THE UNIVERSITY OF MELBOURNE

COMP90018

Mobile Computing Systems Programming

WEEK 4 – Data Storage

Atiq Shaikh,

atiq.shaikh@unimelb.edu.au

1 Data Storage

2 SharedPreferences

3 SQLite

4 Content Provider

5 File Storage

6 Cloud Platform(Firebase)

Intended Learning Outcomes

Data Storage

Data Storage

When building mobile applications you will often need to ***store persistent data*** on the device itself.

This type of storage allows your applications to

1. function effectively when there is ***no connection***;
2. offering more advanced and often faster ***functionality*** than server side storage only.

Data Storage

Android provides ***several options*** for you to save your app data:

- *SharedPreferences;*
- *SQLite;*
- *File Storage;*
- *Content Provider;*
- *Cloud Storage;*
- …

SharedPreferences

SharedPreferences

If you have a relatively ***small collection of key-values*** that you'd like to save, you should use the ***SharedPreferences*** APIs.

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

SharedPreferences

A ***SharedPreferences object*** points to a file containing ***key-value pairs*** and provides simple methods to ***read and write*** them.

Each SharedPreferences file is managed by the framework and can be ***private or shared***.

SharedPreferences – Code

Save

```
SharedPreferences.Editor editor = getSharedPreferences(PREFERENCE_NAME, MODE_PRIVATE).edit();
editor.putString(OBJECT_NAME, String.valueOf(name.getText()));
editor.putInt(OBJECT_AGE, Integer.valueOf(String.valueOf(age.getText())));
editor.apply();
```

Load

```
SharedPreferences pref = getSharedPreferences(PREFERENCE_NAME, MODE_PRIVATE);
name.setText(pref.getString(OBJECT_NAME, s1: ""));
age.setText(String.valueOf(pref.getInt(OBJECT_AGE, i: 0)));
```

SharedPreferences – Demonstration

SQLite

Android and SQLite

Android provides ***full support for SQLite databases***.

Any database you create is ***accessible only by your app***.

https://developer.android.com/guide/topics/data/data-storage#db

For Reading – SQLite

SQLite is a ***C-language library*** that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.

SQLite is the most used database engine in the world. SQLite is built into all ***mobile phones*** and most computers and comes bundled inside countless other applications that people use every day.

https://www.sqlite.org/index.html

SQLite

Saving data to a database is ideal for ***repeating or structured data***, such as contact information.

SQLite can perform the operations of ***insert, query, update and delete*** as other relational database.

https://developer.android.com/training/data-storage/sqlite.html

SQLiteOpenHelper

When creating and maintaining database and tables, ***SQLiteOpenHelper*** class contains a useful set of APIs for managing your database.

All you need to do is call ***getWritableDatabase()*** or ***getReadableDatabase()***.

https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

SQLite

- *Create a Database*

Define a Schema

One of the main principles of SQL databases is the ***schema***: *a formal declaration of how the database is organized*.

The schema is reflected in the SQL statements that you use to create your database.

Extend SQLiteOpenHelper

```java
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    public static final String CREATE_CATEGORY = "create table Category ("
            + "id integer primary key autoincrement, "
            + "category_name text, "
            + "category_code integer)";

    private Context mContext;

    public MyDatabaseHelper(Context context, String name,
                            SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        db.execSQL(CREATE_CATEGORY);
        Toast.makeText(mContext, text: "Create succeeded", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("drop table if exists Book");
        db.execSQL("drop table if exists Category");
        onCreate(db);
    }
```

Create a subclass that overrides the ***onCreate()*** and ***onUpgrade()*** callback methods.

Hold the instance of SQLiteOpenHelper

Hold a instance of SQLiteOpenHelper

```
dbHelper = new MyDatabaseHelper( context: this,  name: "BookStore.db",  factory: null,  version: 2);
```

Get writable database
via the instance of SQLiteOpenHelper

```
dbHelper.getWritableDatabase();
```

Execute the defined onCreate() method


```java
public class MyDatabaseHelper extends SQLiteOpenHelper {

    public static final String CREATE_BOOK = "create table Book ("
            + "id integer primary key autoincrement, "
            + "author text, "
            + "price real, "
            + "pages integer, "
            + "name text)";

    public static final String CREATE_CATEGORY = "create table Category ("
            + "id integer primary key autoincrement, "
            + "category_name text, "
            + "category_code integer)";

    private Context mContext;

    public MyDatabaseHelper(Context context, String name,
                            SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_BOOK);
        db.execSQL(CREATE_CATEGORY);
        Toast.makeText(mContext, text: "Create succeeded", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("drop table if exists Book");
        db.execSQL("drop table if exists Category");
        onCreate(db);
    }
```

When the database does not exist, the instance of SQLiteOpenHelper will create a database ***according to onCreate() method***.

