Computer Programming – CS 6011 Lecture 22: Android Part 2

MASTER OF SOFTWARE DEVELOPMENT (MSD) PROGRAM

J. DAVISON DE ST. GERMAIN

FALL 2023

Miscellaneous

- Questions?
- Missing grades…

Lecture 22 – Topics

- Android
- Final Exam Review Topics

Android Versions

- Android 14 (Upside Down Cake)
  - Released Oct. 2023 - 14th major version of Android
- Android 13 (Tiramisu) – API 33
  - Released Aug. 2022
- Android 12L (Snow Cone v 2) – API 32
- Android 12 (Snow Cone) – API 31
  - 13.5% of devices
- …
- Android 5.1 (Lollipop) – API 22
  - Released Mar. 2015
  - App will run on 98% of Android phones.

WebSocket Client Library For Java

- Strangely enough, Java does not have a built-in WebSocket library. Therefore, we will use:
  - https://github.com/TakahikoKawasaki/nv-websocket-client
- Note, this library is used for the *client-side* communication.
- Make sure you add the Gradle (Module) dependency to your project.
  - implementation("com.neovisionaries:nv-websocket-client:2.14")
  - This will tell Android Studio to download and install the library automatically for you (when you press the build button).
- What methods do we need on the client side (to handle (JavaScript) WebSocket events)?
  - onopen, onclose, onmessage, onerror
- How do you think we will go about providing our methods (callbacks / handler functions) so they can be used?
  - The nv-WebSocket Library specifies the functions, but we have to code them… how does this work in Java?
    - Interfaces vs Inheritance
    - Implements vs Extends

WebSocket Client Library For Java

- WebSocketListener Interface specifies 20+ methods, do you want to implement them all?
- Because we only want to write code for the functions we are interested in, the library provides an "empty" WebSocketAdapter class.
  - It implements WS listener interface with *stub methods* – methods with no actual code, ie:
    - void onConnected(WebSocket ws) {}
- We will extend the adapter, so that we don't have to write all the methods we aren't going to use.

- onStateChanged
- onConnected
- onConnectError
- onDisconnected
- onFrame
- onContinuationFrame
- onTextFrame
- onBinaryFrame
- onCloseFrame
- onPingFrame
- onPongFrame
- onTextMessage - String
- onTextMessage - Byte array
- onBinaryMessage

- onSendingFrame
- onFrameSent
- onFrameUnsent
- onError
- onFrameError
- onMessageError
- onMessageDecompressionError
- onTextMessageError
- onSendError
- onUnexpectedError
- handleCallbackError
- onSendingHandshake
- onThreadCreated
- onThreadStarted
- onThreadStopping

Android Permissions

- Manifest
  - Edit the AndroidManifest.xml to add permissions that the App user will have to agree to allow the App to have.
  - What permissions do Apps ask for?
    - Location, Bluetooth, Internet, Vibrate, Read/Write Calendar, Camera, Read/Write Contacts, Record Audio, Send/Read/Receive SMS, etc.
  - We are only interested in permission to use the network:
    - `<uses-permission android:name="android.permission.INTERNET" />`
  - If you add a permission after "installing" the app on the emulator, you must uninstall it and then run again (from Android Studio) in order for the resource to be allowed to be used. [Although sometimes it just works.]

Localhost

- The (App) Chat Client needs to talk to the server… what do we use to specify this?
  - *localhost*, but on the phone, localhost is the phone…
- The emulator can use a special IP address for the computer you are running on:
  - 10.0.2.2
  - ws://10.0.2.2:8080/endpoint
    - Note, the "endpoint" is necessary for this library (though any word works…)

Android Studio Output / Debugging

- Logcat – like the console of the web browsers
- Lots of "spam" is sent by the phone (emulator) to logcat, so you need to specify what to look for
- To send your own (debug) messages to the logcat, you use:
- Log.d("tag", "message")
  - "tag" will let us filter for our messages.
    - My tag is usually something like: "CC:mainActivity".
    - I can then filter on "CC:" to get only my messages, or "mainActivity" to get only messages for the main activity.
  - Warning, when you filter based on a "tag", you will not see system exception messages…

Android UI and Threading

- Google does not want Apps to freeze…
  - Or more specifically, their UIs to freeze.
- Threads
  - UI Thread – updating the View
    - For example, the `onCreate` (for the activity) method – can't do network stuff here
  - Other threads
    - Web Socket Thread (In our application)
    - Worker threads – things that take more than a split second (ie, that could cause the UI to freeze up)
- Need to run the WebSocket functions, including starting it up in its own thread. How can we do this?
  - Create our own Runnable class and add it to a thread… or:
- `ws.connectAsynchronously()`
  - Runs the handshake stuff in a different thread for us. ☺
- `ws.onTextMessage()`
  - If the WebSocket was started asynchronously, then all of its callbacks will be on that thread.
  - However, what do we want to occur when `onTextMessage()` happens?
    - Update the UI with the message…
  - runOnUIThread(Runnable)

Displaying the Chat Messages

- Start with a list of things you want to display:
  - `ArrayList<String> messages;`
- What type of widget will we use to display the messages?
  - A ListView
- Create an *Adapter* that will help the `ListView` display the `messages`:

```
adapter = new ArrayAdapter( this,
                            android.R.layout.simple_list_item_1,
                            messages );
```

- Attach the adapter to the `ListView`

```
lv_.setAdapter( adapter );
```

- Update the UI using the approach on the next slide.

