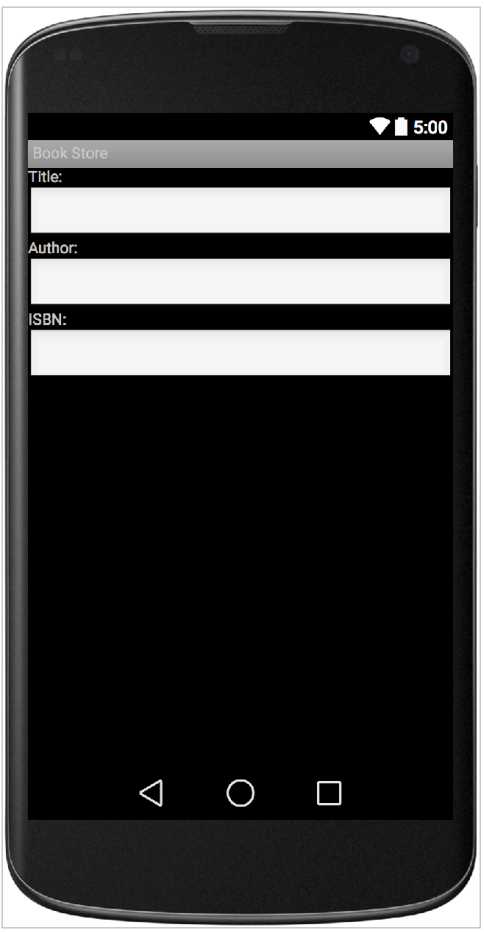
CS 522—Fall 2018
Mobile Systems and Applications
Assignment Two—Bookstore & Chat App

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 in the second part of the assignment, 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.

Part 1: Bookstore

You are given an implementation of a bookstore application for an Android device. You should complete this project and submit it. You can view the parts to be completed by looking for TODO items in the provided project. The project provides an activity for a book store shopping cart, with sub-activities for adding a book, viewing the details of an individual book, and checking out. The add activity just returns the book information entered, while checkout just asks for shipping and billing information and then clears the shopping cart.



In the activity for displaying a list of books, the adapter should be of type `ArrayAdapter<Book>`, where `Book` is an entity object for a book's state (title, author(s), ISBN and price). You will need to define a custom adapter that displays the fields of a book entity object in each row of the list view:

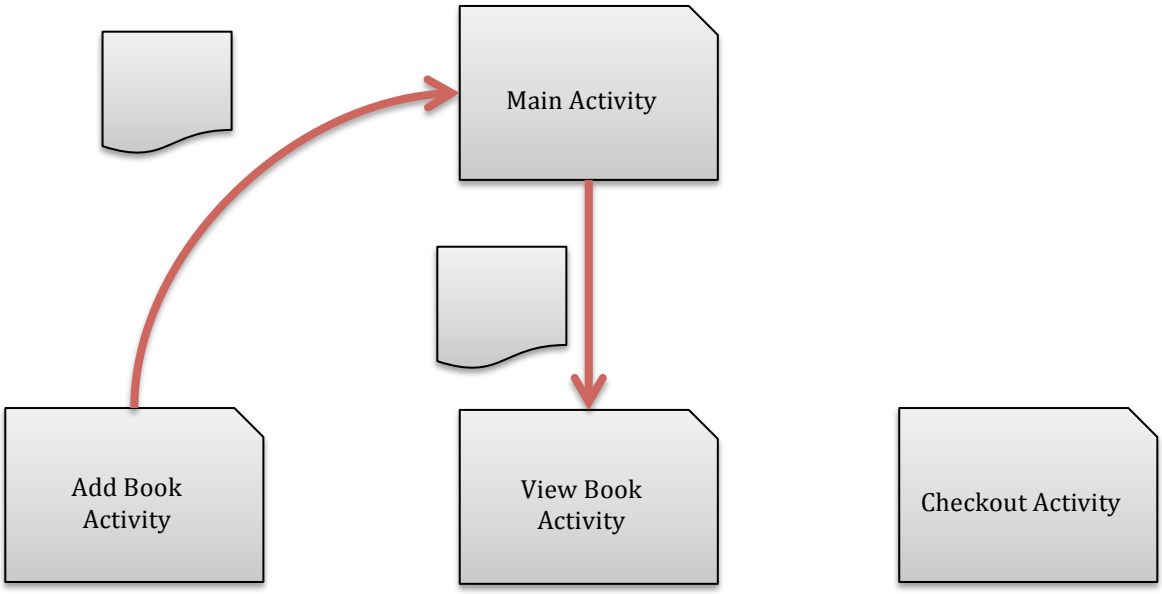
```
public class BooksAdapter extends ArrayAdapter<Book> {
    public BooksAdapter(Context context, ArrayList<Book> books) {
       super(context, 0, books);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
       // Get the data item for this position
       Book book = getItem(position);
       // Check if an existing view is being reused, otherwise inflate the view
```

```
        if (convertView == null) {
           convertView = LayoutInflater.from(getContext())
                                 .inflate(R.layout.cart_row, parent, false);
        }
        // Lookup view for data population
        TextView title = (TextView) convertView.findViewById(R.id.cart_row_title);
        TextView author = (TextView) convertView.findViewById(R.id.cart_row_author);
        // Populate the data into the template view using the data object
        title.setText(book.title);
        author.setText(...);
        // Return the completed view to render on screen
        return convertView;
    }
}
```

