08027 Lab 6 JSON and SQLite

Goal

The goal of this lab is to give you some experience of downloading some data using JSON and storing that data in a SQLite database. The portion regarding SQLite is heavily based on a tutorial in the Android documentation available here:

http://developer.android.com/training/basics/data-storage/databases.html

You should refer to that for more information. Both that tutorial and this lab assume some prior understanding of databases and SQL. If you don't there is a video resource that will give you some of the basics of SQL here:

https://www.youtube.com/watch?v=7Vtl2WgqqOg

If you want to verify your understanding by practicing some SQL in isolation of the android environment there is an online SQL tool here:

https://www.khanacademy.org/computer-programming/my-sqlite-tutorial/6392526208237568

Check out and open the lab 6 application called Pirate Crew. In this application we will create a database and use it to store information about pirates and pirate ships. Each ship can have a crew of pirates. If you run the application you will see a series of buttons enabling you do download data, add new pirates and ship and edit existing pirates and ships. We will be downloading this information from a JSON file on the internet and using it to populate and edit a SQLite database.

The JSON file that contains information about pirates and ships is available here:

http://www.hull.ac.uk/php/349628/08027/labs/pirates.json

You can open the link in internet explorer to take a look at the data. You should see that it contains a Pirates object that contains a list of Pirate objects. Each Pirate object contains a name and a nationality, and optionally a Nickname.

Adding Activities to the Manifest

If you run the code and click any of the buttons the application will crash. This is because although both the associated java code files and xml layout files are present in Android Studio they have not been added to the application manifest file, so are not part of the application. To remedy this add all the Activities we will use to the application in the manifest file like this:

```
<activity android:name=".DisplayJSONActivity"/>
<activity android:name=".EditPirateActivity"/>
<activity android:name=".EditShipActivity"/>
<activity android:name=".SelectPirateActivity"/>
<activity android:name=".SelectShipActivity"/>
```

Downloading JSON Data

The first step is to download the JSON file. Currently if you click the download crew data button an new activity is launched that just displays the string "Hello, JSON!". Firstly we will download the JSON data and display that instead. Remember, because downloading something from the internet is a time consuming task we cannot do it on the UI thread, so we should create an AsyncTask to deal with this for us on a separate thread.

In the DisplayJSONActivity file add a private class downloadJSON that extends AsyncTask like this:

```
private class downloadJSON extends AsyncTask<String, String, String> {

}
```

The compiling is complaining because if you inherit from AsyncTask you need to implement the doInBackground method. For now we'll just get the JSON as a String to check that it works. Add a doInBackground method to the downloadJSON class like this:

```
protected String doInBackground(String... args) {
    String result = "";
    try
    {
        InputStream stream = (InputStream)new URL(args[0]).getContent();
        BufferedReader reader = new BufferedReader(new InputStreamReader(stream));
        String line = "";
        while(line != null)
        {
            result += line;
            line = reader.readLine();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
```

Finally, once this method has finished we want to update the JSONTextView on the main UI thread. To do that we can use AsyncTask's onPostExecute method, which will be called on the UI thread when doInBackground has completed its work. Add an onPostExecute method to the downloadJSON class like this:

```
protected void onPostExecute(String pResult){
    TextView textView = (TextView)findViewById(R.id.JSONTextView);
    textView.setText(pResult);
}
```

Next, when the activity is created we need to start the new downloadJSON task. In the onCreate method of the DisplayJSONActivity activity replace the code that sets the textView tesxt to "Hello, JSON!" with this code, to create and execute a downloadJSON object instance.

```
TextView textView = (TextView)findViewById(R.id.JSONTextView);
textView.setText("Downloading JSON!");
new downloadJSON().execute("http://www.hull.ac.uk/php/349628/08027/labs/pirates.json");
```

Finally, because we're accessing the internet we need to add the appropriate permission to the AndroidManifest.xml file. Inside the manifest tag, but outside the application tag add the tag:

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

Run your code and test it. When you click the Download Crew Data button you should see the data like this:


```
{  "Pirates": [{        "Name": "Cheung Po
Tsai",        "Nationality": "Chinese"      },{       "Name":
"William Kidd",       "Nationality": "Scottish"      },
{        "Name": "Edward Teach",       "Nationality":
"English",       "Nickname": "Blackbeard"      },
{  "Name": "Bartholomew
Roberts",  "Nickname": "Black
Bart",        "Nationality": "Welsh"      },{       "Name":
"Henry Every",       "Nationality":
"English",       "Nickname": "Long Ben"      },
{        "Name": "Anne Bonny",        "Nationality":
"Irish"      },{       "Name": "Sir Henry
Morgan",       "Nationality": "Welsh"      },{       "Name":
"Francois l'Olonnais",        "Nationality": "French"     },
{        "Name": "Sir Francis Drake",       "Nationality":
"English"      }, {        "Name": "Ching
Shih",       "Nationality": "Chinese"}     ],  "Ships":
["Royal Fortune", "Golden Hind", "Roebuck", "The
Adventure Galley", "The CSS Alabama"]}
```


