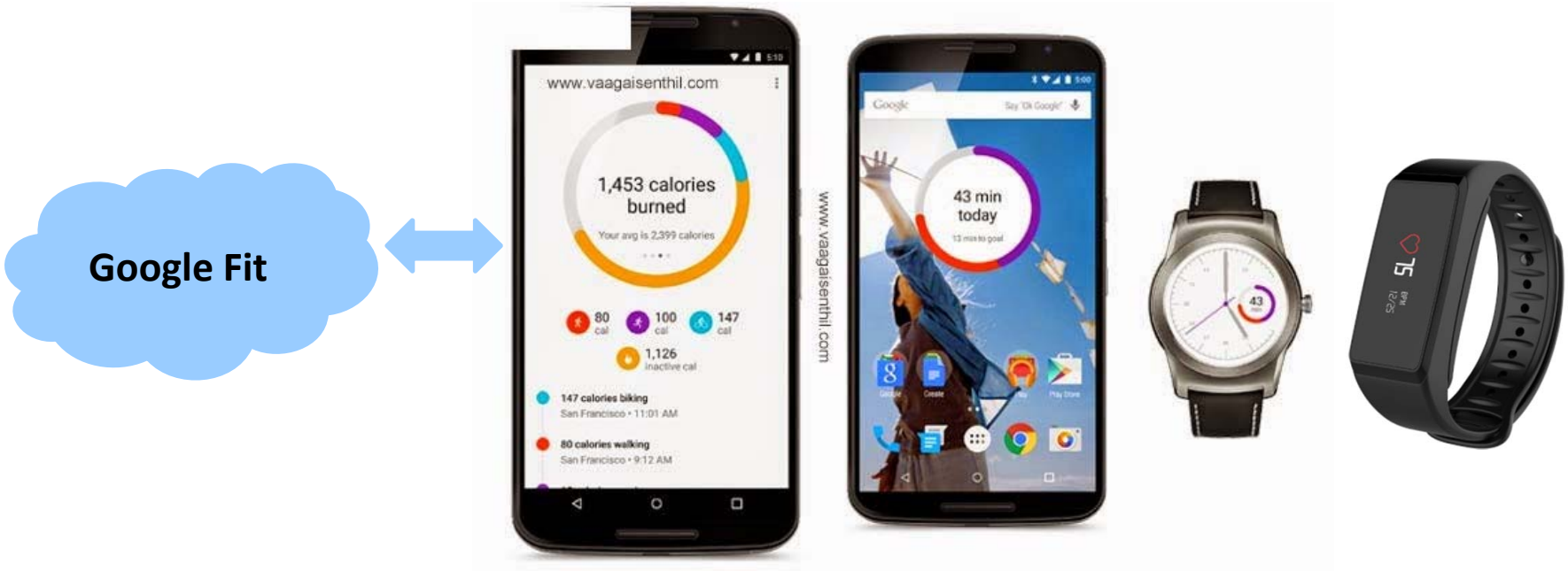
Google Fit

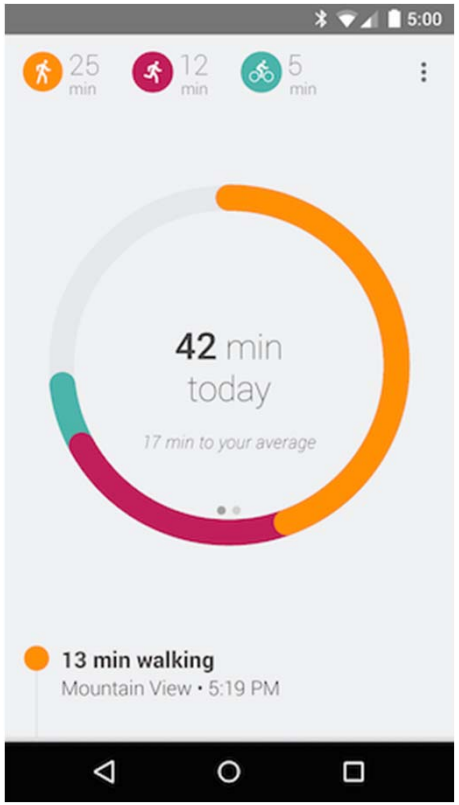
- Google Fit, part of Google Play Services
- Single cloud storage record for all user's fitness apps (myfitnesspal), gadgets (fitbit), etc
- Third-party app developers can read-write user's health data in single Google Fit repository, tied to gmail account (cloud storage)
- User can track health data using multiple devices, apps, store data



