CS683
Mobile Application Development
Module 4: Storage

Yuting Zhang
BU METCS

References

- Textbook
- Android developer website: https://developer.android.com/guide/topics/data/data-storage.html#pref

Topics

- **SQLite Database & Room**
- **Shared Preferences**
- File
- Content Provider
- *Firebase Storage*
- *Retrofit*

MVVM - Model

- Find all entity objects in your application (persistent data, business logic)
- Define model classes (in POJO (Plain Old Java Object))
- Decide the storage type or model for your entity objects.
- Define the database schema if needed.

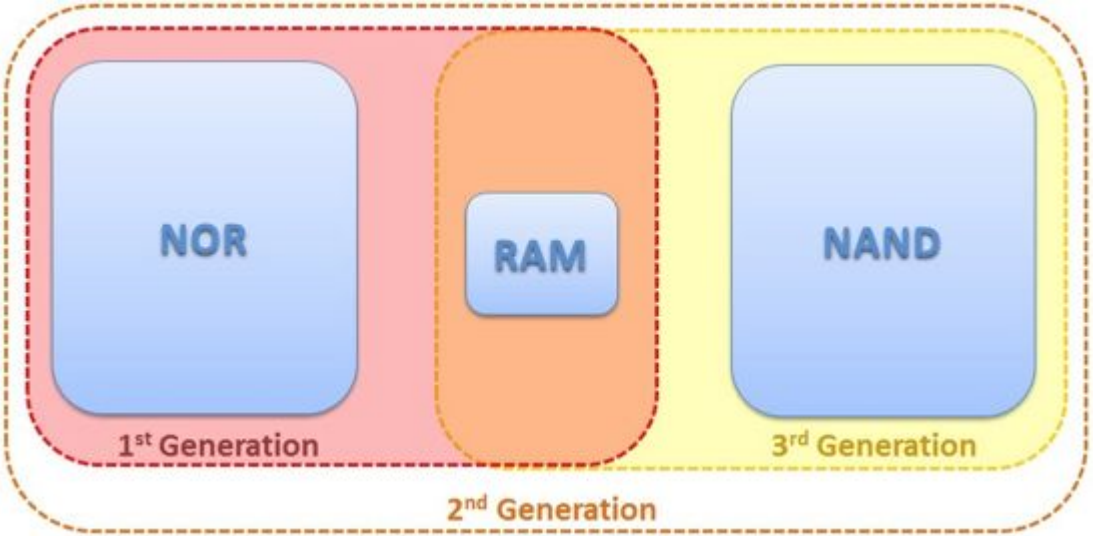
Questions to Ask

- Why need storage? What storage units are provided by mobile devices? What type of data should be stored?
- How to store these data? What formats are used to store these data?
- Where to store these data?
- What APIs are used to manage these data?
- When to use which format/method?
- What are common pitfalls (and security problems) when using these APIs?
- What are limitations?

Why Need Storage?

- Here we are talking about permanent storages (nonvolatile storages).
- Persistent application data need be accessed or stored on device flash memory (internal storage) or SD cards (external storage), or on remote server, or on cloud storage.
- Persistent application data include
  - Structured application data usually stored in database (e.g. email content, chat history, SMS messages etc)
  - Persistent application settings or user preferences across user sessions (remembered when the application is relaunched)
  - Additional structured application or unstructured data stored in files (e.g. email attachment)

Memory Evolution



From (Ayers et al., 2014) NIST SP800-101r1

- 1st: NOR (system and user data)
- 2nd: NOR (system), NAND (user data)
- 3rd: NAND (system and user data)

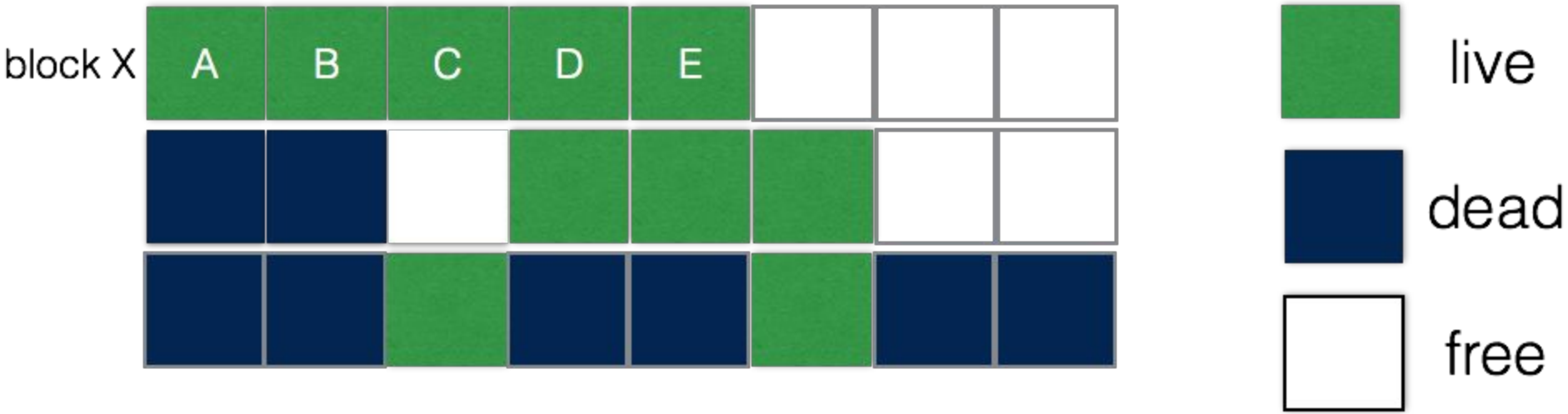
- NOR Flash: faster read times, slower write times than NAND, nearly immune to corruption and bad blocks, allow random access. Mostly feature phones contain NOR flash and RAM
- NAND: higher capacity, less stable, only allows sequential access

NAND

- Serial cell structure, high storage density, no mechanical platter, faster than magnetic disks.
- It is page-based (512B - 4KB). Pages are combined into blocks. (e.g. 64 pages ->1 block)
- Operations: read (page), write (page), erase(block).
- Pages has to be read sequentially. (No random access)
- Pages can only be written once without erasing the complete block.
- Initially all are set to 1s. Later, it can only write 0s to a page, or erase the whole block (by reset to all 1s)
- Each block has a limited write/erase lifespan (Wear-leveling techniques to improve lifespan)
- Has high error rate, often shipped with bad blocks

