Lecture 7

Data Handling in Mobile Platforms

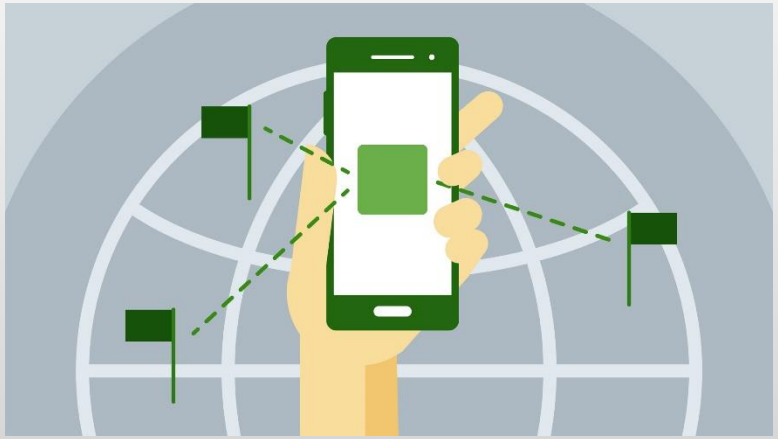
- Mobile Mindset
- Mobile Platforms and Application Development fundamentals
- Introduction to Android Operating System
- Android Interface Design Concepts
- Main Components of Android Application
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Kotlin Language to develop Android Mobile Apps
- Android Application Testing and security aspects

Learning Outcomes

- At the end of this Lecture, students should be able to
  - ✓ Identify the persistence techniques in Android Applications.
  - ✓ Understand the database handling of mobile technology using SQLite as the database tool.
  - ✓Develop mobile applications using SQLite as the database.

Persistence techniques in Android Applications

- Android provides several options to save your app data.
- Your solution depends on:
  - How much space your data requires
  - What kind of data you need to store
  - Data should be private to your app or accessible to the other apps



File storage options

1. **Internal file storage** - Store app-private files on the device file system.

2. **External file Storage** - Store files on the shared external file system. This is usually for shared user files, such as photos.

3. **Shared preferences** - Store private primitive data in key-value pairs.

4. **Database** - Store structured data in a private database.

Internal Storage

- By default, files saved to internal storage are private to your app.
- The system provides a private directory on the file system for each file.
- When the user uninstalls your app, the files saved on the internal storage are removed.

External Storage

- A storage space that users can mount to a computer as an external storage device.

- Physically removable.

