
COMPUTER SECURITY LAB
RICE UNIVERSITY

Building a “real” Android app: the story of CalWatch

Dan S. Wallach (Rice University)

Announcements

No labs this week (finish your project!)

Friday's lecture will be a review / Q&A for the final exam

Final exam is December 11 (watch Piazza for room assignments)

We start at 2pm sharp. We finish at 4:50pm sharp. Don't be late!

Same "open Internet" exam policies as last time.

Prof. Wallach's Java chops are rusty

But Comp215 is in Java, and in May 2014, Motorola gave me an idea...

9TO5Google
BEYOND GOOD & EVIL

PLUS ANDROID CHROME/OS SEARCH GOOGLE APPS GOOGLETV YOUTUBE

There is one comment; join the conversation!

Motorola asks community to submit Moto 360 watch face designs ahead of launch

Daily Deals

Motorola has spent a lot of time since first introducing the Moto 360 talking about how its Android Wear smartwatch will be better than the other guys. That super streamlined watch face it's been

CATEGORY: GOOGLE CORPORATE

Written by:
JORDAN KAHN
May 21, 2014 / 8:20 am
SHARE 1 COMMENT

SEARCH 9TO5GOOGLE

LATEST NEWS
9TO5Mac
APPLE INTELLIGENCE

Next
y to get a
ce every year.

AT&T introduces longer Next 24 monthly installment program for paying off your iPhone


Raymond
U.S.


34 mins
to work
Light traffic on
US-101 S.
FRI SAT SUN MON
61 63 57 64
48 46 42 47
9:20
SFO → JFK


10:09
Sunrise 6:45AM
Sunset 7:27PM
Cupertino, CA
14:59
7:00
00:46
TUE 9
15:00
10:09

Let's build a watchface for Android Wear!

Initial idea for the Moto 360 contest, ~30 minutes in Photoshop & Illustrator:

Let's build a watchface for Android Wear!

Initial idea for the Moto 360 contest, ~30 minutes in Photoshop & Illustrator:

Floating reminder text

Transparency & shadows

Free/busy time on the dial

Easy to read, analog indicia (dive watch "tool" styling)

Android development 101

Android apps are written in Java 7 (no lambdas, no streams)

Download and install Java SE, Android Studio (IntelliJ), and tons of Android SDKs

- Java8 support is new as of summer 2016; all sample code you'll find online doesn't use Java8

Start building a basic "Hello World" and work up from there

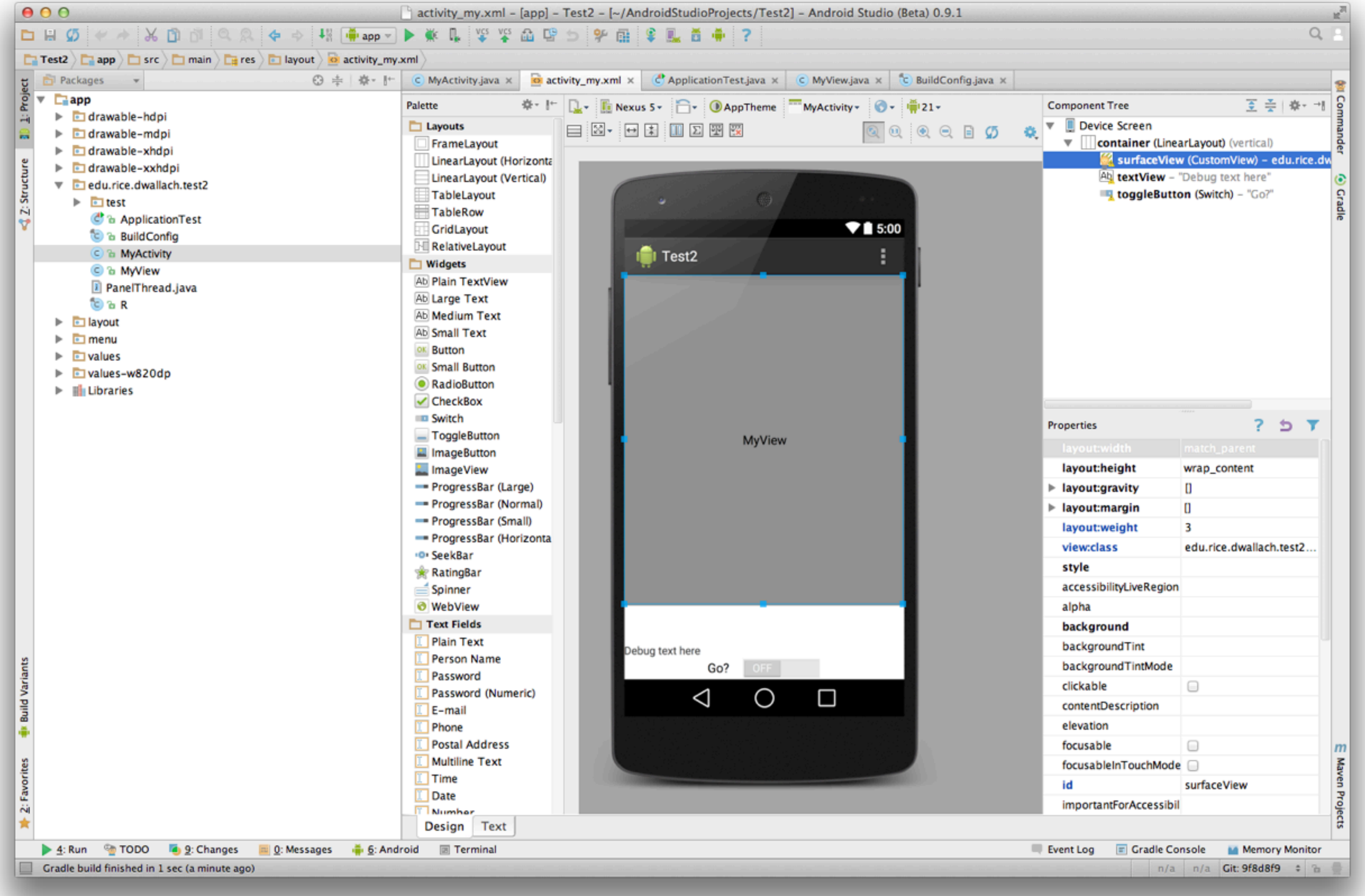
Android Studio generates lots of boilerplate

You lay out your UI with a graphical tool

Basic Java code to put it together is auto-generated

Android "emulator" is easy to use

Build a fake phone with "AVD" (Android Virtual Device)


activity_my.xml – [app] – Test2 – [~/AndroidStudioProjects/Test2] – Android Studio (Beta) 0.9.1
Test2 app src main res layout activity_my.xml
MyActivity.java activity_my.xml ApplicationTest.java MyView.java BuildConfig.java
Packages
app
drawable-hdpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
edu.rice.dwallach.test2
test
ApplicationTest
BuildConfig
MyActivity
MyView
PanelThread.java
R
layout
menu
values
values-w820dp
Libraries
Palette
Layouts
FrameLayout
LinearLayout (Horizonta
LinearLayout (Vertical)
TableLayout
TableRow
GridLayout
RelativeLayout
Widgets
Plain TextView
Large Text
Medium Text
Small Text
Button
Small Button
RadioButton
CheckBox
Switch
ToggleButton
ImageButton
ImageView
ProgressBar (Large)
ProgressBar (Normal)
ProgressBar (Small)
ProgressBar (Horizonta
SeekBar
RatingBar
Spinner
WebView
Text Fields
Plain Text
Person Name
Password
Password (Numeric)
E-mail
Phone
Postal Address
Multiline Text
Time
Date
Nexus 5
AppTheme
MyActivity
21
5:00
Test2
MyView
Debug text here
Go?
OFF
Component Tree
Device Screen
container (LinearLayout) (vertical)
surfaceView (CustomView) – edu.rice.dw
textView – "Debug text here"
toggleButton (Switch) – "Go?"
Properties
layout:width match_parent
layout:height wrap_content
layout:gravity []
layout:margin []
layout:weight 3
view:class edu.rice.dwallach.test2...
style
accessibilityLiveRegion
alpha
background
backgroundTint
backgroundTintMode
clickable
contentDescription
elevation
focusable
focusableInTouchMode
id surfaceView
importantForAccessibil
Design Text
4: Run TODO 9: Changes 0: Messages 6: Android Terminal
Event Log Gradle Console Memory Monitor
Gradle build finished in 1 sec (a minute ago)
Git: 9f8d8f9

Two days later... (August 13)

Running clock

Second hand ticking at 5Hz

Timezone is wrong

Crashes if you close, rotate, etc.