Once you've done that commit your code to SVN with an appropriate message like *"Completed lab 6 task 1 download JSON data and displayed it in unformatted text"*

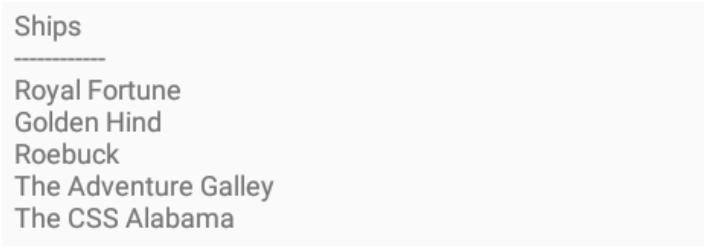
Parsing JSON Data

So far we've downloaded the JSON data, but we've just displayed it as a string and it looks pretty ugly. Instead of downloading JSON data and saving the text let's do some work to format it in a better way.

At the top of the doInBackground method create an empty String called formattedResult. Then, after the while loop that reads the result String from the reader use the following code:

```
JSONObject json = new JSONObject(result);

formattedResult = "Ships\r\n-----------\r\n";

JSONArray ships = json.getJSONArray("Ships");

for(int i = 0; i < ships.length(); ++i){
    formattedResult += ships.get(i) + "\r\n";
}
```

This code creates a JSONObject called json from the result String, writes a title and an underline, and then uses the getJSONArray method on the JSONObject to get the array called "Ships". Finally it iterates over that array adding a newline containing the value of each ship from the ship array to the formattedResult String. Finally, remember to return the formattedResult String instead of the result String. You should end up with something like this:

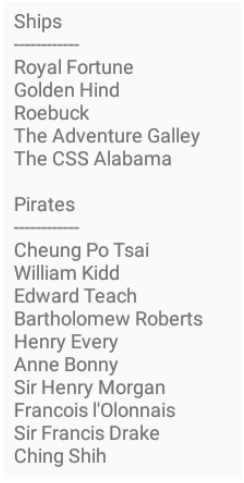


Check your code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 2 formatted the ship data from the JSON Object"*

The next part of the JSON object is and array of objects. This makes the code to parse it a little more complicated. First add a "Pirates" title to the formattedResult String along with an underline to separate the title from the content. Next get the "pirates" JSONArray from the JSONObject. Then write a for look to iterate over that JSONArray, and for each item in the array get the JSONObject and use that to get the name from that object, adding that to the formattedResult array. This code inside the loop should look like this:

```
JSONObject pirate = pirates.getJSONObject(i);
formattedResult += pirate.getString("Name") + "\r\n";
```

And the result should look like this:



Check your code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 3 formatted the ship data and the pirate data from the JSON Object"*

To test your understanding of what is going on add the pirate nationality, and if they have one, their nickname too.

```
Ships
----------
Royal Fortune
Golden Hind
Roebuck
The Adventure Galley
The CSS Alabama

Pirates
----------
Cheung Po Tsai,Chinese
William Kidd,Scottish
Edward Teach,Blackbeard,English
Bartholomew Roberts,Black Bart,Welsh
Henry Every,Long Ben,English
Anne Bonny,Irish
Sir Henry Morgan,Welsh
Francois l'Olonnais,French
Sir Francis Drake,English
Ching Shih,Chinese
```

Check your code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 4 formatted the ship data and the pirate data from the JSON Object including nationalities and nicknames"*

Storing data in a SQLite database

As detailed previously this section on SQLite is heavily based on a tutorial in the Android documentation available here:

http://developer.android.com/training/basics/data-storage/databases.html

You should refer to that for more information. If you are not familiar with databases they provide a way of storing large amounts of structured data and are optimized for common operations such as searching for records that fulfil particular filters. Databases contain tables of data, and each table contains records (rows) which are made up of field (columns). For example, one of the tables we will be using will be the Pirate table, which has fields called Name, Nickname and Nationality.

_ID	NAME	NICKNAME	NATIONALITY
1	Cheung Po Tsai		Chinese
2	William Kidd		Scottish
3	Edward Teach	Blackbeard	English
4	Bartholomew Roberts	Black Bart	Welsh
5	Henry Every	Long Ben	English
6	Anne Bonny		Irish
7	Sir Henry Morgan		Welsh
8	Francois l'Olonnais		French
9	Sir Francis Drake		English
10	Ching Shih		Chinese

SQL stands for Structured Query Language. It is used to send instructions to databases and typically looks like "SELECT * FROM Pirate" which selects all the records in a table called Pirate.

