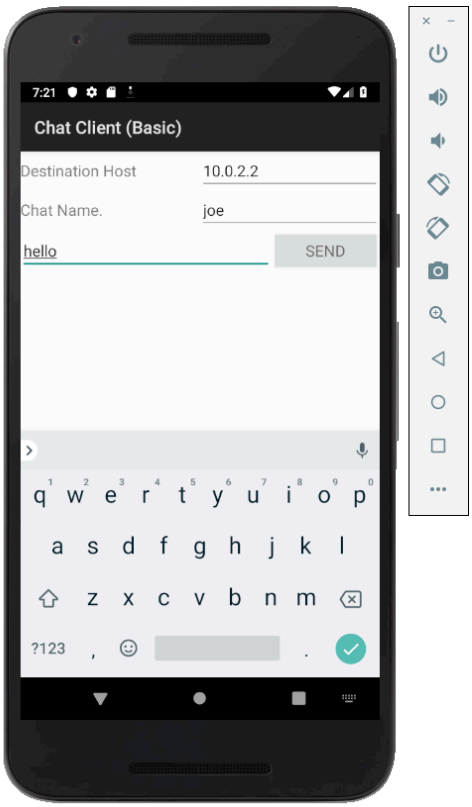
CS 522—Spring 2019
Mobile Systems and Applications
Assignment Two—Chat App UI

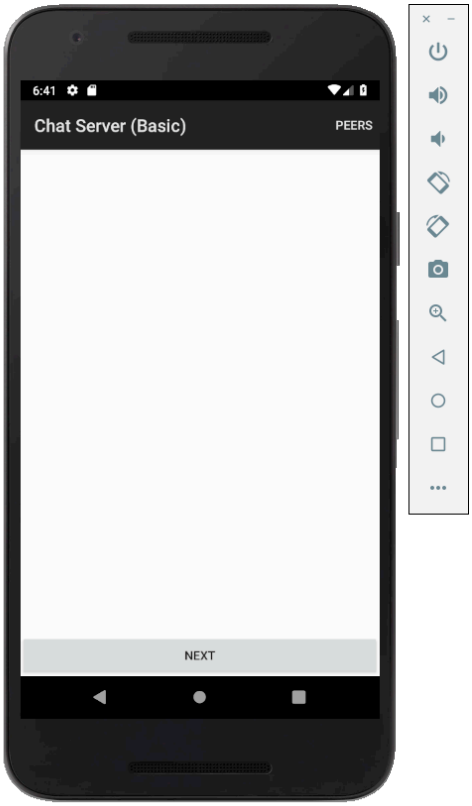
Target the Lollipop (Android 5.1, API 22) version of the Android platform for your submission for this assignment (i.e., set `minSdkVersion` to 22). Test your assignment on emulated Nexus 5X devices.

Since you will need to obtain permissions for the chat app, and with API 23 you are required to obtain permissions dynamically (in your code), set the `targetSdkVersion` to 22 for now. The `compileSdkVersion` should be set to the latest stable version of Android.

You will complete two apps so that they will speak to each other peer-to-peer using UDP sockets. These apps are very simple and violate some of the design guidelines for Android apps, such as not performing network communication on the main thread. We will see later how to fix this.

You are provided with two apps. ChatServer has a single button, Next. When you press it, it waits to receive a UDP datagram packet from the network on a designated port, and appends the message in the packet to a list of messages that are displayed on the screen. ChatClient has a message editor window and a button, Send. When you press Send, the contents of the message edit window are sent to the server.





On real devices, the client and server apps are intended to communicate by broadcasting over a Bluetooth or WIFI network that they are both connected to. Since the emulator does not implement the WIFI[1] or Bluetooth stacks, for the sake of this assignment we will just have the server bind to a server port, and the client then sends messages to that port. For machine addressing, the client will send to the loopback interface for the host machine on which both the client and server AVDs execute, and redirect messages to the server UDP port on the host machine to the corresponding port on the server AVD.

Every message that the client sends should have a name prepended to it (at the front of the message). On the server, the name is separated from each incoming requests. Currently the name is specified in the client UI. Eventually, in a future assignment, you will add an activity that allows this name to be defined as an app preference.

In addition, provide another UI (accessible using the options menu from the action bar) for displaying a list of the peers. This UI just lists peers by name in a list view. The list of peers is maintained in the main chat server activity, and passed to the subactivity for displaying the list of peers as a parcelable array list extra.

