Lab on apps development for tablets, smartphones and smartwatches

Week 5: Firebase and Data Management

Dr. Giovanni Ansaloni, Prof. David Atienza

Ms. Halima Najibi, Ms. Farnaz Forooghifar,
Mr. Renato Zanetti, Mr. Saleh Baghersalimi, Mr. Alireza Amirshahi

Institute of Electrical Engineering (IEL) – Faculty of Engineering (STI)

A step back: using WearService

- Communication with Android Wear devices

- Two Wear APIs:
  - *Message API*
    - Short arrays (typically strings)
    - one-way requests

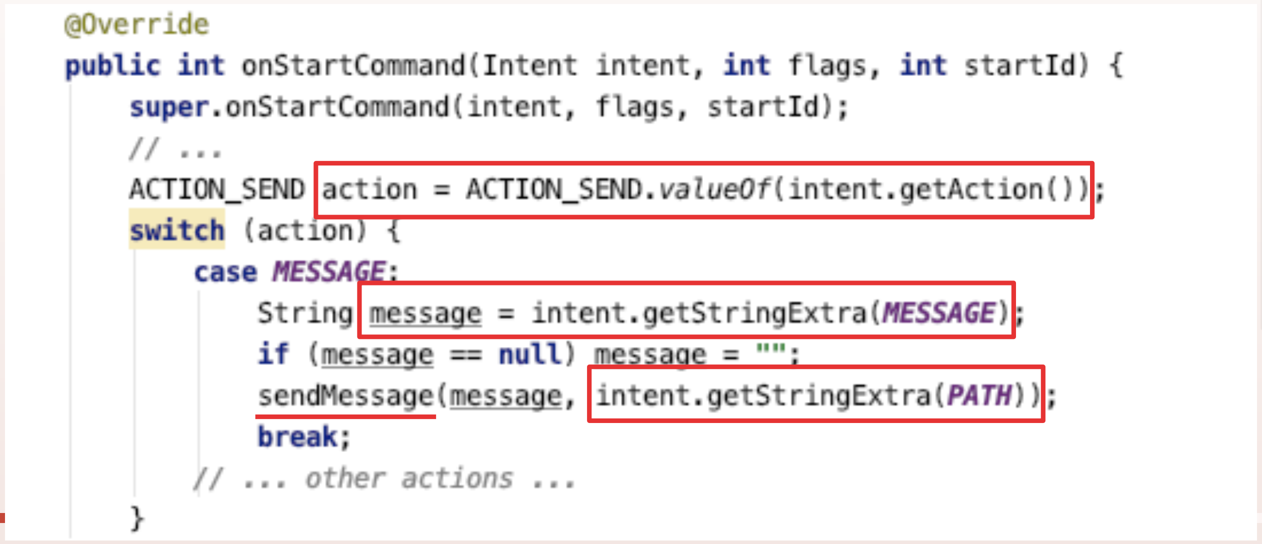
  - *Data API*
    - DataMaps (structured bundles of data)
    - From one node to **all connected nodes**
    - Synchronizes data, similar to shared memory

- Activities interact with WearService with Intents on sender and receiver side

A step back: using WearService

- To send a Message, WearService needs information on:
  - action
  - content
  - path

sent to watch



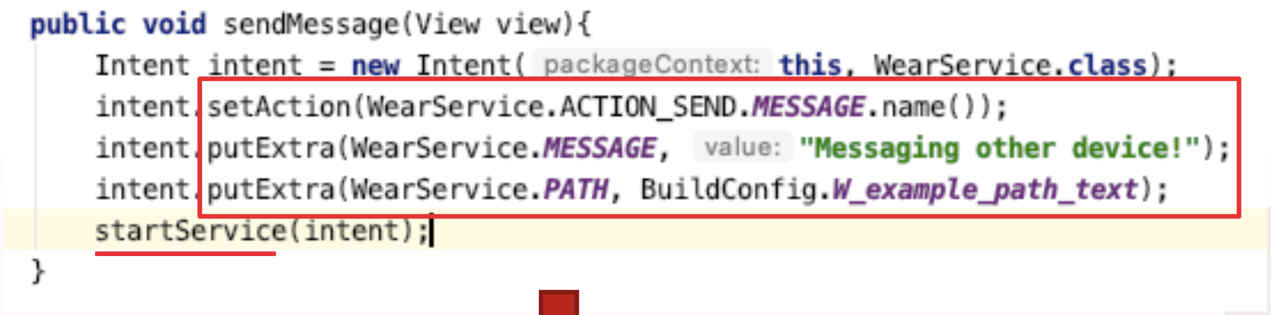
```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    super.onStartCommand(intent, flags, startId);
    // ...
    ACTION_SEND action = ACTION_SEND.valueOf(intent.getAction());
    switch (action) {
        case MESSAGE:
            String message = intent.getStringExtra(MESSAGE);
            if (message == null) message = "";
            sendMessage(message, intent.getStringExtra(PATH));
            break;
        // ... other actions ...
    }
```

WearService

onStartCommand()

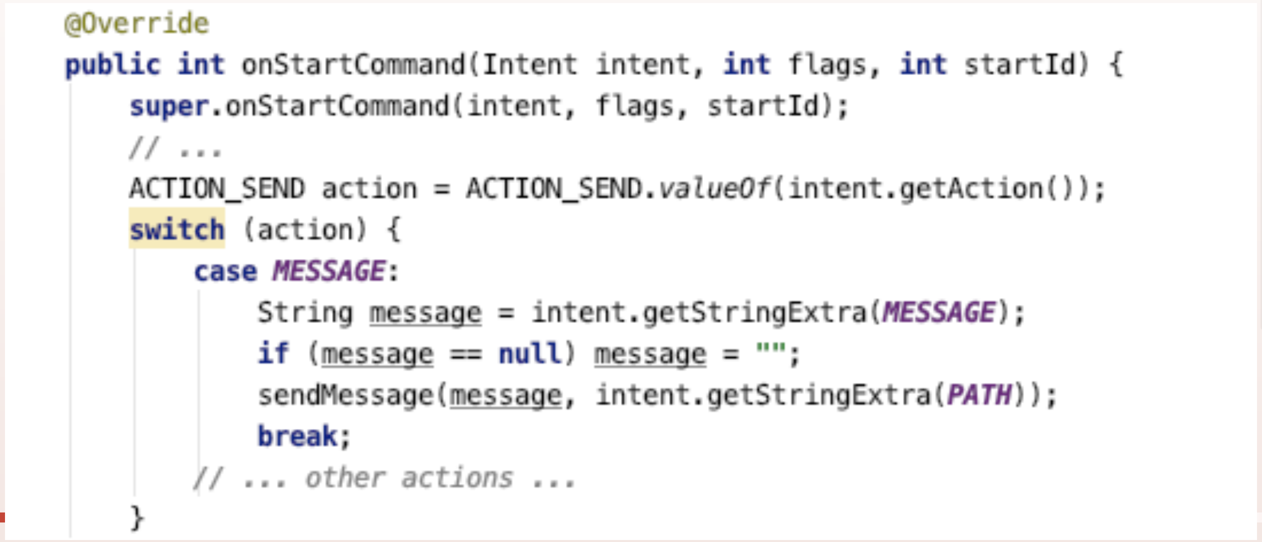
A step back: using WearService

- Intent from application



```
public void sendMessage(View view){
    Intent intent = new Intent( packageContext: this, WearService.class);
    intent.setAction(WearService.ACTION_SEND.MESSAGE.name());
    intent.putExtra(WearService.MESSAGE,  value: "Messaging other device!");
    intent.putExtra(WearService.PATH, BuildConfig.W_example_path_text);
    startService(intent);
}
```

- action
- content
- path



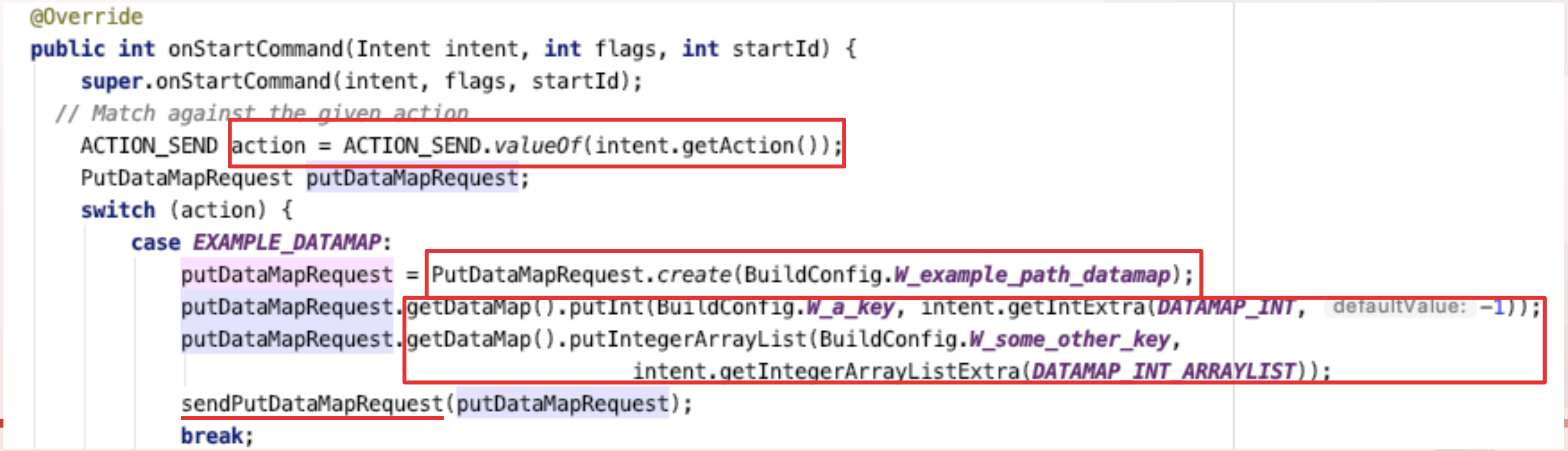
```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    super.onStartCommand(intent, flags, startId);
    // ...
    ACTION_SEND action = ACTION_SEND.valueOf(intent.getAction());
    switch (action) {
        case MESSAGE:
            String message = intent.getStringExtra(MESSAGE);
            if (message == null) message = "";
            sendMessage(message, intent.getStringExtra(PATH));
            break;
        // ... other actions ...
    }
```