Total code written

MyActivity.java: 124 lines

MyView.java: 350 lines

XML layout, manifest, etc.: ~60 lines

Ignore Android Wear for now

I didn't have one yet

Easier to test and debug on a phone

Two days later... (August 13)

Running clock

Second hand ticking at 5Hz

Timezone is wrong

Crashes if you close, rotate, etc.

Total code written

MyActivity.java: 124 lines

MyView.java: 350 lines

XML layout, manifest, etc.: ~60 lines

Ignore Android Wear for now

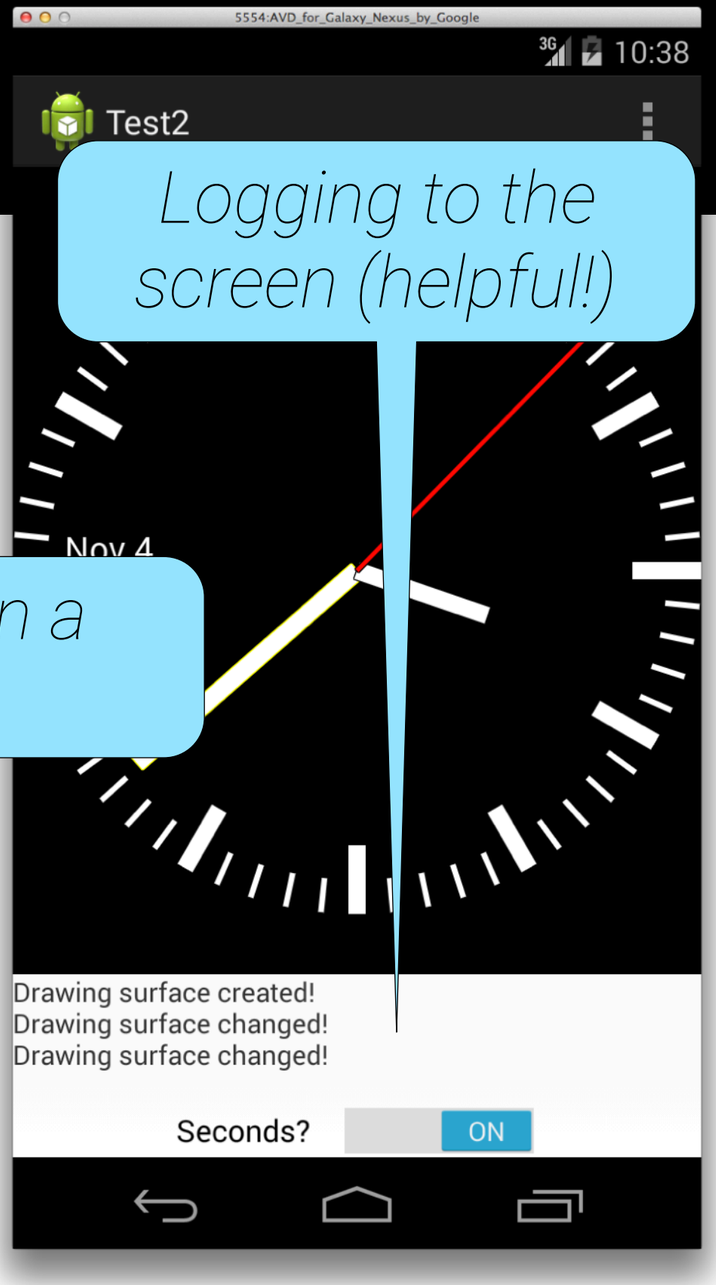
I didn't have one yet

Easier to test and debug on a phone

"Application lifecycle"

Drawing on a Canvas

Machine generated (mostly)



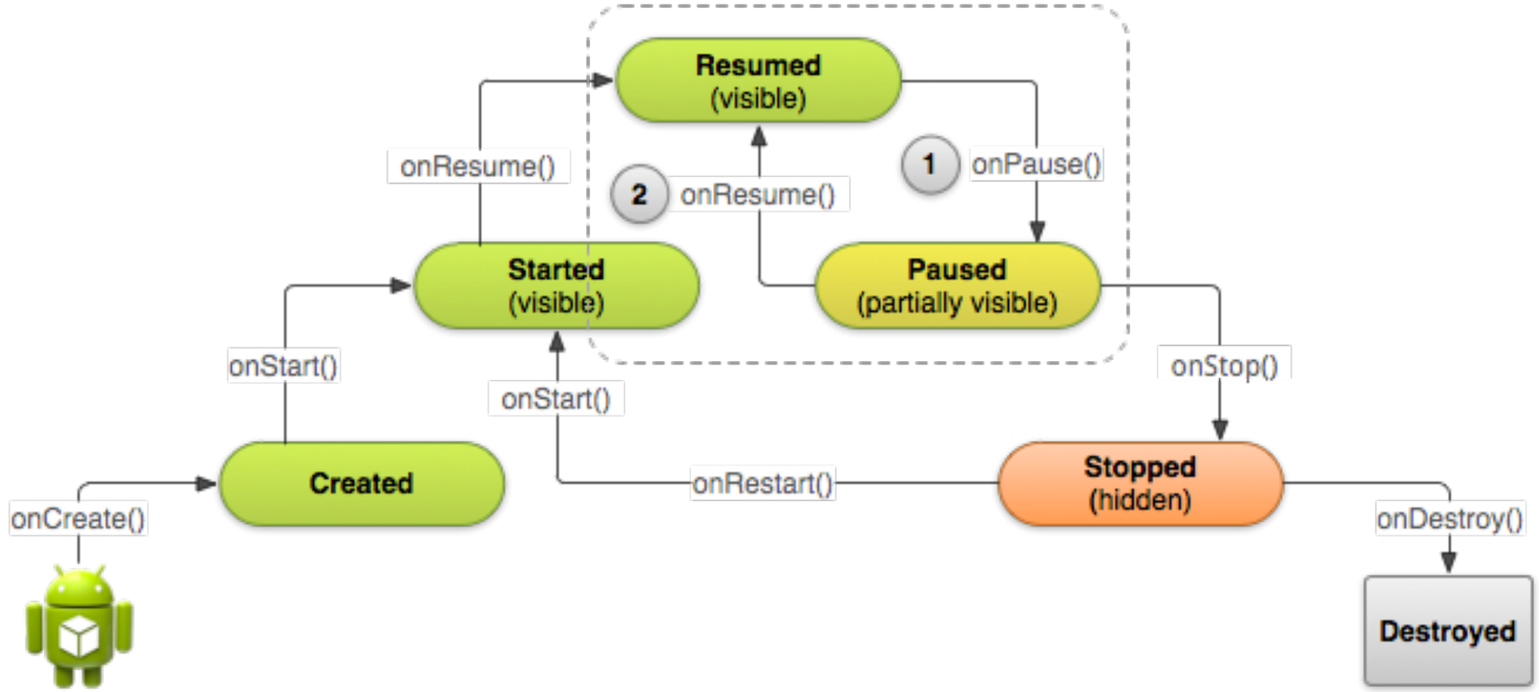
The Android “Application Lifecycle”

You implement a subclass of *Activity* and override a bunch of methods

onCreate, onStart, onStop, onPause, etc.

Callback-style programming

Set everything up, sit back, and wait for the system to call you



Single-threaded UI programming

Blocking is not allowed! Schedule things for the future.

Set “alarms”

Use “animators”

Or, start a separate thread, and deal with multithreading headaches

Pay careful attention to the application lifecycle.

Turn off alarms, kill auxiliary threads, etc.

Hardest thing for me to ultimately debug

Debugging an Android app

Logging, logging, logging!

```
import android.util.Log;
public class Foo {
    private static final String TAG = “Foo”;
    ...
    Log.v(TAG, “something interesting happened here”);
}
```

Logs are captured by Android Studio or by “adb” command line tool

Also, there are apps on the phone that can show you the logs

When your app crashes in the field, logs can be recovered post facto

Note: do not write “private” or user-sensitive info to the logs!