Google Fit

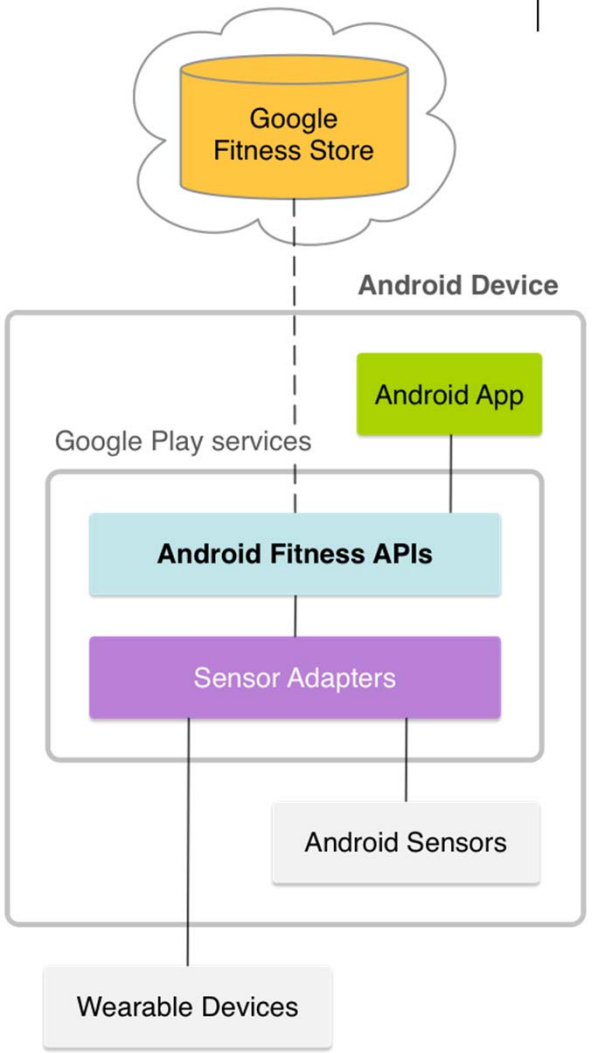
Ref:http://en.wikipedia.org/wiki/Google_Fit

- User data not lost if user upgrades, change or loses device
- **Google Fit components:**
  - **App:** Free from Google, supports fitness tracking, unified view of progress, accessible from multiple devices
  - **Cloud Storage:** Single repository of user's fitness data
  - **API:** Third-party Developers can program app to access, read, write Google Fit record



Google Fit Features

- **Sensors API:** Allows app access raw information from sensors on user's devices (including smartphones and Android wear devices)
- **Recording API:** Allows app to automate storage of fitness data using subscriptions.
  - Specific data are automatically stored in the background
  - App can access this data on any device user has granted permission to



Google Fit Features

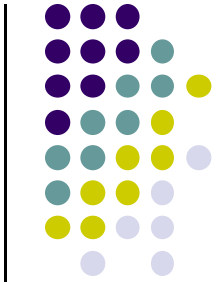
- **History:** App can access user's fitness history
  - Supports inserting, deleting and querying previously stored fitness data
  - Can also import batch data into Google Fit
- **Bluetooth Low Energy:** Access data directly from Bluetooth, store data from them
  - Apps can find nearby Bluetooth devices and store data from them

