Lecture 10: Permissions and Data Management

Jianjun Chen (jianjun.chen@xjtlu.edu.cn)

*slides based on the version written by Prof. Dawei Liu

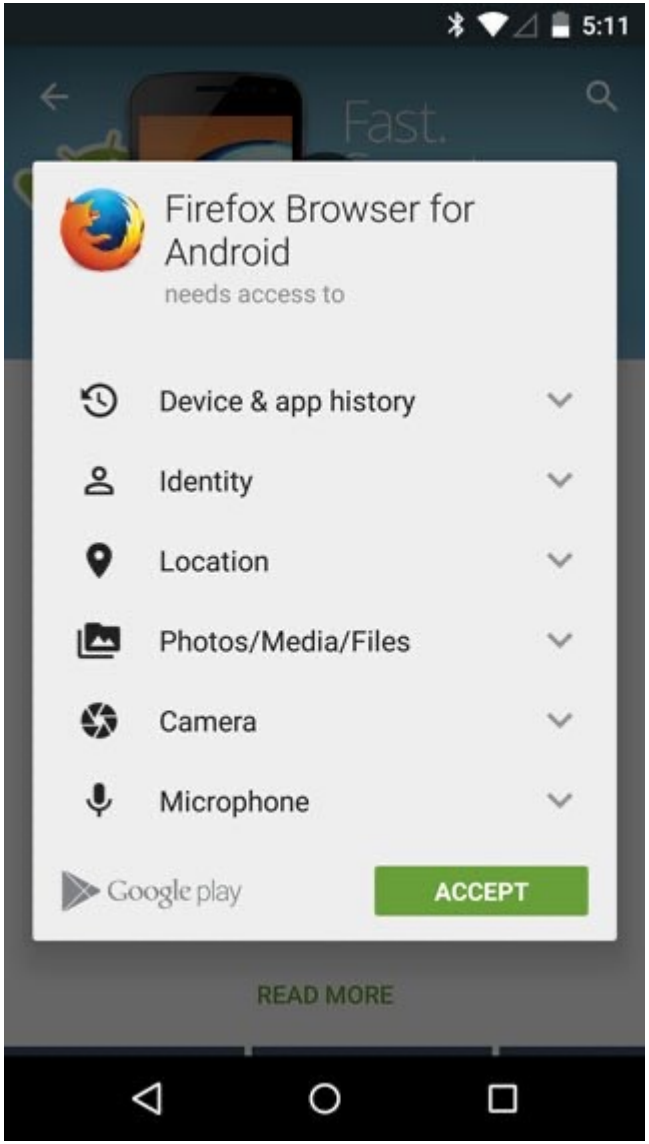
Permissions

Permission types, declaration

Permissions

- The purpose of a permission is to protect the privacy of an Android user.

- Android apps must request permission to access sensitive user data (such as contacts and SMS) and certain system features (such as camera and internet)



Permission

- If an application has permission to perform any operations, it could adversely impact other applications. E.g.
  - Read & write data
  - Make a call
  - …
- By default, an android app has no associated permissions.

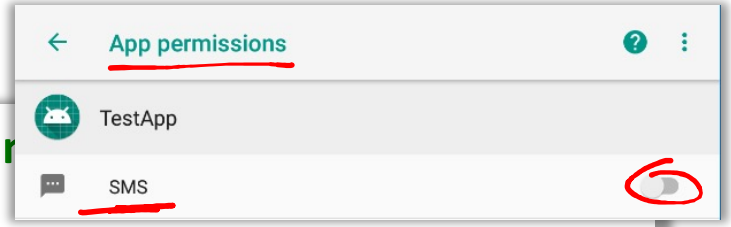
Permission Types

- Normal permissions
  - Don't pose much risk to the user's privacy or the device's operation (E.g. Request for setting time zone).
  - System automatically grants those permissions when the app is installed.
- Dangerous permissions
  - Potentially affect the user's privacy or the device's normal operation, such as the SEND_SMS permission.
  - User must explicitly agree to grant those permissions at runtime.