Debugging an Android app

Logging, logging, logging!

```
import android.util
public class Foo {
    private static final String TAG = "Foo";
    ...
    Log.v(TAG, “something interesting happened here”);
}
```

5 log “levels” (verbose, error, info, ...).
See also, ***`Log.wtf()`***

Optional 3rd argument: any Exception
(prints stack trace to the log)

Logs are **and line tool**

Also, there s

When your app crashes in the field, logs can be recovered post facto

Note: do not write “private” or user-sensitive info to the logs!

So I wanted to read from the calendar...

```
// first, get the list of calendars
Log.v(TAG, "CalendarFetcher starting to load content");
final String[] calProjection =
        new String[]{
                CalendarContract.Calendars._ID,
                CalendarContract.Calendars.NAME,
                CalendarContract.Calendars.ACCOUNT_NAME,
                CalendarContract.Calendars.ACCOUNT_TYPE,
                CalendarContract.Calendars.CALENDAR_COLOR,
                CalendarContract.Calendars.CALENDAR_COLOR_KEY,
                CalendarContract.Calendars.VISIBLE
        };
Cursor calCursor = ctx.getContentResolver().
        query(CalendarContract.Calendars.CONTENT_URI,
                calProjection,
                CalendarContract.Calendars.VISIBLE + " = 1",
                null,
                CalendarContract.Calendars._ID + " ASC");

int calendarsFound = 0;
```

```
if (calCursor.moveToFirst()) {
    do {
        int i = 0;
        CalendarResults.Calendar cal =
            new CalendarResults.Calendar();

        cal.ID = calCursor.getInt(i++);
        cal.name = calCursor.getString(i++);
        cal.accountName = calCursor.getString(i++);
        cal.accountType = calCursor.getString(i++);
        cal.calendarColor = calCursor.getInt(i++);
        cal.calendarColorKey = calCursor.getString(i++);
        cal.visible = (calCursor.getInt(i++) != 0);
        // Log.v(TAG, "Found calendar. ID(" + cal.ID + "), name("
+ cal.name + "), color(" + Integer.toHexString(cal.calendarColor) + "),
colorKey(" + cal.calendarColorKey + "), accountName(" + cal.accountName +
"), visible(" + Boolean.toString(cal.visible)+ ")");

        cr.calendars.put(cal.ID, cal);
        calendarsFound++;
    }
    while (calCursor.moveToNext()) ;
}
Log.v(TAG, "calendars found ("+ calendarsFound + ")");

calCursor.close();
```

So I wanted to read from the calendar...

```
// first, get the list of calendars
Log.v(TAG, "CalendarFetcher starting to load content");
final String[] calProjection =
        new String[]{
                CalendarContract.Calendars._ID,
                CalendarContract.Calendars.NAME,
                CalendarContract.Calendars.ACCOU
                CalendarContract.Calendars.AC      NT_TYPE,
                CalendarContract.Calenda    .CALENDAR_COLOR,
                CalendarContract.Cal      ndars.CALENDAR_COLOR_KEY,
                CalendarContract.Calendars.VISIBLE
        };
Cursor calCursor = ctx.getContentResolver().
        query(CalendarContract.Calendars.CONTENT_URI,
                calProjection,
                CalendarContract.Calendars.VISIBLE + " = 1",
                null,
                CalendarContract.Calendars._ID + " ASC");

int calendarsFound = 0;
```

```
if (calCursor.moveToFirst()) {
    do {
                cal.name = calCursor.getString(i++);
                cal.accountName = calCursor.getString(i++);
                cal.accountType = calCursor.getString(i++);
                cal.calendarColor = calCursor.getInt(i++);
                cal.calendarColorKey = calCursor.getString(i++);
                cal.visible = (calCursor.getInt(i++) != 0);
                // Log.v(TAG, "Found calendar. ID(" + cal.ID + "), name("
+ cal.name + "), color(" + Integer.toHexString(cal.calendarColor) + "),
colorKey(" + cal.calendarColorKey + "), accountName(" + cal.accountName +
"), visible(" + Boolean.toString(cal.visible)+ ")");

                cr.calendars.put(cal.ID, cal);
                calendarsFound++;
        }
        while (calCursor.moveToNext()) ;
    }
    Log.v(TAG, "calendars found ("+ calendarsFound + ")");

    calCursor.close();
```

Context: your Activity (or equiv.)

Building a SQL query

So I wanted to read from the calendar...

```
// first, get the list of calendars
Log.v(TAG, "CalendarFetcher starting to load content");
final String[] calProjection =
        new String[]{
                CalendarContract.Calendars._ID,
                                                    EY,
                CalendarContract.Calendars.VISIBLE
        };
Cursor calCursor = ctx.getContentResolver().
        query(CalendarContract.Calendars.CONTENT_URI,
                calProjection,
                CalendarContract.Calendars.VISIBLE + " = 1",
                null,
                CalendarContract.Calendars._ID + " ASC");

int calendarsFound = 0;
```

Load results into my own data structure

```
if (calCursor.moveToFirst()) {
    do {
        int i = 0;
        CalendarResults.Calendar cal =
            new CalendarResults.Calendar();

        cal.ID = calCursor.getInt(i++);
        cal.name = calCursor.getString(i++);
        cal.accountName = calCursor.getString(i++);
        cal.accountType = calCursor.getString(i++);
        cal.calendarColor = calCursor.getInt(i++);
        cal.calendarColorKey = calCursor.getString(i++);
        cal.visible = (calCursor.getInt(i++) != 0);
        // Log.v(TAG, "Found calendar. ID(" + cal.ID + "), name("
+ cal.name + "), color(" + Inte
colorKey(" + cal.calendarColorK
"), visible(" + Boolean.toStrin

        cr.calendars.pu
        calendarsFound
    }
    while (calCurs .moveToNext()) ;
}
Log.v(TAG, "calendars found ("+ calendarsFound + ")");

calCursor.close();
```

Don't forget to clean up afterwards

How do you figure all this out?

Google searches, StackOverflow searches, much head scratching

Initially, I was querying the full calendar

Found an open-source library for dealing with RFC 2245 “recurring events”

Didn’t always work (solved for real later; *stay tuned*…)

Save URLs in code comments

`grep http *.java` → 17 different places where I borrowed ideas and code

Warning: much advice on the Internet is incomplete or flat-out wrong

Overlapping calendar events

Initial thought: design a greedy algorithm

For each event: If no overlap with prior events, insert and we're done

If there is overlap, then add a "new level" and squish overlapping events

Stretch prior inserted events to fill these new levels

Analysis: O(n^3) worst case (if they all overlap), but n is small

Mostly worked, but obscure bugs and layout wasn't pretty

Good enough for now, fix it later (*stay tuned*...)

Time to port the code to run on Android Wear

Code size, August 24, 2014 (before Wear port)

CalendarFetcher.java	417 lines
CalendarResults.java	95 lines
ClockFace.java	574 lines
EventLayout.java	137 lines
MyActivity.java	245 lines
MyViewAnim.java	300 lines
miscellaneous Java	47 lines
XML files	158 lines
Total	1973 lines

Graphics code now in a separate class; Added graphics code for the pie wedges

Running the graphics in a separate thread, many headaches

Android Wear: It's just Android

APIs for implementing a "watchface" were undefined; start with an "app"

My app was running immediately on the watch emulator

New "layout" (XML) and minor tweaks, but it ran right away

There's a CalendarProvider on the watch, but it returns no calendars, no events

New engineering necessary: build a phone → watch data path

Two choices: DataAPI or MessageAPI (both over Bluetooth)

DataAPI: key/value store, synchronized magically (eventually)

MessageAPI: here's an array of bytes, get it over there or fail now

How to send calendar data?

DataAPI key/value store doesn’t handle object-arrays

Need reliable, simple serialization

Protocol buffers to the rescue!

Invented by Google, widely supported, efficient, compact, extensible, ...