WearService

onStartCommand()

A step back: using WearService

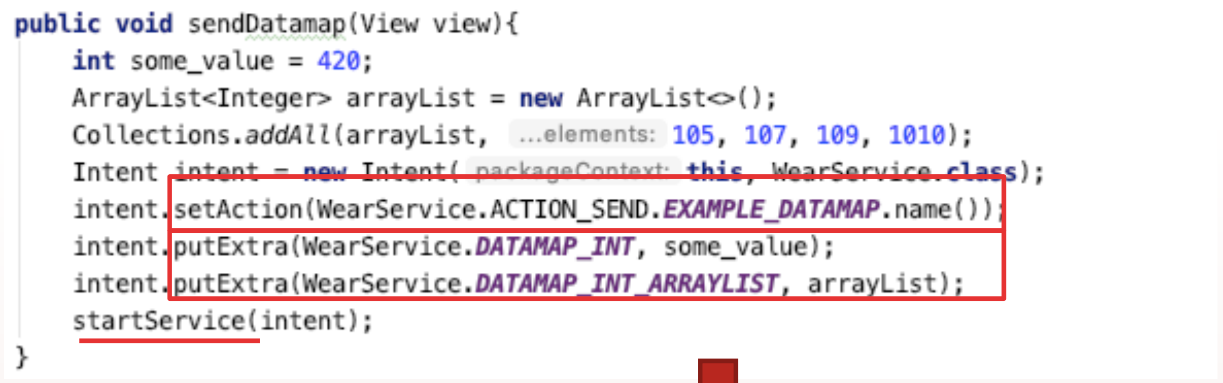
- Sending DataMaps, is similar (we use sendPutDataMapRequest())
  - action
  - content → *Key/value pairs, keys shared among tablet and watch*
  - path



```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    super.onStartCommand(intent, flags, startId);
  // Match against the given action
    ACTION_SEND action = ACTION_SEND.valueOf(intent.getAction());
    PutDataMapRequest putDataMapRequest;
    switch (action) {
        case EXAMPLE_DATAMAP:
            putDataMapRequest = PutDataMapRequest.create(BuildConfig.W_example_path_datamap);
            putDataMapRequest.getDataMap().putInt(BuildConfig.W_a_key, intent.getIntExtra(DATAMAP_INT,  defaultValue: -1));
            putDataMapRequest.getDataMap().putIntegerArrayList(BuildConfig.W_some_other_key,
                                     intent.getIntegerArrayListExtra(DATAMAP_INT_ARRAYLIST));
            sendPutDataMapRequest(putDataMapRequest);
            break;
```

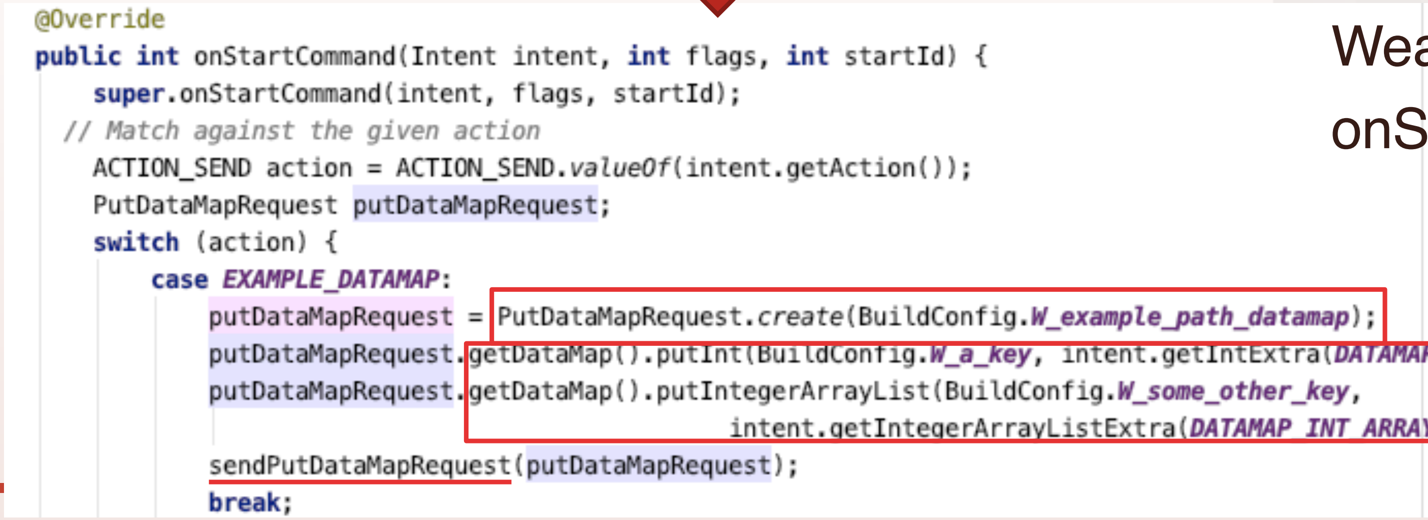
A step back: using WearService

- Intent from application



```
public void sendDatamap(View view){
    int some_value = 420;
    ArrayList<Integer> arrayList = new ArrayList<>();
    Collections.addAll(arrayList, ...elements: 105, 107, 109, 1010);
    Intent intent = new Intent( packageContext: this, WearService.class);
    intent.setAction(WearService.ACTION_SEND.EXAMPLE_DATAMAP.name());
    intent.putExtra(WearService.DATAMAP_INT, some_value);
    intent.putExtra(WearService.DATAMAP_INT_ARRAYLIST, arrayList);
    startService(intent);
}
```

- action
- content
- ~~path~~



```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    super.onStartCommand(intent, flags, startId);
  // Match against the given action
    ACTION_SEND action = ACTION_SEND.valueOf(intent.getAction());
    PutDataMapRequest putDataMapRequest;
    switch (action) {
        case EXAMPLE_DATAMAP:
            putDataMapRequest = PutDataMapRequest.create(BuildConfig.W_example_path_datamap);
            putDataMapRequest.getDataMap().putInt(BuildConfig.W_a_key, intent.getIntExtra(DATAMAP_INT,  defaultValue: -1));
            putDataMapRequest.getDataMap().putIntegerArrayList(BuildConfig.W_some_other_key,
                                        intent.getIntegerArrayListExtra(DATAMAP_INT_ARRAYLIST));
            sendPutDataMapRequest(putDataMapRequest);
            break;
```

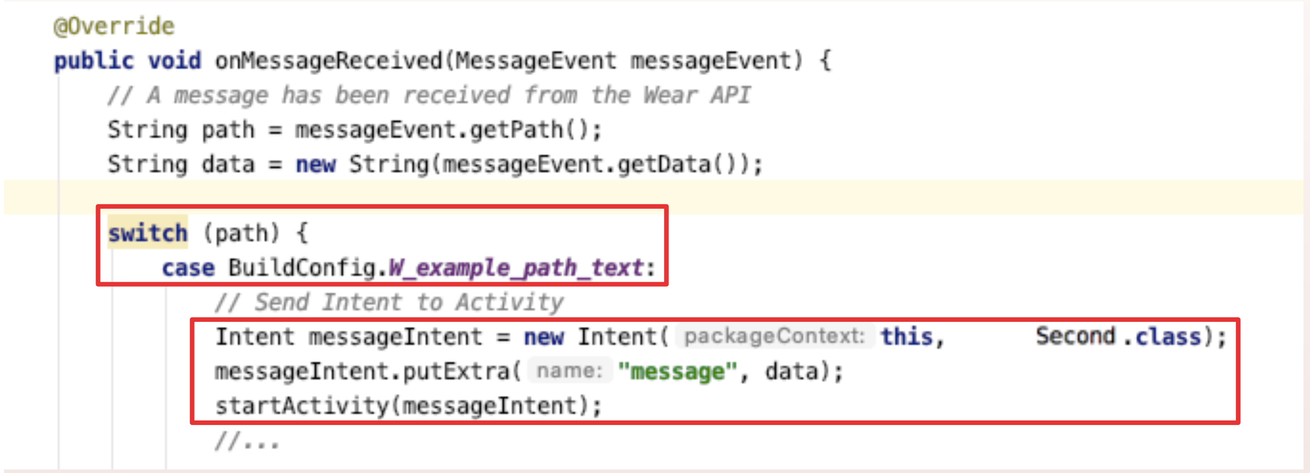
WearService

onStartCommand()