The first thing to do is to consider what our database will look like. We want a table of Pirates and a table of Ships. Our pirate table has strings for Name, Nickname and Nationality, as well as a primary key called _ID. A primary key is a unique identifier for a record that can be used to distinguish that record from any other record that may contain the same data. Our Ship table just contains the Name of the ship, and the _ID primary key.

We also want to be able to create a link between a Pirate and a Ship. To do this we have a table called Crew Member. Records in Crew Member have a primary key just like the Pirate and Ship tables, but it also has two fields that are foreign keys, and represent the primary keys of records in other tables. In this case this forms a link between a Ship and a Pirate that will be preserved even if data like the Pirate or Ship Name is changed.

Now we have defined the tables we need in our database start programming. First we need a Contract class called PirateDBContract which we will use to store information about our database. This class should contain inner classes that represent our database schema as a series of static strings. Through the rest of our code we can use these strings to formulate SQL queries. This means that if we need to change, for example, the name of a field in a database we can just change it in one place and that change will propagate throughout the rest of our code. Create a public final class PirateDBContract. Make a default private constructor (this prevents people from accidently instantiating this class) and then add the following inner classes.

```
public static abstract class PirateEntry implements BaseColumns {
    public static final String TABLE_NAME = "Pirate";
    public static final String COLUMN_NAME_NAME = "Name";
    public static final String COLUMN_NAME_NICKNAME = "Nickname";
    public static final String COLUMN_NAME_NATIONALITY = "Nationality";
}

public static abstract class ShipEntry implements BaseColumns {
    public static final String TABLE_NAME = "Ship";
    public static final String COLUMN_NAME_NAME = "Name";
}

public static abstract class CrewMemberEntry implements BaseColumns {
    public static final String TABLE_NAME = "Crew Member";
    public static final String COLUMN_NAME_PIRATE_ID = "Pirate ID";
    public static final String COLUMN_NAME_SHIP_ID = "Ship ID";
}
```

Note that by inheriting from the BaseColumns class we automatically get a unique ID for each for called _ID (among other things).

Next, we can use these strings to create some SQL queries to help us create and delete our tables.

```
public static final String TEXT_TYPE = " TEXT";
public static final String COMMA_SEP = ",";

public static final String SQL_CREATE_PIRATE_TABLE = "CREATE TABLE " + PirateEntry.TABLE_NAME +
        " (" + PirateEntry._ID + " INTEGER PRIMARY KEY" + COMMA_SEP +
        PirateEntry.COLUMN_NAME_NAME + COMMA_SEP + PirateEntry.COLUMN_NAME_NICKNAME
        + COMMA_SEP + PirateEntry.COLUMN_NAME_NATIONALITY + " )";

public static final String SQL_CREATE_SHIP_TABLE = "CREATE TABLE " + ShipEntry.TABLE_NAME +
        " (" + ShipEntry._ID + " INTEGER PRIMARY KEY" + COMMA_SEP +
        ShipEntry.COLUMN_NAME_NAME  + " )";

public static final String SQL_CREATE_CREW_TABLE = "CREATE TABLE " + CrewMemberEntry.TABLE_NAME +
        " (" + CrewMemberEntry._ID + " INTEGER PRIMARY KEY" + COMMA_SEP +
        CrewMemberEntry.COLUMN_NAME_PIRATE_ID + COMMA_SEP + CrewMemberEntry.COLUMN_NAME_SHIP_ID + " )";

public static final String SQL_DELETE_PIRATE_TABLE = "DROP TABLE IF EXISTS " + PirateEntry.TABLE_NAME;
public static final String SQL_DELETE_SHIP_TABLE = "DROP TABLE IF EXISTS " + ShipEntry.TABLE_NAME;
public static final String SQL_DELETE_CREW_TABLE = "DROP TABLE IF EXISTS " + CrewMemberEntry.TABLE_NAME;
```

Now we've set up all our SQL queries we need a way to call them. We can do that using the SQLiteOpenHelper class. Open the class called PirateDBHelper that extends SQLiteOpenHelper. Note there is an int database version and a String to hold the filename, the constructer uses these, along with the Context to call the SQLiteOpenHelper constructor.

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

public PirateDBHelper(Context pContext) {
    super(pContext, DATABASE_NAME, null, DATABASE_VERSION);
}
```

The onCreate method which takes the SQLiteDatabase as a parameter, and using that database execute the SQL queries we created in PirateDBContract to create the three tables.

