IAT359 Mobile Computing

Fall 2022

Instructor: Hanieh Shakeri
TA: Parnian Taghipour

lecture 5

- Review of starting an Activity for result (new way)
- Data storage options in Android
- Shared preferences
- SQLite Databases for Android

week 5 check-in

- Quiz 2 today
  - Will be released on Canvas after lecture (2:20PM) and will be open for 24 hours (due Wednesday October 12th at 2:20PM)
  - **Must be completed individually**
  - Will cover content from weeks 3 and 4
- Project teams should be formed by TODAY, milestone 1 due on October 18
- Meeting with teams on October 18 during labs
- Assignment 2 due on October 25

startActivityForResult (deprecated)

Old way:

```
@Override
public void onClick(View v) {
    Intent i = new Intent( packageContext: this, ActivityTwo.class);
    startActivityForResult(i, REQUEST_CODE);
}
```

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode==REQUEST_CODE) //check that we're processing the response from WordEntry
    {
        if(resultCode==RESULT_OK) //make sure the request was successful
        {
            if ((data.hasExtra( name: "First Name"))&&(data.hasExtra( name: "Last Name"))&&(data.hasExtra( name: "Selected Color")));
            {
                String firstname_entered = data.getExtras().getString( key: "First Name");
                String lastname_entered = data.getExtras().getString( key: "Last Name");
                String color_selected = data.getExtras().getString( key: "Selected Color");
                displayInfo.setText("Welcome " + firstname_entered + " " +lastname_entered);
                displayInfo.setBackgroundColor((Color.parseColor(color_selected)));
            }
        }
    }

    super.onActivityResult(requestCode, resultCode, data);
}
```

ActivityResultLauncher (new way)

```java
@Override
public void onClick(View v) {
    Intent i = new Intent( packageContext: this, ActivityTwo.class);
    getResult.launch(i);
}
```