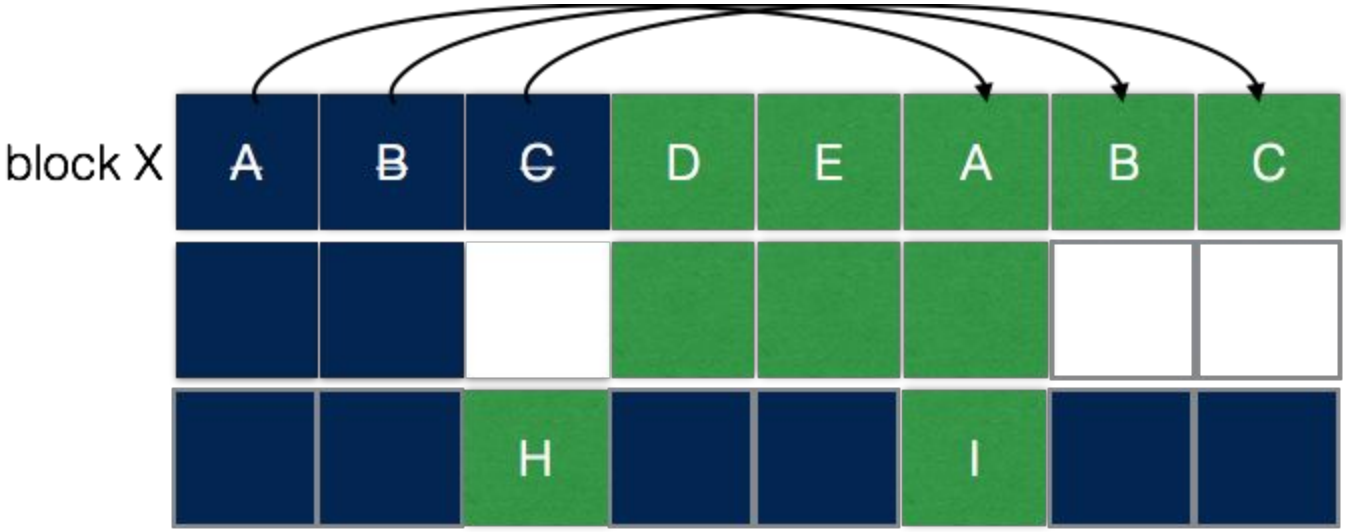
NAND Operation Examples

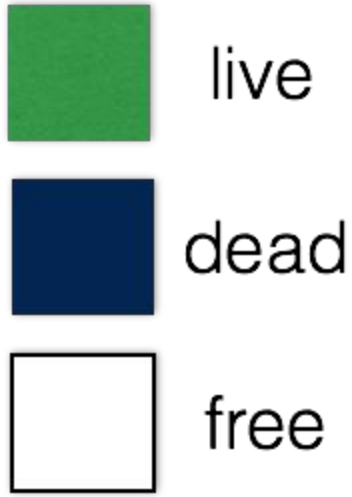
- Modify page A, B, C



NAND Operation Examples

- Modify page A, B, C

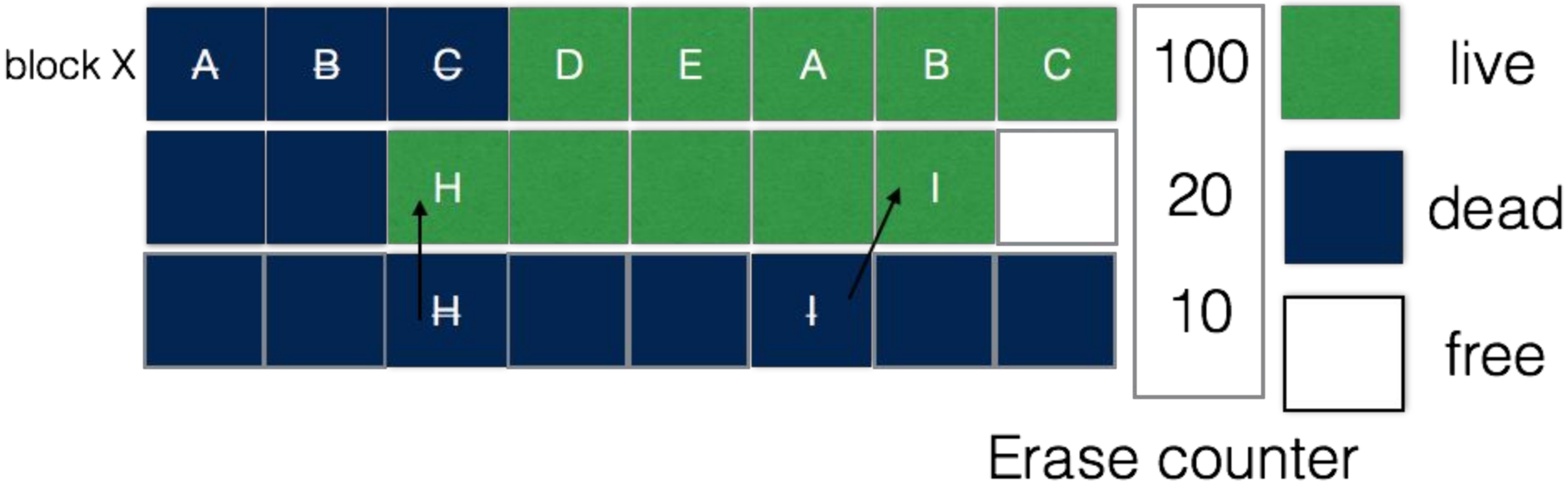




- Erase block X+2

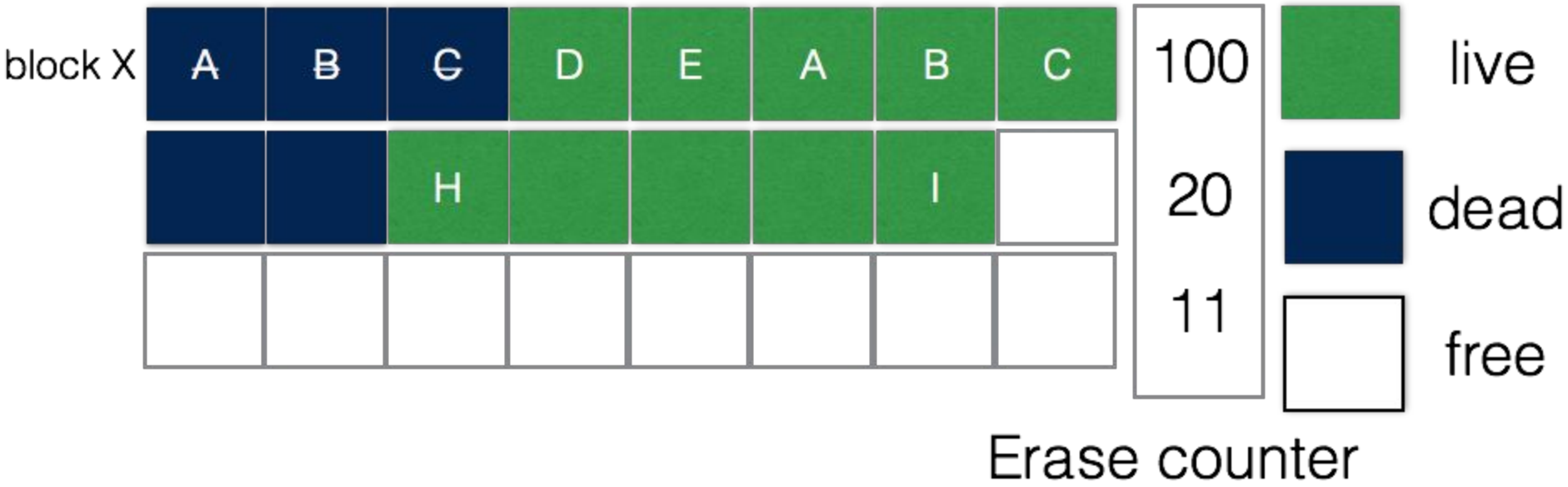
NAND Operation Examples

- Erase block X+2



NAND Operation Examples

- Erase block X+2



Data Formats

- Two special file formats are commonly used by mobile devices.
- SQLite Database - usually store application specific data
- XML file - usually store preferences and settings
  - A small amount of data
  - Key-value pairs
  - Usually primitive data types: booleans, ints, longs, floats, strings
- Data can also be stored in a general file structure, on either the internal flash memory or the external flash memory.

