マルチ OS エンジンを使用した固定記憶域の操作 (テクノロジー・プレビュー) - パート 1

この記事は、インテル® デベロッパー・ゾーンに公開されている「Working with persistent storage using Multi-OS Engine (Technology Preview)- Part 1」の日本語参考訳です。

このチュートリアルは 2 つのパートで構成されており、インテルのマルチ OS エンジンを使用して Java* で iOS* の固定記憶域を操作する方法を示します。

パート 1 では、iOS* と Android* の固定記憶域にファイルを作成し、書き込む方法を説明します。ノートを作成し、最新のノートがテーブルビューのトップに表示される、単純なノート作成アプリケーションを作成します。

このチュートリアルでは、マルチ OS エンジンを使用して iOS* および Android* アプリケーションをビルドする入門ガイドを読了されていることを前提としています。まだお読みになられていない場合は、先にこちら (英語) をお読みください。

サンプル・アプリケーションのプロジェクト一式はこちら (英語) からダウンロードできます。

共通コード:

最初に、iOS* と Android* に共通の Java* クラスを格納する、共通のライブラリー・フォルダーを作成します。そして、ファイルを作成したり、開いたりといった基本的なファイルの I/O 操作を実行するクラスを作成します。

```
package com.example;

import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;


public class Data {

    private HashMap<String,String> allNotes;
    private String key;

    private String baseDir;
    String fileName = "note_taking.txt";
```

```
String filePath;


public Data(String dir){
    baseDir = dir;
    filePath = baseDir + File.separator + fileName;
}

public HashMap<String, String> getAllNotes(){

    File f = new File(filePath);
        try {
            if(!f.exists()) {
                f.createNewFile();
            }
            if(allNotes==null) {
                try{
                    InputStream fis = new FileInputStream(f);
                    ObjectInputStream ois = new ObjectInputStream(fis);
                    allNotes = (HashMap<String, String>)
                    ois.readObject();
                    fis.close();
                    ois.close();

                    }catch (EOFException e){
                    allNotes = new HashMap<String, String>();
                }
            }
        } catch (IOException ioe) {
            ioe.printStackTrace();
        } catch (ClassNotFoundException c) {
            System.out.println("Class not found");
            c.printStackTrace();
        }
    return allNotes;
}

public void setCurrentKey(String key){
    this.key = key;
}

public String getCurrentKey(){
    return key;
}

public void setNoteForCurrentKey(String note){
    allNotes.put(key,note);
}

public String getNoteForCurrentKey(){return allNotes.get(key);}

public void setNoteForKey(String k,String note){
    allNotes.put(k,note);
}

public String getNoteForKey(String k){return allNotes.get(k);}

public void removeNoteForKey(String k){
    allNotes.remove(k);
}
```

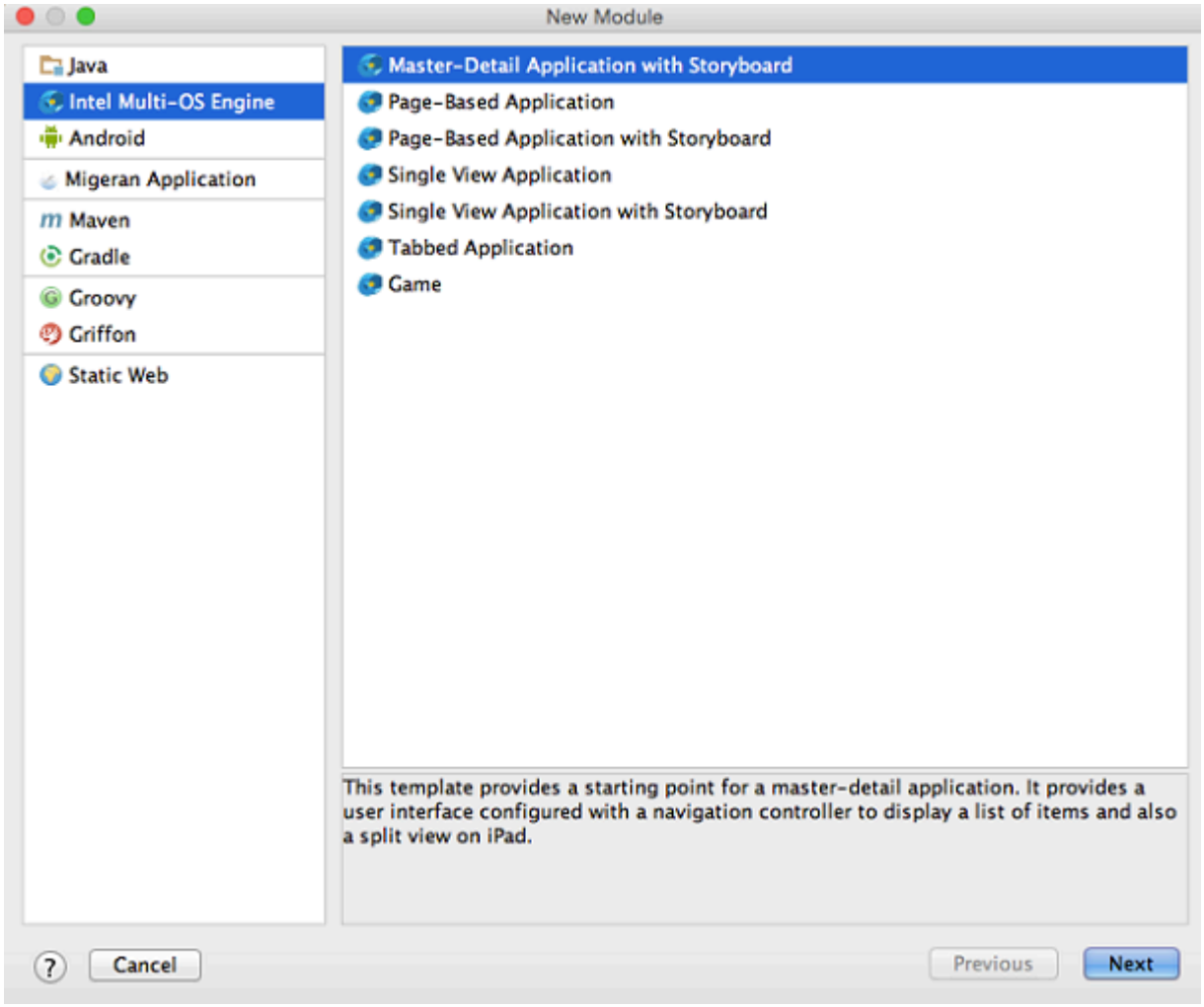
```
    public void saveFile() {
        try {
            File f = new File(filePath );
            if(!f.exists()) {
                f.createNewFile();
            }try {
                FileOutputStream fos = new FileOutputStream(f);
                ObjectOutputStream oos = new ObjectOutputStream(fos);
                oos.writeObject(allNotes);
                fos.close();
                oos.close();

            }catch (Exception e){
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public ArrayList<String> getAllKeys(){
        ArrayList<String> keys = new ArrayList<String>();
        keys.addAll(getAllNotes().keySet());
        Collections.sort(keys,Collections.reverseOrder());
        return keys;
    }

}
```