```java
ActivityResultLauncher<Intent> getResult = registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        new ActivityResultCallback<ActivityResult>() {
            @Override
            public void onActivityResult(ActivityResult result) {
                if (result.getResultCode() == Activity.RESULT_OK) {
                    // There are no request codes
                    Intent data = result.getData();
                    if ((data.hasExtra( name: "First Name"))&&(data.hasExtra( name: "Last Name"))&&(data.hasExtra( name: "Selected Color")));
                    {
                        String firstname_entered = data.getExtras().getString( key: "First Name");
                        String lastname_entered = data.getExtras().getString( key: "Last Name");
                        String color_selected = data.getExtras().getString( key: "Selected Color");
                        displayInfo.setText("Welcome " + firstname_entered + " " +lastname_entered);
                        displayInfo.setBackgroundColor((Color.parseColor(color_selected)));
                    }
                }
            }
        });
```

Data Storage Options – Persistent Storage

SharedPreferences

Key-value pairs, private date

Internal Storage

Device memory, private data

External Storage

Shared external storage for public data

SQLite Database

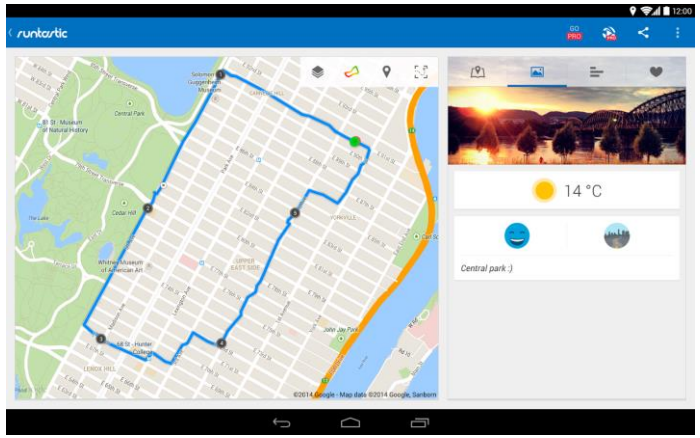
Structured, private data

Network Connection

On the web, with network server

SharedPreferences

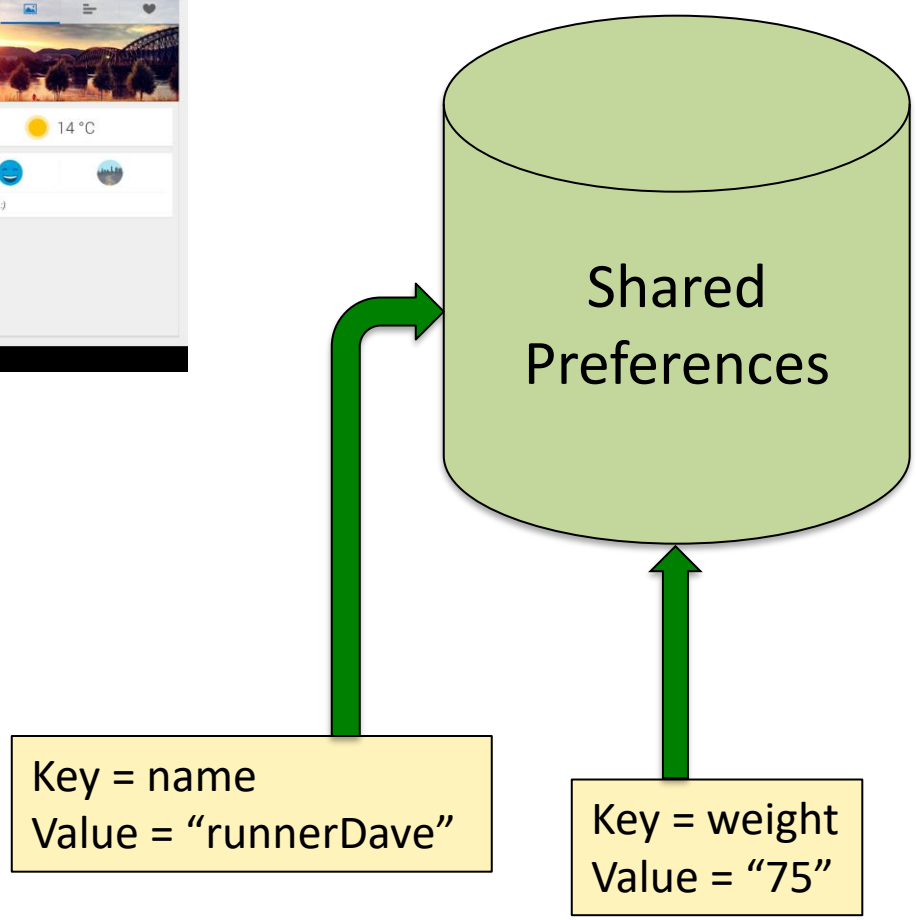
Primitive data stored as key/ value pairs
Private to the app



(Runtastic App)

name: runnerDave
weight: 75 kg

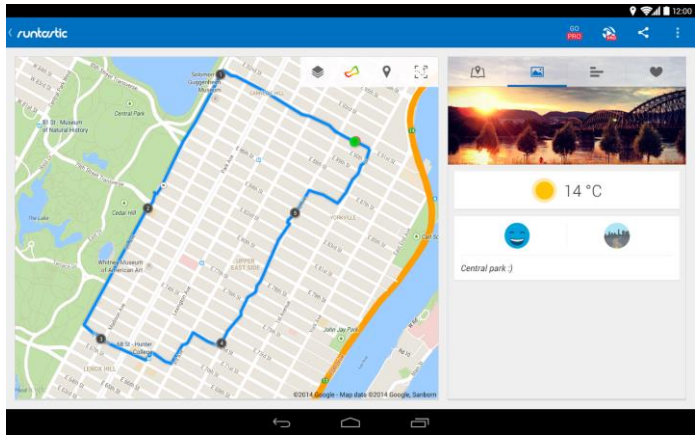
Key / Value pairs



Android internal and external storage

- **Internal Storage:**
  - Save files directly on the device’s internal storage
  - Private files to the app
  - If app is uninstalled, files are removed
- **External storage**
  - Removable storage media (SD card)
  - Non-removable storage
  - Readable by anyone

SQLite Database for Android



(Runtastic App)

Structured data
Private to the app

name: runnerDave
weight: 75 kg

Shared Preferences

SQLite Database

Activity ID	Type	Distance	Time	Calories
27	Running	5.2km	00:38:32	348

Android SharedPreferences

- Simplest way of storing data in Android – XML file
- Key / value pairs

Key	Value
Username	MrSmith
LastName	Smith
FirstName	John
School	SFU

Store the value and use the corresponding key to retrieve the value later

Data stored in XML file in *data/ data/ <package-name>/shared-prefs*

SharedPreferences – what kind of data

- Username, password, app settings, theme settings, etc
- Only allow primitive data types**: boolean, long, int, float, string**
- Not meant for complex data

accessing preferences

- An app can have one or more preference files

getPreferences()

getSharedPreferences()