Data Location

- Each application stores its data in a separate directory under `/data/data` that is inaccessible to other applications.
- Each application data folder is named with the application data.
- Each folder usually has the following subfolders
  - ./shared_pref: store shared preferences
  - ./lib
  - ./files: additional files
  - ./databases: store sqlite database files
  - ...

SQLite

- An Open source, in-process library that implements a self-contained, zero configuration and transactional SQL database engine.
  - No client-server relationship
  - No need to install
- A single cross-platform file containing multiple tables, triggers, views
  - Records are stored in pages
- Portable, reliable and small. Popular database format in many mobile platforms
- May use different extension names: .db, .sqlite, .sqlitedb, or even no extension name.
- Tools:
  - SQLite Browser: http://www.sqlite.org/.
  - SQLite command-line client: http://www.sqlite.org/
  - SQLite command-line utility: sqlite3 (Mac and Linux by default)
  - Android Studio: Database inspector

SQLite in Android

- SQLite provides the foundation for managing private and embedded databases in an Android application
- The Android SDK includes the SQLite software library that implements the SQL (Structured Query Language) database engine
- This library is used for defining the database structure and adding and managing the database content as required by the application
- The Android SDK also includes an SQLite database tool for explicit database debugging purposes
- As a condensed version of SQL (Structured Query Language), SQLite supports the standard SQL syntax and database transactions
- Though SQLite is not a full-featured database, it supports a large set of the SQL standard and is sufficient for Android developers needing a simple database engine to plug into their applications

SQLite Database File

- An SQLite database file is a collection of data organized in a table
  - A row represents a single record, the implicitly structured data entry in the table
  - A column represents a data field, an attribute of a data record
  - A field is a single item that exists at the intersection between one row and one column
- The possible data types for an individual field of data consist of NULL, INTEGER, REAL, TEXT, and BLOB
  - BLOB is a data type used to store a large array of binary data (bytes)
  - SQLite does not provide specific data storage classes for values that need to be represented as Boolean, date, or time. Instead, these values can be stored as an INTEGER or TEXT

CRUD

- The four basic functions of persistent storage. It stands for Create, Read, Update and Delete.
- They are also four major operations implemented in the relational database.
- SQL statement mapping
  - C- INSERT, R- SELECT, U- UPDATE, D - DELETE
- HTTP (used to build RESTful APIs)
  - C - PUT/POST, R - GET, U - PUT/POST/PATCH, D-DELETE

Define a Schema

- Schema: a formal declaration of how the database is organized. It defines the tables, fields, relationships, views, indexes and other elements.
- The schema is reflected in the SQL statements that you use to create your Database
- We can have a contract class to explicitly specify the layout of your schema in a systematic and self- documenting way, such as defining names for URIs, tables, and columns.
- Principle: never use literal constants, instead use symbolic constants.
- Room uses annotations, making schema definition much easier

SQLite3 Commands

- Commands:
  - Connecting to a database: sqlite3 filename.sqlitedb
  - disconnect: .exit
  - dot commands: .tables, .schema table-name, .dump table-name, .output file-name, .headers on, .mode MODE, .help
- SQL Queries
  - SELECT … FROM … WHERE … ORDER by…
- Sqlite browser: https://sqlitebrowser.org/

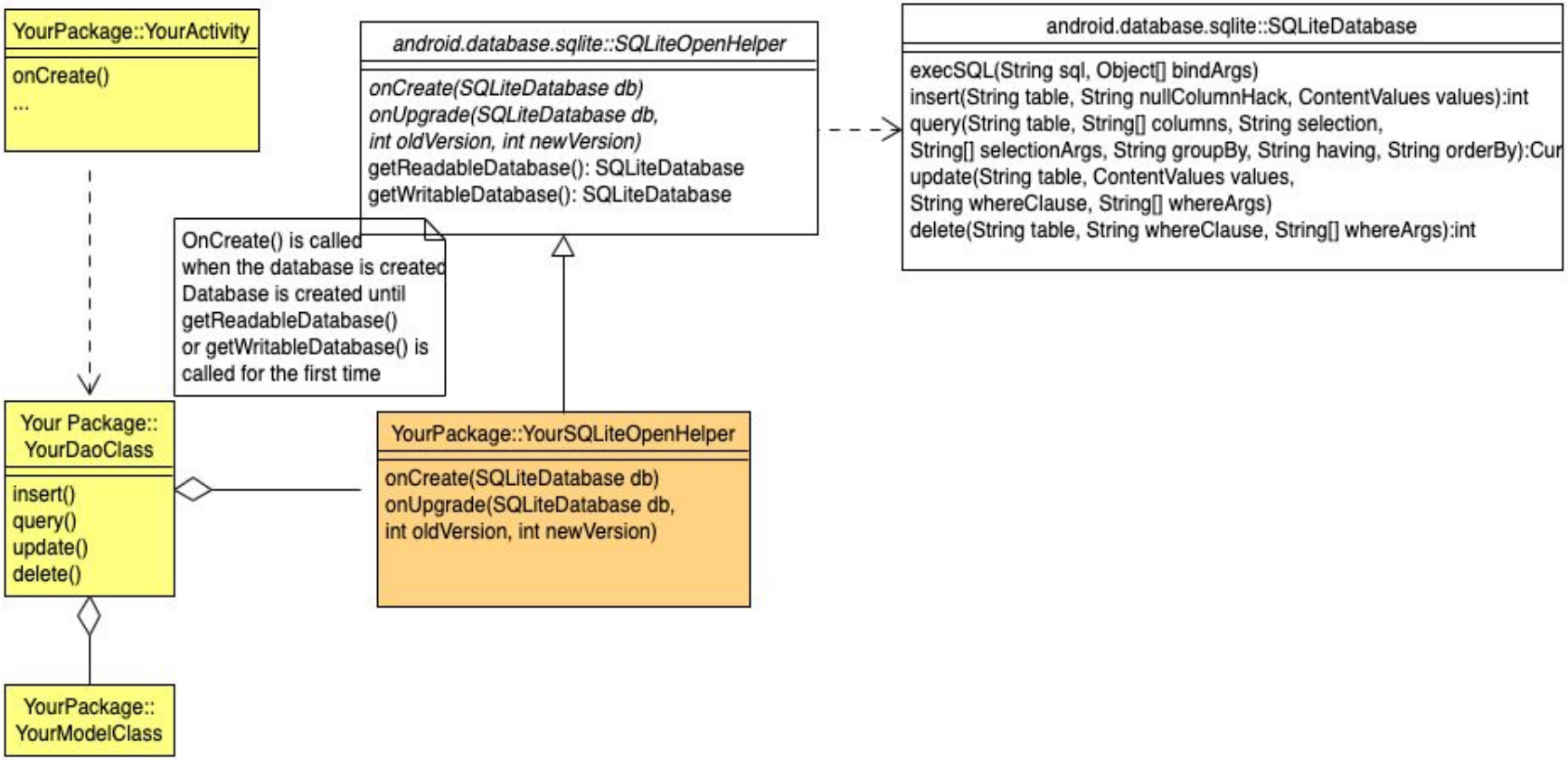
Basic SQL Statements

- CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  column3 datatype,
  ....
  );
- DROP TABLE table_name;
- SELECT column1, column2, ...
  FROM table_name
  WHERE condition;
- DELETE FROM table_name
  WHERE condition;
- https://www.w3schools.com/sql/sql_intro.asp