SQLite

- *CRUD*

Insert Data

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();

// Compose the first data
values.put("name", "The Da Vinci Code");
values.put("author", "Dan Brown");
values.put("pages", 454);
values.put("price", 16.96);
// Insert the first data
db.insert(table: "Book", nullColumnHack: null, values);
values.clear();

// Compose the second data
values.put("name", "The Lost Symbol");
values.put("author", "Dan Brown");
values.put("pages", 510);
values.put("price", 15.96);
// insert the second data
db.insert(table: "Book", nullColumnHack: null, values);
```

Insert data into the database by passing a ***ContentValues*** object to the ***insert()*** method.

Query Data

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
// Query the data in Book database
Cursor cursor = db.query( table: "Book",  columns: null,  selection: null,  selectionArgs: null,  groupBy: null,  having: null,  orderBy: null);
if (cursor.moveToFirst()) {

        // Display all the rows in Book database
        String name = cursor.getString(cursor.getColumnIndex( s: "name"));
        String author = cursor.getString(cursor.getColumnIndex( s: "author"));
        int pages = cursor.getInt(cursor.getColumnIndex( s: "pages"));
        double price = cursor.getDouble(cursor.getColumnIndex( s: "price"));

                msg: "————————————————————\n" +
                        "book name is " + name + "\n" +
                        "book author is " + author + "\n" +
                        "book pages is " + pages + "\n" +
                        "book price is " + price + "\n" +
                        "————————————————————"
        );
    } while (cursor.moveToNext());
}
cursor.close();
```

To read from a database, use the ***query()*** method, passing it your selection criteria and desired columns.

Update Data


```
SQLiteDatabase db = dbHelper.getWritableDatabase();
ContentValues values = new ContentValues();
values.put("price", 10.99);
db.update( table: "Book", values, whereClause: "name = ?", new String[]{"The Da Vinci Code"});
```

When you need to ***modify a subset of your database values***, use the ***update()*** method.

Delete Data


```
SQLiteDatabase db = dbHelper.getWritableDatabase();
db.delete( table: "Book",  whereClause: "pages > ?", new String[]{"500"});
```

To delete rows from a table, you need to provide ***selection criteria*** that identify the rows to the ***delete()*** method.

SQLite

- *greenDao(For Reading)*

For Reading – greenDao

greenDAO is an ***open source Android ORM*** *(Object-Relational Mapping)* making development for SQLite databases fun again.

http://greenrobot.org/greendao/

http://greenrobot.org/greendao/documentation/

greenDao – Add greenDao to your project

```
app
  manifests
  java
    com.example.databasedemo
    com.example.databasedemo
      DBManager
      MainActivity
      UserAdapter
      UserObject
    com.example.databasedemo (androi
    com.example.databasedemo (test)
  generatedJava
  res
Gradle Scripts
  build.gradle (Module: app)
  proguard-rules.pro (ProGuard Rules for
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

```
apply plugin: 'org.greenrobot.greendao' // apply plugin

android {
    compileSdkVersion 28

    defaultConfig {
        applicationId "com.example.databasedemo"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }

    sourceSets {
        main {
            java.srcDirs = ['src/main/java', 'build/generated/source/greendao']
        }
    }

    // Butterknife requires Java 8.
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    implementation 'org.greenrobot:greendao:3.2.2' // add library
```

build.gradle(app level)