Compact open-source “Wire” library from Square, engineered for Android

```
message WireEvent {
    required int64 startTime = 1;
    required int64 endTime = 2;
    required int32 displayColor = 3;
}

message WireUpdate {
    repeated WireEvent events = 1;
    required bool newEvents = 2; // true: the events are new, have a look; false: ignore the events field
    required int32 faceMode = 3; // display mode: ClockState.FACE_TOOL, FACE_NUMBERS, FACE_LITE
    required bool showSecondHand = 4;
    required bool showDayDate = 5;
}
```

How to send calendar data?

DataAPI key/value store doesn't handle object-arrays

Need reliable, simple serialization

Protocol buffers to the rescue!

Invented by Google, widely supported, efficient, compact, extensible, ...

Compact open-source "Wire" library from Square, engineered for Android

```
message WireEvent {
    required int64 startTime = 1;
    required int64 endTime = 2;
```

```
public byte[] getProtobuf() {
    WireUpdate wireUpdate = new WireUpdate(getWireEventList(), true, getFaceMode(), getShowSeconds(), getShowDayDate());
    byte[] output = wireUpdate.toByteArray();

    return output;
}
```

```
    required bool
    required bool
}
```

Constructor to build the data structure (auto-generated)

List<WireEvent>

Boom! Bytes ready to send.

11 calendar events → 250 bytes

Activities vs. Services

Activities (you run them, you see them)

Services (operate in the background)

Both can run in the same process, share the same UI thread.

Services can continue running even when the app isn't visible.

Android might kill your Activity but leave your Service running.

Services can automatically start at boot time.

> We want to feed calendar data from the phone to the watch, even if the user hasn't started the app!

Refactor code to have a service interacting with the calendar provider

Shared state: Activity (on phone) just sees new data to render

Networking: Serialize the state and send it to the watch

Manifest.XML - where it all begins

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dwallach.calwatch">

    <uses-feature android:name="android.hardware.type.watch"
android:required="false"/>

    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.permission.PROVIDE_BACKGROUND" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />

        <activity
            android:name=".PhoneActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".WatchCalendarService"
            android:enabled="true"
            android:exported="false" >
        </service>


        <receiver
            android:name=".WakeupReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="org.dwallach.calwatch.WAKE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PROVIDER_CHANGED"/>
                <data android:scheme="content"/>
                <data android:host="com.android.calendar"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Manifest.XML - where it all begins

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dwallach.calwatch">

    <uses-feature android:name="android.hardware.type.watch"
android:required="false"/>

    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.permission.PROVIDE_BACKGROUND" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />

        <activity
            android:name=".PhoneActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                                                                    />
            android:enabled="true"
            android:exported="false" >
        </service>


        <receiver
            android:name=".WakeupReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="org.dwallach.calwatch.WAKE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PROVIDER_CHANGED"/>
                <data android:scheme="content"/>
                <data android:host="com.android.calendar"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Extra declaration: I'm a hybrid phone/watch app

Manifest.XML - where it all begins

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dwallach.calwatch">

    <uses-feature android:name="android.hardware.type.watch"
android:required="false"/>

    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.permission.PROVIDE_BACKGROUND" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />

        <activity
            android:name=".PhoneActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".WatchCalendarService"
            android:enabled="true"

            android:name=".WakeupReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="org.dwallach.calwatch.WAKE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PROVIDER_CHANGED"/>
                <data android:scheme="content"/>
                <data android:host="com.android.calendar"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Declare Android permissions

Manifest.XML - where it all begins

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dwallach.calwatch">

    <uses-feature android:name="android.hardware.type.watch"
android:required="false"/>

    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.permission.PROVIDE_BACKGROUND" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />

        <activity
            android:name=".PhoneActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".WatchCalendarService"
            android:enabled="true"
            android:exported="false" >
        </service>


        <receiver
            android:name=".WakeupReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="org.dwallach.calwatch.WAKE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PROVIDER_CHANGED"/>

</manifest>
```

The Activity: what the user sees

Manifest.XML - where it all begins

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dwallach.calwatch">

    <uses-feature android:name="android.hardware.type.watch"
android:required="fa

    <uses-permissio
    <uses-permissio
    <uses-permissio
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.permission.PROVIDE_BACKGROUND" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">

        <meta-data android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />

        <activity
            android:name=".PhoneActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".WatchCalendarService"
            android:enabled="true"
            android:exported="false" >
        </service>

        <receiver
            android:name=".WakeupReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="org.dwallach.calwatch.WAKE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PROVIDER_CHANGED"/>
                <data android:scheme="content"/>
                <data android:host="com.android.calendar"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Service: fetches data when the calendar is updated

Manifest.XML - where it all begins

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.dwallach.calwatch">

    <uses-feature android:name="android.hardware.type.watch"
android:required="false"/>

    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.permission.PROVIDE_BACKGROUND" />

    <application
        a
        <
        <
        <
        <
android:
        <a
            android:name=".PhoneActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".WatchCalendarService"
            android:enabled="true"
            android:exported="false" >
        </service>

        <receiver
            android:name=".WakeupReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="org.dwallach.calwatch.WAKE" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.PROVIDER_CHANGED"/>
                <data android:scheme="content"/>
                <data android:host="com.android.calendar"/>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Receiver: Gets notified when we boot and when the calendar changes; kicks the Service

Gradle build system

make → ant → maven → gradle → ...

Gradle is this week's current trendy build system; it has some cool features

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.squareup.wire:wire-runtime:1.5.1'
}
```

Go find the library out there, somewhere, and link it in for me.

```
def getVersionName = { ->
    try {
        def stdout = new ByteArrayOutputStream()
        exec {
            commandLine 'git', 'describe', '--tags', '--dirty'
            standardOutput = stdout
        }
        return stdout.toString().trim()
    }
    catch (ignored) {
        return null;
    }
}
```

General-purpose Groovy scripting language.

Debugging

Paranoid engineering?

Tons of paranoid code throughout the codebase, e.g.,:

```
public void onConnected(Bundle connectionHint) {
    Log.v(TAG, "connected to Google API!");
    readyToSend = true;

    // shouldn't ever happen, but might explain the weird null pointer exceptions that rarely show up in the logs
    if(googleApiClient == null) {
        Log.e(TAG, "unexpected null googleApiClient");
        cleanup();
        initGoogle();
        return;
    }

    try {
        Wearable.MessageApi.addListener(googleApiClient, this);
    } catch (NullPointerException e) {
        Log.e(TAG, "unexpected failure in onConnected (googleApiClient = " + googleApiClient + ")", e);
        cleanup();
        initGoogle();
        return;
    }

    sendAllToWatch();
}
```

Paranoid engineering?

Tons of paranoid code throughout the codel

```
public void onConnected(Bundle connectionHint) {
    Log.v(TAG, "connected to Google API!");
    readyToSend = true;

    // shouldn't ever happen, but might explain the weird                    logs
    if(googleApiClient == null) {
        Log.e(TAG, "unexpected null googleApiClient");
        cleanup();
        initGoogle();
        return;
    }

    try {
        Wearable.MessageApi.addListener(googleApiClient, this);
    } catch (NullPointerException e) {
        Log.e(TAG, "unexpected failure in onConnected (googleApiClient = " + googleApiClient + ")", e);
        cleanup();
        initGoogle();
        return;
    }

    sendAllToWatch();
}
```

This is the callback once we're "connected" to the Google API.

Paranoid engineering?

Tons of paranoid code throughout the codebase, e.g.,:

```
public void onConnected(Bundle connectionHint) {
    Log.v(TAG, "connected to Google API!");
    readyToSend = true;

    // shouldn't ever happen, but might explain the weird null pointer exceptions that rarely show up in the logs
    if(googleApiClient == null) {
        Log.e(TAG, "unexpected null googleApiClient");
        cleanup();
        initGoogle();
        return;
    }

    try {
        Wearable.MessageApi.addListener(googleApiClient, t
    } catch (NullPointerException e) {
        Log.e(TAG, "unexpected failure in onConnected (goo
        cleanup();
        initGoogle();
        return;
    }

    sendAllToWatch();
}
```

But this rarely blows up with a NullPointerException. Which shouldn't happen.

Paranoid engineering?