A step back: using WearService

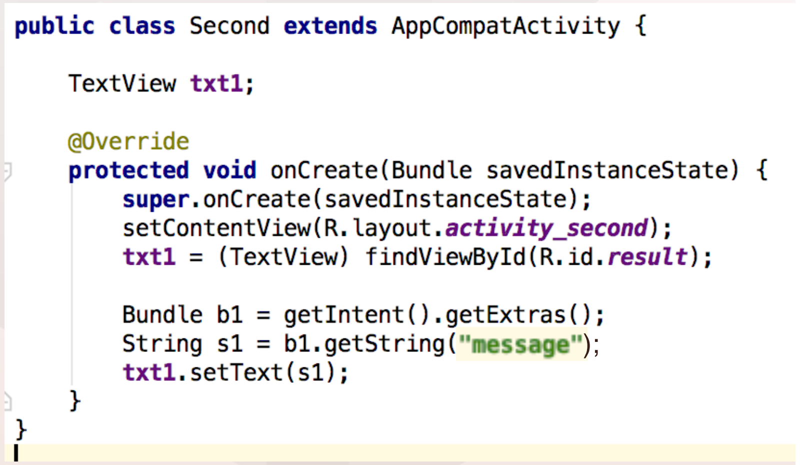
- Messages are processed by WearService in onMessageReceived()
  - Look for matching path
  - Send intent to activity → *explicit intent in this example*

WearService onMessageReceived()



```
@Override
public void onMessageReceived(MessageEvent messageEvent) {
    // A message has been received from the Wear API
    String path = messageEvent.getPath();
    String data = new String(messageEvent.getData());

    switch (path) {
        case BuildConfig.W_example_path_text:
            // Send Intent to Activity
            Intent messageIntent = new Intent( packageContext: this,        Second.class);
            messageIntent.putExtra( name: "message", data);
            startActivity(messageIntent);
            //...
```

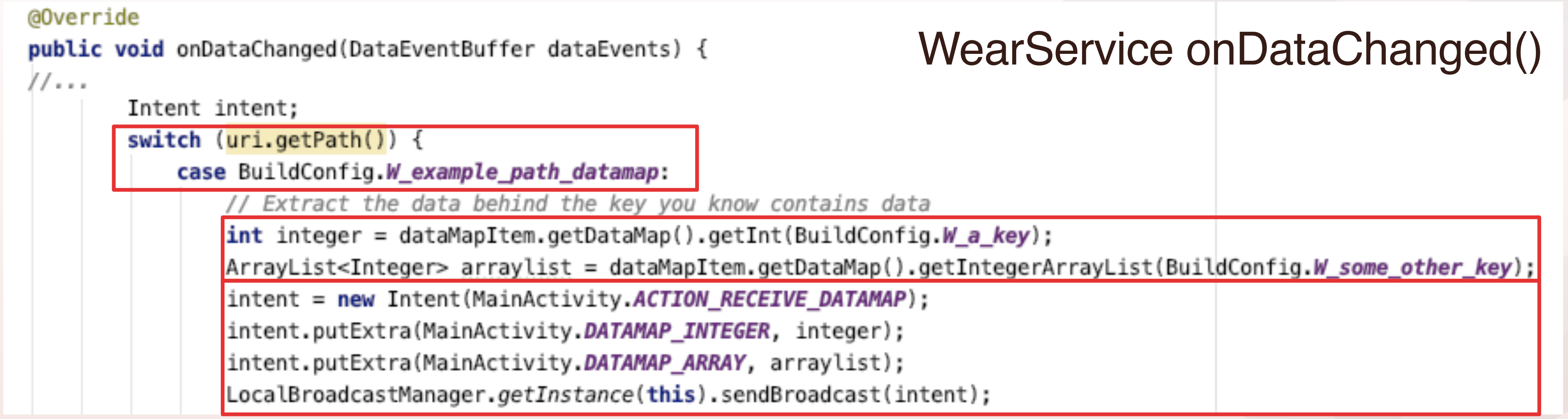
Activity onCreate()



```
public class Second extends AppCompatActivity {

    TextView txt1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        txt1 = (TextView) findViewById(R.id.result);

        Bundle b1 = getIntent().getExtras();
        String s1 = b1.getString("message");
        txt1.setText(s1);
    }
}
```

A step back: using WearService

- DataMaps are processed by WearService in onDataChanged()
  - Look for matching path
  - Extract fields from dataMap
  - Send intent to activity → *implicit intent in this example*



WearService onDataChanged()

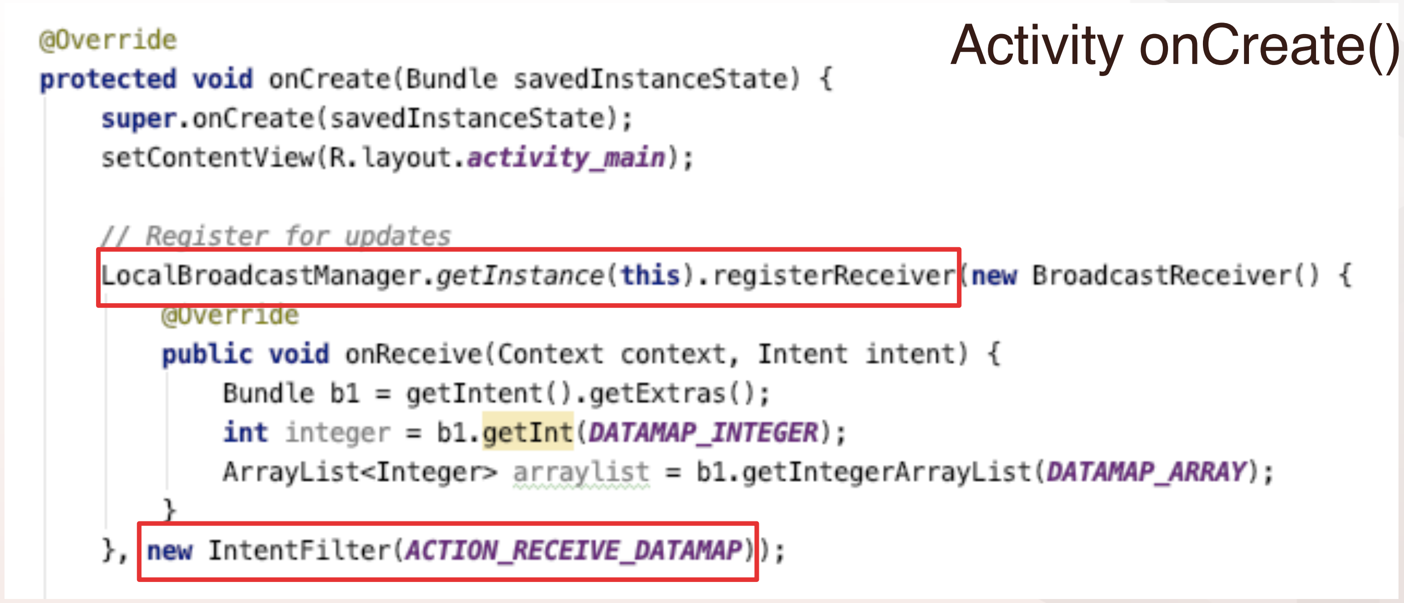
```
@Override
public void onDataChanged(DataEventBuffer dataEvents) {
//...
        Intent intent;
        switch (uri.getPath()) {
            case BuildConfig.W_example_path_datamap:
                // Extract the data behind the key you know contains data
                int integer = dataMapItem.getDataMap().getInt(BuildConfig.W_a_key);
                ArrayList<Integer> arraylist = dataMapItem.getDataMap().getIntegerArrayList(BuildConfig.W_some_other_key);
                intent = new Intent(MainActivity.ACTION_RECEIVE_DATAMAP);
                intent.putExtra(MainActivity.DATAMAP_INTEGER, integer);
                intent.putExtra(MainActivity.DATAMAP_ARRAY, arraylist);
                LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

…

A step back: using WearService

- The receiving activity should register for the broadcasted intent in onCreate()
- …and override the onReceive() method

…



Activity onCreate()

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

    // Register for updates
    LocalBroadcastManager.getInstance(this).registerReceiver(new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle b1 = getIntent().getExtras();
            int integer = b1.getInt(DATAMAP_INTEGER);
            ArrayList<Integer> arraylist = b1.getIntegerArrayList(DATAMAP_ARRAY);
        }
    }, new IntentFilter(ACTION_RECEIVE_DATAMAP));
```

