up with your own test cases and testing strategies (more on this later).
  - You will need to run the graders multiple times and make sure that you get the same results consistently.
- Serialize as much as possible.
  - Do one thing at a time.
  - Use a lock on every shared variable.

First Things to Keep in Mind

- Don't be caught up with just one way of doing things.
  - Start early.
  - Ask and explore if there is any other way of doing the same thing.
  - Learn why one way is better than the other.
  - This often results in a better, faster outcome.
- Use logging statements for debugging.
  - Almost the only way to debug things for distributed systems.
  - Once deployed, the only way to debug your code in any system.
  - Open logcat for each instance in a terminal.
  - It's OK to put a lot of logging statements. Just remove really unnecessary ones later.
  - Be aware that logging statements alter the code behavior.

First Things to Keep in Mind

- Do not use graders as your debugger.
  - Graders do not know what you're doing. They only verify what you're returning.
  - They will try to give you reasons why they are complaining. But this by no means gives you enough information for debugging.
  - When a grader complains about something, the first thing you need to do is NOT deciphering the grader's message; you need to look at your own logs and pinpoint what's going on.

First Things to Keep in Mind

- Do not assume that Android or graders are broken.
  - It's almost always your code. Blaming other systems only reveals your immaturity as a developer ☺
  - In general, if you're developing with a large-scale system, it's always tempting to think that the other system is broken, not yours.
  - That's almost always wrong, unless you have a very well-thought-out reason verified by people knowledgeable about the internals of the other system.
  - OK, I admit that sometimes it might be the case. But it is very rare in the context of our class.
  - (Disclaimer: It is generally good to be suspicious about what's going on outside your code; it gives you learning opportunities and you might actually find a bug that no one has encountered.)

First Things to Keep in Mind

- You will write complex code, so make sure that you have your own testing strategy
  - Graders are convenient, indeed. However, it is just not enough for testing and debugging.
  - This is especially true with concurrency, since a bug may not manifest deterministically. There can be many conditions that lead to deadlocks and livelocks.
  - It's best to develop your own test cases that stress your system.

More Specific Things

- Socket
  - Know how exactly it works. E.g., accept() returns a new socket.
  - Know how to use socket timeout, which only times out when reading (not writing, not accepting, etc.).
  - Use it as a write/read pair (and check on the returning read for failures), or even better, continue using the same socket as long as possible.
- AsyncTask
  - Be aware: there is a limit of how many you can create and execute at the same time (a concept called thread pools). This is system-dependent, but on our emulator, it seems to be 5.
  - You can wait for a result using AsyncTask.get()

More Specific Things

- Threading
  - Understand how threads behave and synchronization works. E.g., what happens if you call Thread.sleep() in the middle of an execution? What happens with wait() and notify()?
  - If you have threads, but don't use locks or synchronized, you're probably doing something very wrong.
- Messages
  - Do not send a whole object (Serializable). Takes a long time, unnecessarily big, and just a lot of overhead.
  - Formatted strings should be enough.

More Specific Things

- Logging
  - Concurrency is difficult to debug and extensive logging is pretty much the only reliable way.
  - You need to be able to trace out all interactions, e.g., communication originated from one peer to all other peers and coming back (use and log message IDs, message types, etc.)
  - You need to be able to keep track of each thread's progress, e.g., is this method call blocked or returning? Is this for loop moving on to the next iteration? Is this loop terminating properly?
- Know how to use adb
  - adb devices
  - adb kill-server/start-server
  - adb install/uninstall/insert/query

More Specific Things

- Grader related
  - Check the output of a grader carefully. They do not always print out a potential error at the end.
  - Graders use string matching to check the results. Be careful about extra whitespaces you might be inserting.
- onCreate()
  - This should not do any long-running operation. If onCreate() takes a long time to return, it might give you unexpected problems, e.g., it might get called multiple times.
  - This is especially true with a ContentProvider. If onCreate() is being executed and there's an insert() or a query() call, onCreate() will be called again.

More Specific Things

- Callback method signatures
  - Do not change the signature of a callback method you implement, e.g., query(), insert(), etc. They are all defined by Android, and Android doesn't expect any change.
  - For example, don't add synchronized to query() or insert(). It will behave in an erratic way.
- Failure detection
  - You can use a regular request/reply as a failure detection mechanism.
  - For this, always reply back with something so the other side knows that you're alive.
- Do not write a custom busy-wait in your code
  - Don't use a loop and check whether or not a condition is true. Use Java primitives such as wait()-notify().