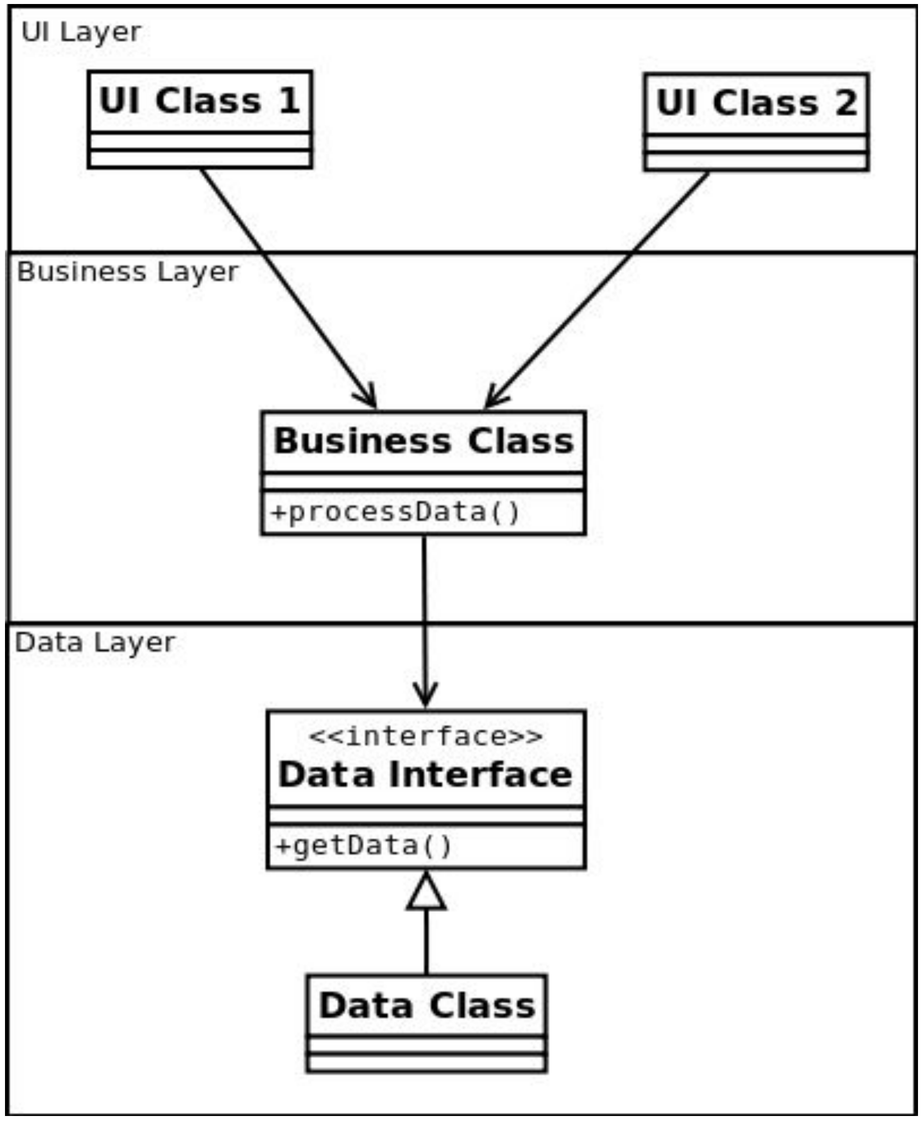
Android Data Storage

Key Considerations

- Storage Options
  - Local
    - File
    - Database
  - Remote
    - Cloud

- Persistence strategies
  - Layered Architecture with Data Access Layer
  - Other strategies (e.g. Serialization, ORM)

Layered Architecture



Data Layer

```
public interface INoteDAO {

     public void save(Hashtable<String,String> attributes);
     public void save(ArrayList<Hashtable<String,String>>
objects);
     public ArrayList<Hashtable<String,String>> load();
     public Hashtable<String,String> load(String id);
}
```

```
public class NoteDAO implements INoteDAO {


    @Override
    public void save(Hashtable<String, String> attributes) {
        // save a single object
    }

    @Override
    public void save(ArrayList<Hashtable<String, String>>
objects) {
        // save multiple objects
    }

    @Override
    public ArrayList<Hashtable<String, String>> load() {
        // load all objects

    @Override
    public Hashtable<String, String> load(String id) {
        // load a single object based upon id
    }
```

Business Layer

```
public class Note {

    private INoteDAO dao = null;
    private String id;
   // declare other attributes

     public void save(){
        if (dao != null){
            Hashtable<String,String> data =new Hashtable<String, String>();
            data.put("id",id);
            // similarly put other attributes
            dao.save(data);
        }
     }

     public void load(Hashtable<String,String> data){
         id = data.get("id");
         // similarly load other attributes after necessary string conversion
     }

     public static ArrayList<Note> load(INoteDAO dao){
        ArrayList<Note> notes = new ArrayList<Note>();
        if(dao != null){

            ArrayList<Hashtable<String,String>> objects = dao.load();
            for(Hashtable<String,String> obj : objects){
                Note note = new Note(dao);
                note.load(obj);
                notes.add(note);
            }
        }
        return notes;
     }
}
```