```
app
  manifests
  java
    com.example.databasedemo
    com.example.databasedemo
      DBManager
      MainActivity
      UserAdapter
      UserObject
    com.example.databasedemo (androi
    com.example.databasedemo (test)
  generatedJava
  res
  build.gradle (Project: DatabaseDemo)
  gradle-wrapper.properties (Gradle Versic
  proguard-rules.pro (ProGuard Rules for
  gradle.properties (Project Properties)
  settings.gradle (Project Settings)
  local.properties (SDK Location)
```

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()
        mavenCentral() // add repository
    }
    dependencies {

        classpath 'org.greenrobot:greendao-gradle-plugin:3.2.2'

        classpath 'com.jakewharton:butterknife-gradle-plugin:10.1.0'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
```

build.gradle(project level)

greenDao – Create Entity class

```java
@Entity
public class UserObject {

    @Id(autoincrement = true)
    private Long id;

    @NotNull
    private String name;

    private Long age;
```

UserObject.java

```java
@Entity
public class UserObject {

    @Id(autoincrement = true)
    private Long id;

    @NotNull
    private String name;

    private Long age;

    @Generated(hash = 1300257024)
    public UserObject() {
    }

    @Generated(hash = 961160741)
    public UserObject(Long id, @NotNull String name, Long age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public Long getId() { return this.id; }

    public void setId(Long id) { this.id = id; }

    public String getName() { return this.name; }

    public void setName(String name) { this.name = name; }

    public Long getAge() { return this.age; }

    public void setAge(Long age) { this.age = age; }

    public String toString() { return "ID:" + id + '\t' + "Name:" + name + '\t' + "Age:" + age; }

}
```

UserObject.java(automatic generated)

greenDao – Create DBManager class

```java
public class DBManager {
    private static DBManager dbManager;
    private DaoSession daoSession;
    private UserObjectDao userObjectDao;

    private DBManager(Context context) {
        DaoMaster.DevOpenHelper helper = new DaoMaster.DevOpenHelper(context, name: "user-db");
        Database db = helper.getWritableDb();
        daoSession = new DaoMaster(db).newSession();
        userObjectDao = daoSession.getUserObjectDao();
    }

    public static synchronized DBManager getDbManager(Context context) {
        if (dbManager == null) {
            dbManager = new DBManager(context);
        }
        return dbManager;
    }

    public void insertUser(UserObject user) {
        daoSession.insert(user);
    }

    public void deleteUser(UserObject user) {
        daoSession.delete(user);
    }

    public void deleteUsers(List<UserObject> users) {
        for (UserObject user : users) {
            daoSession.delete(user);
        }
    }

    public void updateUser(UserObject user) {
        daoSession.update(user);
    }

    public List<UserObject> queryUser(String name, Long age) {
        QueryBuilder<UserObject> query = userObjectDao.queryBuilder();
        query.where(query.and(UserObjectDao.Properties.Name.eq(name),
                UserObjectDao.Properties.Age.eq(age)));
        return query.list();
    }

    public List<UserObject> queryUser(String name) {
        QueryBuilder<UserObject> query = userObjectDao.queryBuilder();
        query.where(UserObjectDao.Properties.Name.eq(name));
        return query.list();
    }

    public List<UserObject> queryUser(Long age) {
        QueryBuilder<UserObject> query = userObjectDao.queryBuilder();
        query.where(UserObjectDao.Properties.Age.eq(age));
        return query.list();
    }

    public List<UserObject> queryAllUser() {
        QueryBuilder<UserObject> query = userObjectDao.queryBuilder();
        return query.list();
    }
}
```

DBManager.java

Create DBManager class to ***encapsulate CRUD methods*** and ***obtain the instances*** of DaoSession, DaoMaster, etc..