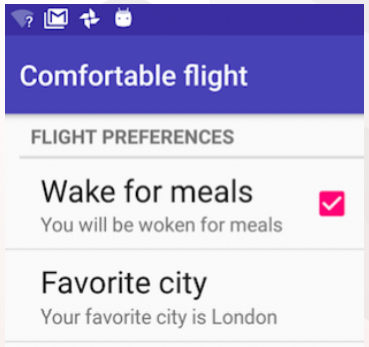
…

Outline of the class

- **Firebase** *(Today's lab 5!)*

- Shared Preferences *(Lab 7)*
  - Creating/saving/restoring prefs.
  - Setting UI

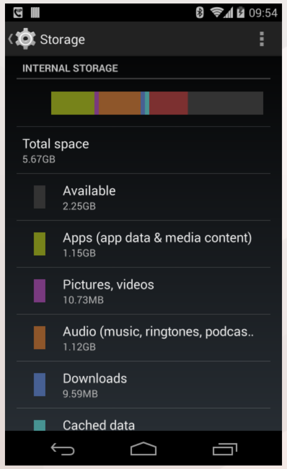
- Internal/External storage
  - Writing to files


Comfortable flight

FLIGHT PREFERENCES

Wake for meals
You will be woken for meals ✓

Favorite city
Your favorite city is London


Storage

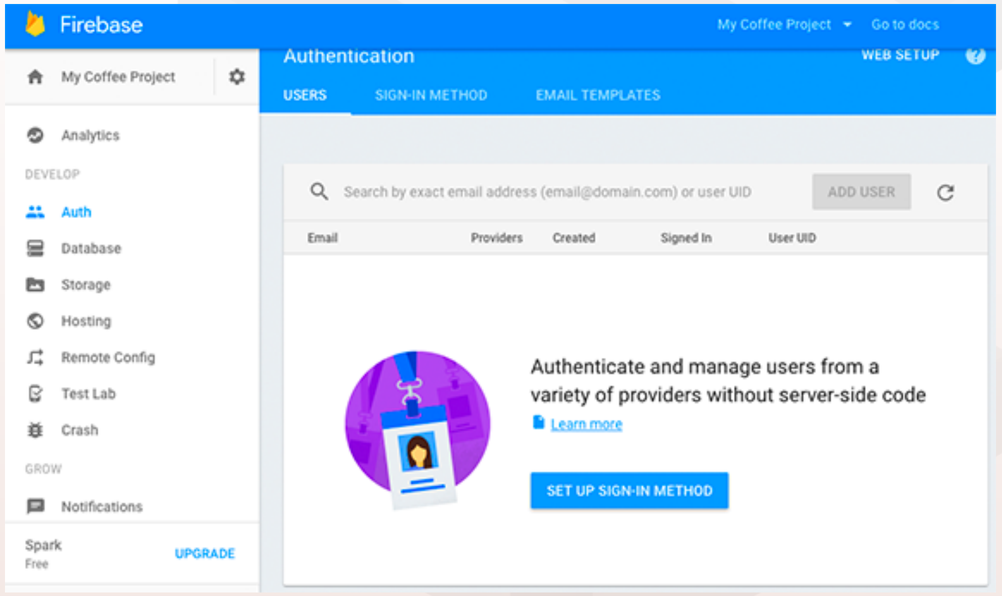
INTERNAL STORAGE

Total space	5.67GB
Available	2.25GB
Apps (app data & media content)	1.15GB
Pictures, videos	10.73MB
Audio (music, ringtones, podcas..	1.12GB
Downloads	9.59MB
Cached data	

What is Firebase?

- Firebase is a platform that provides tools to help you
  - develop your app
  - grow your user base
  - earn money from your app

- We will show you how to use it to sync data to the cloud
  1. Connect your app to your Firebase project
  2. Enable Firebase features in the console
  3. Add code to your app to interface Realtime and Storage databases

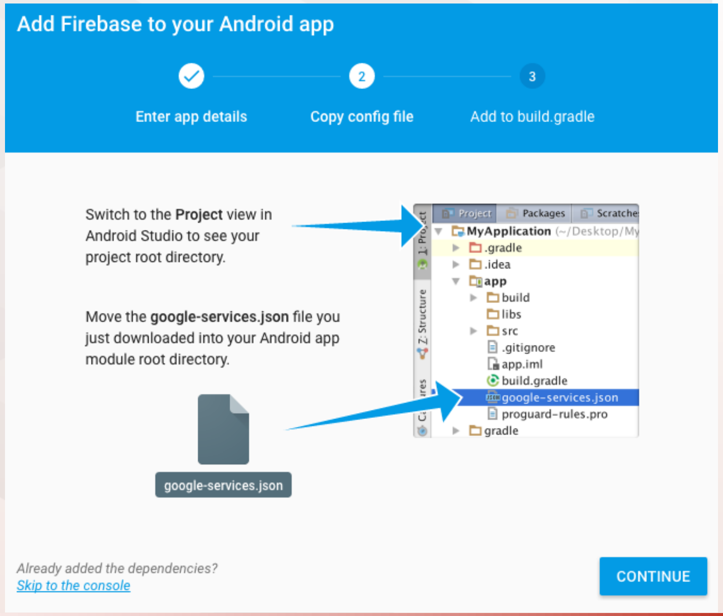


Authenticate and manage users from a variety of providers without server-side code

Firebase console

- Going to firebase.google.com
  - The console allows you to create new projects
  - Firebase creates a config file for your app
    - You can add the config file to your project

- All this happens automatically through Android Studio!
  - We will teach you how to do it, step by step, during the lab.



Add Firebase to your Android app

Switch to the **Project** view in Android Studio to see your project root directory.

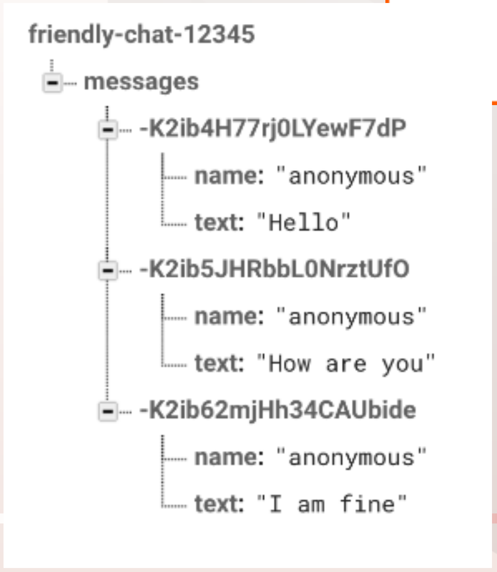
Move the **google-services.json** file you just downloaded into your Android app module root directory.

Already added the dependencies?
Skip to the console

Using Firebase Database / Storage

- We will use Firebase Realtime Database and Cloud Storage features
- Data is synced across all clients, and remains available when your app goes offline
- Realtime Database for profiles
  - data synchronization with listeners
  - key-value database
- Cloud Storage for pictures

Database
Store and sync data in realtime across all connected clients



```
friendly-chat-12345
  messages
    -K2ib4H77rj0LYewF7dP
      name: "anonymous"
      text: "Hello"
    -K2ib5JHRbbL0NrztUfO
      name: "anonymous"
      text: "How are you"
    -K2ib62mjHh34CAUbide
      name: "anonymous"
      text: "I am fine"
```

Interfacing with a Realtime database

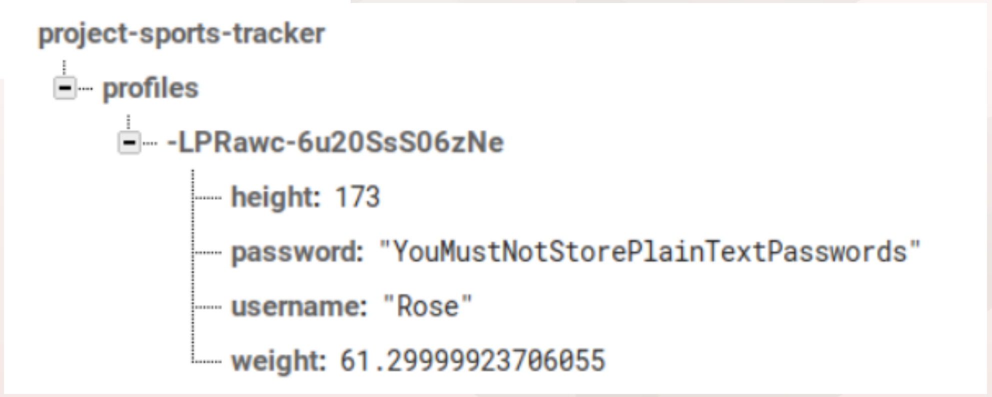
- Your app is connected with Firebase using the GUI-based Firebase assistant
  - Detail in the Lab5 handout
- In your code, you can reference the database and create a key for a new entry via push():

```java
private static final FirebaseDatabase database = FirebaseDatabase
        .getInstance();
private static final DatabaseReference profileGetRef = database
        .getReference("profiles");
private static DatabaseReference profileRef = profileGetRef.push();
```