There are three subactivities of the main activity for viewing the shopping cart:

- `AddBookActivity` displays a screen for entering the details of a book that should be added to the shopping cart. This is invoked by an action in the main activity.
- `ViewBookActivity` displays the details for a particular book. This is invoked by clicking on a book in the list view in the main activity.
- `CheckoutActivity` just displays a blank screen, and allows the shopping cart to be cleared. This is invoked by an action in the main activity.

You need to add option menus for the `AddBook` and `Checkout` activities, as well as handling the launching of sub-activities and the passing of data back and forth with intents. The following picture illustrates the flow of data between activities:



The main activity launches the activity for adding a book in the action bar menu handler:

```
public boolean onOptionsItemSelected(MenuItem item) {
     super.onOptionsItemSelected(item);
     switch(item.getItemId())
        case R.id.add
           Intent addIntent = new Intent(this, AddBookActivity.class);
           startActivityForResult(addIntent, ADD_REQUEST);
           break;
        ...
     }
     return false;
}
```

When the subactivity returns to the main activity, the book that has been added should be an extra in the result intent:

```
protected void onActivityResult(int reqCode, int resultCode, Intent intent) {
    super.onActivityResult(reqCode, resultCode, intent);
    switch(reqCode) {
       case ADD_REQUEST:
          if (resultCode == RESULT_OK) {
            Book result = intent.getParcelableExtra(AddBookActivity.BOOK_KEY);
            ...
          }
          break;
    }
}
```

Remember to update the adapter when adding a book to the shopping cart, by calling `notifyDataSetChanged()`. When viewing a book, the book to be viewed is passed as an extra to the intent used to launch the subactivity:

```
Intent viewIntent = new Intent(this, ViewBookActivity.class);
viewIntent.putParcelableExtra(ViewBookActivity.BOOK_KEY, book);
startActivity(viewIntent);
```

You will need to implement the `Parcelable` interface for `Book` and `Author` classes. When deserializing a book object from a parcel, you will need to deserialize a list of author objects. Do this by passing the classloader for the `Author` class to the operation for reading a parcelable object:

```
public class Book implements Parcelable {
   public Book(Parcel in) {
      …
      in.readParcelable(Author.class.getClassLoader())
      …
   }
}
```

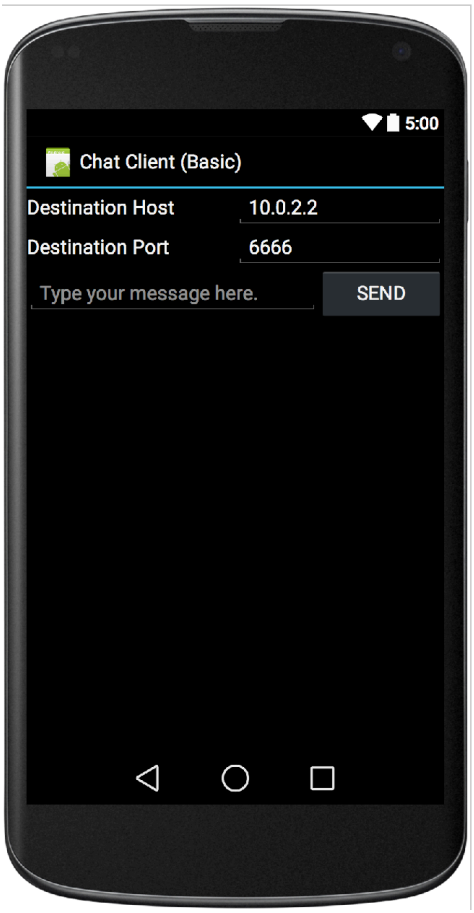
Remember that, for both `Book` and `Author` classes, you should define a static `CREATOR` field with factory methods.