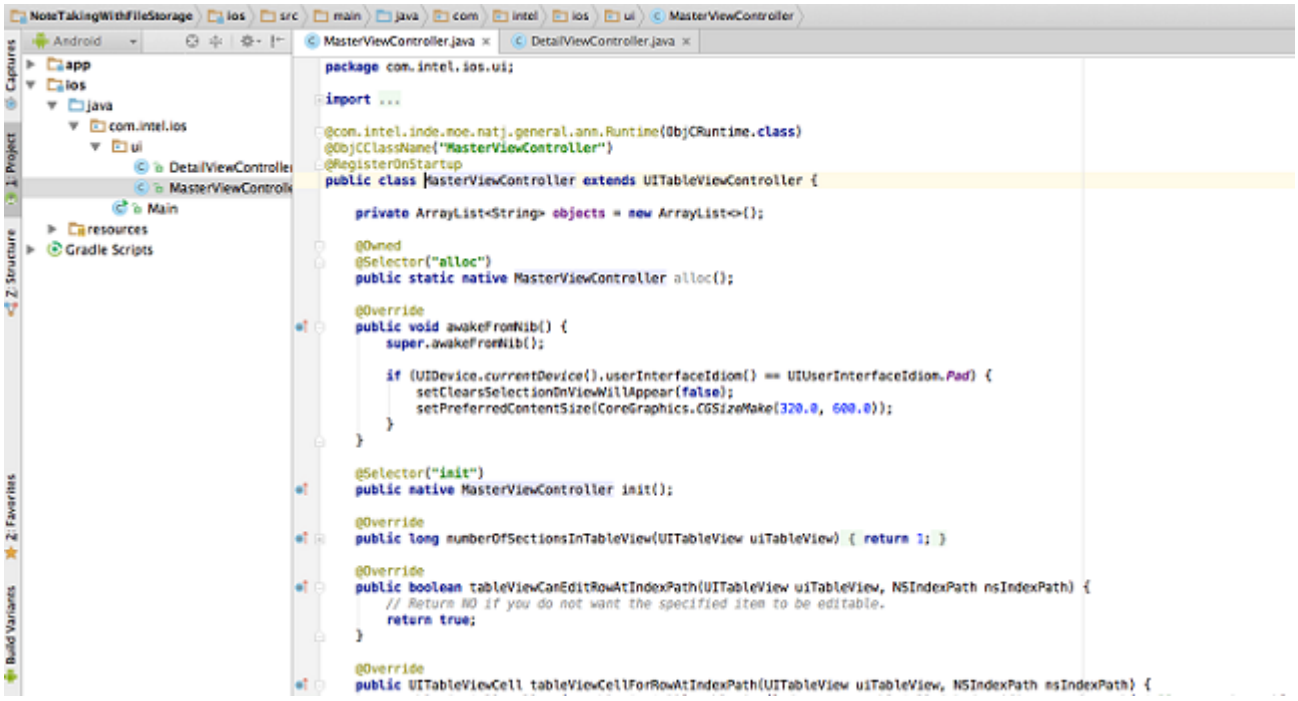
このアプリケーションでは、唯一プラットフォーム固有の部分であるディレクトリー・パスをコンストラクターに渡します。

プラットフォーム固有コード:

iOS プロジェクトのビルド:

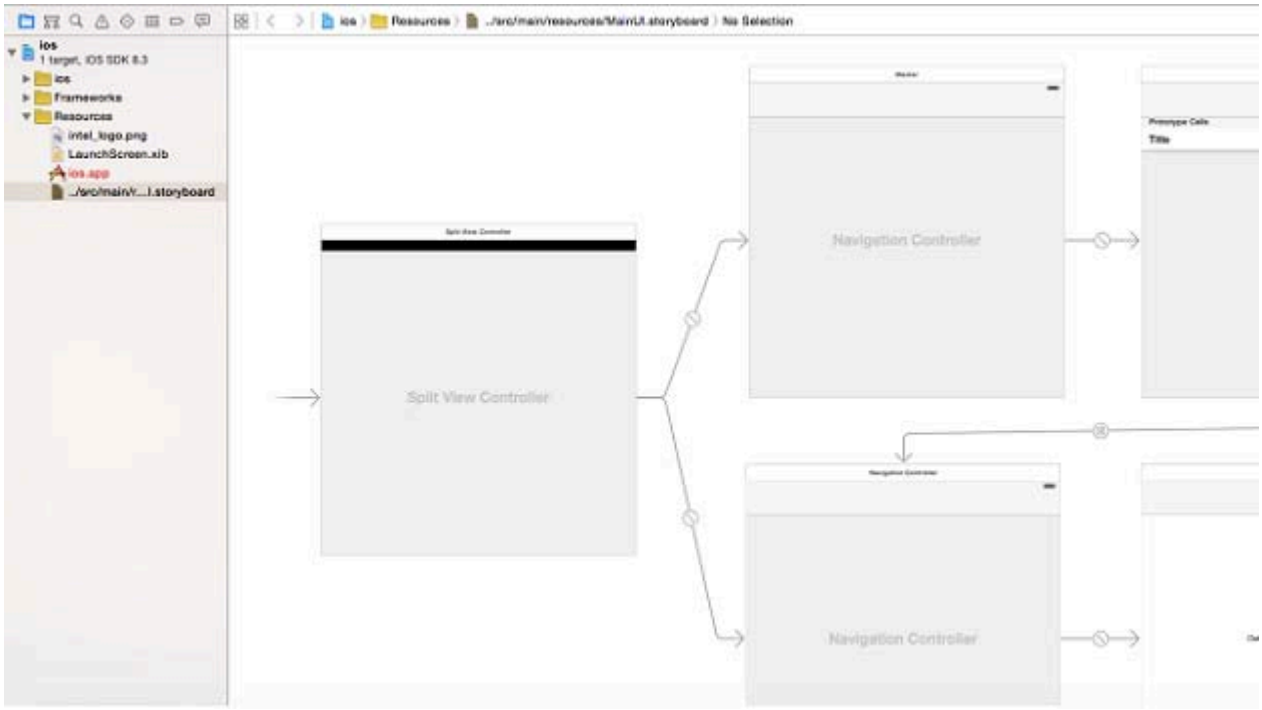
このサンプル・アプリケーションでは、ベース・テンプレートとして 'Master - Detail application with storyboard' を使用します。