Writing to a Realtime database

- A transaction is run using the key

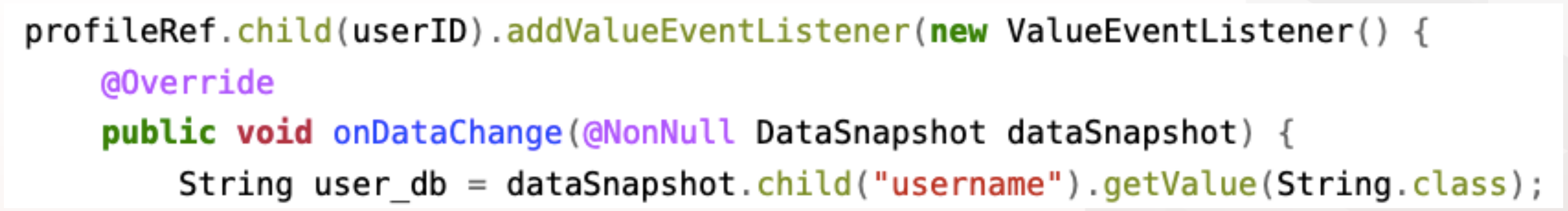
```
profileRef.runTransaction(new Transaction.Handler() {
    @NonNull
    @Override
    public Transaction.Result doTransaction(@NonNull MutableData
                                                mutableData) {
        mutableData.child("username").setValue(userProfile.username);
        mutableData.child("password").setValue(userProfile.password);
        mutableData.child("height").setValue(userProfile.height_cm);
        mutableData.child("weight").setValue(userProfile.weight_kg);
        return Transaction.success(mutableData);
    }
```



```
project-sports-tracker
  profiles
    -LPRawc-6u20SsS06zNe
      height: 173
      password: "YouMustNotStorePlainTextPasswords"
      username: "Rose"
      weight: 61.29999923706055
```

Reading from a Realtime database

- Attach addValueEventListener() to the reference
  - Example: the “username” for a give userID key



```
profileRef.child(userID).addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        String user_db = dataSnapshot.child("username").getValue(String.class);
```

Interfacing with the Cloud storage

- Similar to the Realtime Database case
  - Get the reference to the Cloud storage with

```
StorageReference storageRef = FirebaseStorage.getInstance()
        .getReference();
```

  - Get the reference of the element we want to store

```
StorageReference photoRef = storageRef.child("photos").child
        (profileRef.getKey() + ".jpg");
```

Writing and Reading the Cloud storage

- Write
  - Send the data to Cloud Storage

```
UploadTask uploadTask = photoRef.putBytes(data);
```

  - Use URI returned by the upload process to link the element in the Storage with the proper entry in the RealTime DB

```
userProfile.photoPath = uri.toString();
mutableData.child("photo").setValue(userProfile.photoPath);
```

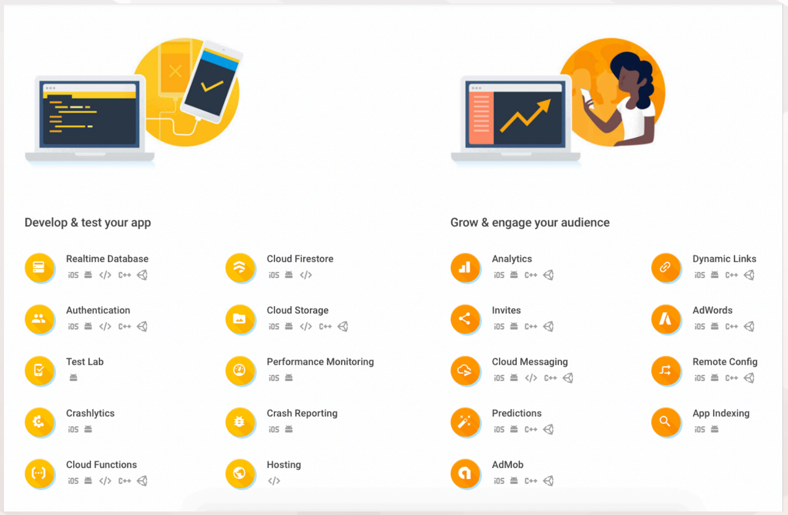
- Read

```
StorageReference storageRef = FirebaseStorage.getInstance()
        .getReferenceFromUrl(userProfile.photoPath);
storageRef.getBytes(Long.MAX_VALUE).addOnSuccessListener(
        new OnSuccessListener<byte[]>() {
```

Today's lab

- Exercising Firebase!
  - Connecting the sports-tracker app to Firebase
  - Adding real-time database to your app
  - Configure Firebase Database rules

- Writing/reading data from Firebase Realtime DBs and Cloud Storage

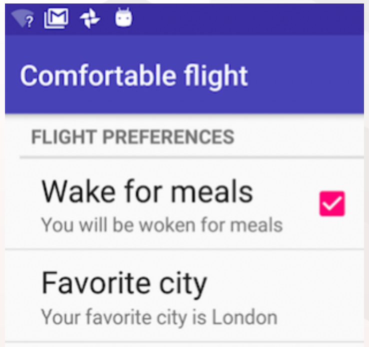


Outline of the class

- Firebase *(Today's lab 5!)*

- **Shared Preferences** *(Lab 7)*
  - Creating/saving/restoring prefs.
  - Setting UI

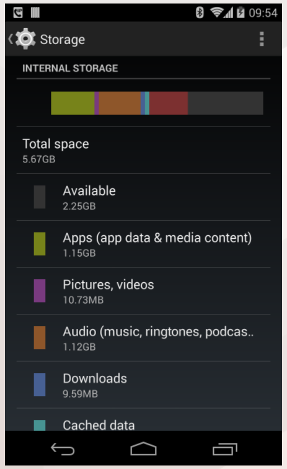
- Internal/External storage
  - Writing to files


Comfortable flight

FLIGHT PREFERENCES

Wake for meals ☑
You will be woken for meals

Favorite city
Your favorite city is London


Storage

INTERNAL STORAGE

Total space
5.67GB

Available
2.25GB

Apps (app data & media content)
1.15GB

Pictures, videos
10.73MB

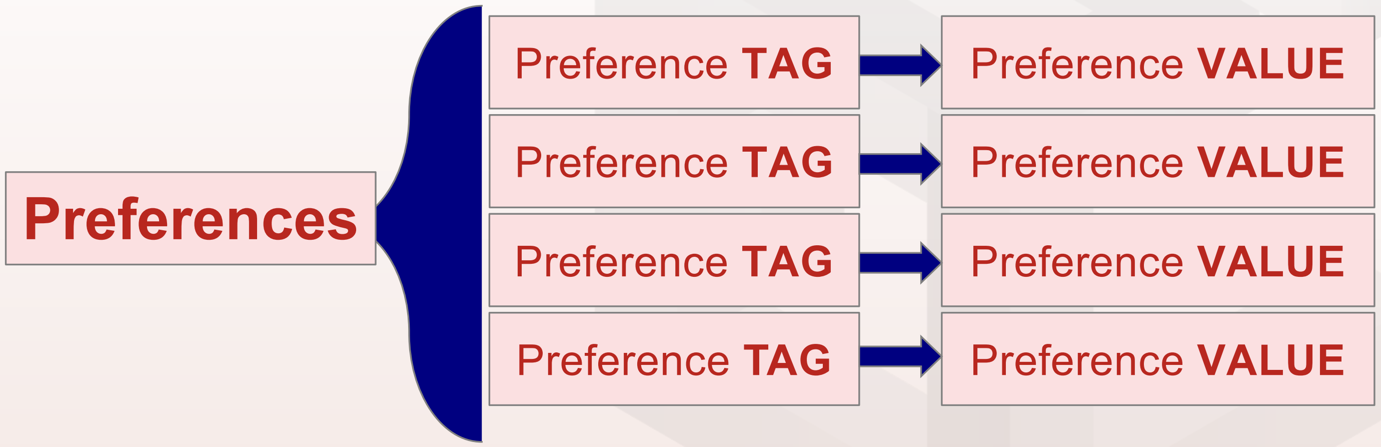
Audio (music, ringtones, podcas..
1.12GB

