CS 160 User Interface Design

Message Passing & APIs

Section 05 // September 25th, 2015

Tricia Fu // **OH** Monday 9:30 - 10:30am // triciasfu@berkeley.edu

Agenda

1. Administrivia
2. Brainstorm Discussion
3. Message Passing Between Wear and Mobile
4. Twitter API

Administrivia

Design Assignment 02	Due October 1st by 10:30am
Design Assignment 03	Due October 1st by 10:30am
Programming Assignment 02	Due October 16th by 11:59pm
Phones!	

Last day TAs are helping with emulator setup next Thursday 10/1

This will be the last painful Android section! Yay!

Brainstorm Discussion

How did it go?

What target audiences did you think about?

Cool ideas?

Feasible within the scope of the class

Progress?

Action Items:

Get to know each other!

Find common causes that you care about

Discuss your schedules

Team Development

Forming	The team act as individuals and there is a lack of clarity about the team's purpose and individual roles.
Storming	Conflict arises as people begin to establish their place in the team.
Norming	There is a level of consensus and agreement within the team. There is clarity about individual roles. The role of the leader is important in managing this.
Performing	The group has a clear strategy and shared vision. It can operate autonomously and resolve issues positively.

Adapted from Tuckman 1965

Fig. 1 Tuckman's Model

Bringing it back – Critiques

Design Critique

- "Unlike a brainstorming meeting, where the goal is to come up with new ideas, a critique meeting is focused on evaluating a set of existing ideas, and possibly identify future directions or changes."

- When are these performed?

- Why are we discussing this now?

- Learn to be critical!

Activity – Group Critique

- Discuss with the person next to you one problem that you might see with each of these brainstormed smartwatch ideas, and how you might modify it to be better.
    - Music Smartwatch App
    - Restaurant Smartwatch App for Waiters
    - Safety Notification App
    - Pet Tracking Smartwatch App
    - Parking App

Wear-Handheld Communication

The Wearable Data Layer

- **Wearable Data Layer (WDL)**: a communication channel between your handheld and wearable apps.
- “The data layer APIs are the only ones you should use to set up communication between wearable and handheld” - Developers Guide (enlighten yourself)

https://developer.android.com/design/wear/principles.html

Main Guide

http://developer.android.com/training/wearables/data-layer/index.html

The following slides show a simplified version!

Wear to Handheld in 5 Steps

Wear to Handheld in 5 Steps

Wear	Handheld
	1. Broadcast capability C
2. Initialize Google API Client	
3. Discover node N with capability C	
4. Send message M to node N	
	5. Handle message M

Step 1

Listen up, world:
I can do this
important thing.

Create a wear.xml with the capability

```
<resources>
    <string-array name="android_wear_capabilities">
        <item>do_stuff</item>
    </string-array>
</resources>
```

Name of capability

Do this in the *mobile* app.

Make the file at res/values/wear.xml

Steps 2 - 4: Send a Message

Is there anyone out there?

I’ve got something to say!

Step 2: Send a Message

GoogleApiClient: the main entry point for any of the Google Play services APIs

APIs include:

- **CapabilityAPI**: find nearby nodes with a "capability" (e.g., the handheld with the capability "do_stuff")
- **MessageAPI**: sends messages

Creating a Google API

We make an API Client *builder*, and then build the client.

```
this.mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(new
GoogleApiClient.ConnectionCallbacks() {
            @Override
            public void onConnected(Bundle bundle) {
                  // Do something
            }
            @Override
            public void onConnectionSuspended(int i) {
                  // Do something
            }
        })
        .addOnConnectionFailedListener(new
GoogleApiClient.OnConnectionFailedListener() {
            @Override
            public void onConnectionFailed(ConnectionResult
connectionResult) {
                  // Do something
            }
        })
        .addApi(Wearable.API)
        .build();
this.mGoogleApiClient.connect();
```

We connect specifically to the Wearable API

Creating a Google API

```
this.mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(new
GoogleApiClient.ConnectionCallbacks() {
            @Override
            public void onConnected(Bundle bundle) {
                // Do something
            }
            @Override
            public void onConnectionSuspended(int i) {
                // Do something
            }
        })
        .addOnConnectionFailedListener(new
GoogleApiClient.OnConnectionFailedListener() {
            @Override
            public void onConnectionFailed(ConnectionResult
connectionResult) {
                // Do something
            }
        })
        .addApi(Wearable.API)
        .build();
this.mGoogleApiClient.connect();
```

We make callbacks that say what to do when we connect to the API, and when that connection changes.