```
public SharedPreferences getPreferences (int mode)
```

```
public abstract SharedPreferences getSharedPreferences (String name, int mode)
```

mode

MODE_PRIVATE	• Only your app can access the file
MODE_WORLD_READABLE (deprecated as of KitKat)	• All apps can read the file X
MODE_WORLD_WRITEABLE (deprecated as of KitKat)	• All apps can write to the file X
MODE_MULTI_PROCESS	• Multiple processes can modify the same shared preferences file

why use preferences

User info	• Store user details
Location	• Remember last user location
Updated	• Check when app was last updated
Settings	• Store user settings
First time use	• Check if the user is using the app for the first time

use SharedPrefs to store data

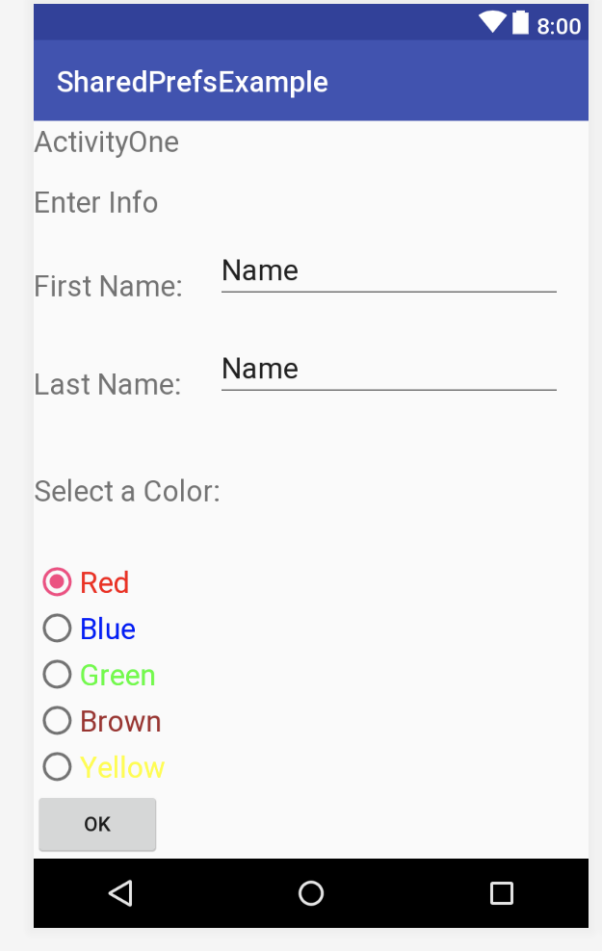
1. Get a reference to the SharedPreferences object
getPreferences() or getSharedPreferences()

2. Call the Editor edit()

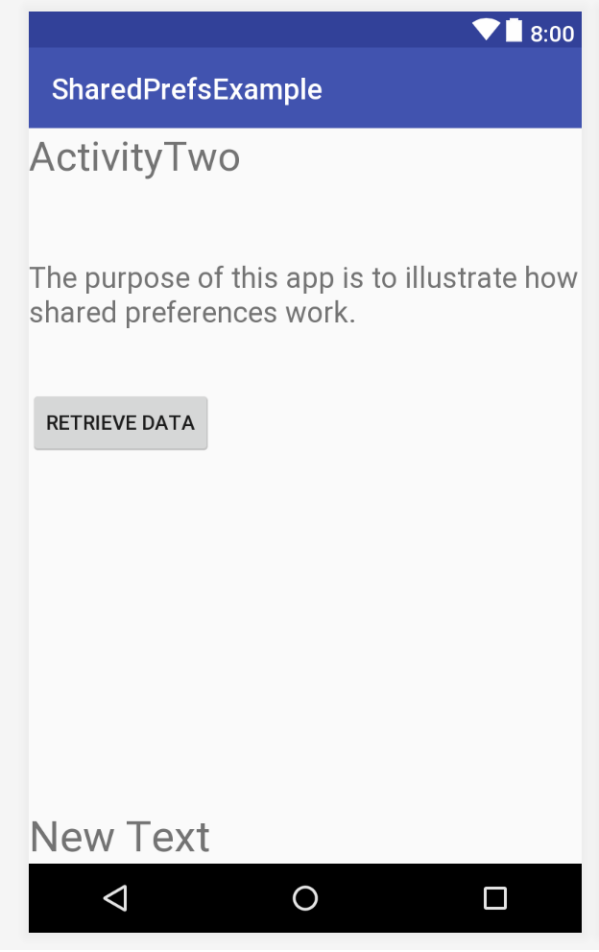
3. Use the editor to add the data with a key
putBoolean(), putString, etc

4. Commit editor changes commit()

example using SharedPrefs

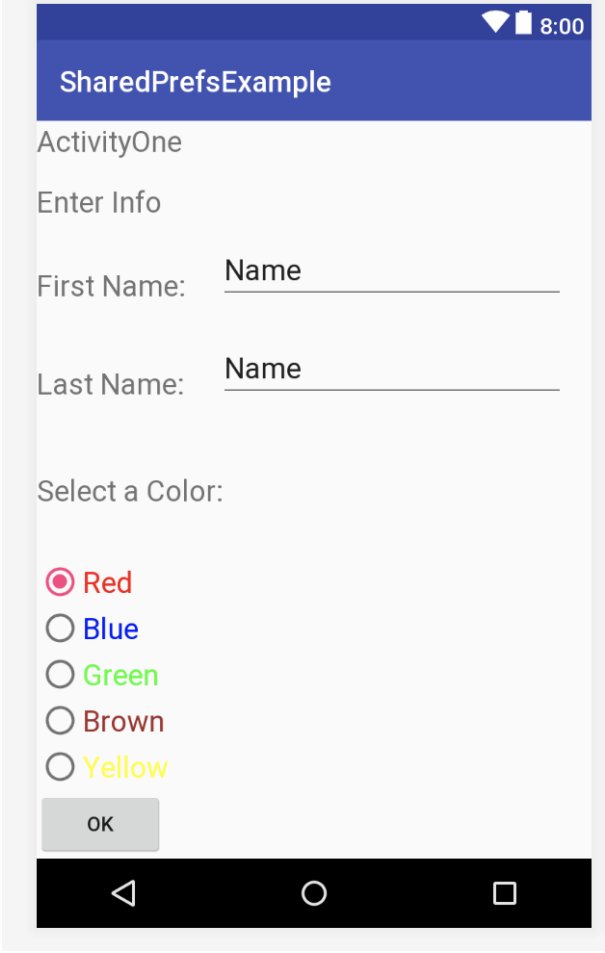


ActivityOne

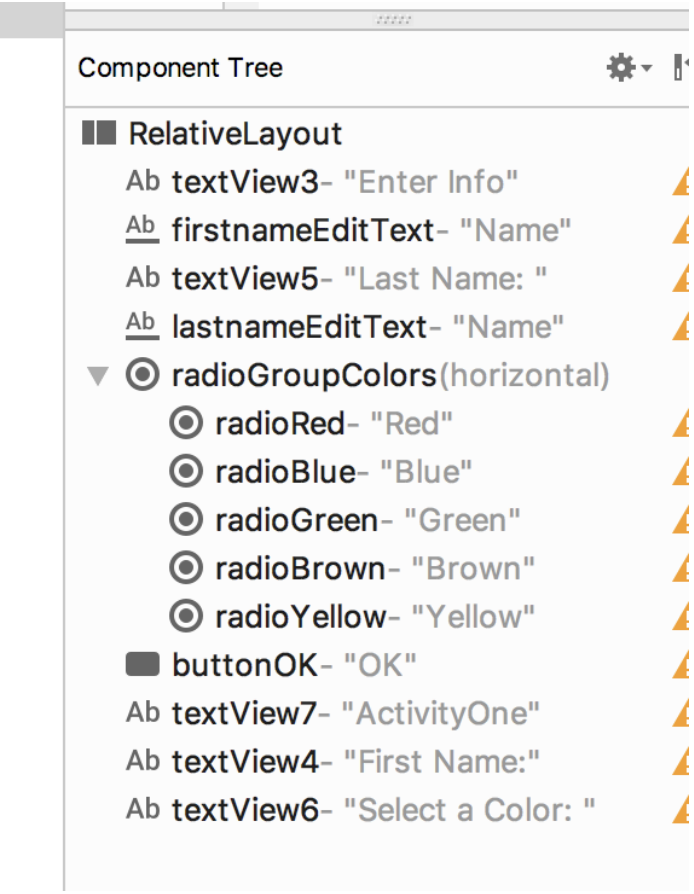


ActivityTwo

ActivityOne



ActivityOne



ActivityOne – get references to UI elements

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

    okButton = (Button)findViewById(R.id.buttonOK);
    okButton.setOnClickListener(this);
    firstNameEditText = (EditText)findViewById(R.id.firstnameEditText);
    lastNameEditText = (EditText)findViewById(R.id.lastnameEditText);
    colorButton = (RadioGroup) findViewById(R.id.radioGroupColors);
    colorButton.setOnCheckedChangeListener(this);
}
```