File

- Flexibility in terms of storage format
  - Customized storage / retrieval of data
  - Popular formats
    - XML
    - JSON
    - CSV, etc.
- Stored on a permanent medium
  - Internal
    - Always available but space may be limited
    - Better security through access restrictions
  - External
    - Not always available (if unmounted)
    - World-readable – suitable for sharing
- I/O
  - Stream-based (Standard Java APIs)

Data Layer

```
public class NoteFileDAO implements INoteDAO {
    File file;
    public NoteFileDAO(File f){
        file = f;
    }

    public void save(Hashtable<String, String> attributes) {
        try{
            BufferedWriter writer = new BufferedWriter(
                                                   new FileWriter(file,true));
            writer.append("[note]");
            writer.newLine();

            Enumeration<String> keys = attributes.keys();
            while (keys.hasMoreElements()){
                String key = keys.nextElement();
                writer.append(key + ":" + attributes.get(key));
                writer.newLine();
            }

            writer.close();

        }catch (Exception ex){

        }
    }

    public void save(ArrayList<Hashtable<String, String>> objects) {
        for(Hashtable<String,String> obj : objects){
            save(obj);
        }
    }
    // continued...
```

Data Layer

```
    public ArrayList<Hashtable<String, String>> load() {
        ArrayList<Hashtable<String,String>> objects =
                                      new ArrayList<Hashtable<String, String>>();
        try {
            Hashtable<String,String> obj =null;
            String line;
            BufferedReader reader =new BufferedReader(new FileReader(file));
            while ((line = reader.readLine()) != null){

                if(line.equals("[note]")){
                    obj = new Hashtable<String, String>();
                    objects.add(obj);
                }
                else {
                    String key = line.substring(0, line.indexOf(":"));
                    String value = line.substring(line.indexOf(":") + 1);
                    obj.put(key, value);
                }

            }

        } catch (Exception ex){
            int i=0;
        }

        return objects;
    }

    public Hashtable<String, String> load(String id) {
        // find the id and load the object in similar fashion
    }
}
```

UI Layer

```
public class NotesActivity extends BaseActivity
{
    ArrayList<Note> notes;
    NoteFileDAO dao;


     public void onCreate(Bundle savedInstanceState)
     {
          super.onCreate(savedInstanceState);
          // other create related operations
          dao = new NoteFileDAO(new File(getFilesDir(),"notes"));
     }

    public void onPause()
    {
        super.onPause();
         for(Note note : notes){
            note.save();
        }
    }

    public void onResume(){
        super.onResume();

        notes = Note.load(dao);
    }

    // other UI operations

}
```

Database

- Sqlite
  - Popular embedded database
  - Single-file based, efficient, structured storage
  - Cross-platform
  - Server-less, zero configuration
  - Suitable as application file format
  - Not very suitable for highly concurrent client-server architectures
- Android Sqlite support
  - SQLiteDatabase and SQLiteOpenHelper
  - ContentValues
  - Cursor

Data Layer

```
public class NotesDbHelper extends SQLiteOpenHelper{

  public static final int DATABASE_VERSION = 1;
  public static final String DATABASE_NAME = "Notes.db";

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

  public void onCreate(SQLiteDatabase db){
    String sql = "CREATE TABLE Notes (Id TEXT PRIMARY KEY, " +
                               "Title TEXT," +
                               "Content TEXT," +
                               "Important INTEGER," +
                               "CreationDateTime TEXT)";
    db.execSQL(sql);
  }

  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS Notes");
    onCreate(db);
  }

  public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    onUpgrade(db,oldVersion,newVersion);
  }

}
```

Data Layer

```
public class NoteDbDAO implements INoteDAO {
    private Context context;

    public NotesDbDAO(Context ctx){
        context = ctx;
    }

    @Override
    public void save(Hashtable<String, String> attributes) {
        NotesDbHelper dbHelper = new NotesDbHelper(context);
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        ContentValues content = new ContentValues();
        Enumeration<String> keys = attributes.keys();
        while (keys.hasMoreElements()){
            String key = keys.nextElement();
            content.put(key,attributes.get(key));
        }

        db.insert("Notes",null,content);
    }


    public void save(ArrayList<Hashtable<String, String>> objects) {
        for(Hashtable<String,String> obj : objects){
            save(obj);
        }
    }

    // continued...
```

Data Layer

```
    public ArrayList<Hashtable<String, String>> load() {
        NotesDbHelper dbHelper = new NotesDbHelper(context);
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        String query = "SELECT * FROM Notes";
        Cursor cursor = db.rawQuery(query,null);

        ArrayList<Hashtable<String,String>> objects = new
                                ArrayList<Hashtable<String, String>>();

         while(cursor.moveToNext()){
            Hashtable<String,String> obj = new Hashtable<String, String>();
            String [] columns = cursor.getColumnNames();
            for(String col : columns){
             obj.put(col.toLowerCase(),
                      cursor.getString(cursor.getColumnIndex(col)));

             }

            objects.add(obj);
        }

        return objects;
    }


    public Hashtable<String, String> load(String id) {
        // find the id and load the object in similar fashion
    }
}
```