Tons of paranoid code throughout the codebase, e.g.,:

```
public void onConnected(Bundle connectionHint) {
    Log.v(TAG, "connected to Google API!");
    readyToSend = true;

    // shouldn't ever happen, but might explain the weird                                    logs
    if(googleApiClient == null) {
        Log.e(TAG, "unexpected null googleApiClient");
        cleanup();
        initGoogle();
        return;
    }

    try {
        Wearable.MessageApi.addListener(googleApiClient, this);
    } catch (NullPointerException e) {
        Log.e(TAG, "unexpected failure in onConnected (googleApiClient = " + googleApiClient + ")", e);
        cleanup();
        initGoogle();
        return;
    }

    sendAllToWatch();
}
```

Paranoia 1: maybe googleApiClient was null (which shouldn't happen).

Paranoia 2: try to clean up and reconnect.

Finally, I got my LG G Watch in the mail

Found bugs that I didn't find in the emulator

Tweaks to the graphics to look good on the watch (shadows, font size, ...)

Integrated sample code (reverse-engineered) to run as a real watchface

Extra work to deal with "ambient" mode

No documentation from Google at all

Many, many fixes to lifecycle bugs

Example: what if the watchface restarts and there's no connected phone?

Solution: use persistent state to remember old calendar data

Tweak graphics for the "flat tire" bottom of the Moto 360

Performance & profiling

Rule #1: Don’t.

Rule #2: Profile your code.

android.os.Debug.*startMethodTracing*( )

Run ~60 seconds, stop, get giant dump, run in analysis tool

Also useful: compute min/mean/max frame rendering times, report in log

Tentative observations:

Recomputing day/date (“Nov 4”) on every redraw is surprisingly expensive

- Caching this result not only sped up redraws, but also eliminated most garbage collection events

Recomputing the geometry of the watch face is also dumb

- Canvas lets you “cache” this in a Path, which you can reuse on subsequent frames

Results? Running at 40+ Hz, using <30% of available CPU time

Folk wisdom:

Keep your *onDraw*( ) method lean (no memory allocation, etc.)

Alpha 1 “release”: September 15, 2014

First point that I’m willing to generate an APK, show others. Still buggy.

CalendarFetcher.java	417→449 lines
CalendarResults.java	95→116 lines
ClockFace.java	574→755 lines
EventLayout.java	137→160 lines
PhoneActivity.java	245→246 lines
MyViewAnim.java	300→302 lines
miscellaneous Java	47→76 lines
XML files	158→283 lines
WatchCalendarService.java	213 lines
WearSender.java	148 lines
BatteryMonitor.java	78 lines
ClockState.java	251 lines
TimeWrapper.java	115 lines
WearActivity.java	241 lines
WearReceiverService.java	196 lines
Total	1973→3555 lines

Alpha 1 “release”: September 15, 2014

First point that I’m willing to generate an APK, show others. Still buggy.

CalendarFetcher.java	417→449 lines
CalendarResults.java	95→116 lines
ClockFace.java	574→755 lines
EventLayout.java	137→160 lines
PhoneActivity.java	245→246 lines
MyViewAnim.java	300→302 lines
miscellaneous Java	47→76 lines
XML files	1973→3555 lines

WatchCalendarService.java	213 lines
WearSender.java	148 lines
BatteryMonitor.java	78 lines
ClockState.java	251 lines
TimeWrapper.java	115 lines
WearActivity.java	241 lines
WearReceiverService.java	196 lines

Model / View / Controller separation

Alpha 1 “release”: September 15, 2014

First point that I’m willing to generate an APK, show others. Still buggy.

CalendarFetcher.java	417→449 lines
CalendarResults.java	95→116 lines
ClockFace.java	574→755 lines
EventLayout.java	137→160 lines
PhoneActivity.java	245→246 lines
MyViewAnim.java	300→302 lines
miscellaneous Java	47→76 lines
XML files	1973→3555 lines

WatchCalendarService.java	213 lines
WearSender.java	148 lines
BatteryMonitor.java	78 lines
ClockState.java	251 lines
TimeWrapper.java	115 lines
WearActivity.java	241 lines
WearReceiverService.java	196 lines

Shared code across phone & watch

Subsequent fixes: lots and lots of bugs

Reading the calendar. *Properly*.

“Instances” not “events”

CalendarFetcher.java: 449→275 lines, and no more recurring-event bugs

Removed RFC2245 library, no longer necessary

***Observation*: led astray by calendar code on the Internet, fixed by careful reading of the Android API documentation.**

And yet more subtle bugs fixed with the app lifecycle.

Maybe once-a-day crashes? Run to a computer and extract logs. Repeat.

***Observation*: Easy to get running. Hard to nail down weird bugs.**

Exasperation: Pasting together bits of code from the Internet is dangerous.

Calendar event layout misbehavior

Weird corner cases where the layout just didn't work right

Attempt #1: Unit testing and careful engineering

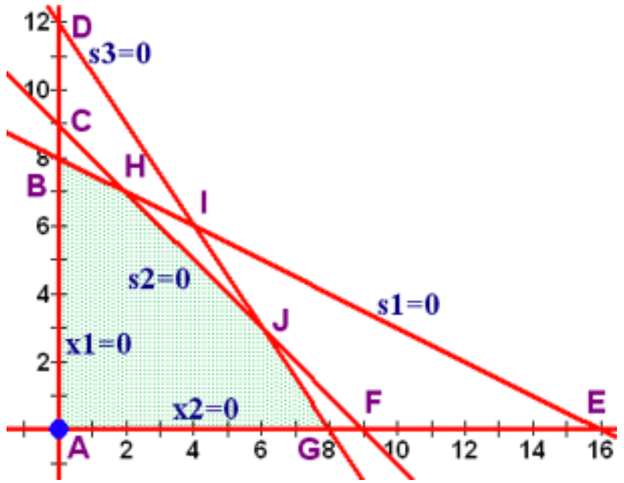
Android has unit testing support. The event layout engine is easy to test on its own.

> *Hypothesis*: My greedy algorithm is the wrong solution. Need to treat like a system of equations or springs.
> *Insight*: Computer science to the rescue. Linear constraint solvers!

Simplex method is decades old

Exponential worst case, fast in practice

Picked the Cassowary solver (Java, open source)



Using a solver is easy

```
ClLinearExpression sumSizes = new ClLinearExpression(0.0);
for (i = 0; i < nEvents; i++) {
    sumSizes = sumSizes.plus(sizes[i]);
    for (j = i + 1; j < nEvents; j++) {
        if (events.get(i).overlaps(events.get(j))) {
            // constraint: base level + its size < base level of next dependency
            ClLinearExpression levelPlusSize = new ClLinearExpression(startLevels[i]).plus(sizes[i]);
            ClLinearInequality liq = new ClLinearInequality(levelPlusSize, CL.LEQ, startLevels[j],
                ClStrength.required);
            solver.addConstraint(liq);

            // weak constraint: constrained segments should have the same size (0.5x weight of other weak constraints)
            ClLinearEquation eqSize = new ClLinearEquation(sizes[i], new ClLinearExpression(sizes[j]),
                ClStrength.weak, 0.5);
            solver.addConstraint(eqSize);
        }
    }
}
// constraint: the sum of all the sizes is greater than the maximum it could ever be under the absolute best of cases
// (this constraint's job is to force us out of degenerate cases when the solver might prefer zeros everywhere)
ClLinearInequality sumSizesEq = new ClLinearInequality(sumSizes, CL.GEQ, new ClLinearExpression(MAXLEVEL*nEvents),
ClStrength.weak);
solver.addConstraint(sumSizesEq);
```

Using a solver is easy

```
ClLinearExpression sumSizes = new ClLinearExpression(0.0);
for (i = 0; i < nEvents; i++) {
    sumSizes = sumSizes.plus(sizes[i]);
    for (j = i + 1; j < nEvents; j++) {
        if (events.get(i).overlaps(events.get(j))) {
            // constraint: base level + its size < base level of next dependency
            ClLinearExpression levelPlusSize = new ClLinearExpression(startLevels[i]).plus(sizes[i]);
            ClLinearInequality liq = new ClLinearInequality(levelPlusSize, CL.LEQ, startLevels[j],
                ClStrength.required);
            solver.addConstraint(liq);

            // weak constraint: constrained segments should have t e same size (0.5x weight of other weak constraints)
            ClLinearEquation eqSize = new ClLinearEquation(sizes[i   new ClLinearExpression(sizes[j]),
                ClStrength.weak, 0.5
            solver.addConstraint(eqS
        }
    }
}
// constraint: the sum of all the si                                    ld ever be under the absolute best of cases
// (this constraint's job is to force us out of degenerate cases when the solver might prefer zeros everywhere)
ClLinearInequality sumSizesEq = new ClLinearInequality(sumSizes, CL.GEQ, new ClLinearExpression(MAXLEVEL*nEvents),
ClStrength.weak);
solver.addConstraint(sumSizesEq);
```