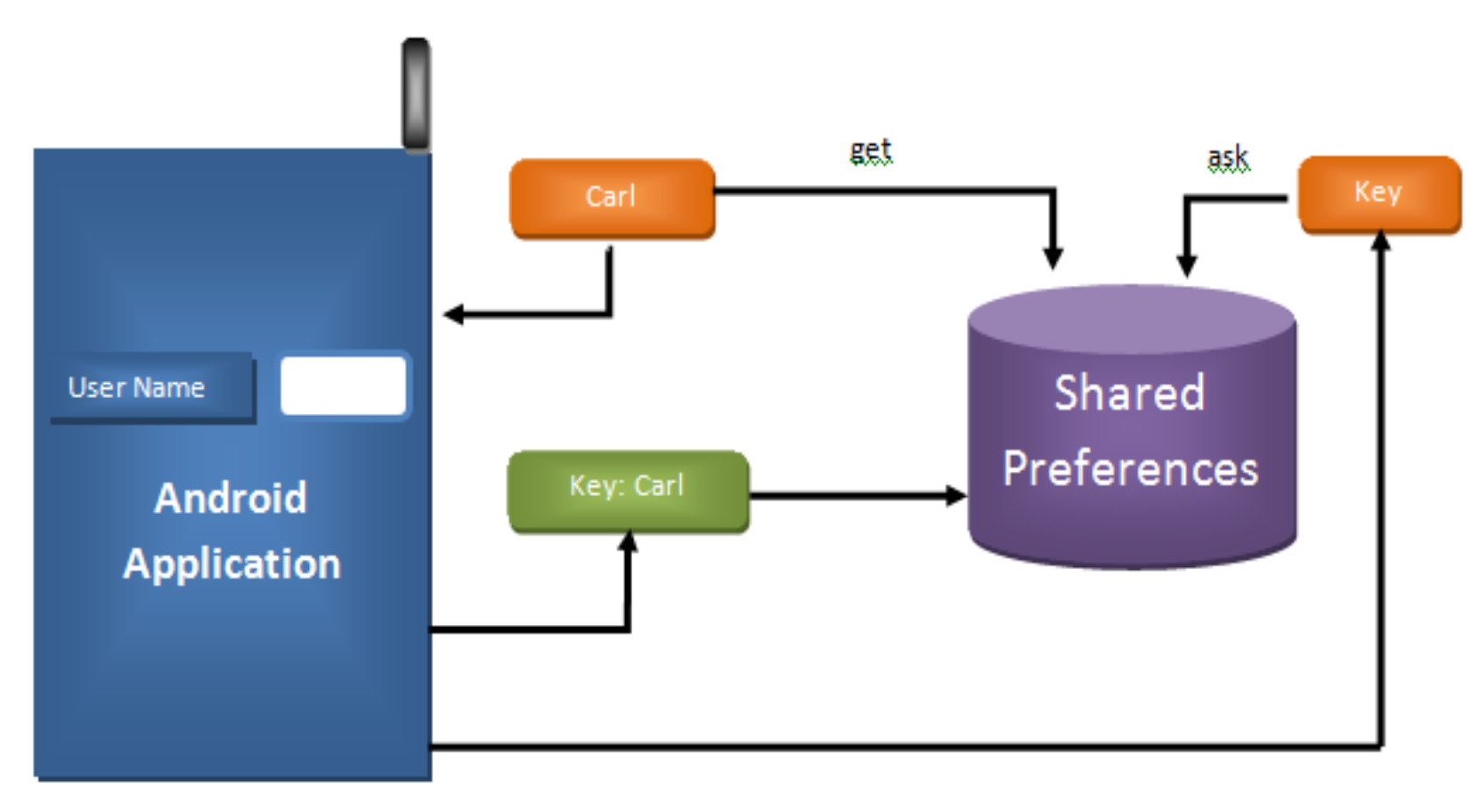
  Eg: SD card

- Can be modified by the user when they enable USB mass storage to transfer files on a computer.

Shared preferences

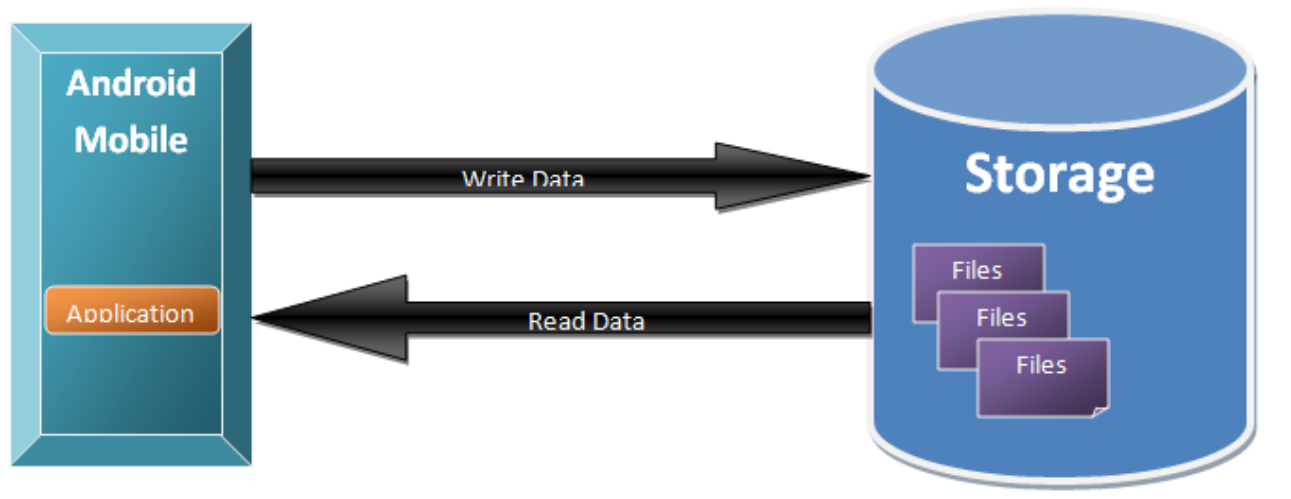
- If you don't need to store a lot of data and it doesn't require structure, you can use Shared preferences.
- The APIs allow you to read and write persistent key-value pairs of primitive data types such as Booleans, floats, ints, longs, and strings.
- The key-value pairs are written to XML files that persist across user sessions, even if your app is killed. You can manually specify a name for the file or use per-activity files to save your data.