Downloads
9.59MB

Cached data

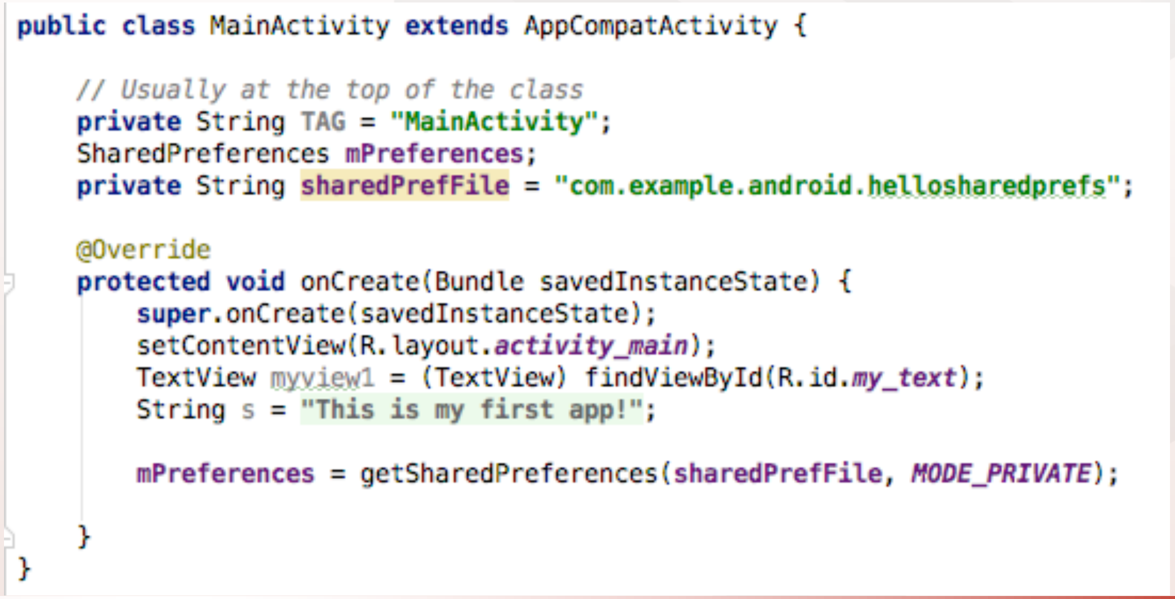
Shared Preferences

- Preferences are a convenient way to store config. parameters
- Read/write small amounts of data as key/value pairs
- Shared among Activities in an app



Creating Shared Preferences

- We need only one Share Preferences file per app.
- Name it with the package name of your app
  - Unique and easy way to associate it with an app.
  - MODE argument for getSharedPreferences() is for backward compatibility—use only MODE_PRIVATE

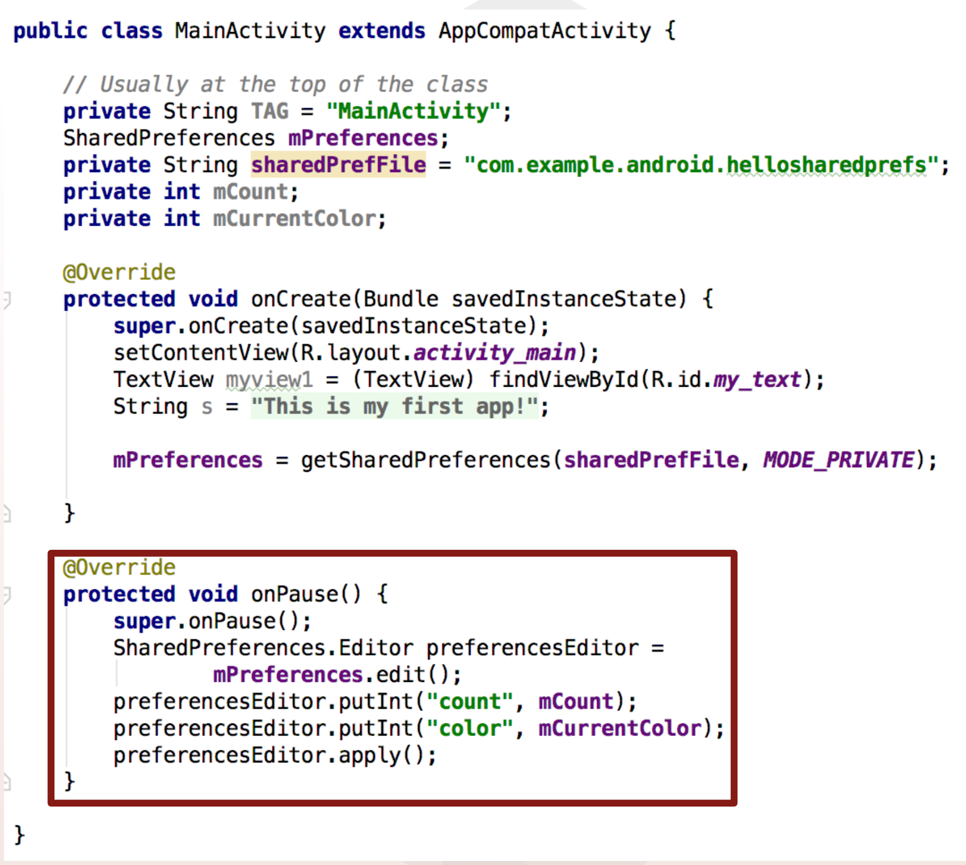


```java
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";
    SharedPreferences mPreferences;
    private String sharedPrefFile = "com.example.android.hellosharedprefs";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

    }
}
```

Saving preferences

- How to edit preferences? → using SharedPreferences.Editor
  - This takes care of all file operations
  - Careful! Overwrite in case the key already exists

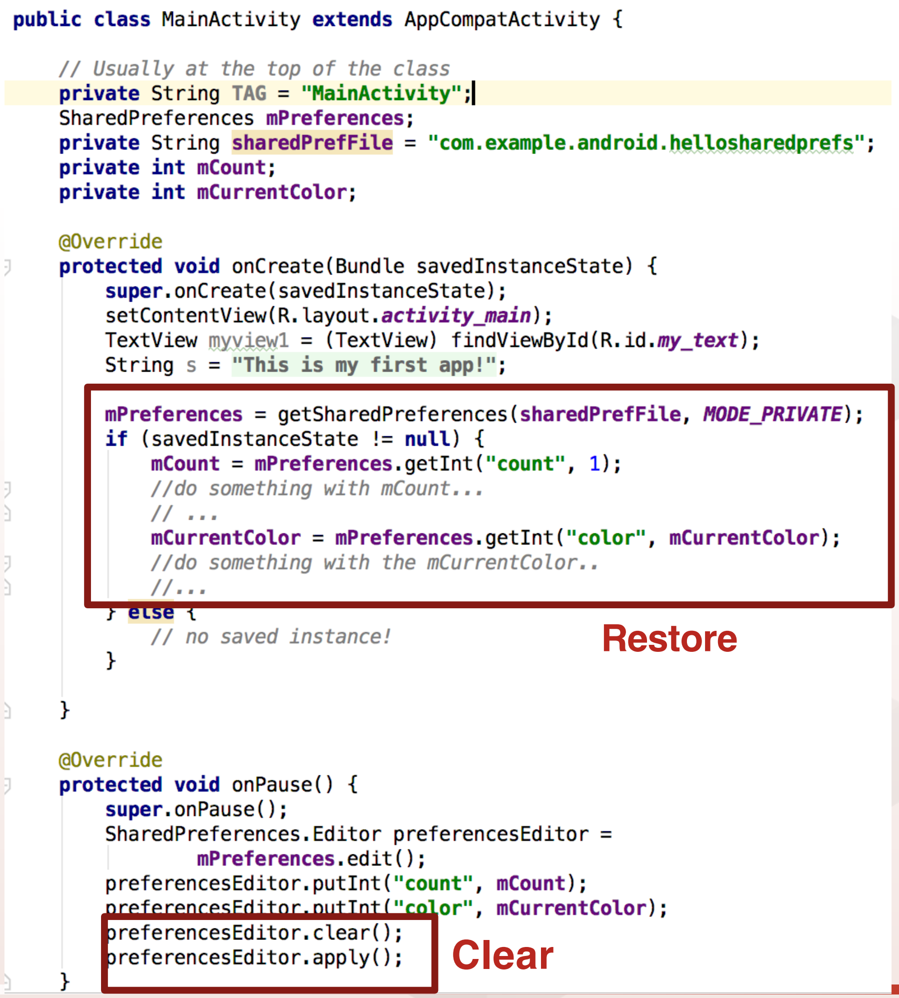
- Be sure to commit operations at the end:
  - apply() saves asynchronously and safely



```java
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";
    SharedPreferences mPreferences;
    private String sharedPrefFile = "com.example.android.hellosharedprefs";
    private int mCount;
    private int mCurrentColor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);

    }

    @Override
    protected void onPause() {
        super.onPause();
        SharedPreferences.Editor preferencesEditor =
                mPreferences.edit();
        preferencesEditor.putInt("count", mCount);
        preferencesEditor.putInt("color", mCurrentColor);
        preferencesEditor.apply();
    }

}
```

