

CS160 Section 7

10-07-2015

Android Backend & Intro to Illustrator

Agenda

- Administrivia
- Android Bits & Pieces
- Intro to Illustrator
- Flowcharting a Wear app
- Flowchart to Code implementation

Administrivia

- PROG2: YourFault due Friday, 10/16
- DESIGN4: Project idea due Friday, 10/16
- RR5: Due Thursday, 10/15
- Midterm: Thursday, 10/22
- API doc updated with Twitter API

Android Bits & Pieces

Intents

- Asynchronous messages that allow Android components to request functionality from other components

- After defining an intent, send it to the Android system

`startActivity(Intent)` to launch an Activity

`broadcastIntent(Intent)` to send to any interested

BroadcastReceiver components

`startService(Intent)` to communicate with a background Service.

- Starting an activity from another activity:

```
Intent i = new Intent(this, ActivityTwo.class);
startActivity(i);
```

Data Transfer with Intents

- Intents can contain data in the form of a Bundle (using key-value pairs)

- Adding data to an intent:

```
Intent i = new Intent(this, ActivityTwo.class);  
i.putExtra("Value1", "Value one for ActivityTwo ");  
i.putExtra("Value2", "Value two ActivityTwo");  
startActivity(i);
```

- Retrieving data from an intent

```
Bundle extras = getIntent().getExtras();  
if (extras == null) {  
    return;  
}  
// get data via the key  
String value1 = extras.getString(Intent.EXTRA_TEXT);  
if (value1 != null) {  
    // do something with the data  
}
```

Android & RESTful APIs

- High level approach:
 - Create HttpURLConnection
 - Make a GET/POST request
 - Store response in string
 - Close connection
 - Parse response using JSONObject

URLConnection

1. Create your URL

```
URL url = new URL("http://www.example.com/?exampleparam=10-09-2015");
```

2. Open your Connection

```
URLConnection connection = (URLConnection) MyURL.openConnection();
```

3. Read the response

```
InputStream in = new BufferedInputStream(connection.getInputStream());  
BufferedReader reader = new BufferedReader(new InputStreamReader(in));  
StringBuilder sb = new StringBuilder();  
String line;  
while ((line = r.readLine()) != null) {  
    sb.append(line);  
}  
stream = sb.toString();
```

4. Disconnect

```
urlConnection.disconnect();
```


JSON Parsing

- Two forms of storage: key-value pairs, arrays

- Create JSONObject from string

```
JSONObject jsonObj = new JSONObject(result_str);
```

- Get String

```
String surname = jsonObj.getString("surname");
```

- Get nested JSONObject

```
JSONObject subObj = jsonObj.getJSONObject("address");  
String city = subObj.getString("city");
```


- Parsing a JSONArray

```
JSONArray jArr = jsonObj.getJSONArray("list");  
for (int i=0; i < jArr.length(); i++) {  
    JSONObject obj = jArr.getJSONObject(i);  
    ....  
}
```

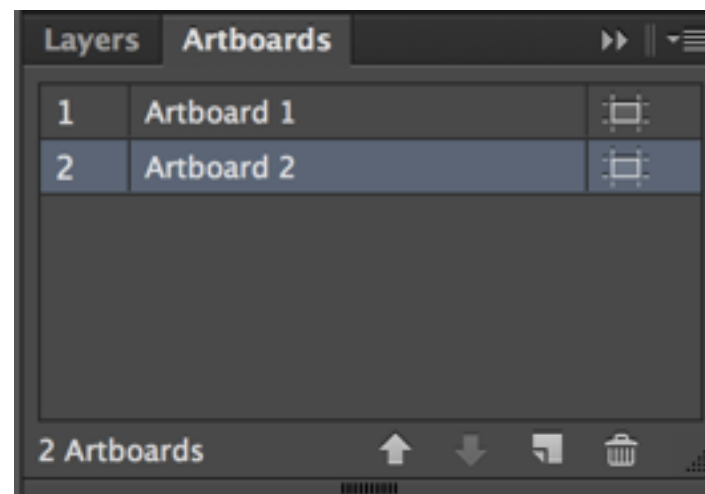
Intro to



Illustrator Workflow: Artboards

- Analogy: Pieces of paper on a desk
- For prototypes: One artboard/screen
- Artboard Tool 