greenDao – Control database through DBManager

```java
public class MainActivity extends AppCompatActivity {
    private String TAG = "GreenDao Demo";
    private DBManager dbManager;
    private UserAdapter userAdapter;

    @BindView(R.id.name)
    EditText name;
    @BindView(R.id.age)
    EditText age;
    @BindView(R.id.insert)
    Button insert;
    @BindView(R.id.delete)
    Button delete;
    @BindView(R.id.listview)
    ListView listView;

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

        dbManager = DBManager.getDbManager(this);

        userAdapter = new UserAdapter(this, R.layout.single_user, dbManager.queryAllUser());
        listView.setAdapter(userAdapter);
    }

    @OnClick(R.id.insert)
    public void insertUser() {
        String user_name = determineName();
        Long user_age = determineAge();

        if (user_name != null && user_age != null) {
            UserObject user = new UserObject();
            user.setName(user_name);
            user.setAge(user_age);
            dbManager.insertUser(user);
            updateList();
        }
    }

    @OnClick(R.id.delete)
    public void deleteUser() {
        String user_name = determineName();
        Long user_age = determineAge();

        if (user_name != null && user_age != null) {
            List<UserObject> users = dbManager.queryUser(user_name, user_age);
            dbManager.deleteUsers(users);
        } else if (user_name != null) {
            List<UserObject> users = dbManager.queryUser(user_name);
            dbManager.deleteUsers(users);
        } else if (user_age != null) {
            List<UserObject> users = dbManager.queryUser(user_age);
            dbManager.deleteUsers(users);
        }

        updateList();
    }

    private void updateList() {
        userAdapter.clear();
        userAdapter.addAll(dbManager.queryAllUser());
        userAdapter.notifyDataSetChanged();
    }

    private String determineName() {
        String user_name = String.valueOf(name.getText());
        if (user_name.equals(""))
            return null;
        else
            return user_name;
    }

    private Long determineAge() {
        String user_age = String.valueOf(age.getText());
        if (user_age.equals(""))
            return null;
        else if (user_age.matches("[0-9]+")) {
            return Long.valueOf(user_age);
        } else {
            return null;
        }
    }
}
```

MainActivity.java

Content Provider

Content Provider

Content Provider is a component of an Android application.

https://developer.android.com/guide/topics/providers/content-provider-basics

Content Provider

Content providers are primarily intended to be ***used by other applications***, which access the provider using a ***provider client*** object.

Providers and provider clients offer a consistent, standard ***interface to data*** that also handles inter-process communication(***IPC***) and secure ***data access***.

Access data through a Content Provider

그림 출처 : arabiannight.tistory.com

URI

- To query a content provider, you specify the query string in the form of a URI which has following format:

- content://<authority>/<data_type>/<id>

This specifies ***the name of the content provider***, like *com.example.datatest.provider.*

Name of database table that this particular provider provides.

Specific record requested (optional).

Content Provider

The Content Provider is required to provide the basic ***"CRUD" (create, retrieve, update, and delete)*** functions of persistent storage.

Content Provider – Create

Content Provider – Overwrite methods

```java
@Override
public Uri insert(Uri uri, ContentValues values) {
    // Insert data from provider
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    Uri uriReturn = null;
    switch (uriMatcher.match(uri)) {
        case BOOK_DIR:
        case BOOK_ITEM:
            long newBookId = db.insert( table: "Book", nullColumnHack: null, values);
            uriReturn = Uri.parse("content://" + AUTHORITY + "/book/" + newBookId);
            break;
        case CATEGORY_DIR:
        case CATEGORY_ITEM:
            long newCategoryId = db.insert( table: "Category", nullColumnHack: null, values);
            uriReturn = Uri.parse("content://" + AUTHORITY + "/category/" + newCategoryId);
            break;
        default:
            break;
    }
    return uriReturn;
}
```

Create - Insert

```java
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    // Query data from provider
    SQLiteDatabase db = dbHelper.getReadableDatabase();
    Cursor cursor = null;
    switch (uriMatcher.match(uri)) {
        case BOOK_DIR:
            cursor = db.query( table: "Book", projection, selection, selectionArgs, groupBy: null, having: null, sortOrder);
            break;
        case BOOK_ITEM:
            String bookId = uri.getPathSegments().get(1);
            cursor = db.query( table: "Book", projection, selection: "id = ?", new String[] { bookId }, groupBy: null, having: null, sortOrder);
            break;
        case CATEGORY_DIR:
            cursor = db.query( table: "Category", projection, selection, selectionArgs, groupBy: null, having: null, sortOrder);
            break;
        case CATEGORY_ITEM:
            String categoryId = uri.getPathSegments().get(1);
            cursor = db.query( table: "Category", projection, selection: "id = ?", new String[] { categoryId }, groupBy: null, having: null, sortOrder);
            break;
        default:
            break;
    }
    return cursor;
}
```