Declaring Permissions

- To declare the permissions needed by an app, one should add one or more <uses-permissions> blocks inside the AndroidManifest XML.
  - E.g. Monitoring incoming SMS messages + sending SMS.

- At application install time, permissions requested by the application are granted to it by the package installer




```
<manifest xmlns:android="http://schemas.ar
  package="cn.edu.xjtlu.testapp">
  <uses-permission android:name="android.permission.RECEIVE_SMS" />
  <uses-permission android:name="android.permission.SEND_SMS" />
```

Declaring Permissions

- There are many permissions defined by the Android system
  - android.permission.ACCESS_COARSE_LOCATION
  - android.permission.ACCESS_NETWORK_STATE
  - …

- A full list is available at:
  - https://developer.android.google.cn/reference/android/Manifest.permission
- A related link:
  - https://developer.android.google.cn/guide/topics/manifest/uses-permission-element

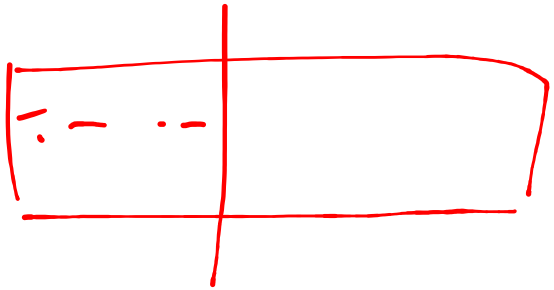
Read about Permissions

- https://developer.android.google.cn/guide/topics/permissions/overview.html
  - Also includes how to specify your own permissions, check the “Permission enforcement” section.

Data Management

Internal/External Files, SharedPreferences, SQLite Database

File



- Android uses a file system that's similar to disk-based file systems on other platforms.
- A `File` object is suited to reading or writing large amounts of data in start-to-finish order without skipping around.
  - `java.io.file`
- A file is identified by a pathname.
- Storage areas are classified as internal or external.

Internal VS External Storage

Internal storage:

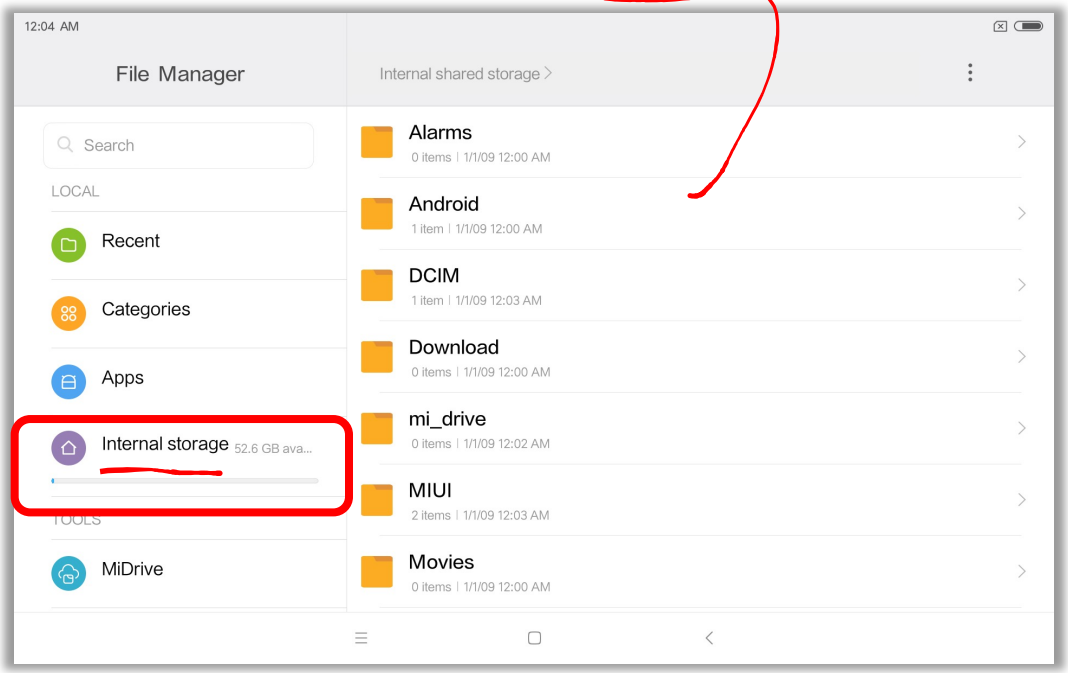
- It's always available.
- Files are accessible only by your app by default.
- When a user uninstalls your application, these files are removed.
- Encrypted by the system since Android 10