```
public void onCreate(SQLiteDatabase pDb){
    pDb.execSQL(PirateDBContract.SQL_CREATE_PIRATE_TABLE);
    pDb.execSQL(PirateDBContract.SQL_CREATE_SHIP_TABLE);
    pDb.execSQL(PirateDBContract.SQL_CREATE_CREW_TABLE);
}
```

There is also an override for the onUpgrade method which is called if the database version is changed. As that is unlikely we can just implement this as an empty method for now.

Next we're going to write to our database. When we click the "Download Crew Data" button instead of launching a new activity we just want to start an asynctask to download the JSON in the background, and then write that information to our database. In order to do this we should start by creating an instance of PirateDBHelper as a datamember of the MainActivity.

```
PirateDBHelper mDbHelper = new PirateDBHelper(this);
```

Next, copy the AsyncTask that downloads JSON from the DisplayJSONActivity class into the MainActivity. If you like you can delete DisplayJSONActivity. We don't need it anymore. When you've done that at the start of the doInBackground method in the MainActivity class call getWritableDatabase on the mDbHelper object. Save the returned database in an SQLiteDatabase object called db.

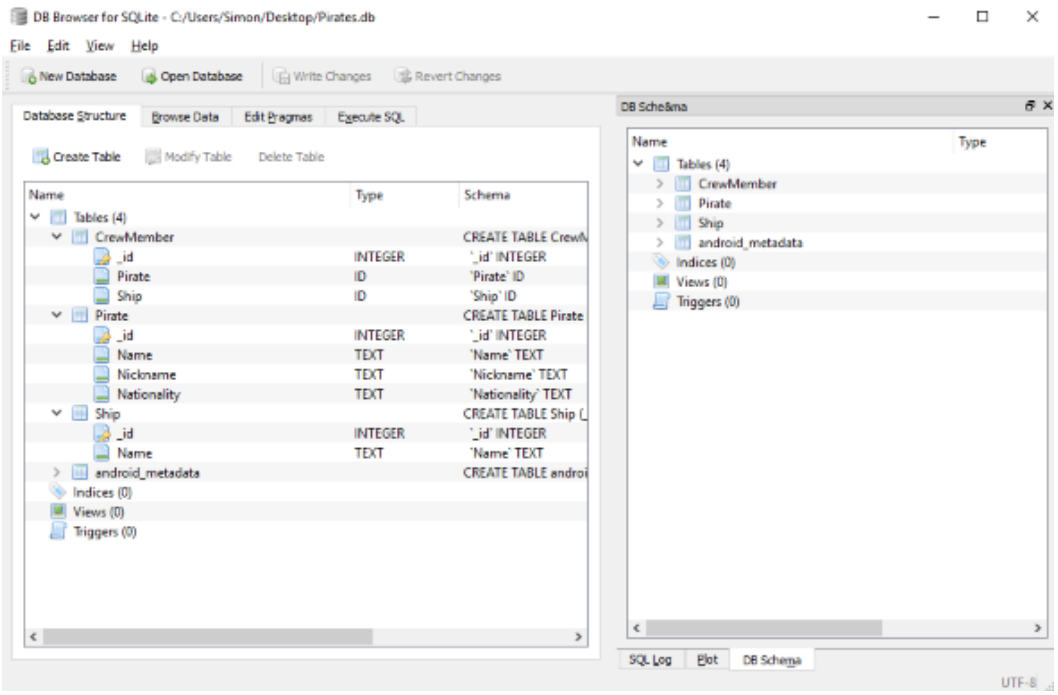
```
SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

You can also remove the contents of the onPostExecute method, as we no longer want to populate a textView with the string returned from the doInBackground method. Finally, in the onClickDownloadData method replace the contents with a line to launch the asynchronous task.

```
new downloadJSON().execute("http://www.hull.ac.uk/php/349628/08027/labs/pirates.json");
```

Run your code. When you click the button you'll notice in the monitor that you get all sorts of errors. If you inspect the output in Android Monitor you can see an error *"Caused by: android.database.sqlite.SQLiteException: near "Member": syntax error (code 1): , while compiling: CREATE TABLE Crew Member (_id INTEGER PRIMARY KEY,Pirate ID,Ship ID)"*. This command isn't being interpreted correctly. There are a number of ways you could fix this, but it seems like one way might be to avoid having a table name with a space in it. Change the name of the table from Crew Member to CrewMember. Luckily, as described previously, we only have to make the change in one place and that change will propagate through. It seems like this mistake was made on purpose just to prove a point ;) .

Run your code. This time your code should not crash, but not a lot else will happen. In order to verify whether the database has been successfully created open DDMS and examine the emulators file system. Navigate to data > data > [the application name] > databases and you should see your database there. If you select the file and click the "pull from device" button in the top right you can extract the database from the emulator and open it in your favourite database software. I suggest DB Browser for SQL Lite from http://sqlitebrowser.org/ - the portable version should work in labs.