- Artboard Panel

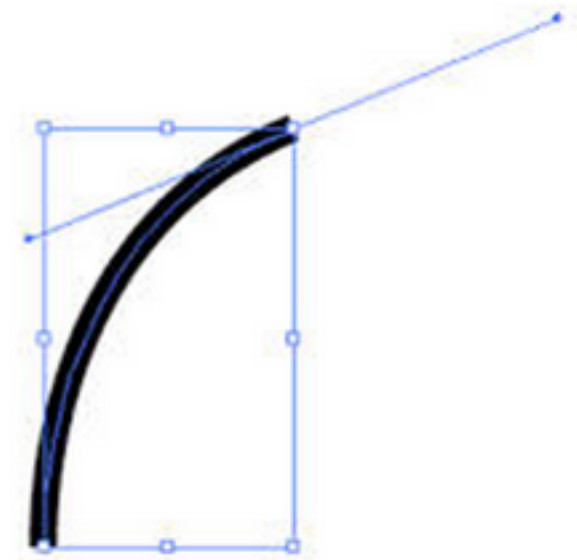


Must-Know Commands



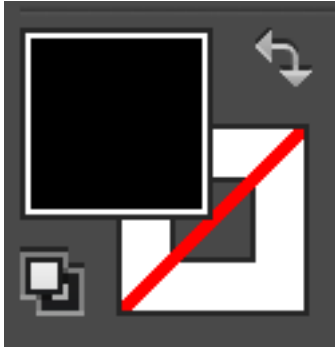
- Alt to copy
- Shift to constrain proportions while resizing
- Ctrl+G to group elements together

Selection vs. Direct Selection Tool

- Selection Tool (V) 
 - Move, resize, modify proportions of elements
- Direct Selection Tool (A) 
 - Modify paths within elements



Shapes

- Shapes The icon for the Shapes tool, showing a dark gray square with a lighter gray square inside, and a small white triangle in the bottom right corner.
- Lines The icon for the Lines tool, showing a dark gray square with a white diagonal line from the top-left to the bottom-right, and a small white triangle in the bottom right corner.
- Attributes: Fill & Stroke The icon for the Attributes panel, showing a dark gray square with a white square inside, a red diagonal line from the bottom-left to the top-right, a small white triangle in the bottom right corner, and a circular arrow icon in the top right corner.

Activity: Flowcharting Mobile + Wear

Cuckoo Clock

- A cuckoo clock is a typically pendulum-regulated clock that strikes the hours with a sound like a common cuckoo's call.
- This watch app triggers images of different cuckoo birds to appear at the strike of each hour.
- We'll be creating a flowchart of the backend structure of this app.