External storage:

- It's not always available.
- Files are world-readable, and can be modified when USB is connected.
- May not be completely removed when the user uninstalls the app.

Internal VS External Storage

- Our Xiaomi tablet has a built-in storage mounted at folder "/storage/emulated/0" (or sdcard0)
  - Part of it is used as internal memory
  - Part of it is used as external memory



Files you see here are from the external memory

Writing An Internal Memory File

To create and write a private file to the internal storage:

Context

- Call `openFileOutput()` with the name of the file and the operating mode. This returns a `java.io.FileOutputStream`.
- Write to the file with `write()`.
- Close the stream with `close()`.

Writing An Internal Memory File

```
String fileName = "myfile";
String inputString ="hello";
try{
    FileOutputStream outputStr =
            openFileOutput(fileName, Context.MODE_PRIVATE);
    outputStr.write(inputString.getBytes());
    outputStr.close();
}
catch (IOException e) {
    e.printStackTrace();
}
```

MODE_PRIVATE will create the file (or replace a file of the same name) and make it private to your application. Other modes available are:

MODE_APPEND
MODE_WORLD_READABLE
MODE_WORLD_WRITEABLE

Reading An Internal Memory File

To read a file from internal storage:

- Call `openFileInput()` and pass it the name of the file to read. This returns a `java.io.FileInputStream`.
- Read bytes from the file with `read()`.
- Then close the stream with `close()`.

Reading An Internal Memory File

```
String fileName = "myfile";
try{
    FileInputStream inputStr = openFileInput(fileName);
    InputStreamReader strRe = new InputStreamReader(inputStr);
    BufferedReader bufRe = new BufferedReader(strRe);
    // reads toward a new line character and move the
    // reader pointer to the beginning of the next line
    bufRe.readLine();
    inputStr.close(); // close the file
}
catch (IOException e) {
    e.printStackTrace();
}
```

External Memory Files

- Removable media may disappear without warning
- Get the current status of external storage: String `Environment.getExternalStorageState()`
  - `MEDIA_MOUNTED` - present & mounted with read/ write access
  - `MEDIA_MOUNTED_READ_ONLY` - present & mounted with read -only access
  - `MEDIA_REMOVED` - not present
- Need permission to write external files

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

Users of Google Pixel and other devices using Android 7 above must enable memory write manually