Android – Running on the UI Thread

- runOnUIThread(Runnable)…
- View.post(Runnable)…
  - Both of the above do the same thing – they tell the application to run the given code (in the Runnable's run() method) on the UI thread as soon as it is available.
  - So, after adding a new message… (ie: adding a string to the `messages` variable (the see previous slide))

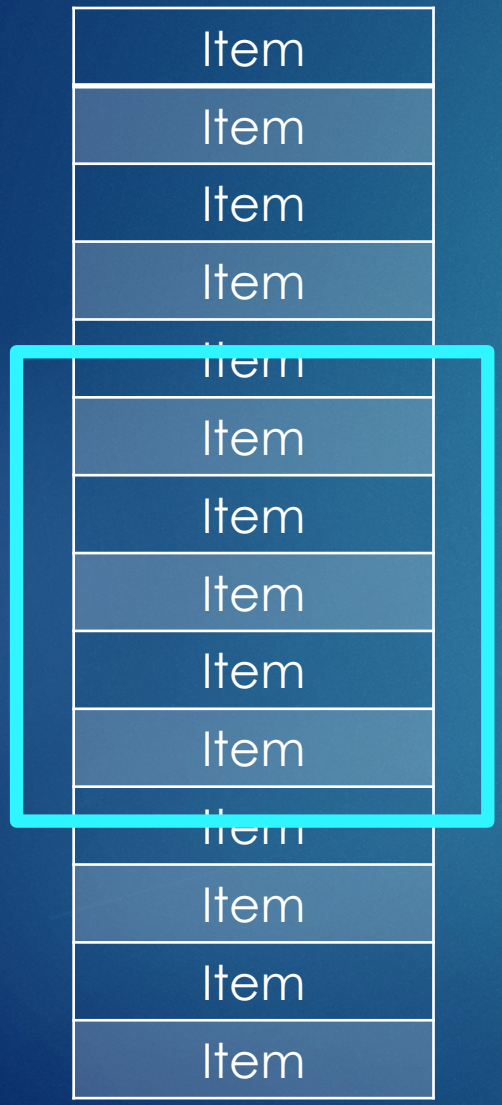
```
lv_.post( new Runnable() {
        @Override
        public void run() {
            chatListAdapter_.notifyDataSetChanged();
            lv_.smoothScrollToPosition( chatListAdapter_.getCount() );
        }
});
```

- What is `lv_`. Where does it come from?
  - A member variable.
  - A view (specifically a ListView) widget.
  - lv_ = findViewById(R.id.chatLV);
- Note, if you try to run UI related code on non-UI threads, then the update will be at best delayed, and at worst not happen at all.

Endless lists…

- Many phone apps have lists of things that go on forever…
  - Twitter
  - FaceBook
  - OurChatApp
- Android has a class that allows for "infinite" scrolling with a limited amount of memory.
- `RecyclerView` will manage this for you.
  - Elements that go off the "screen" will be automatically removed
- However, the `RecyclerView` is fairly complicated to use so I suggest just using a `ListView` – which for our purposes will work just fine.
  - Note, the `RecyclerView` requires the *androidx* Library…(which may already be listed in your Gradle dependencies.)

RecyclerView



Screen Window

- Layout Manager
  - Linear Layout Manager
- Adapter
  - Connection between the Data and the View
- A *Slot* – like an element in the view array
  - View Holder
    - Will contain the view for the item.
  - onCreateViewHolder
    - When recyclerView is making space for a view – create a view object to hold our information
  - onBindViewHolder(int position)
    - Called to "fill in the data" for a view in the list of views.
    - view.setText(message.get(pos))
- XML Inflater – turns XML specification of a View into an actual object.
- https://stackoverflow.com/questions/40584424/simple-android-recyclerview-example

Implicit Intents

- You can send/receive data from any App on your phone that supports the operation.
- You might want to add a "share" button that sends a message to whatever App the user chooses.

```
Intent sendIntent = new Intent();
sendIntent.setAction( Intent.ACTION_SEND );
sendIntent.putExtra( Intent.EXTRA_TEXT, "This is the message I'm sending." );
sendIntent.setType( "text/plain" );
startActivity( Intent.createChooser( sendIntent, getResources().getText( R.string.send_to ) ) );
```

Review Topics – Final Exam

- Cumulative, with a focus on the last two weeks.
- Anything we have discussed in lecture.
- Topics covered in labs / assignments.
- Threading
  - Definition / Purpose / Concurrency / Asynchronous Programs
  - Creation, Runnables, start / join
  - Synchronization / Critical Sections / Shared Variables
  - Communication between threads
- How the web server uses threads to serve multiple clients
- Use of Lambda functions (and/or classes)

Review Topics – Final Exam

- Web Sockets
  - Handshake
    - Request / Response Headers
  - Protocol
    - Message layout / headers / parsing / opcode / length / mask bit – mask bytes
  - Client calls (JavaScript, Java (Android), JavaScript vs Android)
  - Java WebSocket Library – Listener Interface vs Adapter vs MyHandler
    - Purpose of providing MyHandler to the WebSocket, how this works.
- Classes vs Objects
  - Member Variables vs Static Class Variables
  - Static methods
- Chat Server
  - Rooms – getRoom() method (Factory)
  - What does memory look like?

Review Topics – Final Exam

- Exceptions
  - try / catch – and what needs to be done inside of a catch block?
  - throws
- Android
  - Java vs XML
  - Activities / Intents
  - Views, Getting Views from the XML in Java
  - Permissions, Output / Debugging
  - UI Thread vs Other (Worker) Threads
    - runOnUIThread, View.post

Wednesday Assignments

- Code Review – As needed
- Assignment – Finish Android Chat Client

~ Fin ~