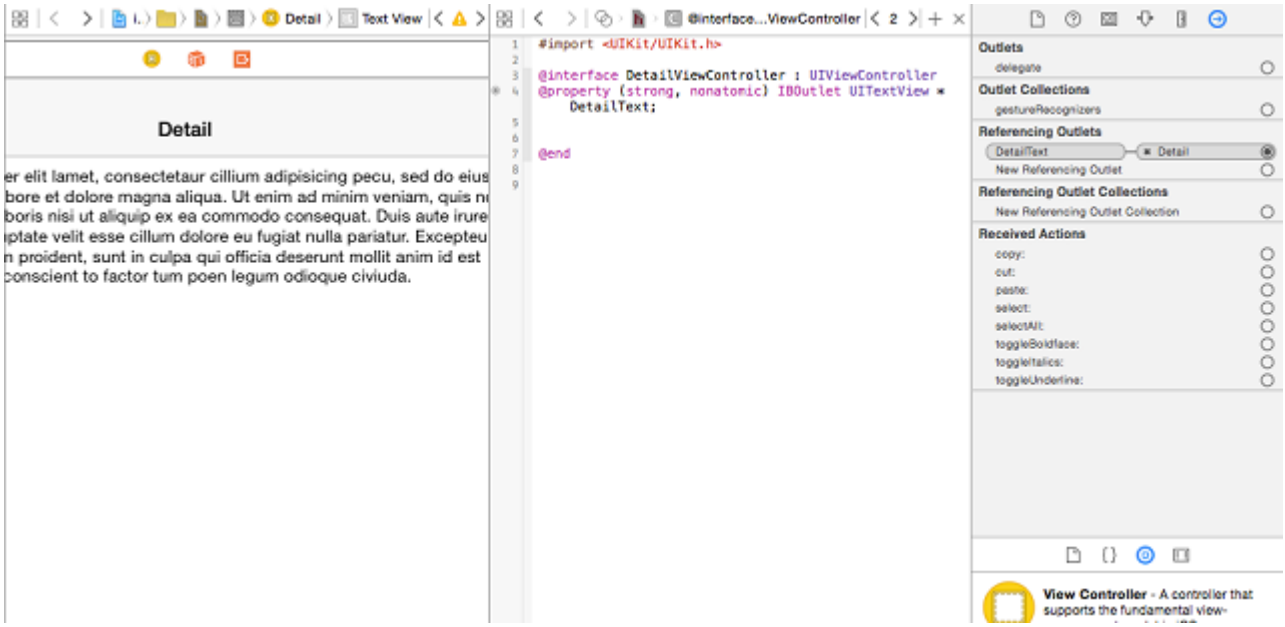


テンプレートには、必要なメソッドとコードがすでに設定されています。

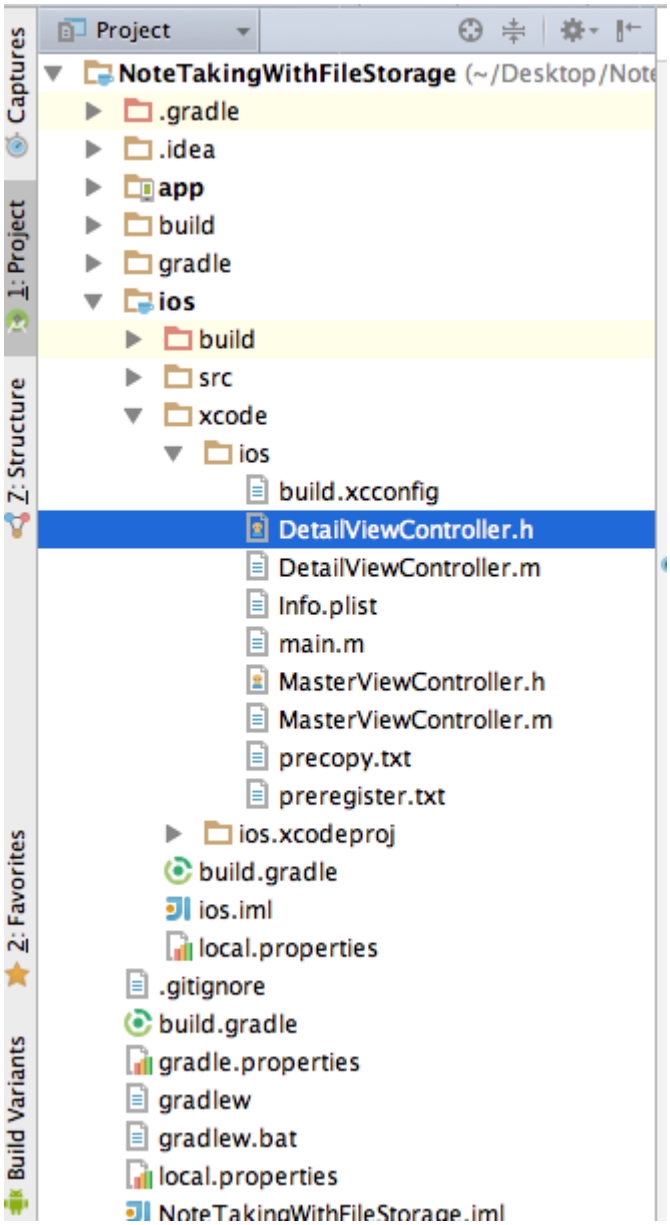
Xcode* でプロジェクトを開くと、対応する MasterViewController.h/.m ファイルおよび DetailViewController.h/.m ファイルが表示されます。また、Storyboard は標準ビュー・オブジェクトで構成されています。



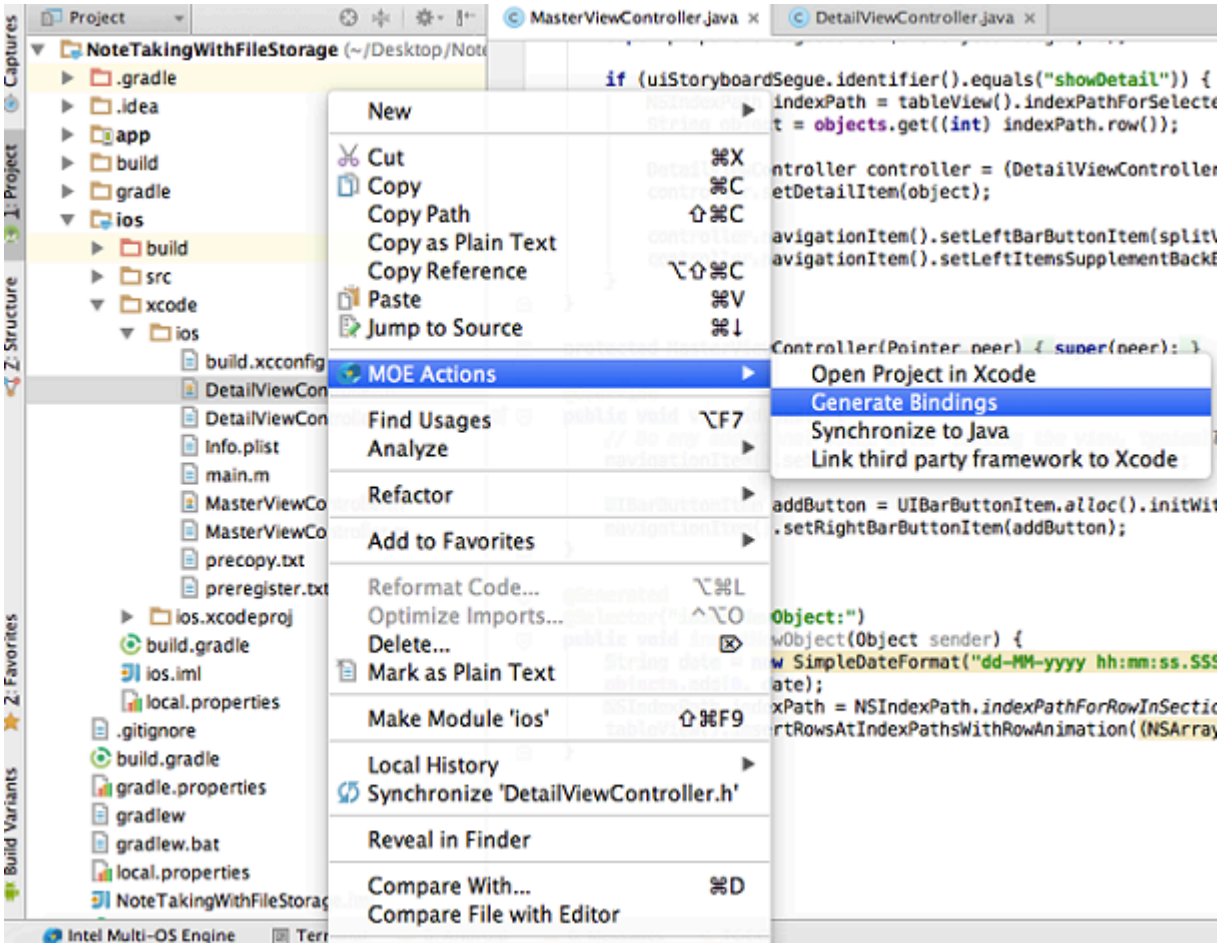
DetailView にオブジェクト・ライブラリーからテキスト・ビュー・オブジェクトを追加し、対応する DetailViewController.h ファイルに参照アウトレットを作成します。