Bluetooth Glucometer

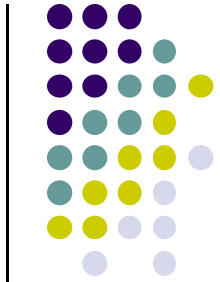
Bluetooth Weight Scale

Google Fit API

http://en.wikipedia.org/wiki/Google_Fit



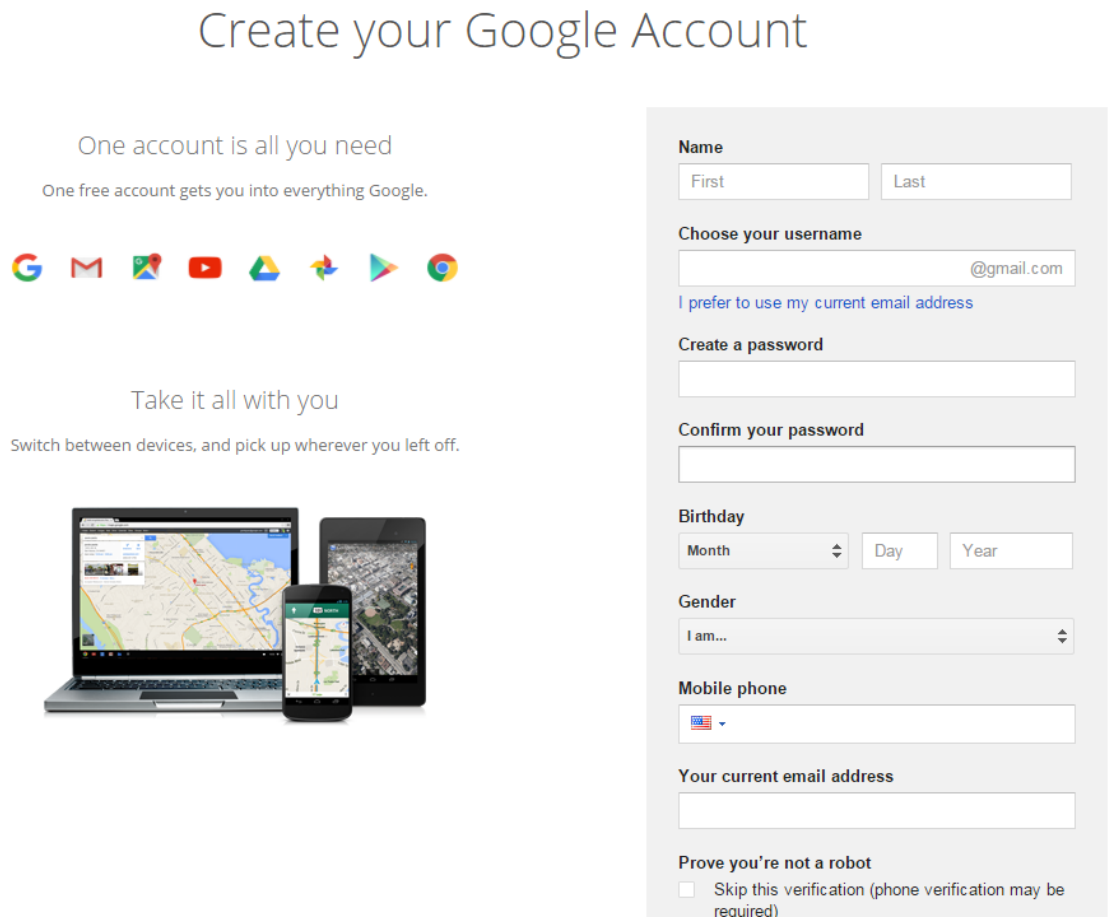
- Google Fit API also has API for step counting
- i.e. Low end phones without step counter can use Google Fit's step counting API
  - Implemented as a Google service
- Also **DetectedActivity** API to detect smartphone user's current activity
- Currently detects 6 states:
  - In vehicle
  - On Bicycle
  - On Foot
  - Still
  - Tilting
  - Unknown



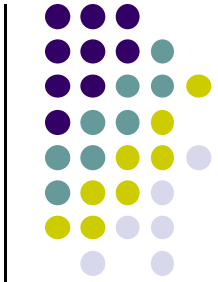
Using Google Fit

Google Fit API Setup:
Step 1: Create Google Account

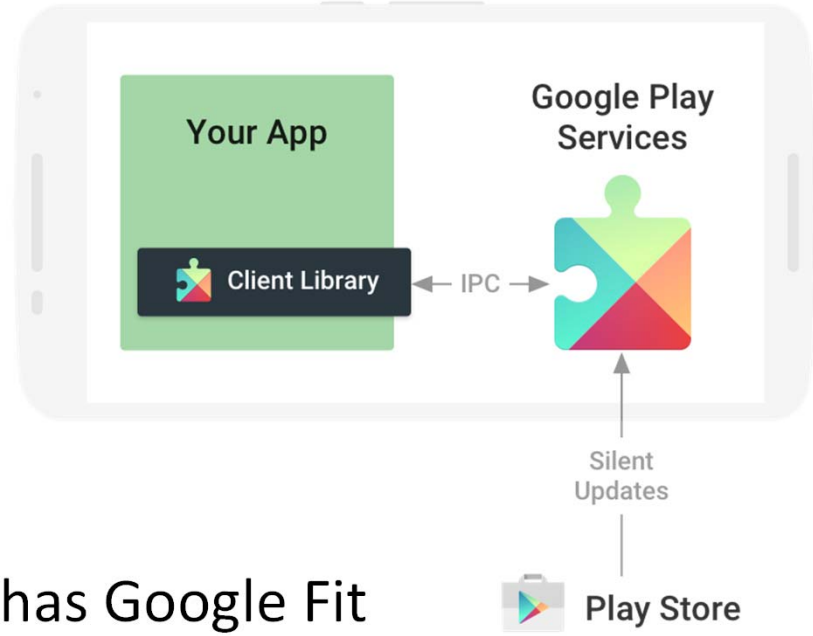
- Can use your existing Google account or create new one for testing