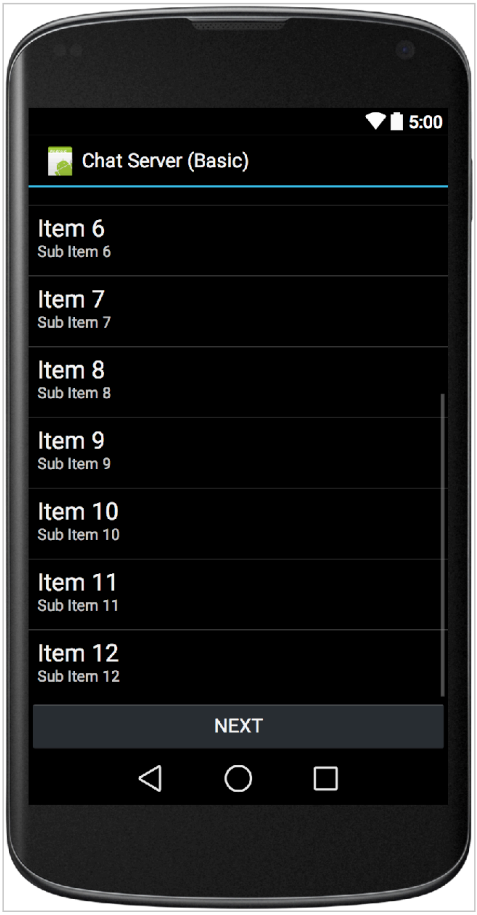
An activity for viewing the details of an individual book may be launched by selecting a book from the list view in the root activity. However you should also support the ability to delete a book that is selected from the list view. There are a couple of ways to accomplish this. One way is to define a context menu for the list view (as distinct from an options menu). The context menu pops up when an item is selected, and offers the choice of viewing or deleting the selected item. You should provide this interface for deleting a book from the cart. In future assignments , you will explore other approaches using contextual action mode.

Part 2: Basic Chat App

In this second part of the assignment, 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.

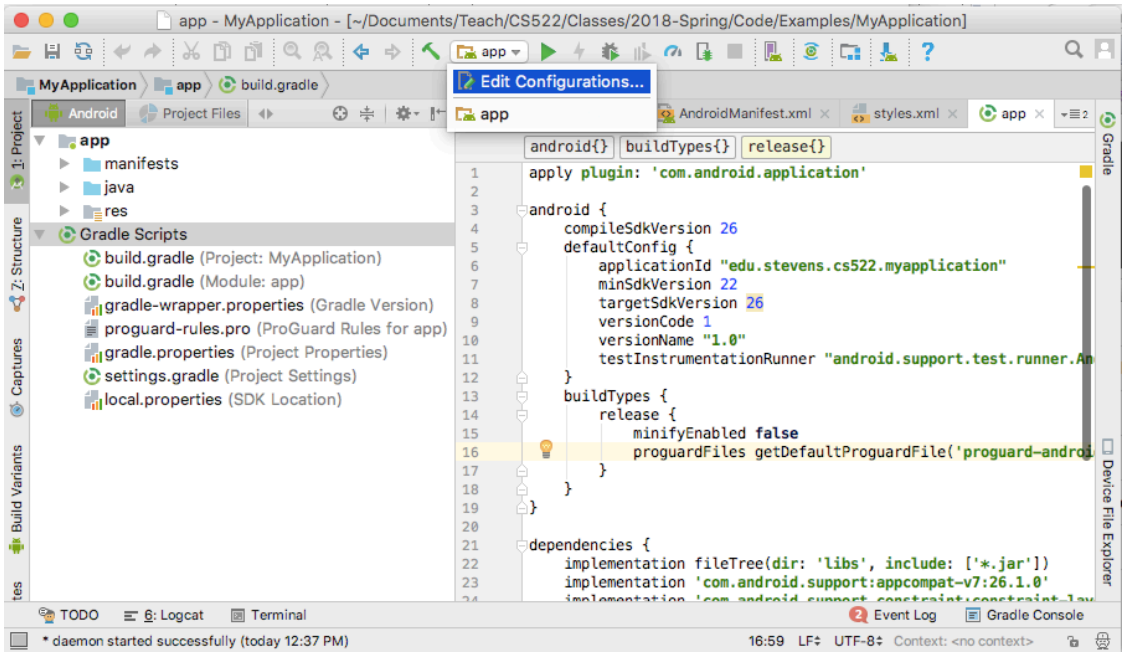
Every message that the client sends should have a name prepended to it (at the front of the message). On the server, the name is separated from each incoming requests. Currently it is okay to define this name as a static string constant in the client project. Eventually, in a future assignment, you will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