Restoring and clearing preferences

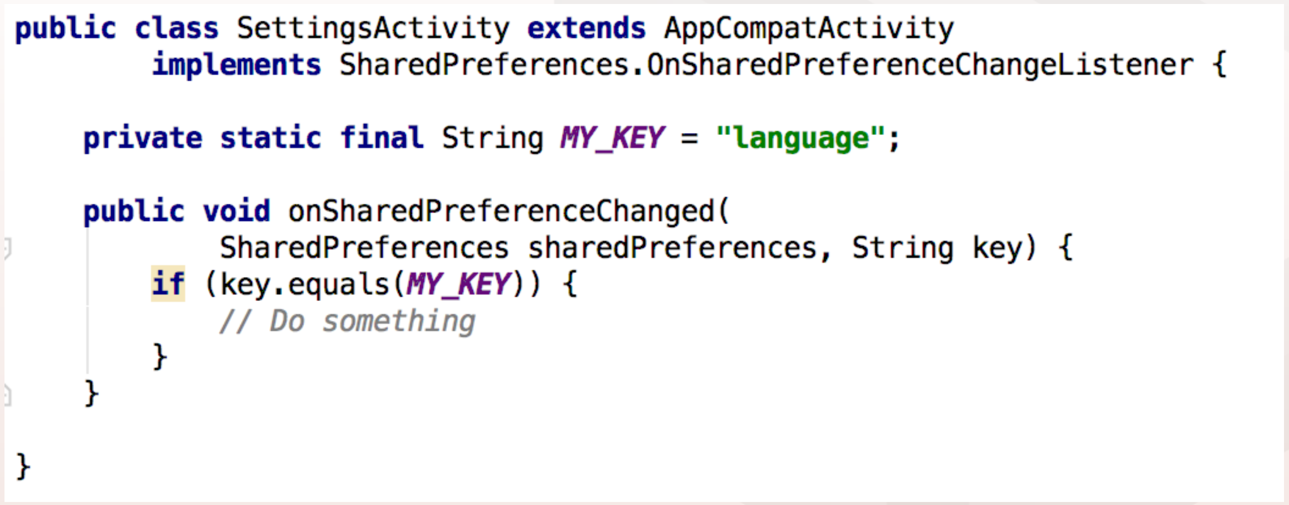
- Restore in onCreate() in Activity
  - Get methods take two arguments:
    - the key
    - the default value if the key cannot be found
  - Use default argument so you do not have to test whether the preference exists in the file
- Clearing:
  - Call clear() on the SharedPreferences.Editor and apply changes



```java
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";
    SharedPreferences mPreferences;
    private String sharedPrefFile = "com.example.android.hellosharedprefs";
    private int mCount;
    private int mCurrentColor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        mPreferences = getSharedPreferences(sharedPrefFile, MODE_PRIVATE);
        if (savedInstanceState != null) {
            mCount = mPreferences.getInt("count", 1);
            //do something with mCount...
            // ...
            mCurrentColor = mPreferences.getInt("color", mCurrentColor);
            //do something with the mCurrentColor..
            //...
        } else {
            // no saved instance!
        }

    }

    @Override
    protected void onPause() {
        super.onPause();
        SharedPreferences.Editor preferencesEditor =
                mPreferences.edit();
        preferencesEditor.putInt("count", mCount);
        preferencesEditor.putInt("color", mCurrentColor);
        preferencesEditor.clear();
        preferencesEditor.apply();
    }
```

Reading preferences

- Listening to changes on the application settings
  - Implement interface SharedPreference.OnSharedPreferenceChangeListener
  - Register listener with registerOnSharedPreferenceChangeListener()
  - Implement on onSharedPreferenceChanged() callback



```java
public class SettingsActivity extends AppCompatActivity
        implements SharedPreferences.OnSharedPreferenceChangeListener {

    private static final String MY_KEY = "language";

    public void onSharedPreferenceChanged(
            SharedPreferences sharedPreferences, String key) {
        if (key.equals(MY_KEY)) {
            // Do something
        }
    }

}
```

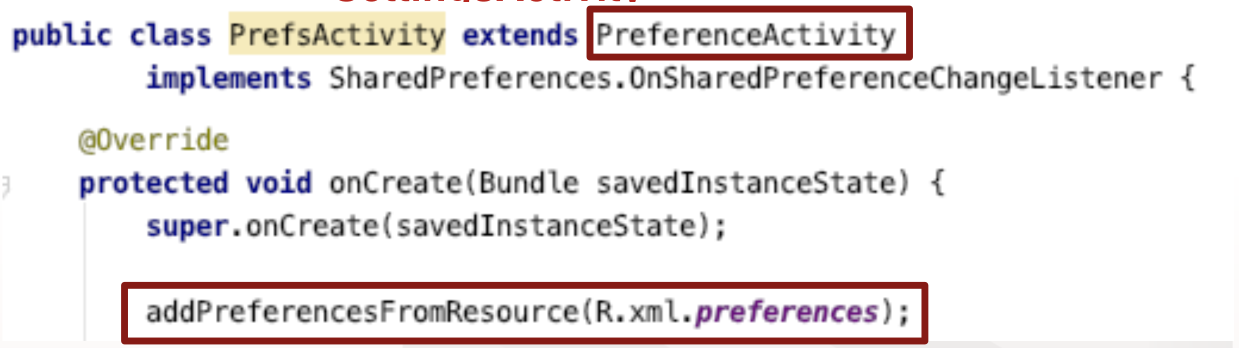
Preference Activity

- Activity extending PreferenceActivity

- Link preference activity to the XML file
  → R.xml.preferences
  - CheckBoxPreference
  - EditTextPreference
  - SwitchPreference

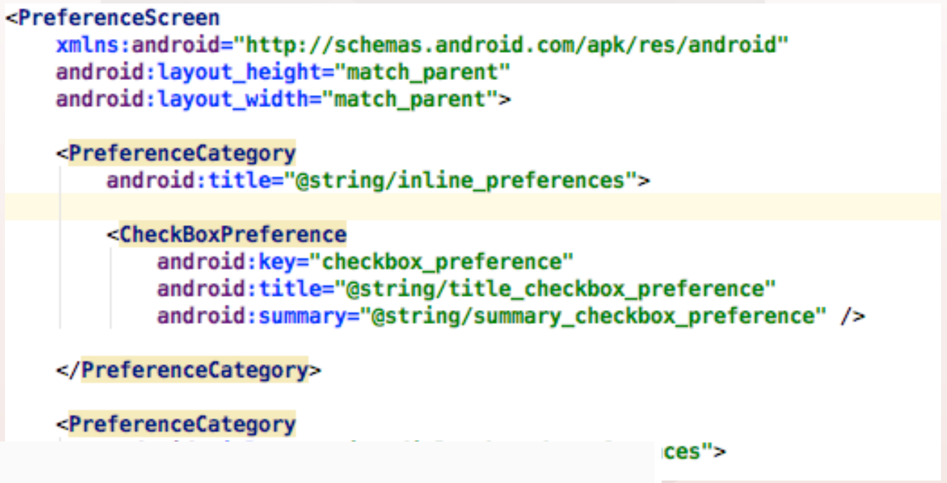
- Use onSharedPreferenceChanged() as before

SettingsActivity

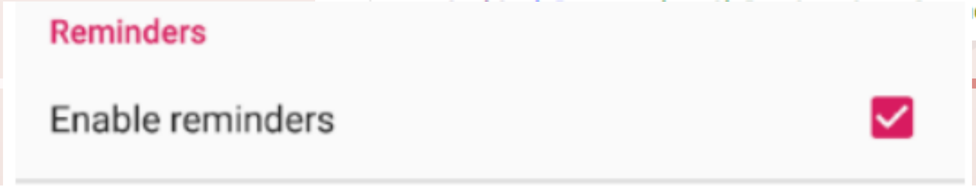


```
public class PrefsActivity extends PreferenceActivity
        implements SharedPreferences.OnSharedPreferenceChangeListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.preferences);
```

preferences.xml



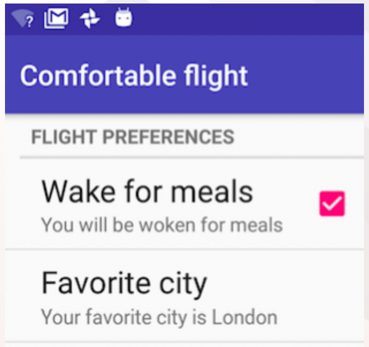
```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="match_parent"
    android:layout_width="match_parent">

    <PreferenceCategory
        android:title="@string/inline_preferences">

        <CheckBoxPreference
            android:key="checkbox_preference"
            android:title="@string/title_checkbox_preference"
            android:summary="@string/summary_checkbox_preference" />

    </PreferenceCategory>

    <PreferenceCategory
                                                ces">
```



Outline of the class

- Firebase *(Today's lab 5!)*

- Shared Preferences *(Lab 7)*
  - Creating/saving/restoring prefs.
  - Setting UI