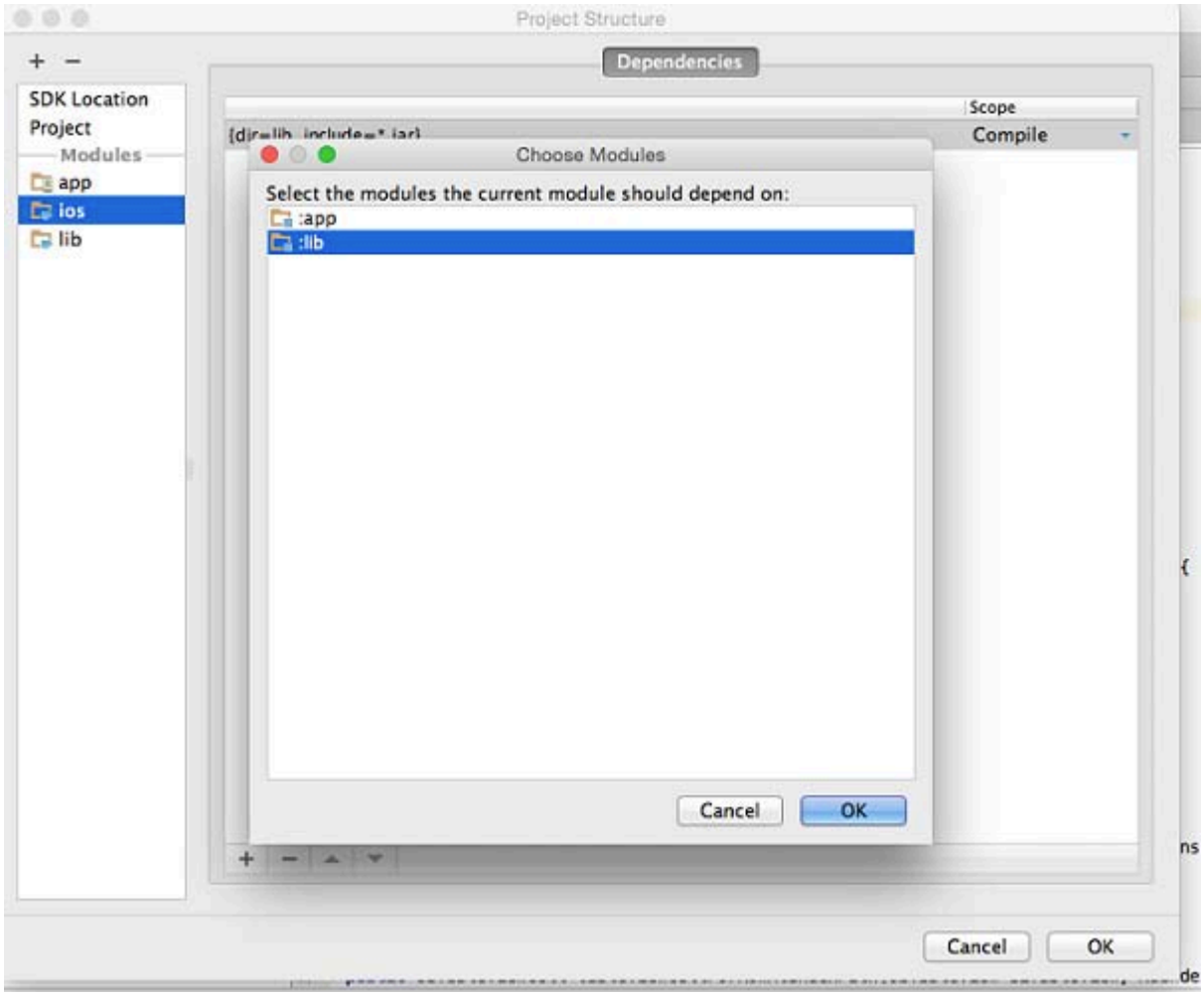
次に、プロジェクトの Android* Studio 側で iOS* モジュールの Xcode* フォルダー以下にある DetailViewController.h ファイルを見つけます。



[MOE Actions] > [Generate Bindings] メニューを選択して、バインディングを生成します。DetailView に追加したテキスト・ビュー・オブジェクトへの参照が生成されます。



依存項目として、共通モジュールを iOS* モジュールと Android* モジュールに追加します。



Android* プロジェクトのビルド:

Android* では、単純な Blank Activity アプリケーションを開始し、UI をビルドして、Activity と Adapter ファイルを追加します。このチュートリアルは、iOS* でアプリケーションをビルドする方法を示すことを目的としているため、ここでは Android* プロジェクトのビルドについては省略します。

ファイル・ディレクトリーの処理 (iOS*):

MasterViewController.java ファイルにコードを追加します。ここで重要なのは、iOS* の filestorage ディレクトリーのパスの取得方法を理解することです。iOS* では、ファイルは Documents ディレクトリーに格納され、そのパスは次のコードによって取得できます。

```
public String applicationDocumentsDirectory() {
    NSArray urls = NSFileManager.defaultManager().URLsForDirectoryInDomains(
        NSSearchPathDirectory.DocumentDirectory,
NSSearchPathDomainMask.UserDomainMask);
    NSURL url = (NSURL)urls.lastObject();
    String path = url.fileSystemRepresentation();
    return path;
}
```