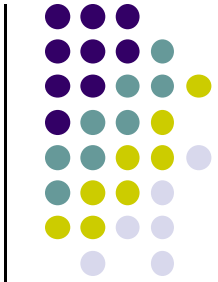
Google Fit API Setup:
Step 2: Get Google Play Services



- Google Play Services: API package and background service
- Allows apps to communicate with Google's services (e.g. maps, Google+, Google Drive, **Google Fit**, etc)
- Google Play Services APK
  - Contains individual Google services
  - Runs as background services on Android client that apps interact with
  - Downloaded through Google Play store
- Google updates Play Services often without depending on software updates by phone makers
- Google Play services 7.0 and higher has Google Fit

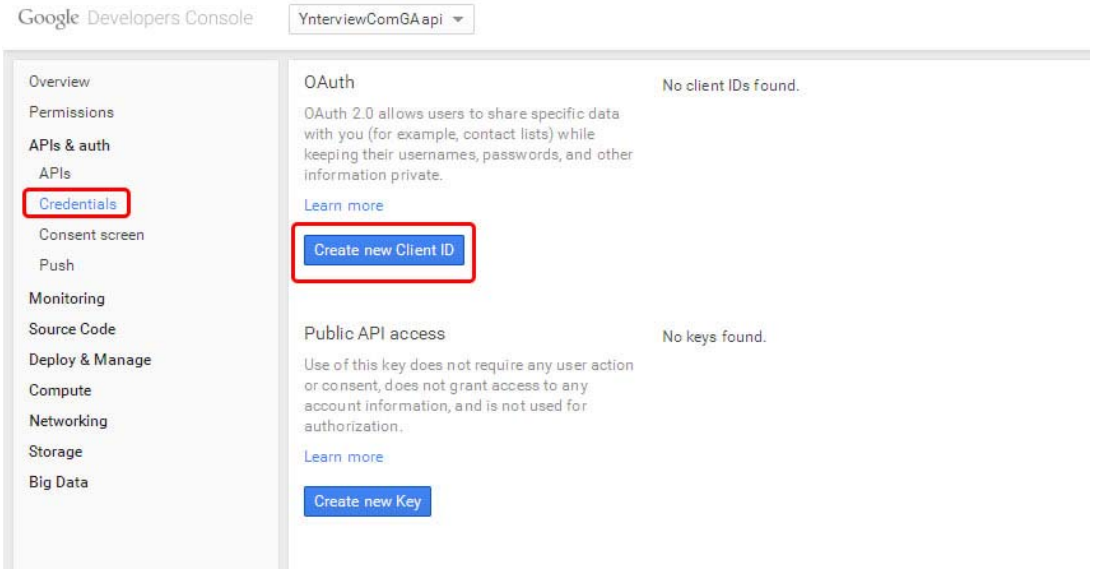


Google Fit API Setup:
Step 3: Get an Oauth 2.0 Client ID

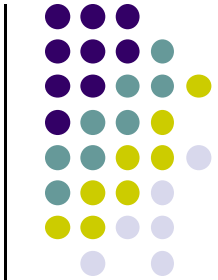


- Oauth 2.0 is open standard for authorization
- Allows users to log into third party websites using their Microsoft, Google, Facebook or Twitter accounts
- Can get Oauth 2.0 client ID through Google Developers Console
- **See: https://developers.google.com/fit/android/get-api-key**
- Oauth 2.0 client ID is string of characters. E.g.

```
780816631155-gbvyo1o7r2pn95qc4ei9d61io4uh48hl.apps.googleusercontent.com
```



Google Fit API Setup:
Step 4: Configure your Project

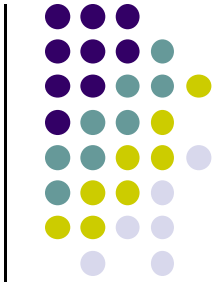


- Android Studio is recommended for development
- Create Android Studio project
- Add Google Play services as dependency in **build.gradle** file

```
apply plugin: 'com.android.application'
...

dependencies {
    compile 'com.google.android.gms:play-services-fitness:8.4.0'
}
```

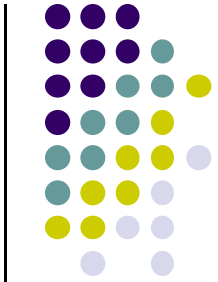
https://developers.google.com/fit/android/get-started

Google Fit API Setup:
Step 5: Connect to the Fitness Service



- Connect to the appropriate fitness service and use it
  - **Fitness.SENSORS_API**
  - **Fitness.RECORDING_API**
  - **Fitness.HISTORY_API**
  - **Fitness.SESSIONS_API**
  - **Fitness.BLE_API**
  - **Fitness.CONFIG_API**