*Each event has a **start** and a **size**. Constrain them when overlapping.*

Using a solver is easy

```
ClLinearExpression sumSizes = new ClLinearExpression(0.0);
for (i = 0; i < nEvents; i++) {
    sumSizes = sumSizes.plus(sizes[i]);
    for (j = i + 1; j < nEvents; j++) {
        if (events.get(i).overlaps(events.get(j))) {
            // constraint: base level + its size < base level of next dependency
            ClLinearExpression levelPlusSize = new ClLinearExpression(startLevels[i]).plus(sizes[i]);
            ClLinearInequality liq = new ClLinearInequality(levelPlusSize, CL.LEQ, startLevels[j],
                ClStrength.required);
            solver.addConstraint(liq);

            // weak constraint: constrained segments should have the same size (0.5x weight of other weak constraints)
            ClLinearEquation eqSize = new ClLinearEquation(sizes[i]   new ClLinearExpression(sizes[j]),
                ClStrength.weak, 0.5
            solver.addConstraint(eqS
        }
    }
}
// constraint: the sum of all the si                                    ld ever be under the absolute best of cases
// (this constraint's job is to force us out of degenerate cases when the solver might prefer zeros everywhere)
ClLinearInequality sumSizesEq = new ClLinearInequality(sumSizes, CL.GEQ, new ClLinearExpression(MAXLEVEL*nEvents),
ClStrength.weak);
solver.addConstraint(sumSizesEq);
```

Event ordering impacts layout. So what order?

Event sorting: for consistency + aesthetics

```
// Primary sort: color, so events from the same calendar will become consecutive wedges

// Secondary sort: endTime, with objects ending earlier appearing first in the sort.
//   (goal: first fill in the outer ring of the display with smaller wedges; the big
//    ones will end late in the day, and will thus end up on the inside of the watchface)

// Third-priority sort: startTime, with objects starting later (smaller) appearing first in the sort.

Collections.sort(cr.instances, new Comparator<CalendarResults.Instance>() {
    public int compare(CalendarResults.Instance lhs, CalendarResults.Instance rhs) {
        if(lhs.displayColor != rhs.displayColor)
            return Long.compare(lhs.displayColor, rhs.displayColor);

        if(lhs.endTime != rhs.endTime)
            return Long.compare(lhs.endTime, rhs.endTime);

        return Long.compare(rhs.startTime, lhs.startTime);
    }
});
```

Event sorting: for consistency + aesthetics

```
// Primary sort: color, so events from the same calendar will become consecutive wedges

// Secondary sort: endTime, with objects ending earlier appearing first in the sort.
//   (goal: first fill in the outer ring of the display with smaller wedges; the big
//    ones will end late in the day, and will thus end up on the inside of the watchface)

// Third-priority sort: startTime, with objects starting later (smaller) appearing first in the sort.

Collections.sort(cr.instances, new Comparator<CalendarResults.Instance>() {
    public int compare(CalendarResults.Instance lhs, CalendarResults.Instance rhs) {
        if(lhs.displayColor != rhs.displayColor)
            return Long.compare(lhs.displayColor, rhs.displayColor);

        if(lhs.endTime != rhs.endTime)
            return Long.compare(lhs.endTime, rhs.endTime);

        return Long.compare(rhs.startTime, lhs.startTime);
    }
});
```

This is what lambdas look like before Java8: ***anonymous inner classes****. Ugly!*

Solver performance?

In typical use, the solver runs for <100ms (on the watch with ~10 events).

Overkill: re-running the solver with even n position swaps (never mind $n!$ total reorderings) could explode the solver time. Not acceptable. (And maybe the current scheme looks better than a packing-optimal one.)

The result is cached.

The cache is only invalidated once per hour or if new calendar data arrives.

And it looks great. Conclusion? Problem solved.

Time to let more people play with it!

“Private” beta

September 24, 2014

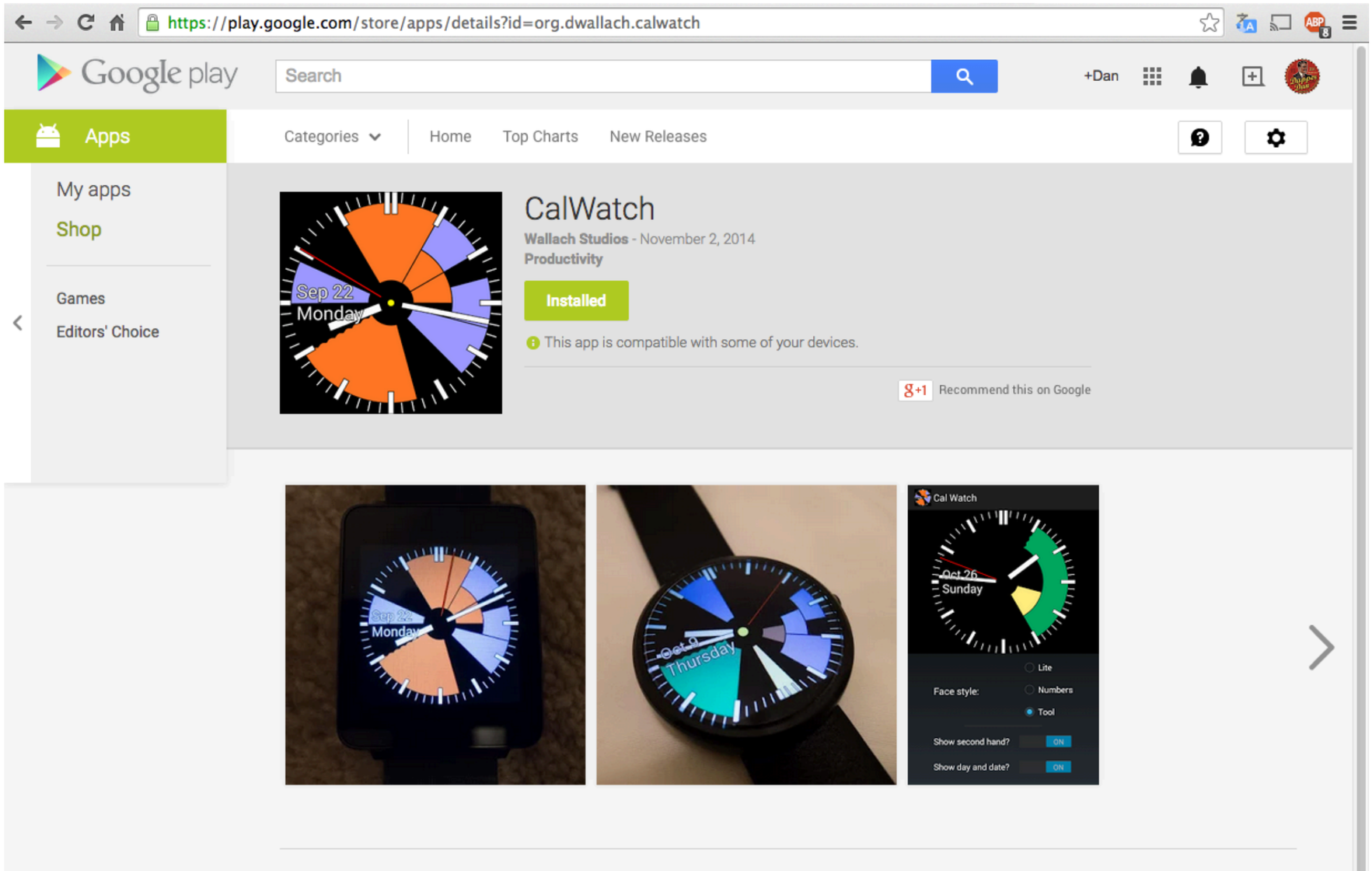
Invited personal friends

Found more bugs

Eventually “public” beta

G+ Community: “Android Wear Developers”

- Picked up 30-40 users
- Little useful bug feedback



Play Store bug reporting

When the app fails, the user can report it. It goes to the Play dev console.