Write (Read) An External Memory File

Provide access to environment variables

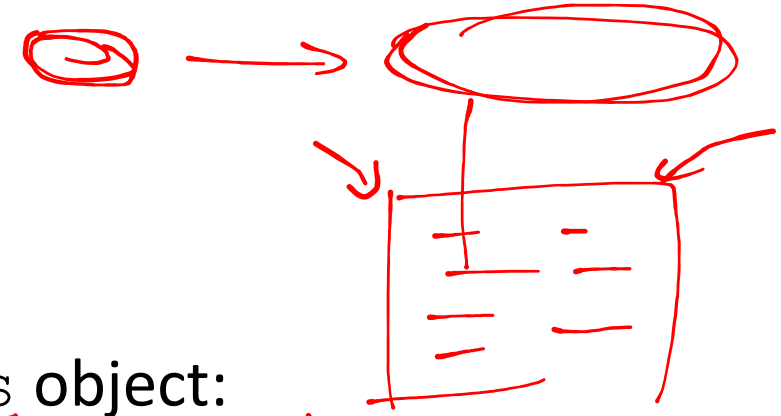
```
String filename = "myfile";
if (Environment.MEDIA_MOUNTED.equals(Environment.getExternalStorageState()))
{
  File file = new File(getExternalFilesDir(null), filename);
  try {
    FileOutputStream fo = new FileOutputStream(file, true);
    String writeToFile = "hello world";
    fo.write(writeToFile.getBytes());
    fo.close();
  }
  catch (IOException e) {}
}
```

The absolute path will be at Android\data\your.project.name\files
Note: You may need to re-connect your device to the computer in order to display a newly created file.

SharedPreference

- Shared Preferences store private primitive data in key-value pairs in internal memory.
- You can use `SharedPreferences` to save primitive data: `booleans`, `floats`, `ints`, `longs`, and `Strings`.
- The data will persist across user sessions (even if your application is killed).
- Shared Preferences are typically used to record
  - User ID
  - Favorite WiFi networks

SharedPreference



To get a `SharedPreferences` object:

- Context.getSharedPreferences(String name, int mode)
  - Retrieve a SharedPreferences object, using the key "name". Only one instance is returned to any callers, meaning they will see each other's edits as soon as they are made.
- Activity.getPreferences(int mode)
  - Retrieve a SharedPreferences object for accessing preferences that are private to this activity.
- The mode parameter could be:
  - MODE_PRIVATE,
  - ~~MODE_WORLD_READABLE~~ (deprecated!)
  - ~~MODE_WORLD_WRITEABLE~~ (deprecated!)

Using SharedPreferences

- Get SharedPreferences:
  - this.getSharedPreferences("id", MODE_PRIVATE);
  - this.getPreferences(MODE_PRIVATE);
  - Then access functions like getInt() …
- Write a sharedPreference:
  - Call the `SharedPreferences.edit()` method and get an instance of editor.
  - Write data using functions like putInt() …
  - Commit the change by calling the method `SharedPreferences.Editor.commit()`.

Example 1

- Record how many times a file has been opened.

```
// or getSharedPreference
SharedPreferences mSharedP = getPreferences(0);
int counter = mSharedP.getInt("fileOpenTimes", 0);
try{
   //open the file
   counter++;
}

SharedPreferences.Editor mEditor = mSharedP.edit();
mEditor.putInt("fileOpenTimes", counter);
mEditor.commit();
```

getInt(String key, int defValue) retrieves an int value from the preferences.
key--The name of the preference to retrieve.
defValue--Value to return if this preference does not exist.

Example 2

```java
public class OutgoingCallReceiver extends BroadcastReceiver {

    String phoneNumber = "null";
    Context context;

    @Override
    public void onReceive(Context context, Intent intent) {

        phoneNumber = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

        //TODO: record outgoing call number here
        SharedPreferences prefs = context.getSharedPreferences("Phone", Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = prefs.edit();
        editor.clear();
        editor.putString("new_number", phoneNumber);
        editor.commit();
    }
}
```

```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.example4_layout);

        SharedPreferences prefs = getSharedPreferences("Phone", MODE_PRIVATE);
        String number = prefs.getString("new_number", "empty");

        TextView textNumb = (TextView) findViewById(R.id.records);
        textNumb.setText(number);
    }
```

SQLite

- SQLite provides in-memory database
- Designed to operate within a very small footprint (<300kB) within a single cross-platform disk file
- Implements most of SQL92
- Supports ACID transactions
  - atomic, consistent, isolated & durable