Creating Google Fit Client

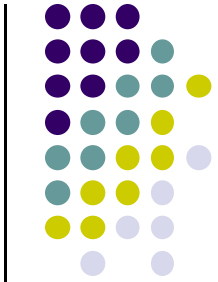


Step 1: Define variables to help track Google Fit connection

```
private GoogleApiClient mClient = null;
```

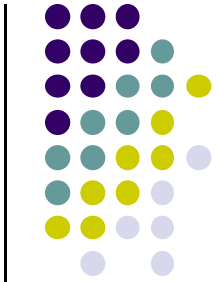
Creating Google Fit Client

Step 2: Connect to Google Fit, Check Permissions



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // This method sets up our custom logger, which will print all log messages to device and logcat
    initializeLogging();

    // When permissions are revoked the app is restarted so onCreate
    if (!checkPermissions()) {
        requestPermissions();
    }
}

@Override
protected void onResume() {
    super.onResume();

    // If user denies permissions then uses Settings to re-enable them, app will start working
    buildFitnessClient();
}
```

Main GoogleFit Client Commands and Callbacks

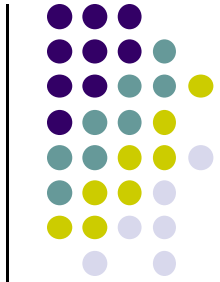


- GoogleApiClient.Builder: Used initially to create GoogleFit client, authenticates user, allows user access Fitness APIs, specifies app scopes
- Scopes? Read/write permissions to different data types

- onConnectionSuspended( ): Called when sensor connection gets lost

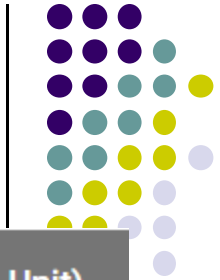
- onConnectionFailed( ): Called when Google Play Services connection fails intentionally
  - Some example reasons for connection failure: User never signed in before, has multiple Google accounts and needs to specify which one to use, etc)

Google Fit Data Types



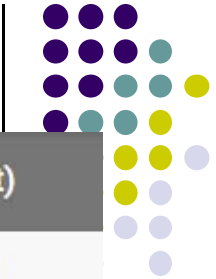
- Google Fit supports:
  - Instantaneous readings with timestamp (e.g. Current user activity)
  - Aggregate statistics over time interval (e.g. Total calories expended over a time interval)
- 3 Google Fit data types
  - **Public data types:** Standard data types that any app can read and write (e.g. Step count)
  - **Private custom data types:** custom types defined by a specific app. Only that app can read/write this data
  - **Shareable data types:** App developers can submit data types which can be shared after reviewed and approved (E.g. types for Nike Fuel)

Example Google Fit Public Instantaneous Data Types



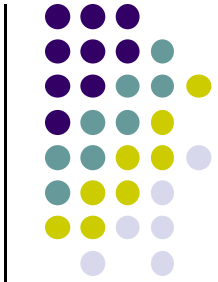
Data Type Name	Description	Permission	Fields (Format—Unit)
com.google.activity.sample	Instantaneous sample of the current activity.	Activity	activity (int—enum) confidence (float—percent)
com.google.activity.segment	Continuous time interval of a single activity.	Activity	activity (int—enum)
(deprecated) com.google.calories.consumed	Total calories consumed over a time interval.	Activity	calories (float—kcal)
com.google.calories.expended	Total calories expended over a time interval.	Activity	calories (float—kcal)
com.google.distance.delta	Distance covered since the last reading.	Location	distance (float—meters)
com.google.heart_rate.bpm	Heart rate in beats per minute.	Body	bpm (float—bpm)
com.google.height	The user's height, in meters.	Body	height (float—meters)
com.google.step_count.cadence	Instantaneous cadence in steps per minute.	Activity	rpm (float—steps/min)
com.google.step_count.delta	Number of new steps since the last reading.	Activity	steps (int—count)

Example Google Fit Public Aggregate Data Types



Data Type Name	Description	Permission	Fields (Format—Unit)
com.google.activity.summary	Total time and number of segments in a particular activity for a time interval.	Activity	activity (int—enum) duration (int—ms) num_segments (int—count)
com.google.heart_rate.summary	Average, maximum, and minimum beats per minute for a time interval.	Body	average (float—bpm) max (float—bpm) min (float—bpm)
com.google.location.bounding_box	A bounding box for the user's location over a time interval.	Location	low_latitude (float—degrees) low_longitude (float—degrees) high_latitude (float—degrees) high_longitude (float—degrees)
com.google.nutrition.summary	User's nutrition intake during a time interval.	Nutrition	nutrients (Map<String, float>—calories/grams/IU) meal_type (int—enum) food_item (String—n/a)
com.google.power.summary	Average, maximum, and minimum power generated while performing an activity.	Activity	average (float—watts) max (float—watts) min (float—watts)
com.google.speed.summary	Average, maximum, and minimum speed over ground over a time interval.	Location	average (float—m/s) max (float—m/s) min (float—m/s)
com.google.weight.summary	Average, maximum, and minimum weight over a time interval.	Body	average (float—kg) max (float—kg) min (float—kg)

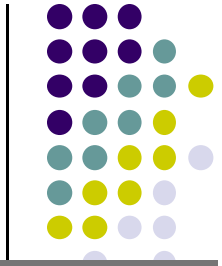
Google Fit Data Scopes



- Scopes are strings that specify
  - Types of data app can access
  - Level of access (Read/write permissions)
- App requests a scope of access during initial connection, access data if permission received

Permission	Scope	Type of Access	Data Types
Activity	FITNESS_ACTIVITY_READ	Read	com.google.activity.sample com.google.activity.segment com.google.activity.summary (deprecated) com.google.calories.consumed com.google.calories.expended com.google.cycling.pedaling.cadence com.google.power.sample com.google.step_count.cadence com.google.step_count.delta com.google.activity.exercise
	FITNESS_ACTIVITY_READ_WRITE	Read and Write	
Body	FITNESS_BODY_READ	Read	com.google.heart_rate.bpm com.google.heart_rate.summary com.google.height com.google.weight com.google.weight.summary
	FITNESS_BODY_READ_WRITE	Read and Write	

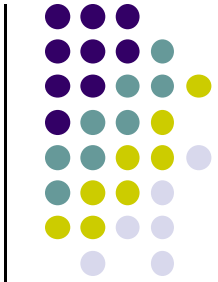
Google Fit Data Scopes



Permission	Scope	Type of Access	Data Types
Location	FITNESS_LOCATION_READ	Read	com.google.cycling.wheel_revolution.cumulative com.google.cycling.wheel.revolutions com.google.distance.delta com.google.location.sample com.google.location.bounding_box com.google.speed com.google.speed.summary
	FITNESS_LOCATION_READ_WRITE	Read and Write	
Nutrition	FITNESS_NUTRITION_READ	Read	com.google.nutrition.item com.google.nutrition.summary
	FITNESS_NUTRITION_READ_WRITE	Read and Write	

Creating Google Fit Client

Step 2: Connect to Google Fit, Check Permissions