saving the data to SharedPrefs

- We have to save the names and color
- Notice the 4 steps in the code below:

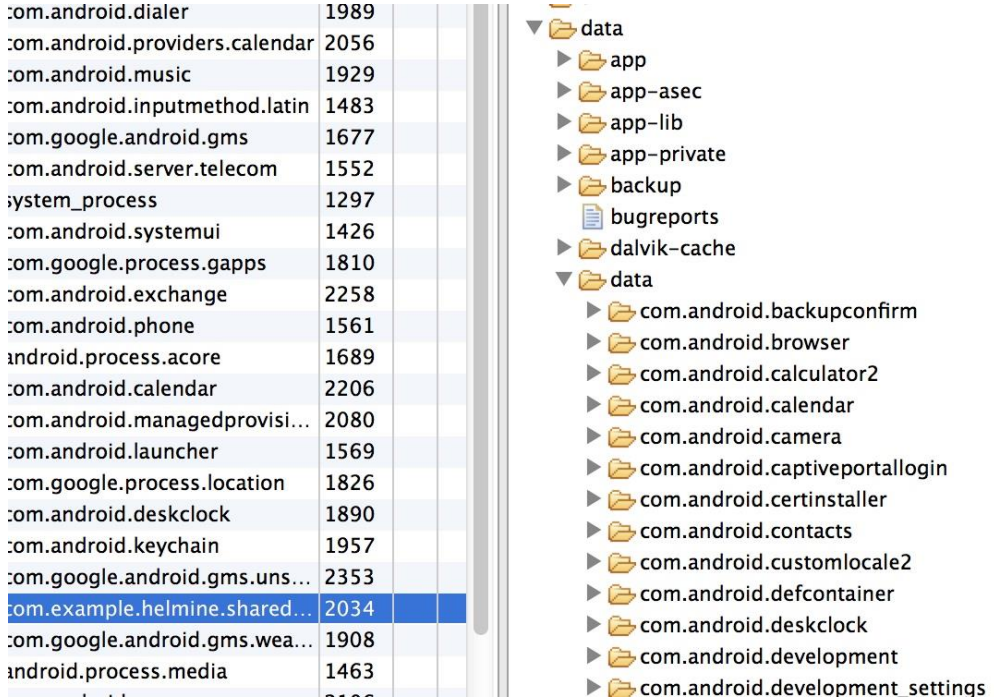
```
@Override
public void onClick(View v) {
    firstName =  firstNameEditText.getText().toString();
    lastName = lastNameEditText.getText().toString();
    Toast.makeText(this, firstName + lastName + color, Toast.LENGTH_SHORT).show();

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

    SharedPreferences sharedPrefs = getSharedPreferences("MyData", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPrefs.edit();
    editor.putString("firstName", firstName);
    editor.putString("lastName", lastName);
    editor.putString("selectedColor", color);
    Toast.makeText(this, "First, last names and color saved to Preferences", Toast.LENGTH_LONG).show();
    editor.commit();

    startActivity(i);
}
```