Database APIs:SQLiteDatabase

- Class: SQLiteDatabase (It is a final class, cannot be extended): Exposes methods to manage a SQLite database.
- Methods to perform CRUD operations:
  - long insert(String table, String nullColumnHack, ContentValues values);
  - Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
  - int update (String table, ContentValues values, String whereClause, String[] whereArgs)
  - int delete (String table, String whereClause, String[] whereArgs)
- The database engine already implements these methods. Your app simply calls these methods.
- Each SQLiteDatabase object is associated with a SQLite file.

SQLite Database APIs



Database APIs: SQLiteOpenHelper

- Class: SQLiteOpenHelper
  - This is the help class to manage database creation and version change.
  - Create a subclass of SQLiteOpenHelper
  - Implement methods onCreate(), onUpgrade(), onDowngrade() methods to create your database.
    - onCreate() need to be implemented so that tables are created when the database is first initialized. This is only called once unless the database version changes.
    - onUpgrade() and onDowngrade() are called when the database versions are changed.
  - Instantiate your subclass to create an object
  - Call getWritableDatabase() or getReadableDatabase() to create and/or get the reference of the SQLiteDatabase.
    - onCreate() will not be called until one of these methods is called.
  - Implement CRUD operations by wrapping the database CRUD methods.

Database APIs

- A contract class is often used to explicitly specify the layout of the database schema. It is a container for constants that define names for URIs, tables, and columns.
- Class: ContentValues:
  - Provides a number of put and get methods to access key value pairs. (e.g.
    void put(String key, Integer value), Integer getAsInteger(String key) …)
- Class: Cursor:
  - Provides a number of methods to randomly access the result set returned returned by a database query. (e.g. move(int offset), moveToFirst(), moveToNext(), moveToPosition(int position), getString(int columnIndex), getLong(int columnIndex), getColumnIndex(String columnName) ...)
  - When finished iterating through results, call close() on the cursor to release its resources

Using ORM

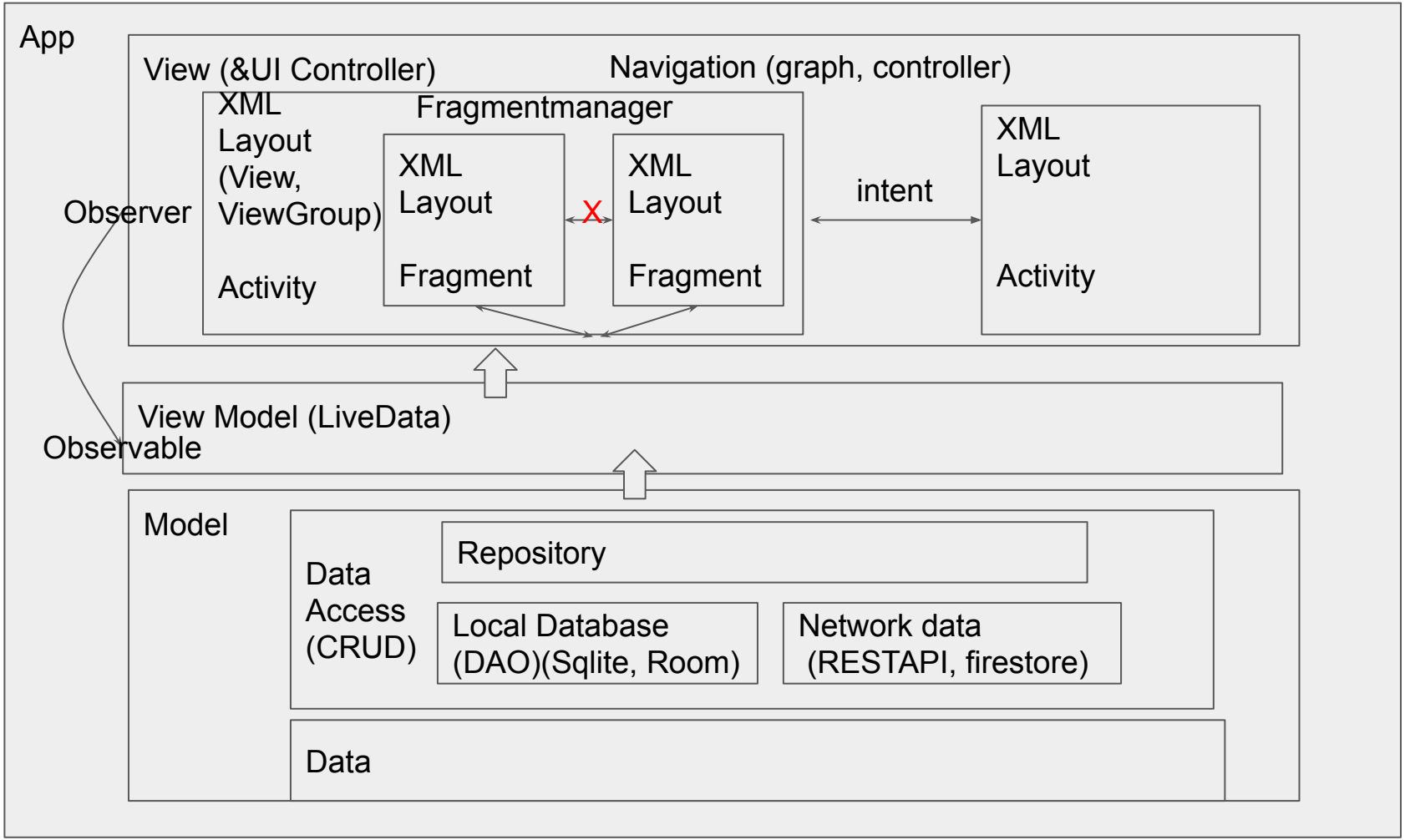
- ORM: Object Relational Mapping
- Android ORMs:
  - **Room**
  - OrmLite
  - ActiveAndroid
  - GreenDAO
  - SugarORM
  - DBFlow
  - http://www.tldevtech.com/best-android-orm-libraries-to-use/

Room

- Provide an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite. https://developer.android.com/jetpack/androidx/releases/room
  - Data entities that represent tables in your app's database.
    - By default, Room uses the class name as the database table name. Set the tableName property of the @Entity annotation if need a different name.
    - Room uses the field names as column names in the database by default. Add the @ColumnInfo annotation to the field and set the name property if need a different name.
  - Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.
    - Define a DAO interface with the @Dao annotation. Room will generate concrete implementation.
  - The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.
    - Define an abstract class with the @Database annotation. Include the abstract methods to return all related Dao objects. Room will generate concrete implementation.