```
private void buildFitnessClient() {
   if (mClient == null && checkPermissions()) {
      mClient = new GoogleApiClient.Builder(this)
            .addApi(Fitness.SENSORS_API)
            .addScope(new Scope(Scopes.FITNESS_LOCATION_READ))
            .addConnectionCallbacks(
                  new GoogleApiClient.ConnectionCallbacks() {
                     @Override
                     public void onConnected(Bundle bundle) {
                        Log.i(TAG, "Connected!!!");
                        // Now you can make calls to the Fitness APIs.
                        findFitnessDataSources();
                     }
```

Create Google Fit client

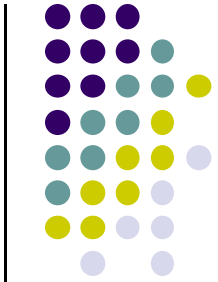
Provide variable for tracking connection

Add Fitness sensor API

Request read access to fitness location data

Creating Google Fit Client

Step 2: Connect to Google Fit, Check Permissions



```
        @Override
        public void onConnectionSuspended(int i) {
            // If your connection to the sensor gets lost at some point,
            // you'll be able to determine the reason and react to it here.
            if (i == ConnectionCallbacks.CAUSE_NETWORK_LOST) {
                Log.i(TAG, "Connection lost.  Cause: Network Lost.");
            } else if (i
                    == ConnectionCallbacks.CAUSE_SERVICE_DISCONNECTED) {
                Log.i(TAG,
                        "Connection lost.  Reason: Service Disconnected");
            }
        }
    }
)
```

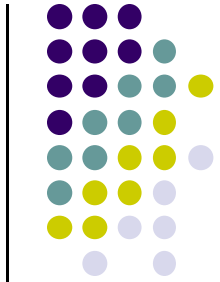
Called if sensor connection lost

Creating Google Fit Client

Step 2: Connect to Google Fit, Check Permissions