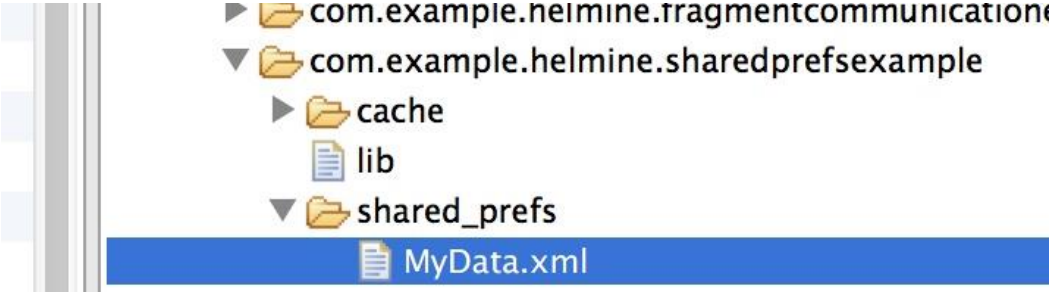
ActivityTwo: retrieve the data from SharedPrefs

```
@Override
public void onClick(View v) {

    SharedPreferences sharedPrefs = getSharedPreferences("MyData", Context.MODE_PRIVATE);
    String firstName = sharedPrefs.getString("firstName", DEFAULT);
    String lastName = sharedPrefs.getString("lastName", DEFAULT);
    String colorSelected = sharedPrefs.getString("selectedColor", DEFAULT);

    displayInfo.setText("Welcome " + firstName + " " + lastName);
    displayInfo.setBackgroundColor((Color.parseColor(colorSelected)));
}
```

the SharedPrefs file

- Device File Explorer
- Find the proper package name
- SharedPrefs file is in *data/ data/ <package-name>/shared-prefs*





the SharedPrefs file

- Pull the file from the device

- Save it in a location of your choice, then open it

AndroidManifest.xml × MyData.xml ×

```xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="firstName">maria</string>
    <string name="lastName">lastname</string>
    <string name="selectedColor">#00ff00</string>
</map>
```

question

What is the main difference between Shared Preferences and onSaveInstanceState?

question

Why is Shared Preferences better suited to storing simple data (rather than complex data)?

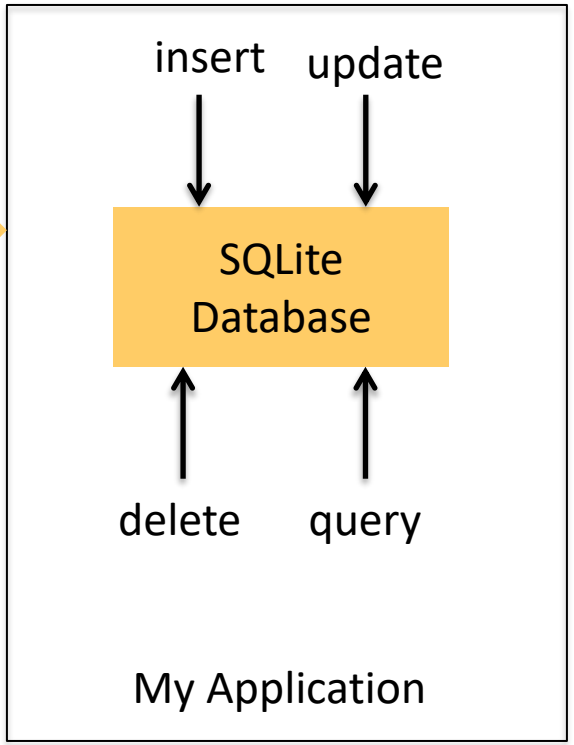
SQLite Databases for Android

- **SQLite**: language for managing data in relational databases
- Apps have their private SQLite databases
  - The database of an app is accessible by any class in the app, but not by other apps
- */data/data/<package-name>/database folder*
- App database
  - Self-contained
  - Transaction-based
  - No server

process

- Programmatically create database

Class: **SQLiteOpenHelper**
Methods: **onCreate** (create the database)
onUpgrade (perform changes to database)



getWritableDatabase(): when this method
gets called on our implementation of SQLiteOpenHelper,
an **SQLite database object** will be returned and **this is the database of our app**

creating an SQLite database schema

- Programmatically create database

1. Define schema – database name, version, table names, column names

2. Create the database – queries to create the database

3. Execute queries – insert, update, delete operations

step 1: define schema

Plant Database

PLANTSTABLE

_id	Name
1	Lavender
2	Rosemary
3	Hibiscus

Notes:

-**_id** is the primary key

- It identifies a row uniquely

- underscore: Android convention for primary keys

```
private static final String DATABASE_NAME = “plantdatabase”;

private static final String TABLE_NAME = “PLANTSTABLE”;

private static final String UID = “_id”;

private static final String NAME = “Name”;

private static final int DATABASE_VERSION = 1;
```

Constant fields, do not change values within our app

Step 2: create the database – SQLiteOpenHelper

- Subclass of SQLiteOpenHelper
  - Implement onCreate() and onUpgrade()

- SQLiteOpenHelper class:
  - Opens the database if it exists
  - Creates the database if it does not exist
  - Updates the database as necessary

SQLiteOpenHelper - methods

- **onCreate()** – this method is called when the database is first created
  - Creation of tables
  - Initial data inside of tables

- **onUpgrade()** – this method is called whenever the database is updated
  - Drop, add tables
  - Anything that updates the database structure

SQLiteDatabase class

- An object representing the database that we have just created

- Method: **public void execSQL(String sql)**
  - Executes a single SQL statement
  - Notice that the method returns void – this is an indication to what type of SQL statements this method can execute

example application

- 1. Create the SQLite Database schema

```
public class HelperClass extends SQLiteOpenHelper{
    private static final String DATABASE_NAME = "plantdatabase";
    private static final String TABLE_NAME = "PLANTSTABLE";
    private static final String UID = "_id";
    private static final String NAME = "Name";
    private static final int DATABASE_VERSION = 1;

    public HelperClass (Context context){
        super (context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

onCreate() method

PLANTSTABLE

_id	Name
1	Lavender
2	Rosemary
3	Hibiscus

- Statement to create database:

CREATE TABLE PLANTSTABLE(_id INTEGER PRIMARY KEY AUTOINCREMENT, Name VARCHAR(255));

This can also be written as:

```
private static final String CREATE_TABLE =
" CREATE TABLE " +TABLE_NAME+ " (" +UID+ " INTEGER PRIMARY KEY AUTOINCREMENT, " +NAME+ " VARCHAR(255));";
```

onCreate() method

```
public class HelperClass extends SQLiteOpenHelper{
    private Context context;
    private static final String DATABASE_NAME = "plantdatabase";
    private static final String TABLE_NAME = "PLANTSTABLE";
    private static final String UID = "_id";
    private static final String NAME = "Name";
    private static final int DATABASE_VERSION = 1;
    private static final String CREATE_TABLE =
    " CREATE TABLE " +TABLE_NAME+ " (" +UID+ " INTEGER PRIMARY KEY AUTOINCREMENT, " +NAME+ " VARCHAR(255));";
```

```
    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(CREATE_TABLE);
        } catch (SQLException e) {
            Toast.makeText(context, "exception onCreate() db", Toast.LENGTH_LONG).show();
        }
    }
```

onUpgrade() method

- In this method we can do anything that will change the database schema
  - Modify the table
  - Delete the table
  - Add, delete columns
  - Drop table

onUpgrade() method

- Example: drop table query

```
private static final String DROP_TABLE = "DROP TABLE IF EXISTS " +TABLE_NAME;
```

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    try {
        db.execSQL(DROP_TABLE);
        onCreate(db);
    } catch (SQLException e) {
        Toast.makeText(context, "exception onUpgrade() db", Toast.LENGTH_LONG).show();
    }
}
```

sample code: main activity

- Create an object of the HelperClass
- Then, run the app to see what happens

```
public class MainActivity extends Activity {

    HelperClass helper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        helper = new HelperClass(this);
```

nothing happens!!!

- Why??
- IMPORTANT: <u>the database gets created only when there is an attempt to access it for the first time</u>
  - Since there was no access to our database, the database was not created
  - This means that onCreate() and onUpgrade() methods have not been called
  - We have to call the method **getWritableDatabase() – returns a database object**