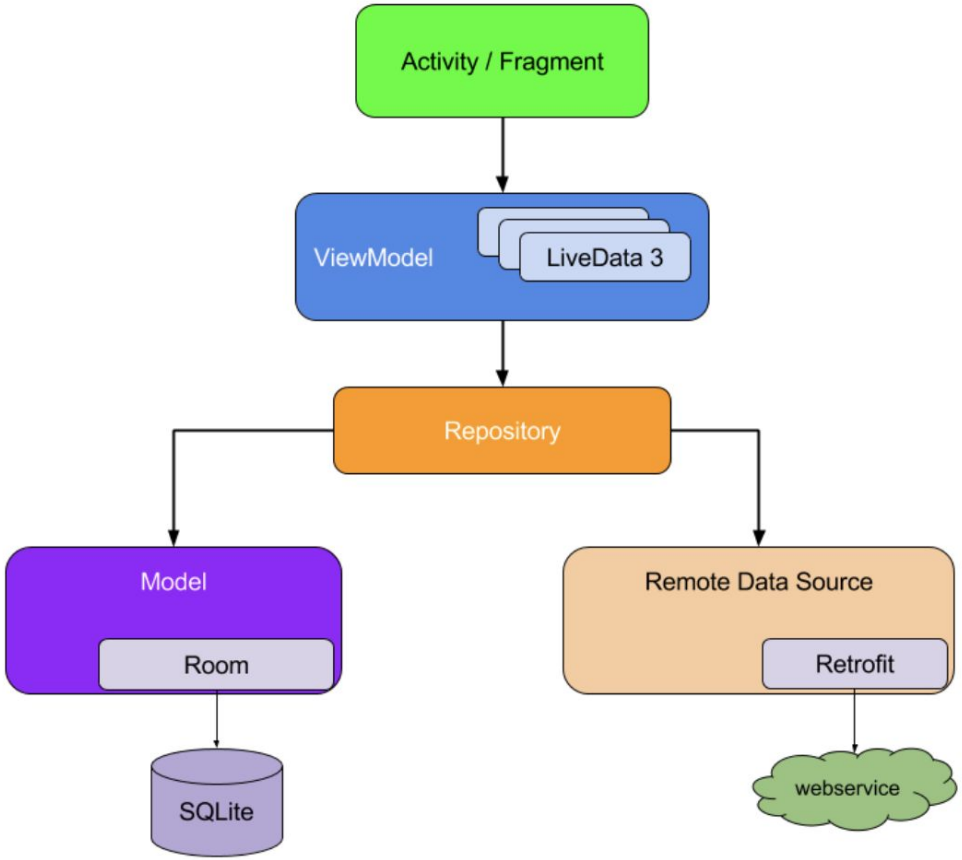
Create Room Database

- Use Room Database Builder
- Make it singular
- Define it in the application class


App
View (&UI Controller)
Navigation (graph, controller)
XML
Layout
(View,
ViewGroup)
Activity
Fragmentmanager
XML
Layout
Fragment
X
XML
Layout
Fragment
intent
XML
Layout
Activity
Observer
View Model (LiveData)
Observable
Model
Data
Access
(CRUD)
Repository
Local Database
(DAO)(Sqlite, Room)
Network data
(RESTAPI, firestore)
Data

Repository

- ViewModel delegates the data-fetching process to a *repository*.
- **Repository** handle data operati from different data source



Review

- What data can be stored for an application?
- Where are data stored in the device?
  - Mostly in the internal storage, under /data/data/<fullpackagename>/.
    - ./shared_pref: store shared preferences
    - ./lib
    - ./files: additional files
    - ./databases: store sqlite database files
  - Each folder has its own uid/gid. Cannot be accessed by other applications.
  - Files can be stored in the external storage, usually under /sdcard. Can be accessed by other applications
  - Also can be stored in the remote server or cloud (e.g. firebase cloud storage)

XML File For Shared Preferences

/data/data/com.google.android.apps.maps/shared_prefs/settings_preference.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <int name="settings_version" value="6" />
    <string name="cohort">748</string>
    <string name="distance_units"></string>
    <string
name="zb">110=ClMHdBarI6dLiMckaSa_lW49BwzClh-w-7mF4B84DmpGbekL-0Wy5_GGzQKQuZ08B
jcnv-NoaInLNYv6YXv_CbqLw89eVwtsS7zqhrm18Wjtwgmxqd36zQ0z3VbMSeDE</string>
    <string name="show_scalebar">fade</string>
    <boolean name="indoor_geofences_set_up" value="false" />
    <boolean name="shake_to_send_feedback_default_off" value="false" />
    <string name="cy">US</string>
</map>
```

APIs to Manage Shared Preferences

- https://developer.android.com/guide/topics/data/data-storage.html#pref
- https://developer.android.com/reference/android/content/SharedPreferences.html
- Classes: `SharedPreferences`
- Provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types.
- The primitive data types are booleans, floats, ints, longs, and strings.
- This data will persist across user sessions (even if your application is killed).
- Preference files are stored in ./shared_pref
- It cannot be accessed by other applications.

APIs to Manage Shared Preferences

- Methods:

```
//called from the activity class to get a Sharedpreferences Object
//if there are multiple preferences files, specify the preference file name to access with the first parameter
SharedPreferences getSharedPreferences (String name,int mode)
// only one preference file (the default file name is the activity name where this is called)
SharedPreferences getPreferences (int mode)
```

APIs to Manage Shared Preferences

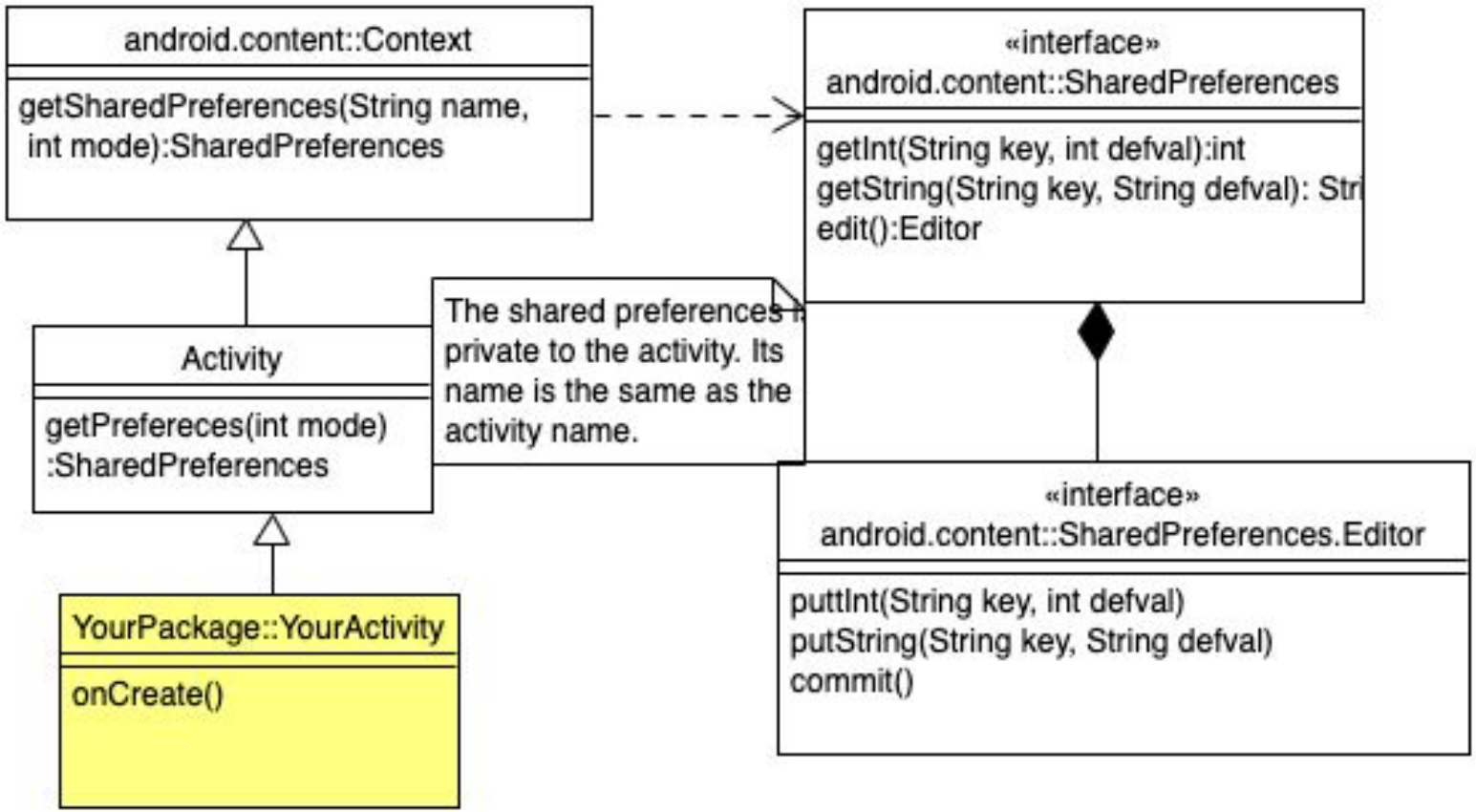
- Methods:

```
// Write the SharedPreferences Object through the editor
SharedPreferences.Editor edit();
SharedPreferences putBoolean(String key,boolean value);
SharedPreferences putString(String key, String value);
commit()
// read the SharedPreferences Object
boolean getBoolean(String key,boolean defValue);
String getString(String key, String defValue);
...
```

APIs to Manage Shared Preferences: Examples

```
Context context = getActivity();
SharedPreferences sharedPref = context.getSharedPreferences(
getString(R.string.preference_file_key), Context.MODE_PRIVATE);

SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);

SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt(getString(R.string.saved_high_score), newHighScore);
editor.commit();

SharedPreferences sharedPref =
getActivity().getPreferences(Context.MODE_PRIVATE);