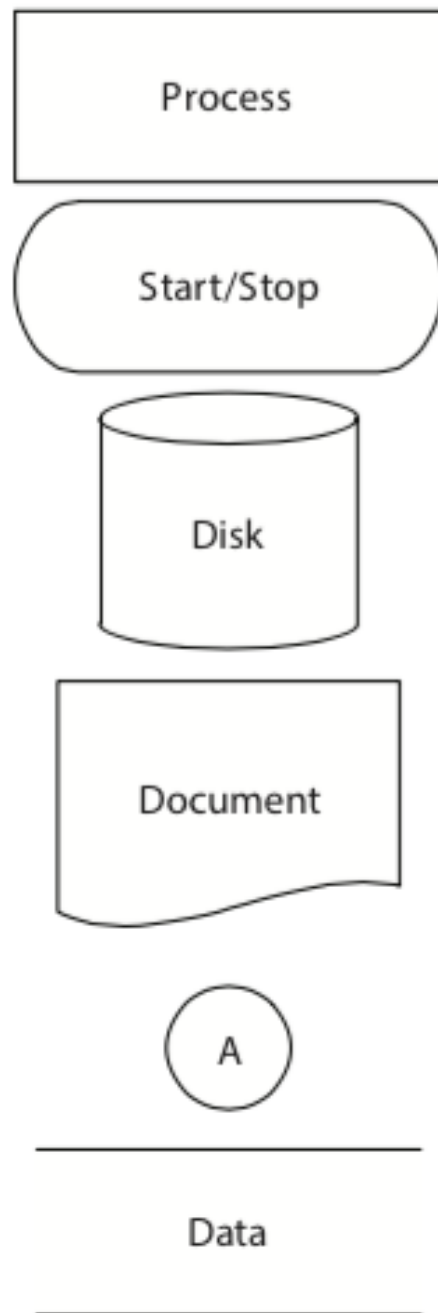
Assets

CODE: <http://tinyurl.com/cs160-cuckoo-alt/cs160-cuckoo-alt>

AI: <http://tinyurl.com/cs160sec7>

Assets Walkthrough

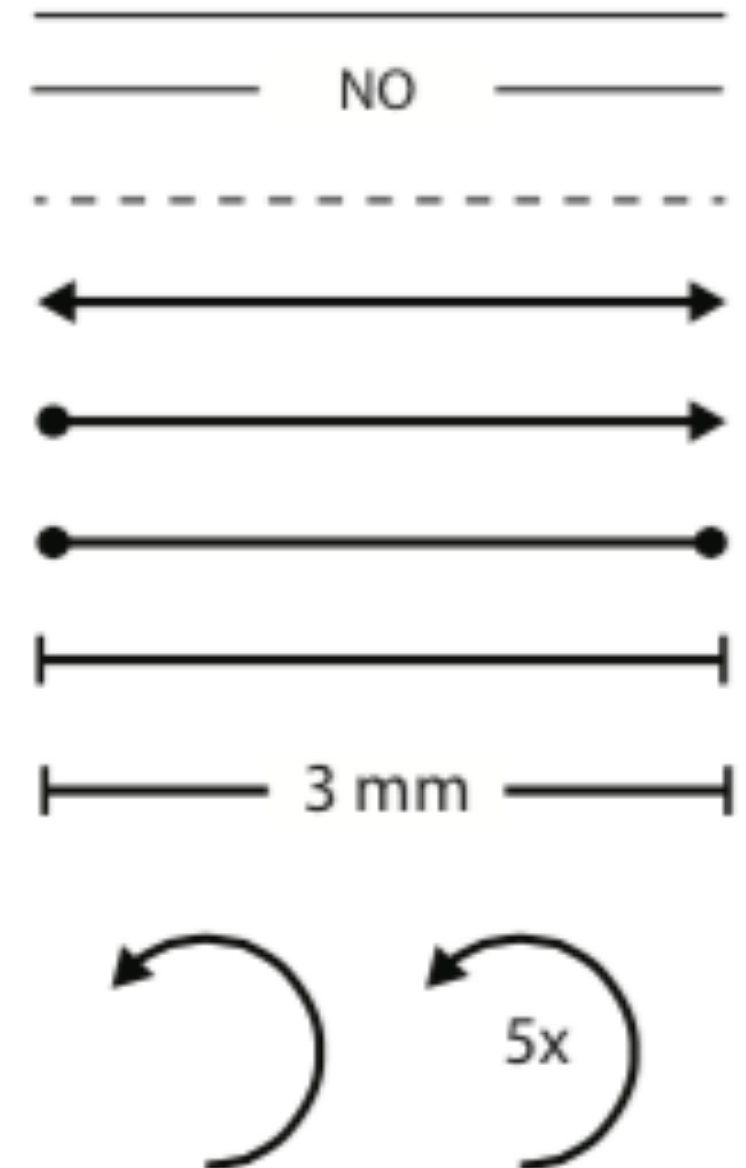
Flowchart elements



Colors



Transition Design



Assets Walkthrough

Phone & Watch Screens



Gestures



Brainstorm

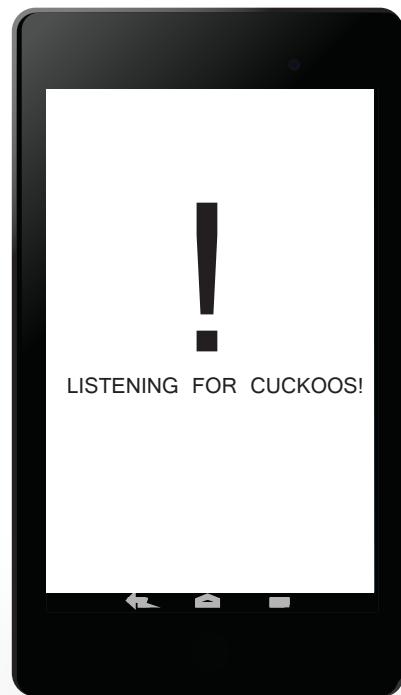
- Which backend components will we be using for this watch app?

Brainstorm

- Backend components:
 - 3 Watch screens
 - One service + API talking
 - Notification Process
- Activity: Flowchart this!

Example Implementation

Walkthrough: Screens & Gestures



Example Implementation

Walkthrough: Service and Notification

WATCHLISTENERSERVICE

Listens for messages from the Wear Data Layer API.



MAIN ACTIVITY

Homelanding page.