Commit to SVN with an appropriate log message like *"Completed lab 6 task 6 Created the database"*

Now we need to save the information from the JSON file to the database by creating the appropriate new records in the database. Currently the code we hacked out of the DisplayJSONActivity downloads a string into the result string, then uses that to create a JSONObject, then uses that to build the formattedResult string that previously was used to populate the text box. Instead, we want to extract the data from each of the ships and pirates and put that data into our new database. To do that we will use the ContentValues class. We can use the put method to add a new value to the ContentValues instance specifying the column name and the value. When we've done that we can

insert the new record into the table using the insert command, passing the table name, any columns that can be assigned null and the ContentValues instance.

For the ships the code looks like this:

```
ContentValues values = new ContentValues();
for (int i = 0; i < ships.length(); ++i) {
    values.clear();
    values.put(PirateDBContract.ShipEntry.COLUMN_NAME_NAME, ships.get(i).toString());
    db.insert(PirateDBContract.ShipEntry.TABLE_NAME, null, values);
}
```

When we insert the record for the ships there are no nullable columns, so we pass in null as the second parameter.

For the pirates the code looks like this:

```
for (int i = 0; i < pirates.length(); ++i) {
    JSONObject pirate = pirates.getJSONObject(i);
    values.clear();
    values.put(PirateDBContract.PirateEntry.COLUMN_NAME_NAME, pirate.getString("Name"));
    values.put(PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY, pirate.getString("Nationality"));
    if (pirate.has("Nickname")) {
        values.put(PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME, pirate.getString("Nickname"));
    }
    db.insert(PirateDBContract.PirateEntry.TABLE_NAME, PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME, values);
}
```

In this case, pirates may or may not have a nickname, so we pass that column as nullable.

Run the code and check the database. You should see the database is now populated with records. Make sure that you successfully export a new version of the database using the device monitor.

_id	Name	Nickname	Nationality
Filter	Filter	Filter	Filter
1	Cheung Po Tsai	NULL	Chinese
2	William Kidd	NULL	Scottish
3	Edward Teach	Blackbeard	English
4	Bartholomew ...	Black Bart	Welsh
5	Henry Every	Long Ben	English
6	Anne Bonny	NULL	Irish
7	Sir Henry Mor...	NULL	Welsh
8	Francois l'Olo...	NULL	French
9	Sir Francis Dr...	NULL	English
10	Ching Shih	NULL	Chinese

Commit to SVN with an appropriate log message like *“Completed lab 6 task 5 Added records to the database”*

Now we’ve saved our data in a database we need to know how to load it from the database.

Initially we want to do this in the SelectPirateActivity. When this activity is launched we want to read the pirates from the database and add them to a list. We will use the Pirate object to help manage the data, and the PirateAdapter to help us get the list of pirates into the UI components.

Initially we need to get data from the database. To do this we need to get hold of the database, tell it which columns we are interested in (the projection) and create a cursor object to help us navigate through the results. In the SelectPirateActivity onCreate method add the following code.

```
SQLiteDatabase db = new PirateDBHelper(this).getReadableDatabase();

String[] projection = {
        PirateDBContract.PirateEntry._ID,
        PirateDBContract.PirateEntry.COLUMN_NAME_NAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY,
        PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME
};

Cursor c = db.query(
        PirateDBContract.PirateEntry.TABLE_NAME,
        projection,
        null,
        null,
        null,
        null,
        null
);
```

Next we need to create an ArrayList of our pirate objects, then use the cursor to iterate over all the results that were returned from the database, adding a new Pirate instance to the list each time. Once we have created all the pirates we can close the Cursor.

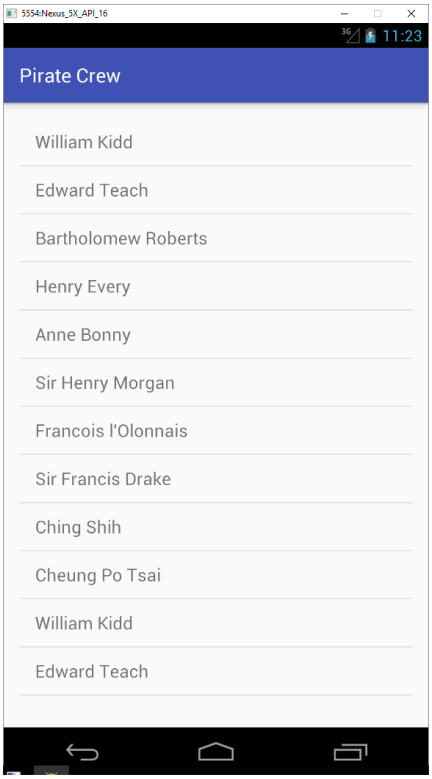
```
ArrayList pirateList = new ArrayList<Pirate>();

c.moveToFirst();

while(c.moveToNext())
{
    String name = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NAME));
    String nationality = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry._ID));
    String nickname  = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME));
    int id = c.getInt(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry._ID));
    pirateList.add(new Pirate(name, nickname, nationality, id));
}

c.close();
```

Finally we can use the ArrayList to create a PirateAdapter which we can use in the pirateListView.

```
PirateAdapter adapter = new PirateAdapter(this, android.R.layout.simple_list_item_1, pirateList);

final ListView pirateListView = (ListView)findViewById(R.id.PirateListView);
pirateListView.setAdapter(adapter);
```

Run the code and click on the button. You should see something like the screen shot below.