With the exception of "onConnected", you can leave these blank

Make sure to run connect(), or else no connection will be made and the 'onConnected' callback will never be run.

Step 3: Discovering the Handheld

Remember the capability the handheld is broadcasting from its *wear.xml*.

Now we have to find the handheld by findings what nodes are broadcasting the capability.

Use the CapabilityApi from the Google Play Services APIs to find nodes with capabilities

This is the Google API client we created in Step 2. It's needed for the call for nodes with the capability.

Finding Nodes with Capabilities

```
CapabilityApi.GetCapabilityResult capResult =
    Wearable.CapabilityApi.getCapability(
        mGoogleApiClient, CAPABILITY_NAME,
        CapabilityApi.FILTER_REACHABLE)
    .await();
```

Search for a specific capability…
on nodes that are "reachable" or nearby

The result of the call contains, among other things, a list of Nodes with the capability that was asked for.

Step 4: Sending the Message

Node: a connected device on the network

MessageApi: a class for remote procedure call (RPC) or starting intents on the wearable from the handheld or vice versa

This time we use the MessageApi instead of the CapabilityApi

The *nodeId* is the address of the receiver of the message.
You can find the nodeId of the receiver from the "capability result" we got from Step 3

```
Wearable.MessageApi.sendMessage(
        mGoogleApiClient, nodeId,
    RECEIVER_SERVICE_PATH, new byte[3]
);
```

RECEIVER_SERVICE_PATH is a string that specifies a routine that will get invoked on the receiver's side.
Set this value to a descriptive name of your choosing and save it for Step 5.

This is the data we're sending.
I just added a placeholder of empty bytes.

Step 5: Receive a Message

What do I hear from the world?

WearableListenerService: a **class** for services that listen for important data layer events

```
public class MyReceiverService extends WearableListenerService {

    private static final String RECEIVER_SERVICE_PATH = "/
receiver-service";

    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public void onMessageReceived(MessageEvent messageEvent) {
        Log.i(TAG, "Got a message");
    }

}
```

Remember me from Step 4? I'm the same value, but on a different device!

Extend WearableListenerService and override onMessageReceived. Do your message reaction in onMessageReceived.

Finally, update your Manifest!

Name of our listener service

```
<service android:name=".MyReceiverService" >
    <intent-filter>
        <action android:name=
        "com.google.android.gms.wearable.BIND_LISTENER" />
    </intent-filter>
</service>
```

This intent-filter stuff lets incoming messages notify our WearableListenerService that they've arrived.

Fabric + Twitter API

API Calls

- Treat all of Twitter's data as a Model in MVC
- Receive data in JSON format
- Calls are typically *asynchronous*

API Calls


Twitter API @twitterapi · 26 Sep 2013
Today, Twitter is updating embedded Tweets to enable a richer photo experience: blog.twitter.com/2013/rich-phot…
Rich photo experience now in embedded Tweets | …
Every day, publishers and journalists all over the world share the best of Twitter with their readers by embedding Tweets in their articles. Perhaps not surprisingly, man...
blog.twitter.com
RETWEETS
137
FAVORITES
97
12:02 PM - 26 Sep 2013 · Details
Reply to @twitterapi

API Calls

JSON Example

```
{
 ...
 "text": "Today, Twitter is updating embedded Tweets to
enable a richer photo experience:
https:\/\/t.co\/XdXRudPXH5",
 "entities": {
  "hashtags": [],
  "symbols": [],
  "urls": [{
   "url": "https:\/\/t.co\/XdXRudPXH5",
   "expanded_url": "https:\/\/blog.twitter.com\/2013\/rich-
photo-experience-now-in-embedded-tweets-3",
   "display_url": "blog.twitter.com\/2013\/rich-phot\u2026",
   "indices": [80, 103]
  }],
  "user_mentions": []
 }
}
```

← getTweet(tweetId, callback { … });

→

```
callback(tweetData) {
    if (success) { print(tweetData); }
    if (fail) {print("failure!");}
}
```

Example Call

Create a service from the TwitterCore module

```
StatusesService statusesService = twitterApiClient.getStatusesService();

statusesService.show(524971209851543553L, null, null, null, new
    Callback<Tweet>() {
    @Override
    public void success(Result<Tweet> result) {
        //Do something with result, which provides a Tweet in result.data
    }

    public void failure(TwitterException exception) {
        //Do something on failure
    }
});
```