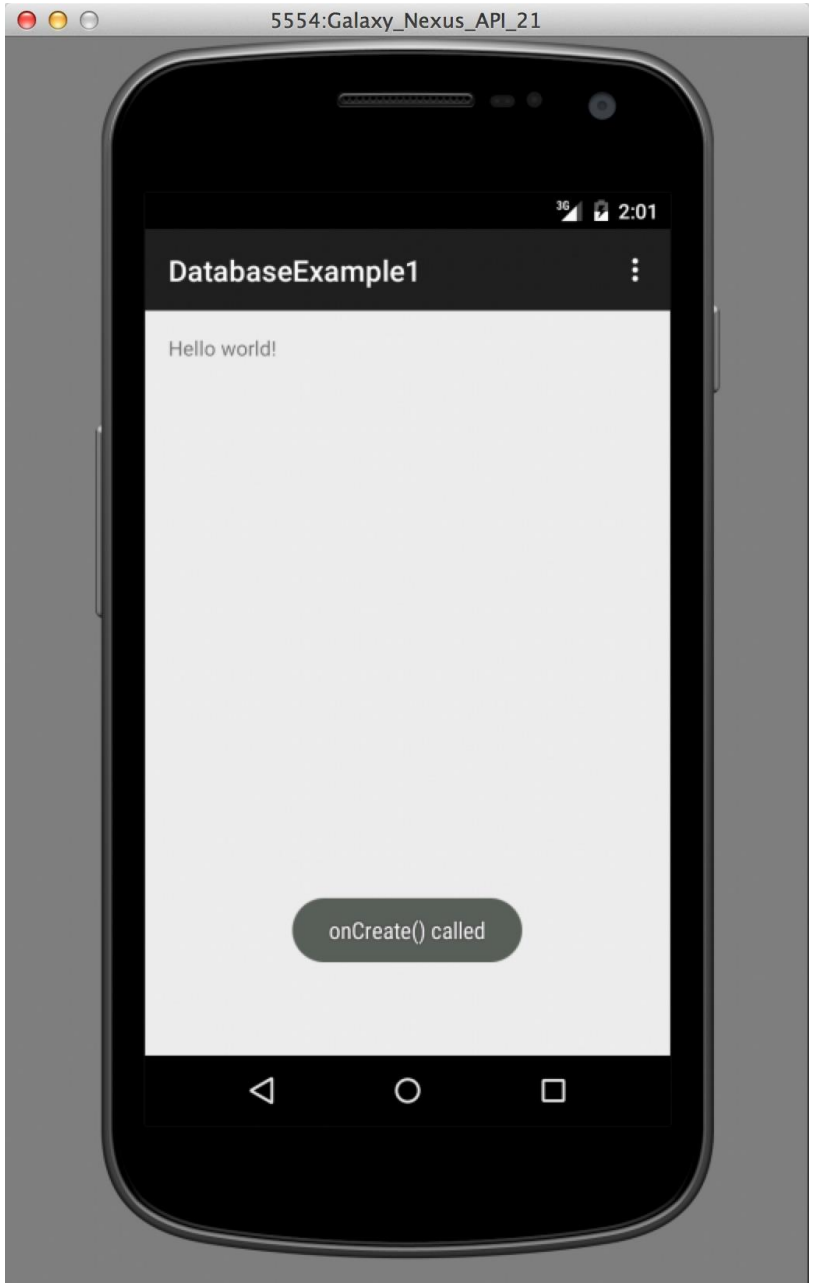
Accessing the database

```
SQLiteDatabase myDatabase = helper.getWritableDatabase();
```

getWritableDatabase() will return an SQLite database object.

The returned object is a reference to the database that we just created.