Shared preferences



Databases

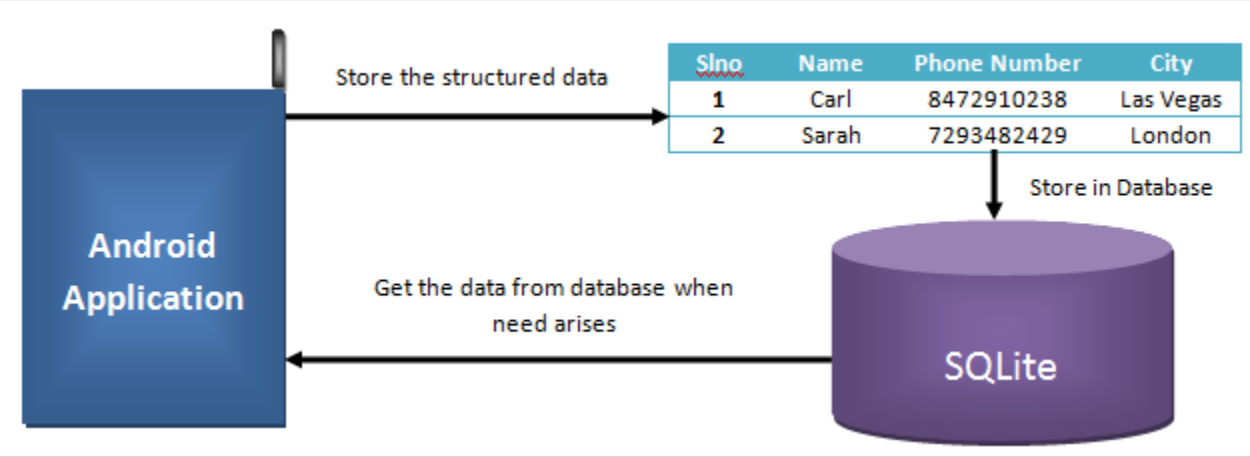
- Android provides full support for SQLite databases.
- Any database you create is accessible only by your app.



SQLite

- A SQL database engine which is developed using C that accesses its storage files directly.

- A software library that implements a self-contained, server less, zero-configuration, transactional SQL database engine.



Why SQLite?

- Serverless (does not require a separate server / ODBC or JDBC queries or system to operate). So. It's a local database.
- Zero-configuration (no setup or administration needed).
- Stored in a single cross-platform disk file.
- Very small and light-weight (250 KB- 400KB).
- Self-contained (no external dependencies and embeddable.
- Includes all basic functionalities.

Advantages of SQLite

- **Portable –** Uses only ANSI-standard C and VFS, file format is cross platform (little vs big endian, 32 vs 64 bit)
- **Reliable –** Has 100% test coverage, open source code and bug database, tranactions are ACID even if power fails
- **Small –** 300kb library, runs in 16kb stack and 100kb heap

Disadvantages of SQLite

- **High concurrency –** Reader / writer loacks on the entire file
- **Huge database –** Db file can’t exceed file system limit or 2TB
- **Access control –** there isn’t any

Create Database

- The SQLiteDatabse and SQLiteOpenHelper libraries are required.
- They include necessary information for the Db handling.

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

- Create a java class with above two imports and extend the SQLiteOpenHelper class. Implement thenecessary constructors and required onCreate() and onUpgrade() methods.