int defaultValue =
getResources().getInteger(R.string.saved_high_score_default);
long highScore = sharedPref.getInt(getString(R.string.saved_high_score),
defaultValue);
```

Shared Preferences

- 

File Storage

- Additional data can be stored in files, particularly large data files, such as images, audio and video files.
- Files can be stored on the device internal flash memory. This is always available, as long as there is space.
- File can also be stored on the external storage unit such as SD cards, which may not be available.
- When using internal storage, files are inaccessible to other applications by default. They are usually stored in /data/data/<appname>/files/. When the application is uninstalled, the files are deleted too.
- When using external storage, it is out of control who will access files. Therefore, only use external storage when no access restriction is required.

File Storage

- Internal Storage:
  - Private, inaccessible to other apps
    - Before Android 7.0 (API level 24), internal files could be made accessible to other apps by means of relaxing file system permissions. This is no longer the case in Android 7.
    - If you wish to make the content of a private file accessible to other apps, your app may use the FileProvider
  - Under /data/data/<appname>/files
  - When the app is uninstalled, all data here will be deleted.
  - Mostly use Java File I/O APIs to manage these files.
  - A File object is suited to reading or writing large amounts of data in start-to-finish order without skipping around.

File Storage

- External Storage
  - Worldly accessible. Other applications can access.
  - To read/write from/to the external storage, you may need to request the READ_EXTERNAL_STORAGE/WRITE_EXTERNAL_STORAGE permission in your manifest file:

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

  - Public files (usually in /sdcard, the root directory on the sdcard)
    - The path is returned by the method getExternalStorageDirectory()
    - When the user uninstalls your app, these files should remain available to the user. For example, photos captured by your app or other downloaded files
  - Private files (usually /sdcard/Android/data/<appname>/files/, app specific folder)
    - The path is returned by the method getExternalFilesDir()
    - When the user uninstalls your app, the system removes your app's files from

File APIs to Use Internal Storage

- https://developer.android.com/guide/topics/data/data-storage.html#pref
- https://developer.android.com/reference/java/io/FileOutputStream.html
- https://developer.android.com/reference/java/io/FileInputStream.html
- Classes:
  - File, FileOutputStream, FileInputStream
- Methods:
  - OpenFileOutput(), write()
  - OpenFileInput(), read()
  - getFilesDir(): return an internal directory for your app
  - getCacheDir(): Return an internal directory for your app's temporary cache files
    - Be sure to delete each file once it is no longer needed and implement a reasonable size limit for the amount of memory you use at any given time, such as 1MB. If the system begins running low on storage, it may delete your cache files without warning

File

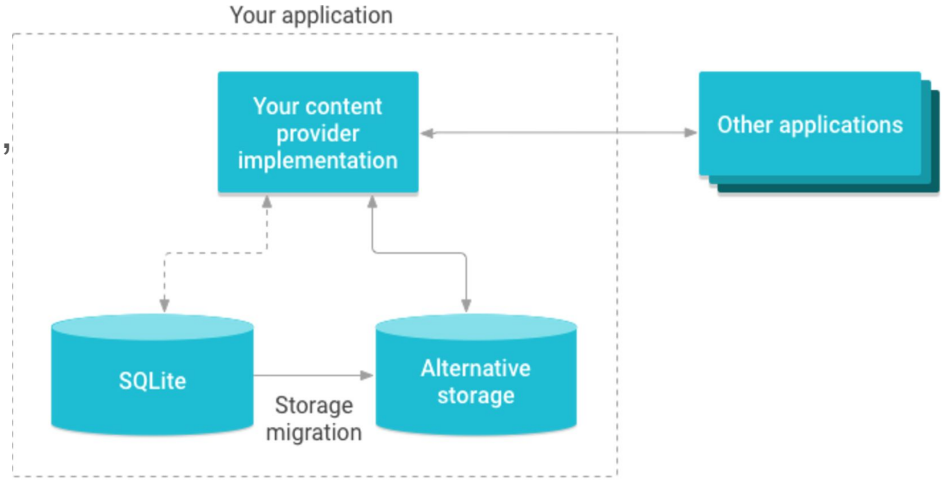
OpenFileOutput(): return a FileOutputStream for writing

OpenFileInput(): return a FileInputStream for reading

API	Path	deleted?	accessible
getFilesDir()	internal storage under ./files/ subfolder	deleted after uninstall	not accessible by other apps
getCacheDir()	internal storage under ./cache/ subfolder	deleted after uninstall	not accessible by other apps
getExternalFilesDir()	private external storage file folder	deleted after uninstall	accessible by other apps
getExtenralCacheDir()	private external storage cache folder	deleted after uninstall	accessible by other apps
getExternalPublicStorageDir()	public external storage	not deleted	accessible by other apps

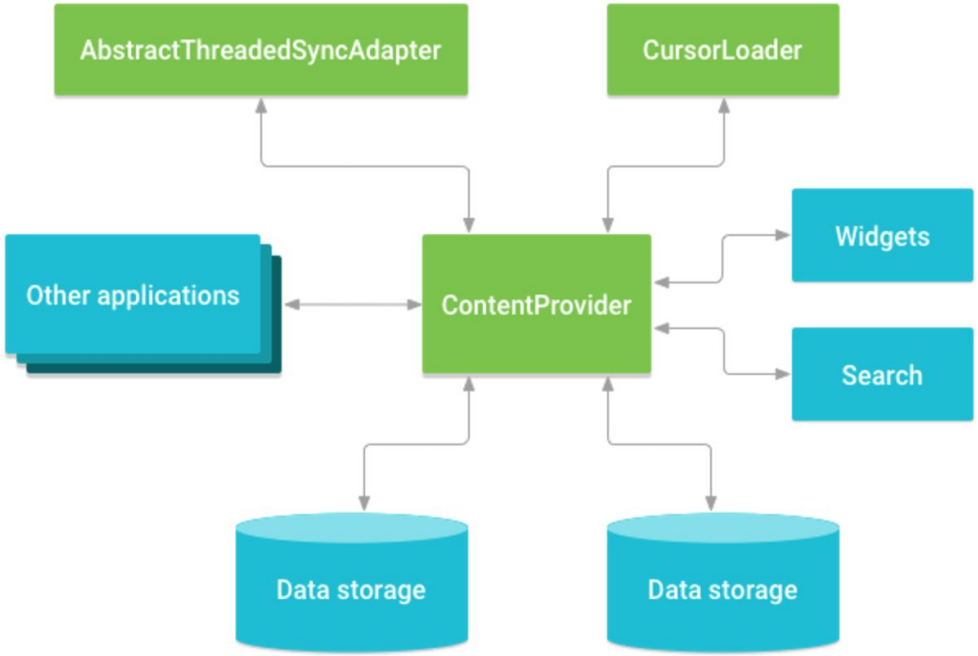
Content Provider

- Content providers are the standard interface that connects data in one process with code running in another process.
  - https://developer.android.com/guide/topics/providers/content-providers.html
  - They encapsulate the data, and provide mechanisms for defining data security.
  - Particular useful to provide secure data share between applications.
  - The underneath storage can be database or files.
  - Can handle common data types, Including videos, audios and Contact data.



Content Provider

- Sharing access to your application data with other applications
- Sending data to a widget
- Returning custom search suggestions for your application through the search framework usi SearchRecentSuggestionsProvid
- Synchronizing application data wi your server using an implementation of
- AbstractThreadedSyncAdapter
- Loading data in your UI using a CursorLoader



Content Provider APIs

- Package: android.Content
- Class: ContentProvider (abstract class)
- Methods:
  - onCreate() which is called to initialize the provider
  - query(Uri, String[], Bundle, CancellationSignal) which returns data to the caller
  - insert(Uri, ContentValues) which inserts new data into the content provider
  - update(Uri, ContentValues, String, String[]) which updates existing data in the content provider
  - delete(Uri, String, String[]) which deletes data from the content provider
  - getType(Uri) which returns the MIME type of data in the content provider onCreate()
- A content provider is created as a subclass of the ContentProvider class.

Database APIs:SQLiteDatabase

- Class: SQLiteDatabase (It is a final class, cannot be extended): Exposes methods to manage a SQLite database.
- Methods to perform CRUD operations:
  - long insert(String table, String nullColumnHack, ContentValues values);
  - Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
  - int update (String table, ContentValues values, String whereClause, String[] whereArgs)
  - int delete (String table, String whereClause, String[] whereArgs)
- The database engine already implements these methods. Your app simply calls these methods.
- Each SQLiteDatabase object is associated with a SQLite file.

Content URI

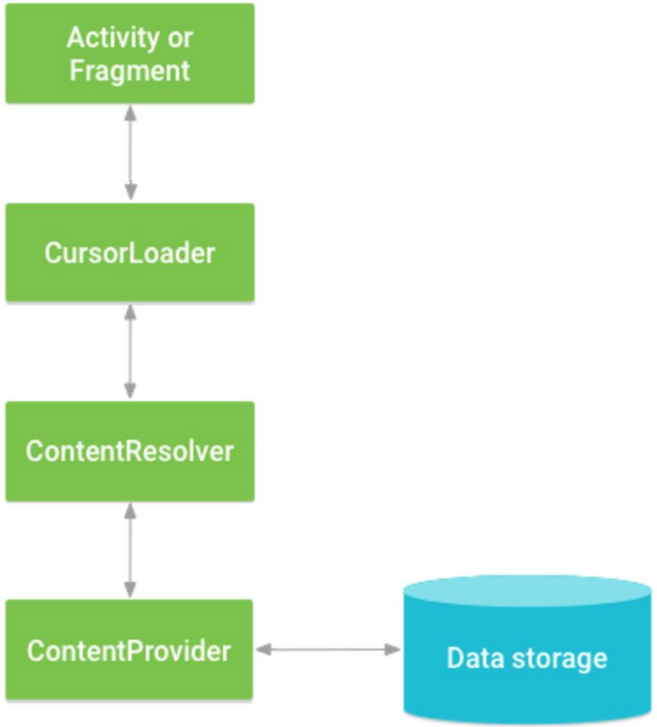
- An application may have more than one content provider, and a single content provider may provide access to multiple forms of content.
- A content URI is a URI that identifies data in a provider.
  - content://authority/path/id (content:// is the scheme)
- A Content URI includes the symbolic name of the entire provider (its authority) and a name that points to a table (a path).
  - content://user_dictionary/words
- Can also specify a single row to access. The row id is appended.
  - content://user_dictionary/words/3
- Can user UriMatcher to aid in matching URIs in content providers.
- https://developer.android.com/reference/android/content/ContentUris.html

UriMatch

- A Utility class to aid in matching URIs in content providers.
- Methods:
  - Public constructors UriMatcher(int code):
    Creates the root node of the URI tree.
  - void addURI(String authority, String path, int code):
    Add a URI to match, and the code to return when this URI is matched.
  - int match(Uri uri):
    Try to match against the path in a url, return the match code
- To use this class, first build up a tree of UriMatcher objects
- https://developer.android.com/reference/android/content/UriMatcher.html

ContentResolver

- A ContentResolver object is used to access the ContentProvider objects.
- Call **getContentResolver() (or contentResolver)** to obtain a reference to its content resolver from the context of the app.
- This ContentResolver object has methods that call identically-named methods in the provider object, which is specified using the provider's URI.