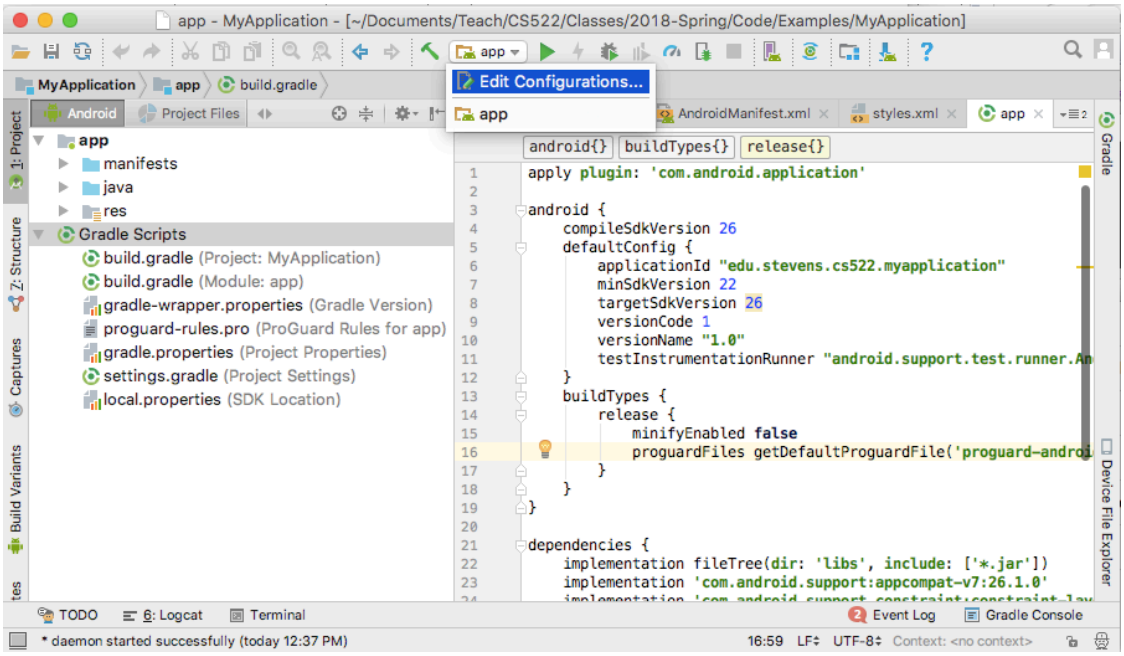
If the user selects one of these peers, then provide in a third UI information about that peer (their currently known IP address, and the last time that a message was received from that user). The peer information is passed as a parcelable extra to this list.

To prepare for future assignments, and get some practice with parcels and the `Parcelable` interface, define a peer entity class that contains the interesting state about a chat peer, and complete the implementation of the `Parcelable` interface for this. This involves writing a method that writes the state of the entity object to an output parcel, and a constructor that reconstitutes the state of an entity from an input parcel. Also complete the implementation of the `CREATOR` static field for the entity class, the implements the `Parcel.Creator<Peer>` interface. Although not necessary for this assignment, you should also complete the implementation of the `Message` entity class, that also implements the `Parcelable` interface. The library you are provided with includes utilities that you will find helpful: `DateUtils.readDate` and `DateUtils.writeDate` for dates, and `InetAddressUtils.readAddress` and `InetAddressUtils.writeAddress` for IP addresses. The latter uses the Guava library to avoid DNS lookup in the standard `InetAddress` constructors!

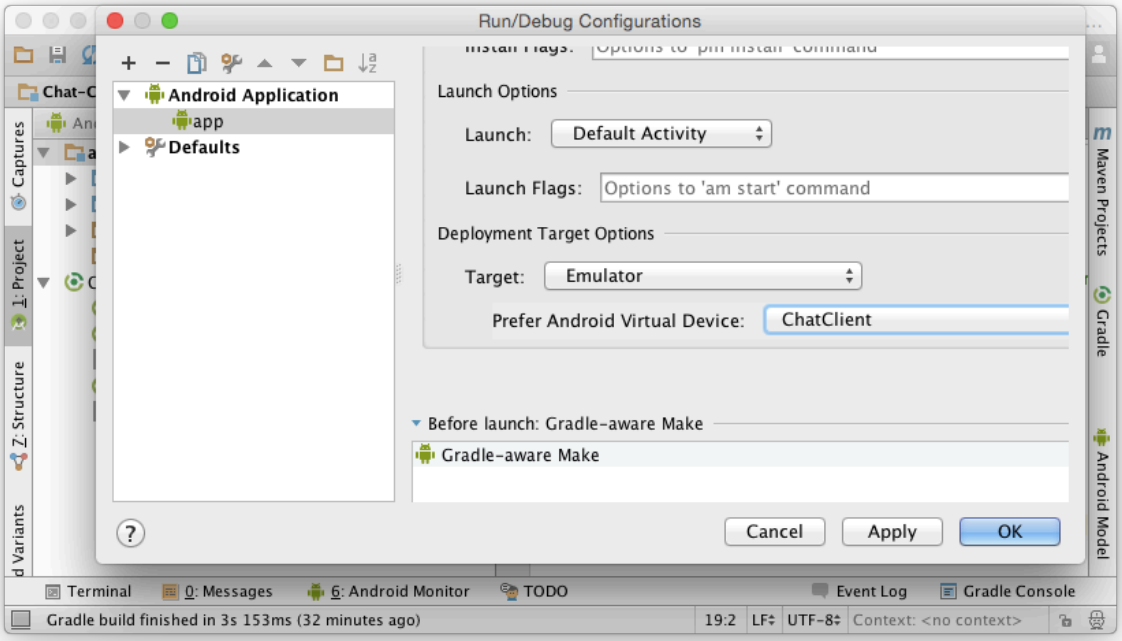
You should follow this strategy to get the client and server to talk to each other:

1. Create separate virtual devices (AVDs) for the client and server. Let's say you call these ChatClient and ChatServer, respectively.
2. In the view for the ChatClient project, lick on the "app" drop-down menu and pick "Edit Configurations":

[1] The emulator with a system software stack for version 25 or greater does support WIFI, but we will work with point-to-point communication rather than debugging WIFI broadcast.



For the Device Target Option, choose Emulator and then pick ChatClient as the device to run the application on:



This is to make sure that the client app runs on the ChatClient device.

3. Similarly, make sure that the server app runs on the ChatServer device.

4. Run the server and client chat apps. Assuming that you ran the server first, followed by the client, the corresponding AVDs have administrative port numbers 5554 and 5556, respectively.
5. The apps are not yet communicating, because they are running on their own network stacks. In particular, the server has bound to a UDP port on the server guest machine. You now need to bind that to a UDP port on the host machine upon which you're running these two AVDs. This is easy to do. The server binds to UDP port 6666 on its guest Ethernet interface (10.0.2.15), and the client will try to send messages to port 6666 on the host loopback interface (10.0.2.2 on the AVD, 127.0.0.1 on the host machine). Use the `adb` command in the platform-tools subdirectory of the Android installation to perform this redirection[2], by typing this line in the OS (bash or Windows) shell:

```
adb –s emulator-5554 forward tcp:6666 tcp:6666
```

6. You may want to change the first port (the first 6666, the chat server port on the host machine) if you are running on a machine where another client has bound to that address. You don't need to change the second 6666 since that is the server port on the guest device.

You can find out more about network redirection here:
https://developer.android.com/studio/run/emulator-networking.html#connecting

The client and server apps include some functionality in an external module, imported as an AAR file[3]. The settings.gradle file should include this line:

```
include ':app', ':cs522-library'
```

The build.gradle file for the app should include these dependencies:

```
    implementation project(":cs522-library")
    implementation group: 'com.google.guava', name: 'guava', version:
'26.0-android'
```

Note: You must make sure to declare in the manifest the permissions that the app will require for network and internet access. See the Android documentation for more information about which permissions you require.

Submitting Your Assignment

Once you have your apps working, please follow these instructions for submitting your assignment:

1. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory Humphrey_Bogart.
2. In that directory you should provide the Android Studio projects for your Android apps.

---

[2] Connecting to the AVD via telnet and using the `redir` command does not appear to work. Also, we are forwarding TCP rather than UDP, and using a class `DatagramSendReceive` that provides a datagram socket API wrapped around TCP, to work around an apparent bug in the emulator.

[3] To add this module into an existing project, click `File | New | New Module` and select Import .JAR/.AAR Package.

3. Also include in the directory a completed rubric for your assignment. This is to help you self-evaluate, though a falsified rubric is grounds for (potentially significant) penalty points.
4. In addition, record short MP4 or Quicktime videos (*note the allowable formats*) demonstrating your deployments working. Make sure that your name appears at the beginning of the video. For example, put your name in the title of the app. *Do not provide private information such as your email or cwid in the video*. Be careful of any "free" apps that you download to do the recording, there are known cases of such apps containing Trojan horses, including key loggers.
5. Your video should record client devices sending multiple messages and those messages being received on the server. You can use one client device to send messages with different chat names. Include both client and server devices, running on the emulator, in a video. Show the server list of peers subactivity being launched, and the details for a single peer activity being launched from this.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have three Android projects, for the apps you have built, as well as the completed rubric and videos demonstrating your app working.