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

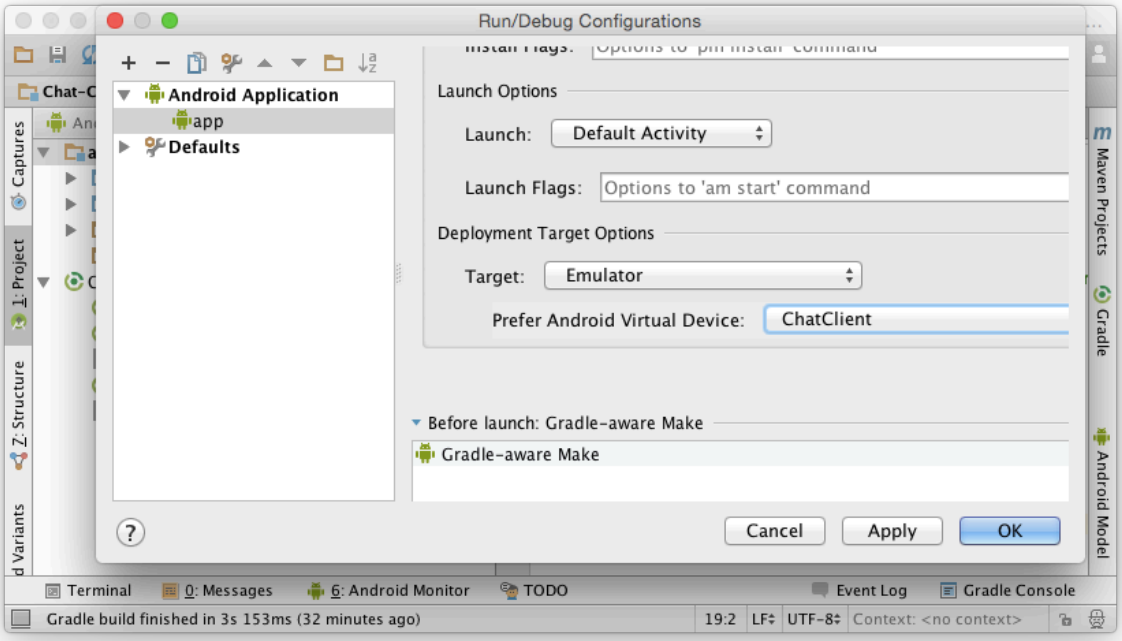
1. Create separate virtual devices (AVDs) for the client and server. Let's say you call these ChatClient and ChatServer, respectively.

[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.

2. In the view for the ChatClient project, lick on the “app” drop-down menu and pick “Edit Configurations”:



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. ~~Telnet to the *server* AVD administrative console, and use the redir command to redirect from a host machine port to the server guest port. 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). Type these lines in a shell on the host:~~

   ```
   telnet localhost 5554
   redir add udp:6666:6666
   quit
   ```

   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'
```

---

[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.

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.
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. For the bookstore app, your video should record adding at least two books to the shopping cart, viewing the details for a book, deleting a book, and checking out. For the chat app, your video should record the client device sending multiple messages being sent and received. Include both devices, running on the emulator, in a video.

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.