Sadly, logs aren't included.

Also, doesn't seem to include on-watch crashes.

Better error reporting is available with Crashalytics library, but network permissions are required.

I don't *want* network permissions!

by device

Nexus 5 (hammerhead)	3	75.0%
Moto X (ghost)	1	25.0%

STACK TRACES | User Messages

Page **2** of **4** ◀ ▶ Go to page Go

Last reported
Oct 24 1:45 PM

Reports this week
0

Reports total
1

Application version
beta9 1

Android version
Android 4.4 1

Device
Moto X (ghost) 1

```
java.lang.NullPointerException
    at com.google.android.gms.wearable.internal.ag.a(Unknown Source)
    at com.google.android.gms.wearable.internal.ag.addListener(Unknown Source)
    at org.dwallach.calwatch.WearSender.onConnected(WearSender.java:124)
    at com.google.android.gms.common.internal.d.a(Unknown Source)
    at com.google.android.gms.common.api.a.bn(Unknown Source)
    at com.google.android.gms.common.api.a.d(Unknown Source)
    at com.google.android.gms.common.api.a$2.onConnected(Unknown Source)
    at com.google.android.gms.common.internal.d.a(Unknown Source)
    at com.google.android.gms.common.internal.d.y(Unknown Source)
    at com.google.android.gms.common.internal.c$g.a(Unknown Source)
    at com.google.android.gms.common.internal.c$g.d(Unknown Source)
    at com.google.android.gms.common.internal.c$b.bN(Unknown Source)
    at com.google.android.gms.common.internal.c$a.handleMessage(Unknown
    Source)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
```

Play Store bug reporting

When the app fails, the user can report it. It goes to the Play dev console.

Sadly, logs aren't included.

Also, doesn't seem to include on-watch crashes.

Better error reporting is available with Crashalytics library, but network permissions are required.

I don't *want* network permissions!

by device

Nexus 5 (hammerhead)	3	75.0%
Moto X (ghost)	1	25.0%

STACK TRACES | User Messages

Page 2 of 4 ◀ ▶ Go to page Go

Last reported
Oct 24 1:45 PM

Reports this week
0

Reports total
1

Application version
beta9 1

```
java.lang.NullPointerException
    at com.google.android.gms.wearable.internal.ag.a(Unknown Source)
    at com.google.android.gms.wearable.internal.ag.addListener(Unknown Source)
    at org.dwallach.calwatch.WearSender.onConnected(WearSender.java:124)
    at com.google.android.gms.common.internal.d.a(Unknown Source)
    at com.google.android.gms.common.api.a.bn(Unknown Source)
    at com.google.android.gms.common.api.a.d(Unknown Source)
    at com.google.android.gms.common.api.a$2.onConnected(Unknown Source)
    at com.google.android.gms.common.internal.d.a(Unknown Source)
    at com.google.android.gms.common.internal.d.y(Unknown Source)
```

Later on: open-source to the rescue.
https://github.com/PomepuyN/AndroidWearCrashReport

Finally, I think I nailed it

Careful attention to the Android SDK documents for lifecycle events

But zero documentation for watchfaces, so it's a guessing game

Careful attention to multithreaded discipline

Simultaneous redrawing from two threads will (rarely) crash the whole watch

→If the draw thread is active, suppress redraws from elsewhere

Many bugs found/fixed specifically for the Moto 360

Different behavior for "ambient" mode, etc.

Try/catch blocks everywhere

35 try blocks across ~4700 lines of code

Minor stuff

Android 5 "material design" widgets, if the phone is running Android 5

Proper handling of daylight savings time

Finally, I think I nailed it

Careful attention to the Android SDK documents for lifecycle events

But zero documentation for watchfaces, so it's a guessing game

Careful attention to multithreaded discipline

Simultaneous redrawing from two threads will (rarely) crash the whole watch

→If the draw thread is active, suppress redraws from elsewhere

Many bugs found/fixed specifically for the Moto 360

Different behavior for "ambient" mode, etc.

Try/catch blocks everywhere

35 try blocks across ~4700 lines of code

Minor stuff

Android 5 "material design" widgets, if the phone is running Android 5

Later on: official Android Watchface API eliminated all need for multithreading.

Beta 11 release: November 2, 2014

This time, for sure. No, really. (Line counts relative to alpha 1.)

CalendarFetcher.java	449→296 lines
CalendarResults.java	116→62 lines
ClockFace.java	755→771 lines
EventLayout.java	160→160 lines
PhoneActivity.java	246→407 lines
MyViewAnim.java	302→507 lines
misc Java + unit tests	76→304 lines
XML files	283→455 lines
WatchCalendarService.java	213→137 lines
WearSender.java	148→179 lines
BatteryMonitor.java	78→83 lines
ClockState.java	251→296 lines
TimeWrapper.java	115→204 lines
WearActivity.java	241→371 lines
WearReceiverService.java	196→256 lines
EventLayoutUniform.java	128 lines
Total	3555→4746 lines

As of late 2014...

Very rare misbehaviors. Stab-in-the-dark code to catch them.

E.g., Rarely, the event loop keeps running even after I kill it.

Reading over the logcat, some evidence of subtle Android system bugs.

Messages about running out of file descriptors.

Could be me leaking memory, object finalizers not running.

Only seems to happen multiple days after the app starts running.

Recovery after a crash is at least solid.

User sees "*Unfortunately, CalWatch has stopped working. [Ok]*".

Single click and it's back again like nothing happened.

As of late 2015...

Google *finally* announced real APIs for building watchfaces

I was able to delete a bunch of code!

Plus lots of new features, like “interactive” watchfaces

Also added a copy of the calendar on the watch (no need for me to send it!)

I also wrote a stopwatch app that communicates with my watchface

“Binder” IPC system: any app can “subscribe” to my stopwatch

Newer versions of Android Studio have sophisticated static checking

Specifically identifies common bug patterns among Android devs

Example: You use anon. inner classes to handle callbacks, but they’re also *closures*.

This can keep everything from being garbage collected when the app is being shut down.

Final code size numbers (release3e)

(Line counts relative to alpha 1.)

CalendarFetcher.java	449→310 lines
CalendarResults.java	116→0 lines
ClockFace.java	755→1083 lines
EventLayout.java	160→166 lines
PhoneActivity.java	246→221 lines
MyViewAnim.java	302→216 lines
misc Java + unit tests	76→576 lines
XML files	283→386 lines
Stopwatch IPC support	0→203 lines
WatchCalendarService.java	213→0 lines
WearSender.java	148→182 lines
BatteryMonitor.java	78→91 lines
ClockState.java	251→294 lines
TimeWrapper.java	115→192 lines
WearActivity.java	241→276 lines
WearReceiverService.java	196→245 lines
EventLayoutUniform.java	134 lines
Total	3555→4787 lines

What's next?

Need to test with other languages and locales

Right-to-left (Arabic, Farsi, Hebrew, ...) will be fun to test and debug

At least Android has good support for internationalization and localization

Open source?

Dual license: it's easy to share (GPLv3) and license commercially

http://www.cs.rice.edu/~dwallach/calwatch/

https://github.com/danwallach/CalWatch

Was it worth it?