Opening A Database

Recommended method relies on a helper class called `SQLiteOpenHelper`

- Create a subclass of `SQLiteOpenHelper`
  - Override `onCreate()`
  - Override `onUpgrade()`
  - Create other functions that operate on the database
- Use Constructor to instantiate the subclass
- Use `SQLiteOpenHelper` methods to open & return underlying database

SQLite Example

- We want to build a table "Contacts" to record contact information. It consists of
  - `contact_id(int)`
  - `contact_name(string)`
  - `contact_phone_number(string)`

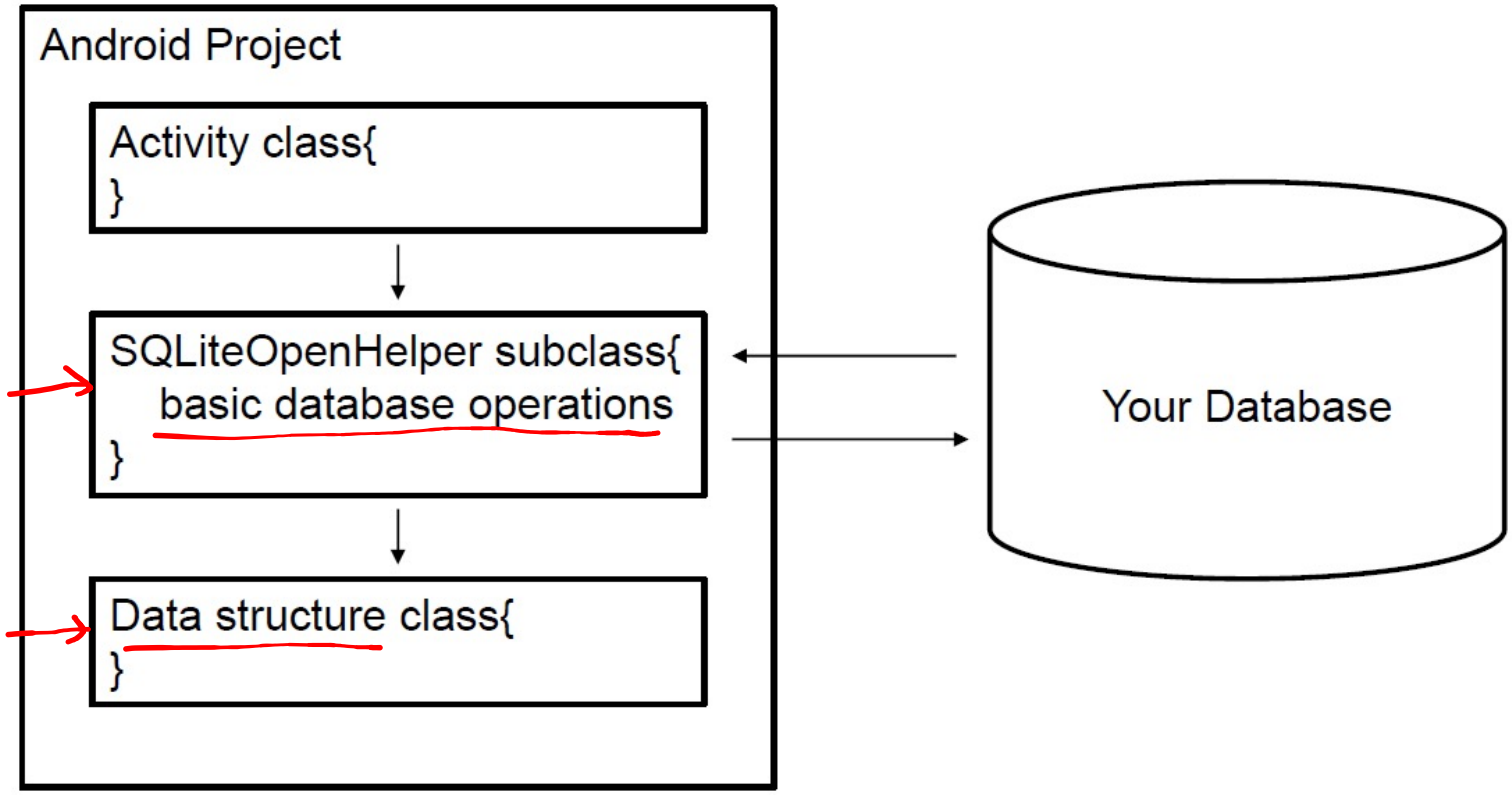
Table Structure AndroidHive

Table Name: Contacts

Field	Type	Key
id	INT	PRI
name	TEXT	
phone_number	TEXT	

SQLite Example: Solution

- Overall Framework



SQLite Example (cont.)

- Step 1: Build a contact class (data structure)

```
public class Contact {
  int _id;
  String _name;
  String _phone_number;
  public Contact(int id, String name, String _phone_number){
    this._id = id;
    this._name = name;
    this._phone_number = _phone_number;}
  public int getID() { return this._id;}
  public void setID(int id) { this._id = id;}
  public String getName() { return this._name;}
  public void setName(String name) { this._name = name;}
  public String getPhoneNumber() { return this._phone_number;}
  public void setPhoneNumber(String phone_number)
            { this._phone_number = phone_number;}
}
```

Step 2: Create SQL Table

```
public class MyContactDatabase extends SQLiteOpenHelper {
  private static final int DATABASE_VERSION = 1; // Database Version
  private static final String DATABASE_NAME = "contactsManager"; // Database Name
  private static final String TABLE_CONTACTS = "contacts"; // Contacts table name
  private static final String KEY_ID = "id"; // Contacts Table Columns names, so as below
  private static final String KEY_NAME = "name";
  private static final String KEY_PH_NO = "phone_number";

  public MyContactDatabase(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
        + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
        + KEY_PH_NO + " TEXT" + ")";
    db.execSQL(CREATE_CONTACTS_TABLE);
  }

  @Override
  public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {}
```

Increasing the version here, onUpgrade() will be called

only run when the database file did not exist and was just created