このパスが、前述の共通コードで作成した Data クラスのコンストラクターに渡されます。

```
@Override
    @Selector("viewDidLoad")
    public void viewDidLoad() {
        // ビューをロード (通常は nib から) した後、追加のセットアップを行う
        navigationItem().setLeftBarButtonItem(editButtonItem());

        data = new Data(this.applicationDocumentsDirectory());
}
```

ファイル・ディレクトリーの処理 (Android*):

Android* 側では、取得済みのベース・ディレクトリーの文字列を、onCreate メソッドの Data クラスに渡します。

```
String baseDir =
android.os.Environment.getExternalStorageDirectory().getAbsolutePath();
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    data = new Data(baseDir);
    objects = new ArrayList<String>();
    makeObjects();
    ListView list = (ListView) findViewById(android.R.id.list);

    listAdapter = new ListAdapter(this,objects,data);
    list.setAdapter(listAdapter);
    list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
            data.setCurrentKey(objects.get(position));
            Intent intent = new Intent(MainActivity.this, EditorActivity.class);

           // Uri uri = Uri.parse(NotesProvider.CONTENT_URI + "/" + id);
            intent.putExtra(DEFAULT_TEXT,
data.getAllNotes().get(data.getCurrentKey()));
            startActivityForResult(intent, EDITOR_REQUEST_CODE);
        }
    });
}
```

テーブルへのノートの追加処理 (iOS*):

tableView を生成に必要な ArrayList オブジェクト変数が、テンプレートによってすでに定義されています。Data クラスを使用してファイルからすべてのノートを取得し、ArrayList に格納します。iOS* 側で、次のメソッドを使用してテーブルを生成します。

```
@Override
public UITableViewCell tableViewCellForRowAtIndexPath(UITableView uiTableView,
    NSIndexPath nsIndexPath) {
        UITableViewCell cell = (UITableViewCell)
            tableView().dequeueReusableCellWithIdentifierForIndexPath("Cell",
nsIndexPath);
        String objkey = objects.get((int) nsIndexPath.row());
        cell.textLabel().setText(data.getAllNotes().get(objkey));
        return cell;
}
```

テーブルへのノートの追加処理 (Android*):

Android* 側でテーブルを生成するには、Data クラスからリストを受け取り、テーブルを生成する ListAdapter へ渡します。

```
objects = new ArrayList<String>();
makeObjects();
ListView list = (ListView) findViewById(android.R.id.list);
listAdapter = new ListAdapter(this,objects,data);
list.setAdapter(listAdapter);
```

2 つ目のビューへの移行処理 (iOS*):

iOS* 側では、あるビューから別のビューへの移行に Segue が使用されます。Segue は、テンプレートによってすでのセットアップされています。次のように、メソッドの一部を変更します。

```
@Override
public void prepareForSegueSender(UIStoryboardSegue uiStoryboardSegue,
    @Mapped(ObjCObjectMapper.class) Object o) {
        super.prepareForSegueSender(uiStoryboardSegue, o);
        String object;
        if (uiStoryboardSegue.identifier().equals("showDetail")) {
            NSIndexPath indexPath = tableView().indexPathForSelectedRow();
            if(indexPath==null){
                object = objects.get(0);
            }
            else {
                object = objects.get((int) indexPath.row());
            }
            DetailViewController controller = (DetailViewController)
((UINavigationController)

uiStoryboardSegue.destinationViewController()).topViewController();
            controller.setDetailItem(object,data);
            controller.navigationItem().setLeftBarButtonItem(
                splitViewController().displayModeButtonItem());
            controller.navigationItem().setLeftItemsSupplementBackButton(true);
        }
}
```

2 つ目のビューへの移行処理 (Android*):

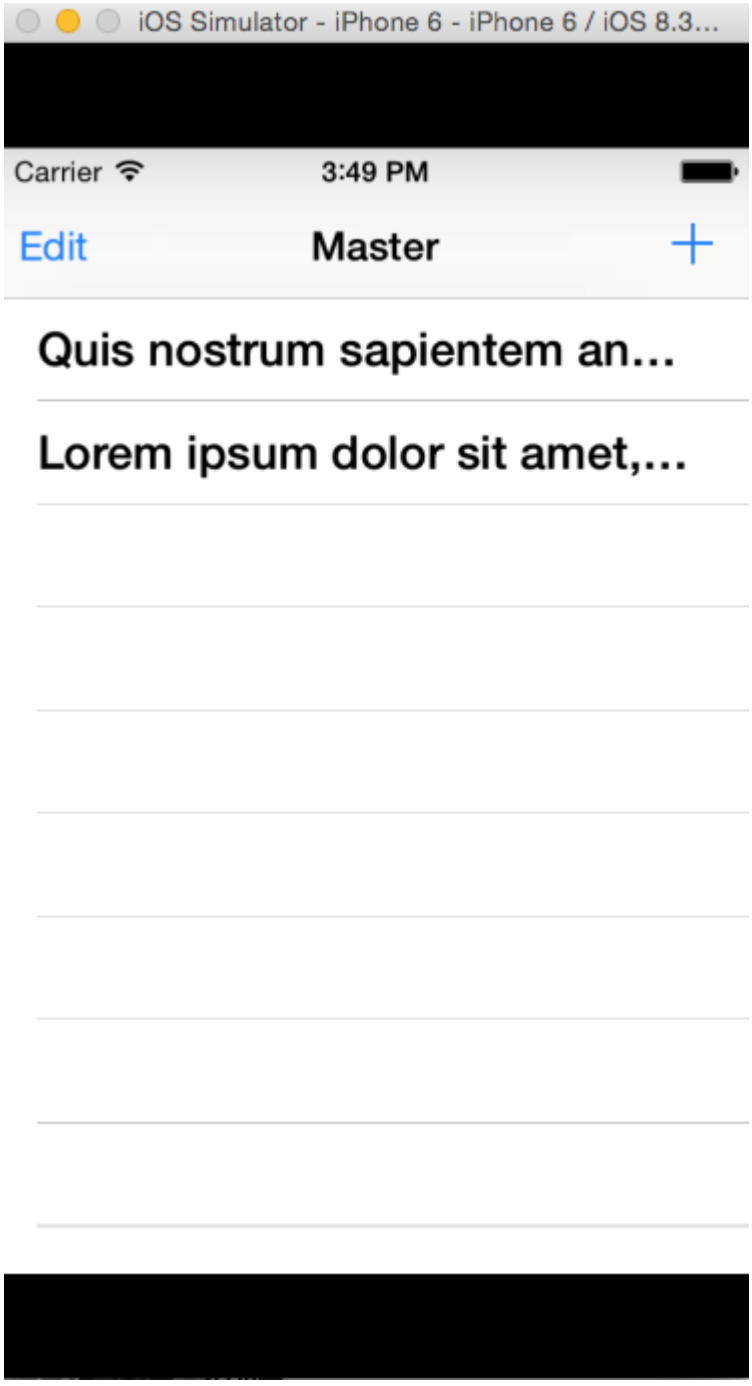
Android* 側では、リスト項目のクリック時に 2 つ目の Activity を初期化する ListView にクリックリスナーを設定します。