Late 2014/early 2015:

"Big" rush of users

Fixed lots of bug fixes

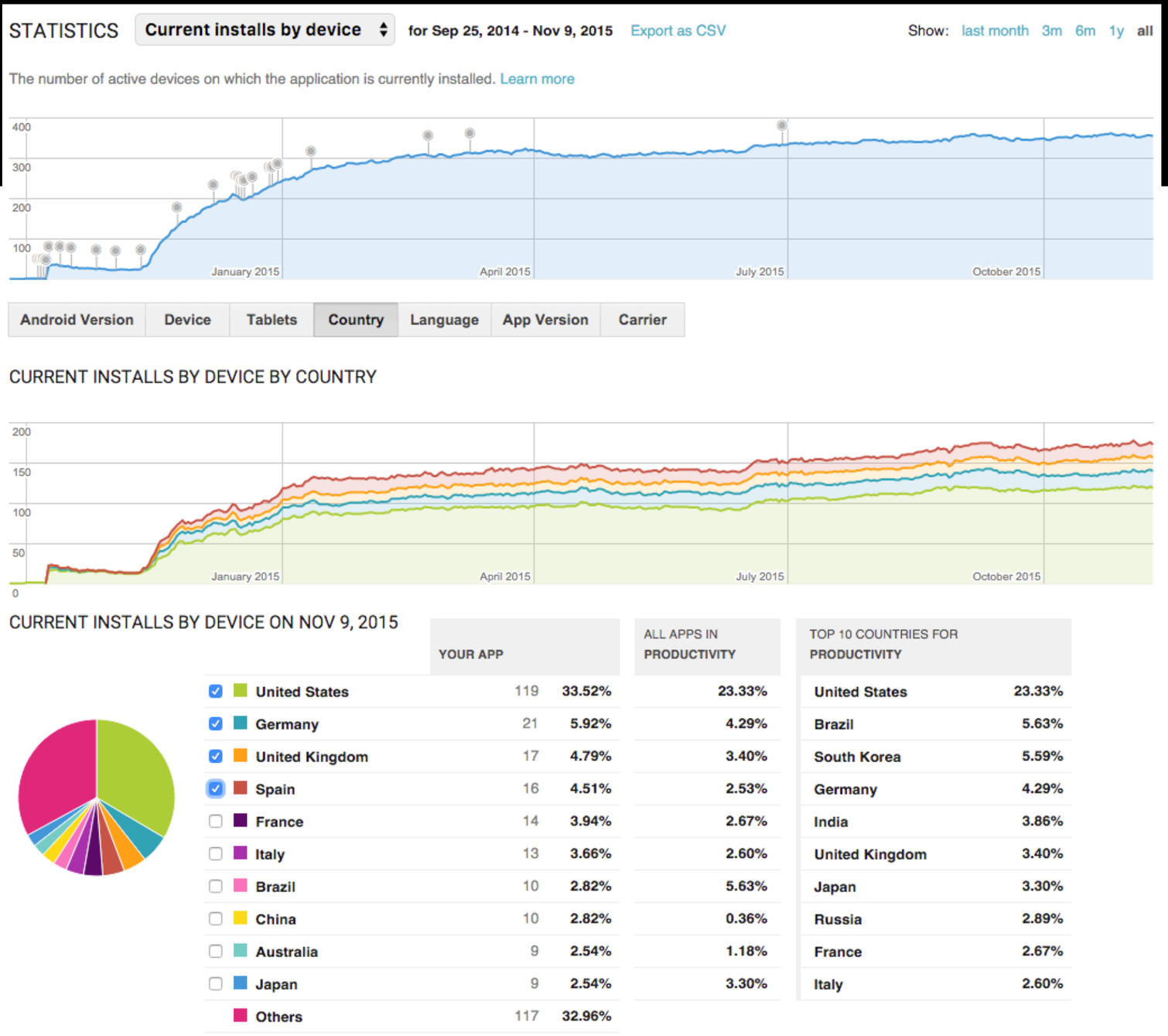
After they made the official APIs, Google promoted "hand picked" watchfaces

CalWatch wasn't "picked"

Growth is now negligible

But it was fun

Cool to have *my* watch running *my* code, showing *my* calendar!



Acknowledgements...

https://android.googlesource.com/platform/packages/providers/CalendarProvider/+/master/src/com/android/providers/calendar/CalendarReceiver.java

http://blog.timmattison.com/archives/2014/07/16/common-android-wear-tasks-for-developers/

http://constraints.cs.washington.edu/cassowary/

https://developer.android.com/google/auth/api-client.html

http://developer.android.com/reference/android/app/Activity.html

http://developer.android.com/reference/android/service/dreams/DreamService.html

http://developer.android.com/training/monitoring-device-state/battery-monitoring.html

https://developer.android.com/training/wearables/data-layer/events.html

http://sourabhsoni.com/how-to-use-intent-action_time_tick/

http://stackoverflow.com/questions/17097263/automatically-versioning-android-project-from-git-describe-with-android-studio-g

http://stackoverflow.com/questions/7597742/what-is-the-purpose-of-looper-and-how-to-use-it

http://toastdroid.com/2014/08/18/messageapi-simple-conversations-with-android-wear/

http://www.doubleencore.com/2014/07/create-custom-ongoing-notification-android-wear/

http://www.grokkingandroid.com/use-contentobserver-to-listen-to-changes/

http://www.hascode.com/2013/01/using-the-android-daydream-api/

http://www.tech-recipes.com/rx/49586/how-do-i-connect-an-android-wear-emulator-to-a-real-phone/

http://www.vogella.com/tutorials/AndroidServices/article.html

https://gist.github.com/kentarosu/52fb21eb92181716b0ce

https://github.com/square/wire

https://github.com/twotoasters/watchface-gears/blob/master/library/src/main/java/com/twotoasters/watchface/gears/widget/Watch.java

The future of Android?

Google is famously, amazingly secretive about whatever's coming next.
Example: What about Java8 support? Only *just* announced (summer 2016).

The good news: Android's market share is immense.
Massive tool support from industry & open source.
Example 1: You like Apple's Swift? Try JetBrains's Kotlin. Built into newer IntelliJ.
Example 2: "Functional-reactive" libraries rather than callbacks (RxAndroid, etc.)

The open-source world is remarkably helpful to you.

Android Wear 2 (currently in beta) has watch "complications".
But do I have time to support it? They're not making it "easy".

Kotlin? What's that?

For more "fun", I rewrote CalWatch in Kotlin

Super easy to do. IntelliJ has a built-in Java to Kotlin translator.

Kotlin has all sorts of features that aren't in Java

Lots of fun: writing "extension methods" for system classes

- vs. Java8 default methods on interfaces: Kotlin's version is more general

Also borrows some ideas from Apple's Swift

- A reference that might be **null**, and what to do with it
- Really, let's you code with Options, with language support to make it clean

Event sorting: Java code vs. Kotlin code

// Java7

```
Collections.sort(cr.instances, new Comparator<CalendarResults.Instance>() {
    public int compare(CalendarResults.Instance lhs, CalendarResults.Instance rhs) {
        if(lhs.displayColor != rhs.displayColor)
            return Long.compare(lhs.displayColor, rhs.displayColor);

        if(lhs.endTime != rhs.endTime)
            return Long.compare(lhs.endTime, rhs.endTime);

        return Long.compare(rhs.startTime, lhs.startTime);
    }
});
```

// Kotlin

```
cr.sortedWith(compareBy<WireEvent> { it.displayColor }
                .thenBy { it.endTime }
                .thenByDescending { it.startTime })  // functional lists: we then return this value
```

Event sorting: Java code vs. Kotlin code

// Java7

```
Collections.sort(cr.instances, new Comparator<CalendarResults.Instance>() {
    public int compare(CalendarResults.Instance lhs, CalendarResul
        if(lhs.displayColor != rhs.displayColor)
            return Long.compare(lhs.displayColor, rhs.displayColor

        if(lhs.endTime != rhs.endTime)
            return Long.compare(lhs.endTime, rhs.endTime);

        return Long.compare(rhs.startTime, lhs.startTime);
    }
});
```

Fun Kotlin feature: lambdas look like curly-braced code blocks.

// Kotlin

```
cr.sortedWith(compareBy<WireEvent> { it.displayColor }
                .thenBy { it.endTime }
                .thenByDescending { it.startTime })  // functional lists: we then return this value
```

Kotlin: the bottom line

It's just Java underneath, so you generally know what's going on

Gradually took advantage of Kotlin's new features, simplifying my code

APK actually *shrunk* when compiling my Kotlin version!
Kotlin's very clever about how it compiles down to Java

Detailed writeup:
https://discuss.kotlinlang.org/t/experience-porting-an-android-app-to-kotlin/1399

Maybe in future years, Comp215 will move to Kotlin?
IntelliJ itself is increasingly written in Kotlin. It's pretty nice.

Android: the bottom line

If you know Java, you're ready for Android

If you only know Java8 from Comp215, Android code is mostly Java7. Ugly.

API documentation is good

But you need to understand the "application lifecycle" model, which isn't obvious

StackOverflow and example apps are helpful, but often flawed

Don't think you're going to build something and immediately get users

"App discovery" is a messy process.