Read - Query

Content Provider – Overwrite methods


```java
@Override
public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
    // Update data from provider
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    int updatedRows = 0;
    switch (uriMatcher.match(uri)) {
        case BOOK_DIR:
            updatedRows = db.update( table: "Book", values, selection, selectionArgs);
            break;
        case BOOK_ITEM:
            String bookId = uri.getPathSegments().get(1);
            updatedRows = db.update( table: "Book", values, whereClause: "id = ?", new String[] { bookId });
            break;
        case CATEGORY_DIR:
            updatedRows = db.update( table: "Category", values, selection, selectionArgs);
            break;
        case CATEGORY_ITEM:
            String categoryId = uri.getPathSegments().get(1);
            updatedRows = db.update( table: "Category", values, whereClause: "id = ?", new String[] { categoryId });
            break;
        default:
            break;
    }
    return updatedRows;
}
```

Update

```java
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
    // Delete data from provider
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    int deletedRows = 0;
    switch (uriMatcher.match(uri)) {
        case BOOK_DIR:
            deletedRows = db.delete( table: "Book", selection, selectionArgs);
            break;
        case BOOK_ITEM:
            String bookId = uri.getPathSegments().get(1);
            deletedRows = db.delete( table: "Book", whereClause: "id = ?", new String[] { bookId });
            break;
        case CATEGORY_DIR:
            deletedRows = db.delete( table: "Category", selection, selectionArgs);
            break;
        case CATEGORY_ITEM:
            String categoryId = uri.getPathSegments().get(1);
            deletedRows = db.delete( table: "Category", whereClause: "id = ?", new String[] { categoryId });
            break;
        default:
            break;
    }
    return deletedRows;
}
```

Delete

Content Provider – Overwrite methods

getType()

```
@Override
public String getType(Uri uri) {
    switch (uriMatcher.match(uri)) {
        case BOOK_DIR:
            return "vnd.android.cursor.dir/vnd.com.example.databasetest.provider.book";
        case BOOK_ITEM:
            return "vnd.android.cursor.item/vnd.com.example.databasetest.provider.book";
        case CATEGORY_DIR:
            return "vnd.android.cursor.dir/vnd.com.example.databasetest.provider.category";
        case CATEGORY_ITEM:
            return "vnd.android.cursor.item/vnd.com.example.databasetest.provider.category";
    }
    return null;
}
```

Providers need to maintain ***MIME data*** type information for each content URI they define.

You can use the MIME type information to:

1. find out if your application can ***handle data that the provider offers***,
2. ***choose a type of handling*** based on the MIME type.

https://developer.android.com/guide/topics/providers/content-provider-basics#MIMETypeReference

ContentResolver – Client of ContentProvider

When you want to access data in a content provider, you use the ***ContentResolver*** object in your application's Context to ***communicate with the provider as a client***.

https://developer.android.com/reference/android/content/ContentResolver.html

ContentResolver

Create – Insert from ContentResolver

```
Uri uri = Uri.parse("content://com.example.databasetest.provider/book");
ContentValues values = new ContentValues();
values.put("name", "A Clash of Kings");
values.put("author", "George Martin");
values.put("pages", 1040);
values.put("price", 55.55);
Uri newUri = getContentResolver().insert(uri, values);
newId = newUri.getPathSegments().get(1);
```

Read – Query from ContentResolver

```
Uri uri = Uri.parse("content://com.example.databasetest.provider/book");
Cursor cursor = getContentResolver().query(uri, projection: null, selection: null, selectionArgs: null, sortOrder: null);
if (cursor != null) {
    while (cursor.moveToNext()) {
        String name = cursor.getString(cursor.getColumnIndex( s: "name"));
        String author = cursor.getString(cursor.getColumnIndex( s: "author"));
        int pages = cursor.getInt(cursor.getColumnIndex( s: "pages"));
        double price = cursor.getDouble(cursor.getColumnIndex( s: "price"));
        Log.d( tag: "MainActivity", msg: "book name is " + name);
        Log.d( tag: "MainActivity", msg: "book author is " + author);
        Log.d( tag: "MainActivity", msg: "book pages is " + pages);
        Log.d( tag: "MainActivity", msg: "book price is " + price);
    }
    cursor.close();
}
```