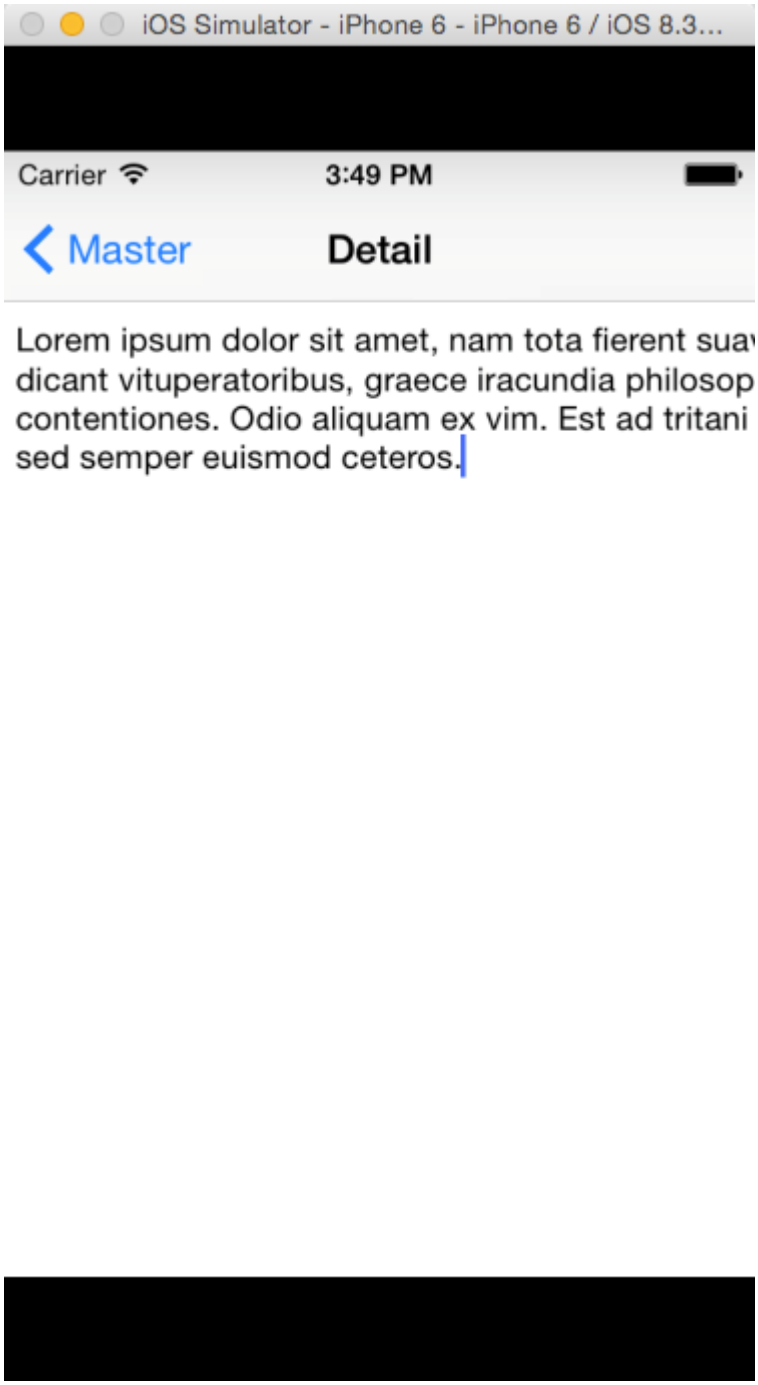
```
list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
        data.setCurrentKey(objects.get(position));
        Intent intent = new Intent(MainActivity.this, EditorActivity.class);
        intent.putExtra(DEFAULT_TEXT,
data.getAllNotes().get(data.getCurrentKey()));
        startActivityForResult(intent, EDITOR_REQUEST_CODE);
    }
});
```