FAIL ACTIVITY

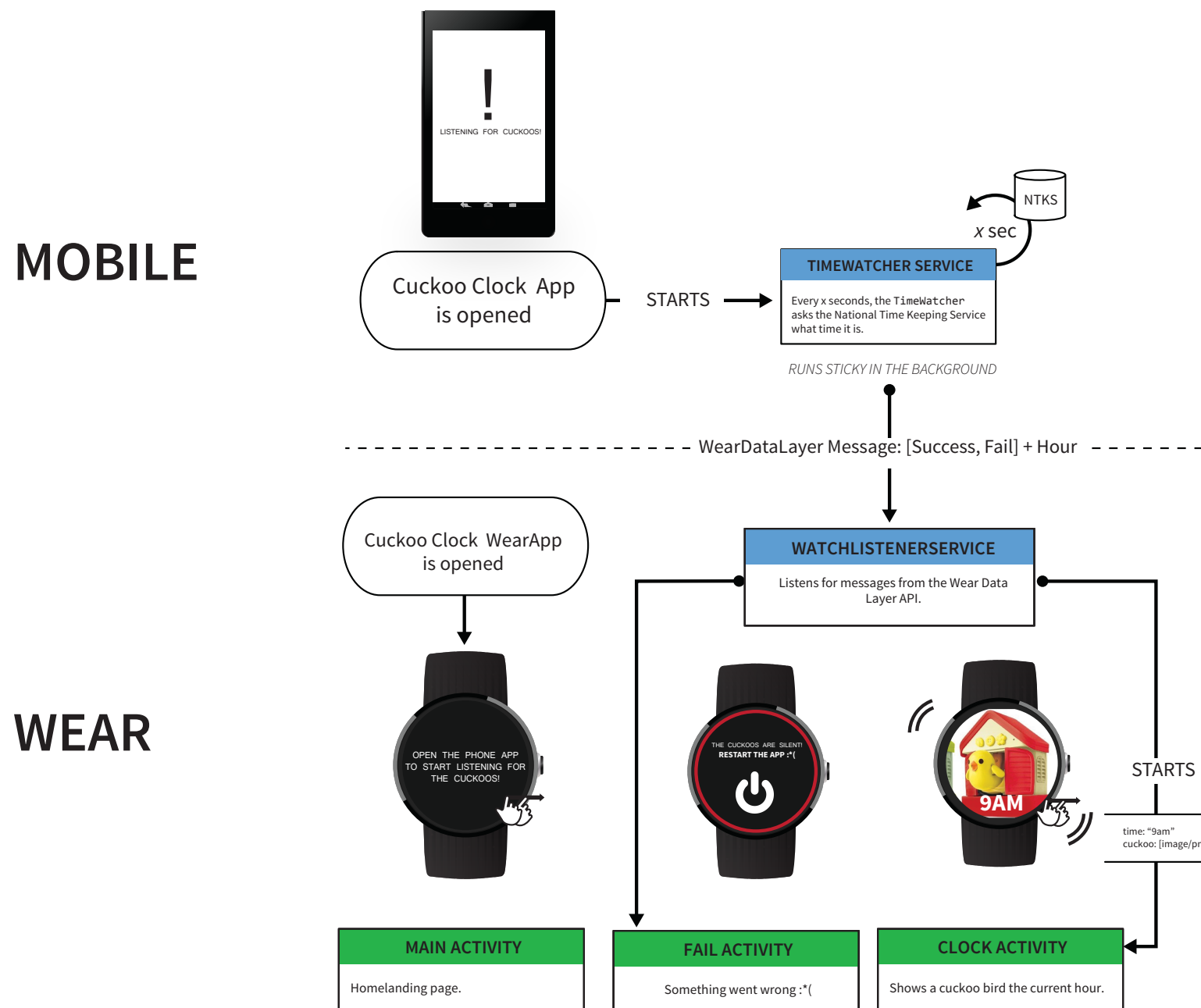
Something went wrong :*(

CLOCK ACTIVITY

Shows a cuckoo bird the current hour.

Example Implementation

Walkthrough: Flow



Flowchart to Code

Phone Activity to Phone Service

- Instantiate an intent with `myService.class` as a parameter
- `startService` runs the service specified in the intent

```
Intent i = new Intent(getApplicationContext(), FastTimeWatcherService.class);  
startService(i);
```

Using an HTTP API in the Service

- In this example we data from a server via bare URL, so we setup an `HttpURLConnection`. Fabric gives us a nice Java wrapper instead.
- In this example we get super simple raw text instead of in a JSON format. With Twitter, you get JSON that you have to parse.


```
urlConnection = (HttpURLConnection) url.openConnection();  
urlConnection.connect();  
InputStream in = urlConnection.getInputStream();  
Scanner scanner = new Scanner(in);  
mTimeResponse = scanner.nextLine(); //read a single line from scanner object
```

Phone Service to WatchListenerService

- WatchListenerService is always on. We do this by making it a `BIND_LISTENER` in the manifest.
- Sender instantiates a `GoogleApiClient` object and we define the `sendMessage` method. Call it when it's cuckoo time!

```
private void sendMessage( final String path, final String text ) {  
    new Thread( new Runnable() {  
        @Override  
        public void run() {  
            // dense API code goes here  
        }  
    }).start();  
}  
sendMessage(START_ACTIVITY, hour)
```

We also send a path (`String`) for the watch, which we need to differentiate between starting different watch activities



WatchListenerService to Watch Activity

- Check the MessageEvent for the path and decide what to do
- Here there's only one path: START_ACTIVITY = “/start_activity”
- Create a new intent, but use
`intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)`
- Add pertinent information for the watch activity with
`intent.putExtra` and call `startActivity`