In order to display a more meaningful list we must override the toString method in the Pirate class, perhaps to just return the name of the pirate. Once you have done that you should get a more meaningful view.



Commit to SVN with an appropriate log message like *"Completed lab 6 task 6 Loaded pirate records from the database into a listView"*

Next we need to write some code to allow us to get hold a pirate when they are clicked. To do this we need to add a click listener to the view, and make it return the pirate ID. Remember, this activity was started using startActivityForResult, so once the SelectPirateActivity is finished the onActivityResult in the MainActivity class will be called. We just want to set the result code and create an Intent to pass back the ID of the selected pirate.

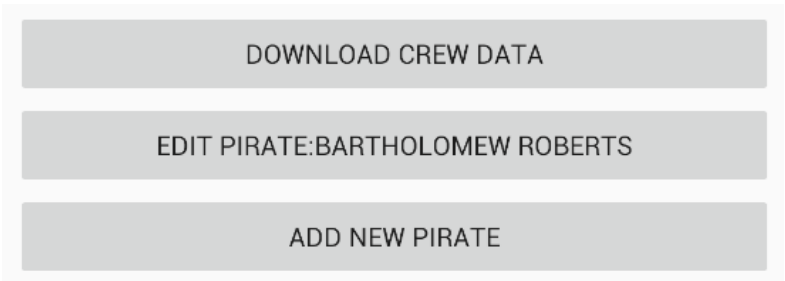
After you set the list adapter in the onCreate method of the SelectPirateActivity add the following code.

```
pirateListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Pirate selectedPirate = (Pirate)pirateListView.getItemAtPosition(position);
        Intent data = new Intent();
        data.putExtra("PirateKey", String.valueOf(selectedPirate.ID()));
        setResult(RESULT_OK, data);
        finish();
    }
});
```

Then, in the onActivityResult method in the MainActivity class there is already code that check that the result code is ok, and that we are responding to the select pirate acitivty. Modify that code to get hold of our database and find the name of the selected pirate. Then use that to modify the editPirate button.

```
SQLiteDatabase db = new PirateDBHelper(this).getReadableDatabase();

String[] projection = {
        PirateDBContract.PirateEntry.COLUMN_NAME_NAME
};

String selection = PirateDBContract.PirateEntry._ID + " = ?";
String[] selectionArgs = { pData.getStringExtra("PirateKey")};

Cursor c = db.query(
        PirateDBContract.PirateEntry.TABLE_NAME,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        null
);

c.moveToFirst();
String name = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NAME));
c.close();
Button button = (Button)findViewById(R.id.editPirateButton);
button.setText("Edit Pirate:" + name);
```

Check this code works by running it and selecting a pirate.



Commit to SVN with an appropriate log message like *"Completed lab 6 task 6 added listener to listView and changed button name in the MainActivity"*

This is all well and good, but it's not really what we want to do. What we should be doing is starting the EditPirateActivity. All we need to do in the MainActivity is get the appropriate pirate ID from the returned intent and pass it to a new EditPirateActivity. Then in the onCreate method of the EditPirateActivity we can load the appropriate pirate from the database and use it to populate the User Interface.

First let's start the EditPirateActivity from the MainActivity window. Replace the code you just wrote with the following code.

```
Intent intent = new Intent(this, EditPirateActivity.class);
intent.putExtra("PirateKey", pData.getStringExtra("PirateKey"));
startActivity(intent);
```

Next in the EditPirateActivity we need to get hold of the database, create a projection of the columns that we are interested in, then a selection for the rows. The only row we care about is the one with the primary key that was passed into the activity by the intent. You should also save that primary key as a String data member of the EditPirateActivity class. We will want to use this again later when we save any changes.

```
SQLiteDatabase db = new PirateDBHelper(this).getReadableDatabase();

String[] projection = {
        PirateDBContract.PirateEntry.COLUMN_NAME_NAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY
};

String selection = PirateDBContract.PirateEntry._ID + " = ?";
String[] selectionArgs = { getIntent().getStringExtra("PirateKey")};
```

