**Exercise Handling JSON in Android**

**Step 1: Examining the Project Files**

1. Download the [ChatClientJson.zip](http://w02.hkvu.hk/edX/COMP107x/w5/ChatClientJson.zip) file, unzip it and import the project into Android Studio.
2. Open the **friend\_item.xml** file.
3. Open the **Contacts.java** file and the **MyArrayAdapter.java** file.
4. Follow the video instructions.

**Step 2: Updating the Contacts Activity**

1. Open the **Contacts.java** file and navigate to the *processFriendInfo()* method.
2. Replace the first // TODO comment with the following code.
3. GsonBuilder gsonBuilder = new GsonBuilder();
4. Gson gson = gsonBuilder.create();
5. Replace the second // TODO comment with the following code.

friendInfoList = new ArrayList<FriendInfo>();

friendInfoList = Arrays.asList(gson.fromJson(infoString, FriendInfo[].class));

1. Run the app and see the result.

Exercise Asnyctask for background production

**Step 1: Examining the Project Files**

1. Download the [ChatClientAsync.zip](http://w02.hkvu.hk/edX/COMP107x/w5/ChatClientAsync.zip) file, unzip it and import the project into Android Studio.
2. Open the **Contacts.java** file.
3. Open the **MyArrayAdapter.java** file.
4. Follow the video instructions.

**Step 2: Updating the Contacts Activity**

1. Replace the // TODO comment in Contacts.java in the *onCreate()* method with the following code:
2. **FriendsProcessor** mytask = new **FriendsProcessor**();
3. mytask.execute("friendsjson.txt");
4. Replace the // TODO comment in Contacts.java file in the AsyncTask class'*doInBackground()* method with the following code:

inputStream = mContext.getAssets().open(strings[0]);

in = new BufferedReader(new InputStreamReader(inputStream));

String readLine;

StringBuffer buf = new StringBuffer();

while ((readLine = in.readLine()) != null) {

buf.append(readLine);

}

// Convert the **read** in information to a Json string

String infoString = buf.toString();

// now process the string using the method that we implemented in the previous exercise

processFriendInfo(infoString);

if (null != in) {

in.close();

}

**Step 3: Updating MyArrayAdapter Class**

1. Replace the // TODO comment in **MyArrayAdapter.java** file with the following code:
2. ImageView imageView = (ImageView) friendInfoView.findViewById(R.id.avatar);

        Picasso.with(context).load("file:///android\_asset/"+friendInfoList.get(position).imageURL).into(imageView);

1. Run the app and see the result.

Setting up server node js

### Step 1: Installing NodeJS

To install NodeJS on your machine, go to [https://nodejs.org](https://nodejs.org/) and click on the Install button. Depending on your computer's platform (Windows, MacOS or Linux), the appropriate installation package is downloaded. Follow along the instructions to install NodeJS on your machine.

### Step 2: Downloading Server Configuration Files

Download [server.zip](http://w02.hkvu.hk/edX/COMP107x/w5/server.zip) and unzip it at a convenient location on your machine. This will create a sub-folder named **server** on your machine that contains the configuration files for running a server on your machine. Move to this folder on your machine.

### Step 3: Setting up JSON-Server

1. Open a terminal window on your machine. If you are using a Windows machine, open a cmd window or PowerShell window with admin privileges.
2. To ensure that your NodeJS setup is working correctly, type the following at the command prompt to check for the version of **Node** and **NPM**.
3. node -v
4. npm -v
5. At the command prompt type the following to install a simple JSON server package in NodeJS.
6. npm install -g json-server

Note: On MacOS and Linux you may need to add **sudo** in front of the command. When you are prompted by sudo, just type in your password in the window to proceed with the installation.

### Step 4: Starting your Server

1. To start your server, move to the server folder that you created by unzipping the configuration files above, then type the following at the command prompt.

json-server *--watch db.json*

1. To check that your server is running correctly and serving up the JSON string, type the url below into any browser window.  
     
    **http://localhost:3000/people**
2. Note down the **IP address** of your computer because you need to configure this in the code of the next exercise.
3. You can also cross-check by typing the url below into a browser window to ensure that your server is running correctly. This should show the JSON string in your browser window.  
     
   **http://<Your machine's IP address>:3000/people**

#### Finding your Machine's IP address

1. If you are using a Microsoft Windows based machine, you can use the instructions available at<http://windows.microsoft.com/en-us/windows/find-computers-ip-address#1TC=windows-7> to find your machine's IP address.
2. On a Mac, go to System Preferences->Network to find the IP address of your network adapter.

Exercise Android Http

**Step 1: Examining the Project Files**

1. Download the [ChatClientNetwork.zip](http://w02.hkvu.hk/edX/COMP107x/w5/ChatClientNetwork.zip) file, unzip it and import the project into Android Studio.
2. Open the **Contacts.java** file.
3. Open the **AndroidManifest.xml** file.
4. Follow the video instructions.

**Step 2: Adding User Permissions to the Android Manifest file**

Open the AndroidManifest.xml file in the manifests folder. Since we are accessing a server using the Internet connectivity, we need to request for permission in the manifest file. In addition, since we will be examining the network state, we need to ask for permission to access the network state. This is accomplished by including the following permission requests in the manifest file:

<**uses***-permission* *android:name*="android.permission.INTERNET"/>

<**uses***-permission* *android:name*="android.permission.ACCESS\_NETWORK\_STATE" />

**Step 3: Updating the Contacts Activity**

1. Replace the // TODO comment in the Contacts.java file in the AsyncTask class'*doInBackground()* method with the following code:
2. **HttpURLConnection** urlConnection = null;
3. **Integer** result = 0;
4. try {
5. */\* forming th java.net.URL object \*/*
6. **URL** url = new **URL**(urls[0]);
7. urlConnection = (**HttpURLConnection**) url.openConnection();
8. */\* optional request header \*/*
9. urlConnection.setRequestProperty("Content-Type", "application/json");
10. */\* optional request header \*/*
11. urlConnection.setRequestProperty("Accept", "application/json");
12. */\* for Get request \*/*
13. urlConnection.setRequestMethod("GET");
14. **int** statusCode = urlConnection.getResponseCode();
15. */\* 200 represents HTTP OK \*/*
16. if (statusCode == 200) {
17. inputStream = new **BufferedInputStream**(urlConnection.getInputStream());
18. *// Convert the read in information to a Json string*
19. **String** response = convertInputStreamToString(inputStream);
20. *// now process the string using the method that we implemented in the previous exercise*
21. processFriendInfo(response);
22. result = 1; *// Successful*
23. } else {
24. result = 0; *//"Failed to fetch data!";*
25. }
26. Run the application to see the result.

**Hints**

1. The images are available on the server at the URL http://<Your Server's IP address>:3000/<Image URL from the FriendInfo object>, e.g., http://<Your Server's IP address>:3000/John.png.
2. Use Picasso's image transformations to fit the image into the ImageView:
3. **Picasso**.with(context)
4. .load(url)
5. .resize(50, 50)
6. .centerCrop()
7. .into(imageView)
8. Use Picasso's place holder and error place holders as follows. Use R.mipmap.ic\_launcher for place holder and error place holder:
9. **Picasso**.with(context)
10. .load(url)
11. .placeholder(**R**.drawable.user\_placeholder)
12. .error(**R**.drawable.user\_placeholder\_error)

.into(imageView);