```
            .enableAutoManage(this, 0, new GoogleApiClient.OnConnectionFailedListener() {
                @Override
                public void onConnectionFailed(ConnectionResult result) {
                    Log.i(TAG, "Google Play services connection failed. Cause: " +
                            result.toString());
                    Snackbar.make(
                            MainActivity.this.findViewById(R.id.main_activity_view),
                            "Exception while connecting to Google Play services: " +
                                    result.getErrorMessage(),
                            Snackbar.LENGTH_INDEFINITE).show();
                }
            })
            .build();
    }
}
```

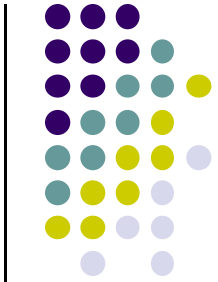
Called if Google Play connection fails



Activity Recognition Using Google Fit

Activity Recognition Using Google Fit

Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

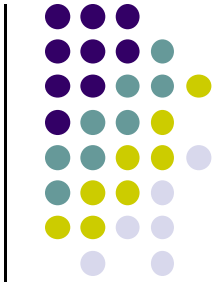


- Google Fit can:
  - Recognize user's current activity (Running, walking, in a vehicle or still)
  - Why? E.g. If user is driving, don't send notifications
  - Track user's steps

- Project Setup similar to previously described case:
  - Create Android Studio project with blank Activity (minimum SDK 14)
  - In build.gradle file, define latest Google Play services (8.4) as dependency

Activity Recognition Using Google Fit

Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

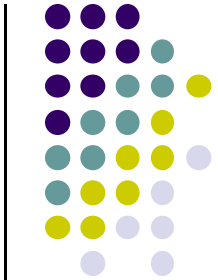


- Create new class **ActivityRecognizedService** which extends **IntentService**
- Throughout user's day, GooglePlay sends user's activity to this IntentService
- Need to program this Intent to handle incoming user activity

```
01  public class ActivityRecognizedService extends IntentService {
02
03      public ActivityRecognizedService() {
04          super("ActivityRecognizedService");
05      }
06
07      public ActivityRecognizedService(String name) {
08          super(name);
09      }
10
11      @Override
12      protected void onHandleIntent(Intent intent) {
13      }
14  }
```

Activity Recognition Using Google Fit

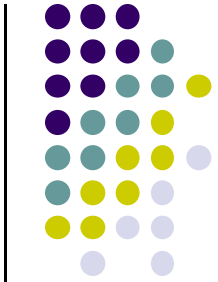
Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials



- Modify **AndroidManifest.xml** to
  - Declare **ActivityRecognizedService**
  - Add com.google.android.gms.permission.ACTIVITY_RECOGNITION permission

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

    <uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:name=".ActivityRecognizedService" />
    </application>

</manifest>
```

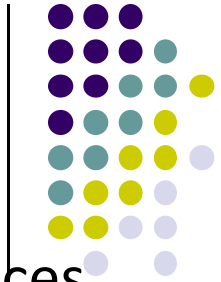
Project Setup



- To connect to Google Play Services, provide **GoogleApiClient** variable type and implement callbacks

```
01 public class MainActivity extends AppCompatActivity implements GoogleApiClient.ConnectionCallbacks,
02 GoogleApiClient.OnConnectionFailedListener {
03
04     public GoogleApiClient mApiClient;
05
06     @Override
07     protected void onCreate(Bundle savedInstanceState) {
08         super.onCreate(savedInstanceState);
09         setContentView(R.layout.activity_main);
10     }
11
12     @Override
13     public void onConnected(@Nullable Bundle bundle) {
14
15     }
16
17     @Override
18     public void onConnectionSuspended(int i) {
19
20     }
21
22     @Override
23     public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
24
25     }
}
```

Requesting Activity Recognition



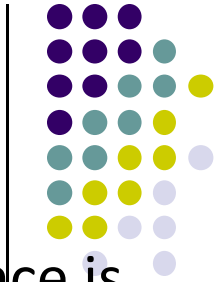
- In onCreate, initialize client and connect to Google Play Services

```
01  @Override
02  protected void onCreate(Bundle savedInstanceState) {
03      super.onCreate(savedInstanceState);
04      setContentView(R.layout.activity_main);
05
06      mApiClient = new GoogleApiClient.Builder(this)
07              .addApi(ActivityRecognition.API)
08              .addConnectionCallbacks(this)
09              .addOnConnectionFailedListener(this)
10              .build();
11
12      mApiClient.connect();
13  }
```

Request ActivityRecognition.API

Associate listeners with our instance of GoogleApiClient

Requesting Activity Recognition



- Once **GoogleApiClient** has connected, **onConnected( )** instance is called
- Need to create a **PendingIntent** that goes to our **IntentService**
- Also set how often API shold check user's activity in milliseconds