こうすることで、両方のプラットフォームで共通のロジックを使用して、ファイルストレージからデータを取得、表示することができます。

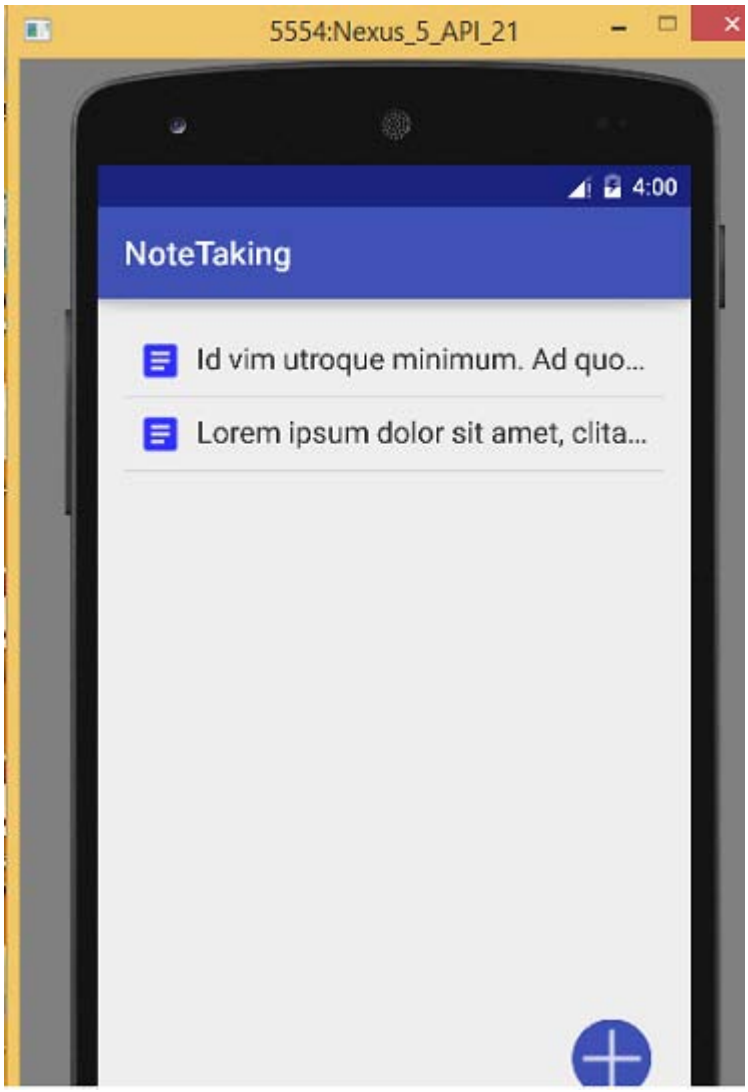
以下は、完成したアプリケーションのスクリーンショットです。

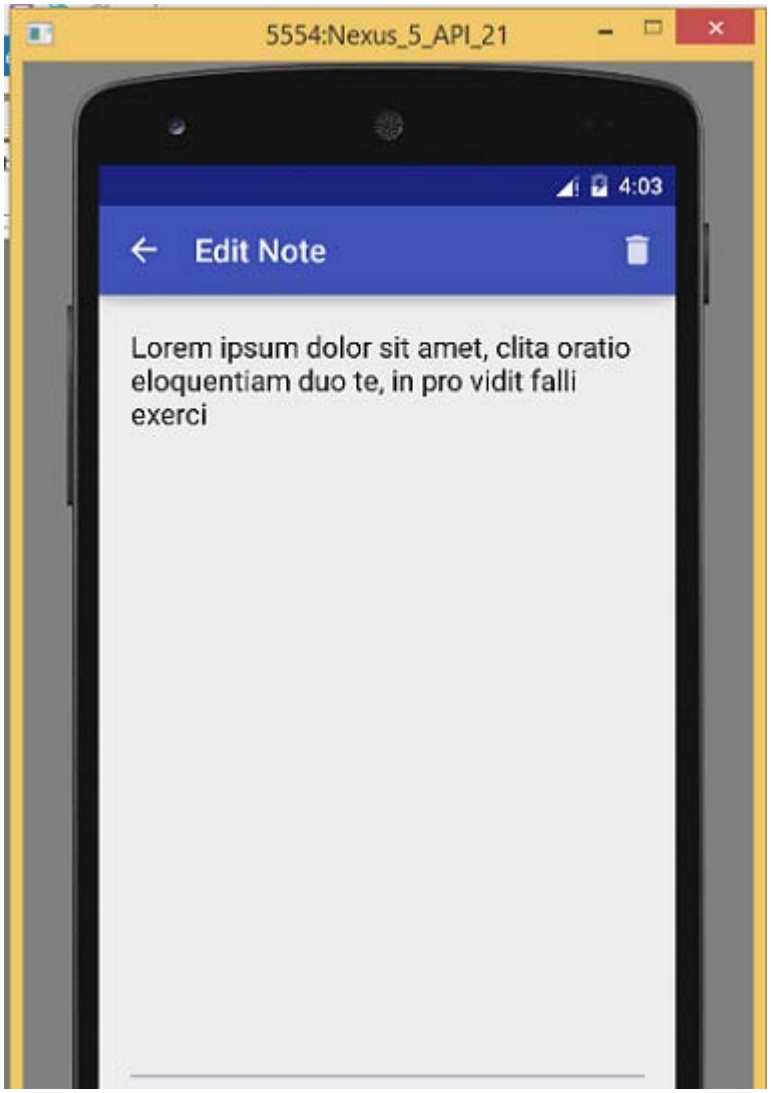
iOS*:




iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.3...
Carrier
3:49 PM
Master
Detail
Lorem ipsum dolor sit amet, nam tota fierent sua
dicant vituperatoribus, graece iracundia philosop
contentiones. Odio aliquam ex vim. Est ad tritani
sed semper euismod ceteros.

Android*:


5554:Nexus_5_API_21
4:00
NoteTaking
Id vim utroque minimum. Ad quo...
Lorem ipsum dolor sit amet, clita...



パート 2 では、アプリケーションに sqlite 機能を追加します。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。