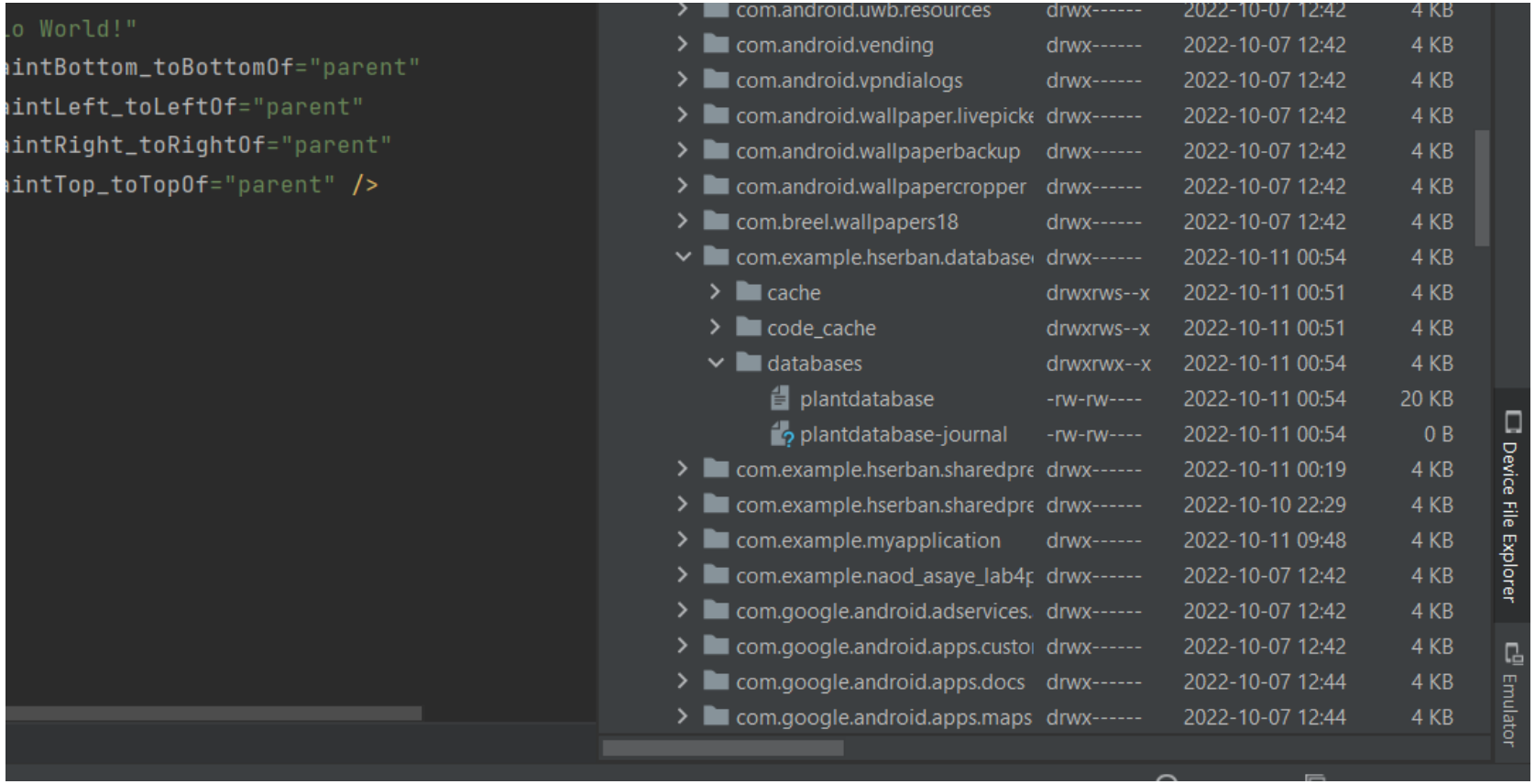
Next: run the code again



onCreate() called because the database was accessed for the first time and this triggered the database creation

viewing the database file

- Device File Explorer



what about onUpgrade?

- When is onUpgrade() called? It is called when a change in the database schema takes place.
- The database exists already, let's modify it by adding one new column: the type of the plant

```
private static final String TYPE = "Type";
```

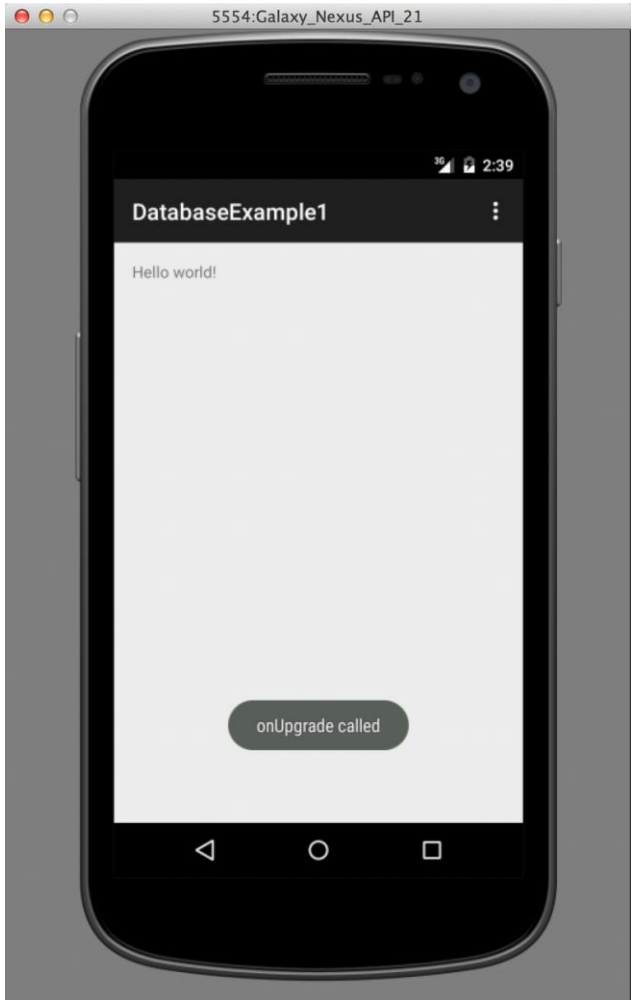
- We also have to change the CREATE TABLE: (only the change is shown below):

```
AUTOINCREMENT, " +NAME+ " VARCHAR(255), " +TYPE+ " VARCHAR(255));";
```

- Database version: this also needs to be changed, because we have changed the structure of the database

```
private static final int DATABASE_VERSION = 2;
```

outcome


5554:Galaxy_Nexus_API_21
2:39
DatabaseExample1
Hello world!
onUpgrade called

question

- Why do we need to call getWriteableDatabase?

question

- What does the Helper class do?

summary of today's class

- Data storage options in Android
  - Shared Preferences (simple, primitive data)
  - Internal and External Storage (not discussed)
  - SQLite Databases (complex, structured data)

- Shared preferences
  - when to use this storage method
  - How to store and retrieve data from SharedPrefs

- SQLite Databases for Android
  - Defining the schema, creating the database
  - We will continue this topic next week

resources

- Saving Data - http://developer.android.com/training/basics/data-storage/index.html
- Using Databases - https://developer.android.com/training/data-storage/sqlite
- SQLite official site - https://www.sqlite.org/index.html
- SQLite Query Language - https://www.sqlite.org/lang.html