- The ContentResolver methods provide the basic "CRUD" functions of persistent storage.
- The content resolver and content provider objects together perform the requested task on behalf of the application



CRUD Operations

- Very similar to database APIs. However, it is called through getContentResolver().
- C (insert new data (a row) into a table at the given URI).
  - `final Uri insert(Uri url, ContentValues values)`
- R (query the given URI, return a Cursor over the result set)
  - `final Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder, CancellationSignal cancellationSignal)`
- U (update row(s) in a table at the given URI)
  - `final int update(Uri uri, ContentValues values, String where, String[] selectionArgs)`
  - If you want to clear the contents of a column, set the value to null
- D (delete row(s) specified by a content URI)
  - `int delete (Uri url, String where, String[] selectionArgs)`
    If the content provider supports transactions, the deletion will be atomic.

Declare The Content Provider in the Manifest File

- The content provider components are declared in the AndroidManifest.xml file using the <provider> element. Attributes include
  - android:authority: the full authority URI of the content provider,
  - android:name: the name of the class that implements the content provider, usually the same value as the authority.
  - android:permission (or android:readPermission, android:writePermission): Permissions that must be held by client applications to get access to the underlying data.
    - If no permissions are declared, the default behavior is for permission to be allowed for all applications in device with API 16 or under, and not allowed in API 17.
    - Permissions can be set to cover the entire content provider, or limited to specific tables and records using <path-permission> tag
  - https://developer.android.com/guide/topics/manifest/provider-element.html
  - https://developer.android.com/guide/topics/manifest/manifest-intro.html#sectperm

Request Access Permission in the Manifest File

- If the provider defines any permissions, the client applications need request permission in their Manifest File using <use-permission>
- If a provider's application doesn't specify any permissions, then other applications have no access to the provider's data. However, components in the provider's application always have full read and write access, regardless of the specified permissions.

Security Issue

- In the devices with API level 16 and lower, if a provider does not define any permission, then all applications can access it by default. If you want limit other applications to access it, always define the permissions (read and write permission).
- In devices with API level 17 and up, a new attribute is introduced:
  - android:exported: if set to true, the provider is available to other applications, if set to false, only available to the applications that have the same UID as the provider, not others.
  - The default value is "false" for devices running API level 17 and higher.

Firebase

- A platform developed by Google for creating mobile and web applications.
- Support the serverless architecture.
- Authentication
- Real-time Database
- FireStorage Database
- …
-

Retrofit

https://square.github.io/retrofit/

- Turn HTTP API into a Java/kotlin interface
  - HTTP verbs: get, post, put, delete, head ...
- First build the Retrofit, providing a base url. GsonConverterFactory can be used to convert between Json and Java objects.
- Define a service interface to specify paths, actions, and response types.
  - Use a Call interface, which is Retrofit's mechanism for executing network requests synchronously (via execute()) or asynchronously (via enqueue(Callback)).
- The Retrofit will generate an implementation of the service interface
- The client code can then use the methods defined in the service.
- The client code need to specify how to handle the http response.

Questions?

Review

- What types of data need to be stored in app?
  - Small amount of setting/configuration data as key-value pairs: xml files as shared preferences
  - Structured data - (SQLite) database
  - Unstructured data - general files
- Export to other applications - content providers

FileProvider

- A special Content Provider. Subclass from ContentProvider
- Define the resource file and add a file provider in the Manifest.xml

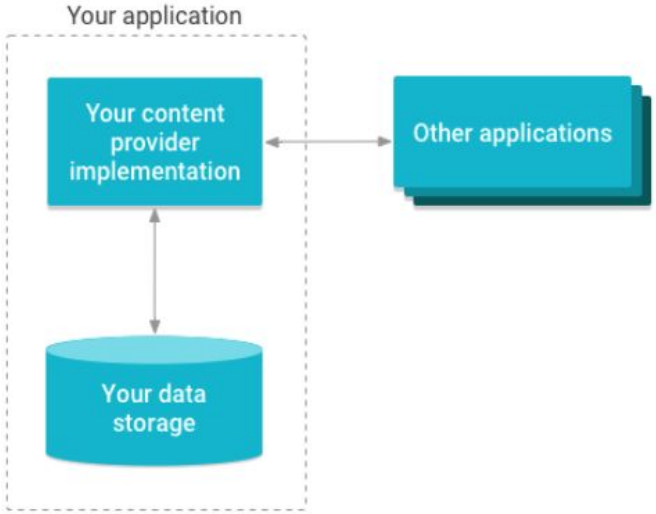
```
<provider
   android:name="androidx.core.content.FileProvider"
   android:authorities="com.android.testable.files"
   android:exported="false"
   android:grantUriPermissions="true">
   <meta-data
       android:name="android.support.FILE_PROVIDER_PATHS"
       android:resource="@xml/file_provider_paths" />
</provider>
```

- Define a class to access the exposed file path.

Content Provider

- Export Data to other applications
- Implement Content Provider
  - Create a UriMatcher and build up a tree of UriMatch objects to match URIs to the underneath storage (e.g. database tables, rows, or files)
  - Create a subclass of ContentProvider, implement its CURD methods using the underneath storage CRUD methods (e.g SQLiteDatabase CRUD methods)
  - Define the ContentProvider in the AndroidManifest.xml file
- Use Content Provider using getContentResolver() and provide URI when performing CRUD operations

Content Provider



- Export Data to other applications
- Implement Content Provider
  - Create a UriMatcher and build up a tree of UriMatch objects to match URIs to the underneath storage (e.g. database tables, rows, or files)
  - Create a subclass of ContentProvider, implement its CURD methods using the underneath storage CRUD methods (e.g SQLiteDatabase CRUD methods)
  - Define the ContentProvider in the AndroidManifest.xml file
- Use Content Provider using getContentResolver() and provide URI when performing CRUD operations

Examples

- To create a new file in one of these directories, you can use the File() constructor, passing the File provided by one of the previous methods that specifies your internal storage directory

```
DFile file = new File(context.getFilesDir(), filename);
```

OR

```
String filename = "myfile"; String string = "Hello world!";
FileOutputStream outputStream;
try {
     outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
     outputStream.write(string.getBytes());
     outputStream.close();
} catch (Exception e) {
     e.printStackTrace();
}
```

File APIs to use External Storage

- Same File APIs can be used for external storage
- However, the external storage may not always available—such as when the user has mounted the storage to a PC or has removed the SD card that provides the external storage—you should always verify that the volume is available before accessing it
- You can query the external storage state by calling getExternalStorageState()
    - If the returned state is equal to MEDIA_MOUNTED, then you can read and write your files

Query Free Space

- Call getFreeSpace() or getTotalSpace() to get current available space and the total space in the storage volume, respectively
- This can help find out whether sufficient space is available without causing an IOException and avoid filling the storage volume above a certain threshold
- However, the system does not guarantee that you can write as many bytes as are indicated by getFreeSpace().
  - If the number returned is a few MB more than the size of the data you want to save, or if the file system is less than 90% full, then it's probably safe to proceed
  - Otherwise, you probably shouldn't write to storage
- You should always delete files that you no longer need. Simply call delete() on the opened file object.

```
myFile.delete();
```