Create Columns for a Table

- Create a final class to define all tables for the database.
- Create an inner class to define columns.

```
import android.provider.BaseColumns;

public final class UsersMaster {
    private UsersMaster() {}

    /* Inner class that defines the table contents */
    public static class Users implements BaseColumns {
        public static final String TABLE_NAME = "users";
        public static final String COLUMN_NAME_USERNAME = "username";
        public static final String COLUMN_NAME_PASSWORD = "password";
    }
}
```

Create Database

```
public class DBHelper extends SQLiteOpenHelper {

    public static final String DATABASE_NAME = "UserInfo.db";

    public DBHelper(Context context) { super(context, DATABASE_NAME, factory: null, version: 1); }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String SQL_CREATE_ENTRIES =
                "CREATE TABLE " + UsersMaster.Users.TABLE_NAME + " (" +
                        UsersMaster.Users._ID + " INTEGER PRIMARY KEY," +
                        UsersMaster.Users.COLUMN_NAME_USERNAME + " TEXT," +
                        UsersMaster.Users.COLUMN_NAME_PASSWORD + " TEXT)";
        // Use the details from the UsersMaster and Users classes we created. Specify the primary key from the BaseColumns

        db.execSQL(SQL_CREATE_ENTRIES);  // This will execute the contents of SQL_CREATE_ENTRIES
    }

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

    }
```

Insert Data

```
public void addInfo(String userName, String password) {
    // Gets the data repository in write mode
    SQLiteDatabase db = getWritableDatabase();

    // Create a new map of values, where column names the keys
    ContentValues values = new ContentValues();
    values.put(Users.COLUMN_NAME_USERNAME, userName);
    values.put(Users.COLUMN_NAME_PASSWORD, password);

    // Insert the new row, returning the primary key value of the new row
    long newRowId = db.insert(Users.TABLE_NAME, nullColumnHack: null, values);
}
```

Read data from the Database

```
public List readAllInfo()
{
    SQLiteDatabase db = getReadableDatabase();

    // define a projection that specifieswhich columns from the database
    // you will actually use after this query
    String[] projection = {
            Users._ID,
            Users.COLUMN_NAME_USERNAME,
            Users.COLUMN_NAME_PASSWORD
    };
    //Filter results WHERE "userName" = 'SLIIT USER'
   // String selection = Users.COLUMN_NAME_USERNAME + " = ?";
    //String[] selectionArgs = {""};

    // How you want the results sorted in the resulting cursor
    String sortOrder = Users.COLUMN_NAME_USERNAME + " DESC";

    Cursor cursor = db.query(
            Users.TABLE_NAME,             // the table to query
            projection,                   // the columns to return
            selection: null,              // the columns for the WHERE clause
            selectionArgs: null,          // the values for the WHERE clause
            groupBy: null,               // don't group the rows
            having: null,                // don't filter by row groups
            sortOrder                    // the sort order
    );
```

Read data from the Database cont..

```
    List userNames = new ArrayList<>();
    List passwords = new ArrayList<>();

    while(cursor.moveToNext()){
        String username = cursor.getString( cursor.getColumnIndexOrThrow(Users.COLUMN_NAME_USERNAME));
        String password = cursor.getString( cursor.getColumnIndexOrThrow(Users.COLUMN_NAME_PASSWORD));
        userNames.add(username);
        passwords.add(password);
    }
    cursor.close();
    return userNames;
}
```

Update a record

```
public void updateInfo(String userName, String password) {
    SQLiteDatabase db = getReadableDatabase();

    //New value for one column
    ContentValues values = new ContentValues();
    values.put(Users.COLUMN_NAME_PASSWORD, password);

    //Which row to update, based on the title
    String selection = Users.COLUMN_NAME_USERNAME + " LIKE ?";
    String[] selectionArgs = {userName};

    int count = db.update(
            Users.TABLE_NAME,
            values,
            selection,
            selectionArgs
    );
}
```

Delete a record

```
//This will delete a particular user from the table
public void deleteInfo(String userName){
    SQLiteDatabase db = getReadableDatabase();
    //Define 'where' part of query
    String selection = Users.COLUMN_NAME_USERNAME + " LIKE ?";
    //Specify arguments n placeholder order
    String[] selectionArgs = { userName };
    //Issue SQL statement
    db.delete(Users.TABLE_NAME, selection, selectionArgs);

}
```