Sends a request to Twitter for a tweet with a given ID

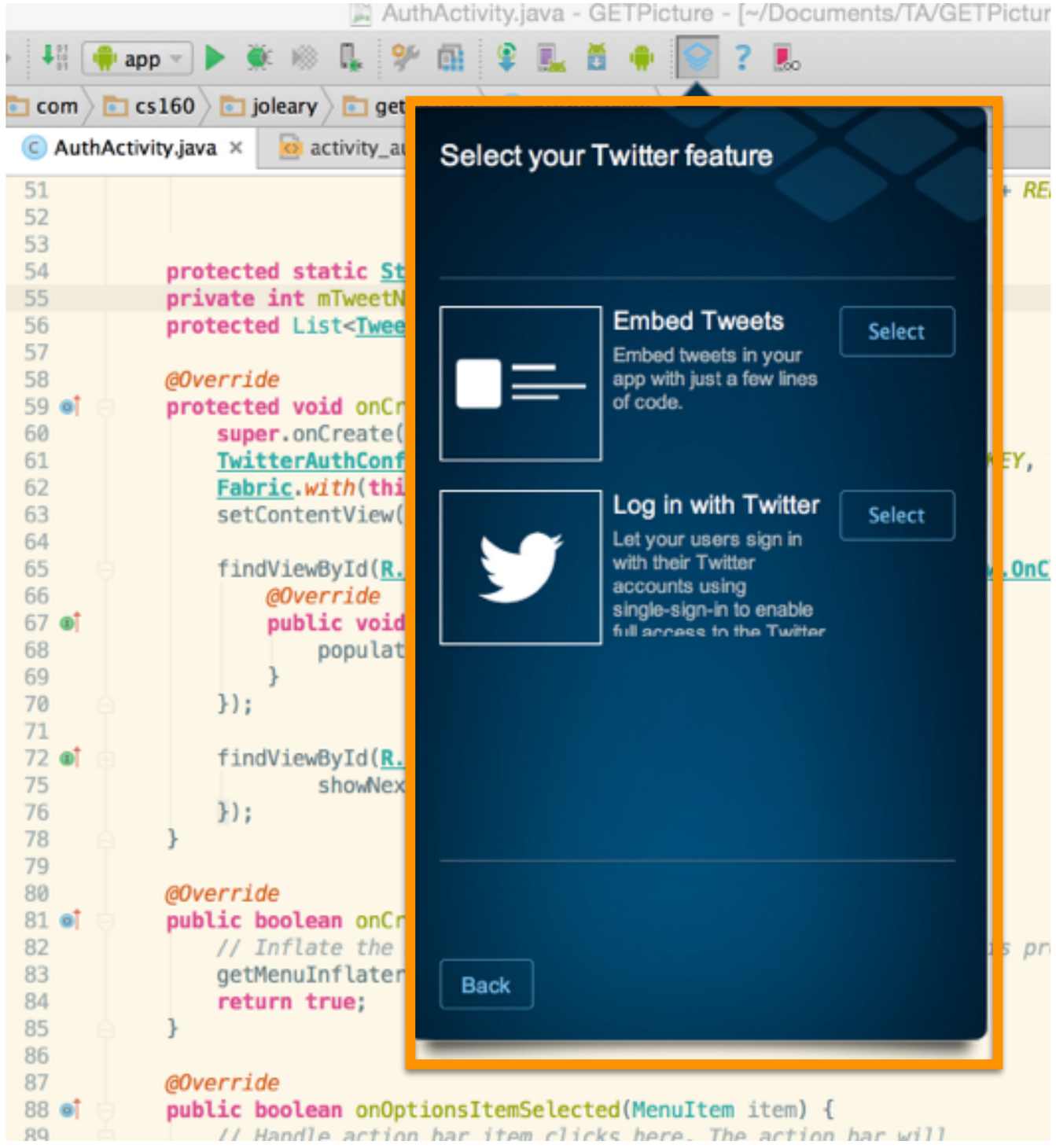
Code that runs once Android receives a response eventually

What does Fabric do for me?

- Create a developer account, get an Android Studio plugin

- *Handles Authentication (OAuth 2.0) for you*

- Auto-adds ~setup code. No need to copy/paste. Get straight to API calls!



Fabric Setup Video

CS160 API Reference Sheet

http://bit.ly/1YEEgrG

This concludes all the Android topics you need to know!

CS 160 User Interface Design

See you next week!

Questions?

Tricia Fu // **OH** Monday 9:30 - 10:30am // triciasfu@berkeley.edu