ContentResolver

Update from ContentResolver

```
Uri uri = Uri.parse("content://com.example.databasetest.provider/book/" + newId);
ContentValues values = new ContentValues();
values.put("name", "A Storm of Swords");
values.put("pages", 1216);
values.put("price", 24.05);
getContentResolver().update(uri, values, where: null, selectionArgs: null);
```

Delete from ContentResolver

```
Uri uri = Uri.parse("content://com.example.databasetest.provider/book/" + newId);
getContentResolver().delete(uri, where: null, selectionArgs: null);
```

File Storage

File Storage

Android uses ***a file system*** that's similar to disk-based file systems on other platforms.

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

Internal and External Storage

All Android devices have two file storage areas: ***"internal"*** and ***"external"*** storage. These names come from the early days of Android, when most devices offered built-in non-volatile memory (***internal storage***), plus a removable storage medium such as a micro SD card (***external storage***).

Internal and External Storage

Many devices now divide the permanent storage space into ***separate "internal" and "external" partitions***.

So even without a removable storage medium, these two storage spaces always exist, and the API behavior is the same ***regardless of whether the external storage is removable***.

Internal and External Storage

Internal storage

1. It's **always available**.
2. Files saved here are accessible by ***only your app***.
3. When the user ***uninstalls*** your app, the system ***removes all your app's files*** from internal storage.

Internal and External Storage

External storage

1. ***It's not always available***, because the user can mount the external storage as USB storage and in some cases remove it from the device.
2. It's ***world-readable***, so files saved here may be read outside of your control.
3. When the user uninstalls your app, the system removes your app's files from here ***only if*** you save them in the directory from ***getExternalFilesDir()***.

Internal and External Storage

Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

Internal Storage Demonstration

Save

```java
public void save(String inputText) {
    FileOutputStream out = null;
    BufferedWriter writer = null;
    try {
        out = openFileOutput( name: "data", Context.MODE_PRIVATE);
        writer = new BufferedWriter(new OutputStreamWriter(out));
        writer.write(inputText);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (writer != null) {
                writer.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Load

```java
public String load() {
    FileInputStream in = null;
    BufferedReader reader = null;
    StringBuilder content = new StringBuilder();
    try {
        in = openFileInput( name: "data");
        reader = new BufferedReader(new InputStreamReader(in));
        String line = "";
        while ((line = reader.readLine()) != null) {
            content.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return content.toString();
}
```

Internal Storage Demonstration

Cached File

If you'd like to keep some data ***temporarily***, rather than store it persistently, you should use the special cache directory to save the data. Each app has a private cache directory specifically for these kinds of files.

https://developer.android.com/guide/topics/data/data-storage#InternalCache

Cached File

When the device is low on internal storage space, Android ***may delete these cache files to recover space***.

Cached File

Save

```java
public void saveCache(String inputText) {
    BufferedWriter writer = null;
    try {
        File file = File.createTempFile(CACHED_FILE_NAME, suffix: "", getCacheDir());
        writer = new BufferedWriter(new FileWriter(file));
        writer.write(inputText);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            if (writer != null) {
                writer.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Load

```java
public String loadCache() {
    BufferedReader reader = null;
    StringBuilder content = new StringBuilder();
    try {
        File[] files = getCacheDir().listFiles(new FilenameFilter() {
            @Override
            public boolean accept(File file, String s) {
                return s.startsWith(CACHED_FILE_NAME);
            }
        });
        if (files.length > 0) {
            File file = files[0];
            reader = new BufferedReader(new FileReader(file));
            String line = "";
            while ((line = reader.readLine()) != null) {
                content.append(line);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (reader != null) {
            try {
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return content.toString();
}
```

Find File through Android Studio

Cloud Platform

Cloud Platform

Modern technologies like ***big data analytics, IoT, artificial intelligence, web and mobile app hosting*** all need heavy computing power.

Instead of building your own backend services, cloud platforms help you build and manage mobile applications through their cloud service, including authentication, database, image storage, etc.

https://en.wikipedia.org/wiki/Category:Cloud_platforms

Cloud Platform – Firebase

In the next tutorial, we will use ***Firebase*** to develop a demonstration application.

THE UNIVERSITY OF MELBOURNE

Thank you