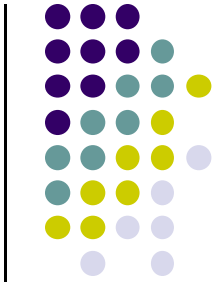
```
@Override
public void onConnected(@Nullable Bundle bundle) {
    Intent intent = new Intent( this, ActivityRecognizedService.class );
    PendingIntent pendingIntent = PendingIntent.getService( this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT );
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates( mApiClient, 3000, pendingIntent );
}
```

Handling Activity Recognition

- Our app now needs to attempt to recognize the user's activity every 3 seconds, send data to **ActivityRecognizedService**
- In **onHandleIntent( )** method of **ActivityRecognizedService**
  - Validate that received intent contains activity recognition data
  - If so, extract **ActivityRecognitionResult** from the Intent
  - Retrieve list of possible activities by calling **getProbableActivities( )** on **ActivityRecognitionResult** object

```
1@Override
2protected void onHandleIntent(Intent intent) {
3   if(ActivityRecognitionResult.hasResult(intent)) {
4      ActivityRecognitionResult result = ActivityRecognitionResult.extractResult(intent);
5      handleDetectedActivities( result.getProbableActivities() );
6   }
7}
```

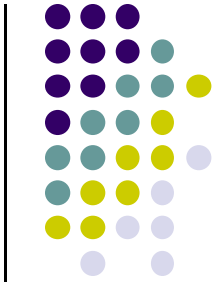
Handling Activity Recognition



- Simply log each detected activity and display how confident Google Play services is that user is performing this activity

```
private void handleDetectedActivities(List<DetectedActivity> probableActivities) {
    for( DetectedActivity activity : probableActivities ) {
        switch( activity.getType() ) {
            case DetectedActivity.IN_VEHICLE: {
                Log.e( "ActivityRecogition", "In Vehicle: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.ON_BICYCLE: {
                Log.e( "ActivityRecogition", "On Bicycle: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.ON_FOOT: {
                Log.e( "ActivityRecogition", "On Foot: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.RUNNING: {
                Log.e( "ActivityRecogition", "Running: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.STILL: {
                Log.e( "ActivityRecogition", "Still: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.TILTING: {
                Log.e( "ActivityRecogition", "Tilting: " + activity.getConfidence() );
                break;
            }
```

Handling Activity Recognition



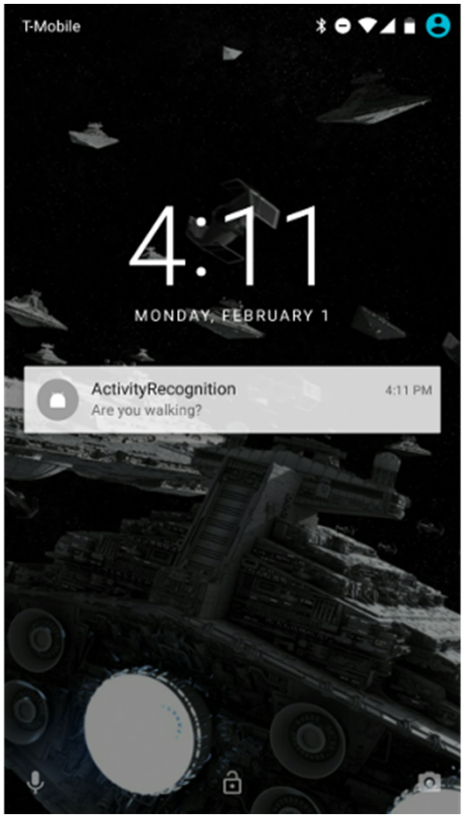
- If confidence is > 75, activity detection is probably accurate

```
            case DetectedActivity.WALKING: {
                Log.e( "ActivityRecogition", "Walking: " + activity.getConfidence() );
                if( activity.getConfidence() >= 75 ) {
                    NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
                    builder.setContentText( "Are you walking?" );
                    builder.setSmallIcon( R.mipmap.ic_launcher );
                    builder.setContentTitle( getString( R.string.app_name ) );
                    NotificationManagerCompat.from(this).notify(0, builder.build());
                }
                break;
            }
            case DetectedActivity.UNKNOWN: {
                Log.e( "ActivityRecogition", "Unknown: " + activity.getConfidence() );
                break;
            }
        }
    }
}
```

Sample Output of Program

- Sample displayed on development console

```
1  E/ActivityRecogition: On Foot: 92
2  E/ActivityRecogition: Running: 87
3  E/ActivityRecogition: On Bicycle: 8
4  E/ActivityRecogition: Walking: 5
```



- Or provided as notification to user

- Full code at: https://github.com/tutsplus/Android-ActivityRecognition