is only called when the database file exists but the stored version number is lower than requested in constructor.

Step 3: Database Operations

Still inside MyContactDatabase

```
public void addContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
    // ContentValues() class is used to store a set of
    //  values that content resolver can handle
    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName()); // Contact Name
    values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone Number
    // Inserting Row
    db.insert(TABLE_CONTACTS, null, values);
    db.close(); // Closing database connection
}
```

Instead of using a content resolver, you can put a sql statement here like
String ROW1 = "INSERT INTO " + …;
db.execSQL(ROW1);

```
public Contact getContact(int id) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
                KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
            new String[] { String.valueOf(id) }, null, null, null, null);
    if (cursor != null)
        cursor.moveToFirst();
    Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
            cursor.getString(1), cursor.getString(2));
    return contact;
}
```

Step 4: Use the database in activity

```
public class MyContactActivity extends AppCompatActivity {

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

        MyContactDatabase db = new MyContactDatabase(this);
        db.addContact(new Contact(1012, "Alice", "88160000"));
        db.addContact(new Contact(913, "Bob", "88160001"));
        // Reading contact
        Contact contact = db.getContact(1);
        String log = "Id: "+ contact.getID()+" ,Name: " + contact.getName() + " ,Phone: " +
                contact.getPhoneNumber();
        // Writing Contacts to log
        Log.d("Name: ", log);
    }
}
```

Extended Readings

Overview about storage:

- https://developer.android.google.cn/training/data-storage

Save files on device storage

- https://developer.android.com/training/data-storage/files

Shared preferences

- https://developer.android.com/training/data-storage/shared-preferences

Practice

- Use SharedPreferences to record
  - How many times your app has been launched (i.e, the onCreate() method called)
  - How many times your app has been shown on the screen (ie, the onStart() method called)
- Build you own database to record contact information (left for you to practice after lab)