Next use these things to get a cursor which we can use to extract the information about the pirate with the primary key ID that was passed into the activity.

```
Cursor c = db.query(
        PirateDBContract.PirateEntry.TABLE_NAME,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        null
);

c.moveToFirst();
String name = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NAME));
String nickname = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME));
String nationality = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY));
c.close();
```

Finally add this information to the appropriate components in the GUI.

```
TextView textView = (TextView)findViewById(R.id.PirateNameEditText);
textView.setText(name);
textView = (TextView)findViewById(R.id.PirateNicknameEditText);
textView.setText(nickname);
textView = (TextView)findViewById(R.id.NationalityEditText);
textView.setText(nationality);
```

When the user clicks the Save Changes button we want to save the information in the GUI back to the database, updating the information that is already there. First you will need to link the button click to a method in the EditPirateActivity class. In that method you should create a new set of ContentValues and populate the values with the information from the form.

```
public void onPirateSaveChangesButtonClick(View pView)
{
    ContentValues values = new ContentValues();

    TextView textView = (TextView)findViewById(R.id.PirateNameEditText);
    values.put(PirateDBContract.PirateEntry.COLUMN_NAME_NAME, textView.getText().toString());
    textView = (TextView)findViewById(R.id.PirateNicknameEditText);
    values.put(PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME, textView.getText().toString());
    textView = (TextView)findViewById(R.id.NationalityEditText);
    values.put(PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY, textView.getText().toString());
```

Next create a selection query using the primary key for the pirate that we stored earlier. Then get hold of the database and finally make a call to update the database, passing in the name of the table we want to update, the values, selection and selection arguments. Once we've done all that we can call finish to close the activity.

```
    String selection = PirateDBContract.PirateEntry._ID + " = ?";
    String[] selectionArgs = { mPirateKey };

    SQLiteDatabase db = new PirateDBHelper(this).getReadableDatabase();
    db.update(PirateDBContract.PirateEntry.TABLE_NAME, values, selection, selectionArgs);
    finish();
}
```

Make sure the code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 7 Wrote code to update pirate records in the database"*.

We can use the same activity to add new pirates to our database. In fact, the add new pirate button in the main activity is already wired up, but if you press it the program will crash because the code that we have written assumes that a primary key is passed in with the intent.

In the onCreate method add some code to test to see if there is a primary key or not using the hasExtra method. If there is not set the value of the mPirateKey datamember to "-1". Next, in the onPirateSaveChangesMethod check the value of the primary key, and if the value is "-1" use the insert method instead of the update method.

Make sure the code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 8 Wrote code to add new pirate records in the database"*.

To check your understanding apply what you have just done to the ships table. You will need to create a ship class that represents the data held in the ship tables, which should be a name and an

ID, and an ArrayList of Pirates for the crew. You will need to create a constructor, some accessors, a toString method and a method to AddCrewMember that takes a Pirate parameter. You will also need to create a ShipAdapter class to help pass an ArrayList of ships to the user interface components. Then you can load the ships into the appropriate ship selection activity. Test your code then commit to SVN with an appropriate log message like *"Completed lab 6 task 9 Loaded ship records from the database into a listView"*

Next create functionality that returns the selected ship to the main activity. You will need to specify the request code as SELECT_SHIP to ensure that you start the appropriate activity to edit ships. The User Interface also includes the ability to add crew to a ship, but ignore that for now.

Commit to SVN with an appropriate log message like *"Completed lab 6 task 10 added listener to listView and launched EditShipActivity"*

Don't forget to enable adding new ships too if the intent that launches the edit ship activity has no extra primarykey for the ship.

Make sure the code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 11 Wrote code to add new ship records in the database"*.

The final step of our application is to allow you to add crew to a ship. If you recall the database schema you will know that this involved a separate table whose records linked the foreign key of a ship with the foreign key of a pirate. This allows each ship to appear in multiple records, so we can have multiple crew members per ship.

To make life a little easier for us we will store the details of the ship in a member variable of type Ship in the EditShipActivity. We will populate this with details of the ship in the onCreate method, and we will save details of this ship back to the database by creating a method that is called when the Save Changes button is pressed. First let's rebuild the ship member variable from the database in the onCreate method. Start by getting hold of the database and setting up a projection, selection, selectionArgs and making a query to get a Cursor object.

```
shipKey = getIntent().getStringExtra("ShipKey");

SQLiteDatabase db = new PirateDBHelper(this).getReadableDatabase();

String[] projection = {
        PirateDBContract.ShipEntry.COLUMN_NAME_NAME,
};

String selection = PirateDBContract.ShipEntry._ID + " = ?";
String[] selectionArgs = { shipKey };

Cursor c = db.query(
        PirateDBContract.ShipEntry.TABLE_NAME,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        null
);
```