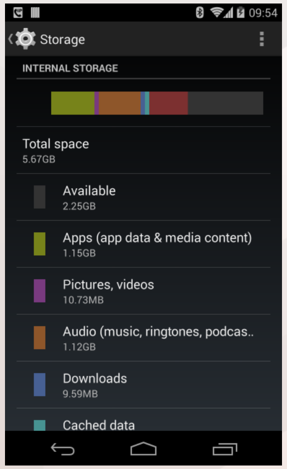
Comfortable flight

FLIGHT PREFERENCES

Wake for meals ✓
You will be woken for meals

Favorite city
Your favorite city is London

- **Internal/External storage**
  - Writing to files



Storage

INTERNAL STORAGE

Total space
5.67GB

Available
2.25GB

Apps (app data & media content)
1.15GB

Pictures, videos
10.73MB

Audio (music, ringtones, podcas..
1.12GB

Downloads
9.59MB

Cached data

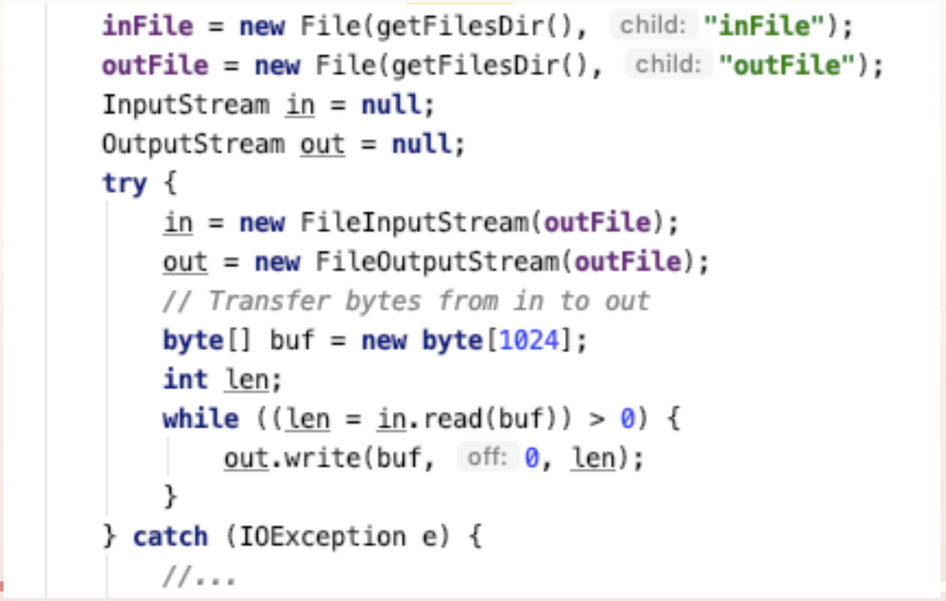
The Android FileSystem

- Internal storage -- Private directories just for your app
  - Always available
  - Uses device's filesystem
  - Only your app can access files, unless explicitly set to be readable or writable
  - On app uninstall, system removes all app's files from internal storage

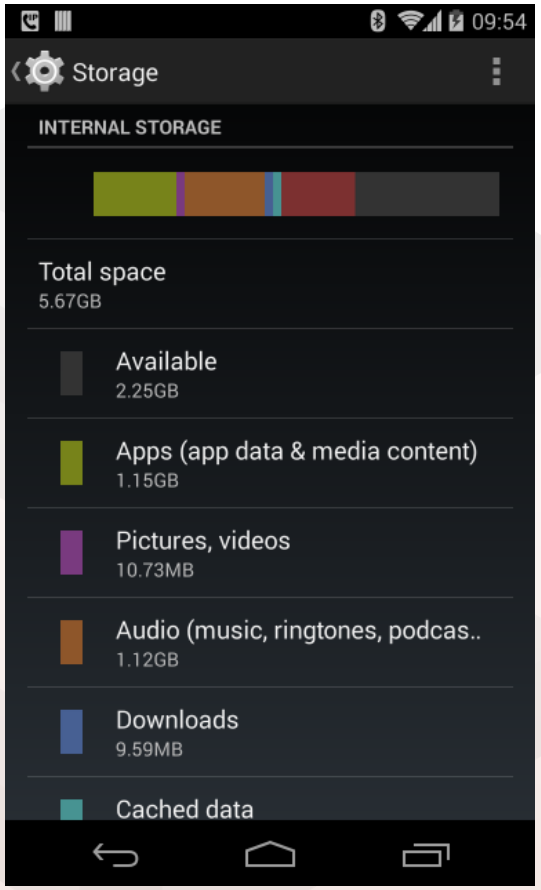
- External storage -- Public directories
  - Not always available, can be removed
  - Uses device's file system or physically external storage like SD card
  - World-readable, so any app can read
  - On uninstall, system does not remove files private to app

Internal storage

- Uses private directories just for your app
- App always has permission to read/write
  - **Where?** /data/data/<package>/files
  - **How?** Use standard java I/O classes
    - Permanent storage directory—getFilesDir()
    - Temporary storage directory—getCacheDir()



```
inFile = new File(getFilesDir(),  child: "inFile");
outFile = new File(getFilesDir(),  child: "outFile");
InputStream in = null;
OutputStream out = null;
try {
    in = new FileInputStream(outFile);
    out = new FileOutputStream(outFile);
    // Transfer bytes from in to out
    byte[] buf = new byte[1024];
    int len;
    while ((len = in.read(buf)) > 0) {
        out.write(buf,  off: 0, len);
    }
} catch (IOException e) {
    //...
```



09:54

Storage

INTERNAL STORAGE

Total space
5.67GB

Available
2.25GB

Apps (app data & media content)
1.15GB

Pictures, videos
10.73MB

Audio (music, ringtones, podcas..
1.12GB

Downloads
9.59MB

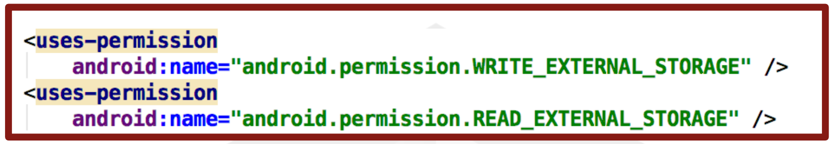
Cached data

External storage

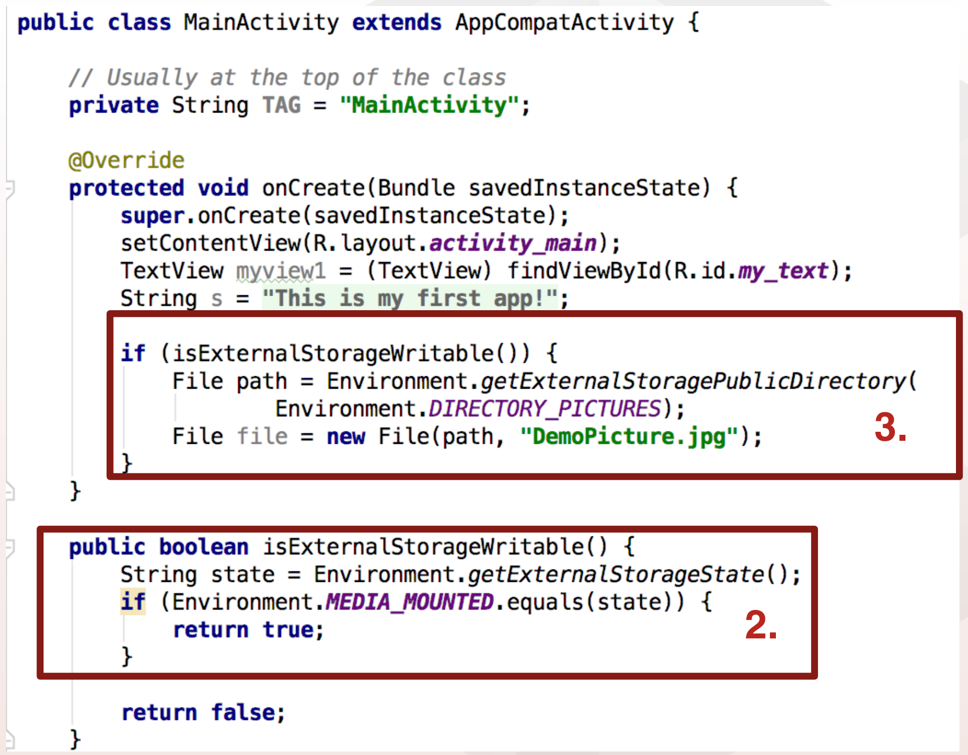
- On device or SD card
  - **Where?** Environment.getExternalStorageDirectory()
  - **How?** Use standard java I/O classes

1. Set permissions in Android Manifest
   - Write permission includes read permission
2. Check availability of storage
3. Get the path to storage folder and create a file

1.



```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE" />
```



```
public class MainActivity extends AppCompatActivity {

    // Usually at the top of the class
    private String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";

        if (isExternalStorageWritable()) {
            File path = Environment.getExternalStoragePublicDirectory(
                    Environment.DIRECTORY_PICTURES);
            File file = new File(path, "DemoPicture.jpg");
        }
    }

    public boolean isExternalStorageWritable() {
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            return true;
        }

        return false;
    }
```

3.

2.

Further notes about storage

- What happens if there is not enough space?
  - If there is not enough space, Android throws IOException
  - If you know the size of the file, check against space
    - getFreeSpace()
    - getTotalSpace().
  - If you do not know how much space is needed
    - try/catch IOException

- When the user uninstalls your app, your app's private storage directory and all its contents are deleted

- Firebase *(Today's lab 5!)*

- Shared Preferences *(Lab 7)*
  - Creating/saving/restoring prefs.
  - Setting UI

- Internal/External storage
  - Writing to files

Questions?