Next lets use the cursor object to get the name of the ship from the database and create the Ship object. For now we'll just add and empty list of Pirate objects as the crew.

```
c.moveToFirst();
String name = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NAME));

TextView textView = (TextView) findViewById(R.id.ShipNameEditText);
textView.setText(name);

ArrayList<Pirate> crew = new ArrayList<Pirate>();

mShip = new Ship(shipKey, Integer.parseInt(shipKey), crew);
```

Next we need to launch the selectPirateActivity when the Add Crew Member button is pressed. We need to start the activity for a result

In order to select crew members we need to create a method to be called when the appropriate button is clicked and use that method to launch the SelectPirateActivity using the startActivityForResult method. When that result returns we need to extract the id of the pirate that was selected. We did something similar to this previously where we called the selectPirateActivity from the MainActivity, so look at that if you need a reminder of what to do. When the selectPirateActivity returns a result we need to get the details of the pirate from the database and use them to create a new Pirate, then add it to the Ship member variable's crew.

First build the query and get the Cursor object.

```
SQLiteDatabase db = new PirateDBHelper(this).getReadableDatabase();

String pirateKey = pData.getStringExtra("PirateKey");

String[] projection = {
        PirateDBContract.PirateEntry.COLUMN_NAME_NAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY
};

String selection = PirateDBContract.PirateEntry._ID + " = ?";
String[] selectionArgs =  {pirateKey };

Cursor c = db.query(
        PirateDBContract.PirateEntry.TABLE_NAME,
        projection,
        selection,
        selectionArgs,
        null,
        null,
        null
);
```

Use the cursor to extract information about the selected pirate from the database

```
c.moveToFirst();
String name = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NAME));
String nickname = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME));
String nationality = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY));
c.close();
db.close();
```

Then build the pirate and add it to the member variable's crew. Finally, use the crew from the ship together with the PirateAdapter to populate the CrewListView.

```
Pirate selectedPirate = new Pirate(name, nickname, nationality, Integer.parseInt(pirateKey));
mShip.AddCrewMember(selectedPirate);

PirateAdapter adapter = new PirateAdapter(this, android.R.layout.simple_list_item_1, mShip.Crew());

ListView pirateListView = (ListView)findViewById(R.id.CrewListView);
pirateListView.setAdapter(adapter);
```

Make sure the code works, then commit to SVN with an appropriate log message like *"Completed lab 6 task 12 Added a pirate to a pirate ship's crew"*.

So far so good, but the changes are only temporary because we haven't saved anything to the database. Create a method to be called when the Save Changes button is clicked. Use that to update the ship, and then the crew. Remember, the crew are stored in a separate table in the database as pairs of foreign keys that link a pirate to a ship.

Also remember that if the EditShipActivity was launched without a ship's ID in the intent then we want to create a new Ship. In this case the member variable ship id is set to -1. First for the ship name we extract the values from the user interface, then we get hold of the database and either insert or update the ship. We need to create a selection and selection arguments using the ship id.

```
ContentValues values = new ContentValues();

TextView textView = (TextView)findViewById(R.id.ShipNameEditText);
values.put(PirateDBContract.ShipEntry.COLUMN_NAME_NAME, textView.getText().toString());

SQLiteDatabase db = new PirateDBHelper(this).getWritableDatabase();
if(mShip.ID() == -1)
{
    db.insert(PirateDBContract.ShipEntry.TABLE_NAME, null, values);
} else {
    String selection = PirateDBContract.ShipEntry._ID + " = ?";
    String[] selectionArgs = { String.valueOf(mShip.ID()) };
    db.update(PirateDBContract.ShipEntry.TABLE_NAME, values, selection, selectionArgs);
}
```

Next, for the crew first we delete the entire crew. Then we replace it with the existing crew from the Ship member variable. Finally we call finish to close the activity.

```
ArrayList<Pirate> crew = mShip.Crew();

String selection = PirateDBContract.CrewMemberEntry.COLUMN_NAME_SHIP_ID + " LIKE ?";
String[] selectionArgs = { String.valueOf(mShip.ID()) };
db.delete(PirateDBContract.CrewMemberEntry.TABLE_NAME, selection, selectionArgs);
for(int i = 0; i < crew.size(); i++)
{
    values.clear();
    values.put(PirateDBContract.CrewMemberEntry.COLUMN_NAME_SHIP_ID, String.valueOf(mShip.ID()));
    values.put(PirateDBContract.CrewMemberEntry.COLUMN_NAME_PIRATE_ID, String.valueOf(crew.get(i).ID()));
    db.insert(PirateDBContract.CrewMemberEntry.TABLE_NAME, null, values);
}

db.close();
finish();
```

Make sure the code works. To do this you will need to add some pirates to a crew, then save changes and then extract the data base from the emulator and take a look at the contents of the CrewMember table. If it works commit to SVN with an appropriate log message like *"Completed lab 6 task 13 Saved a pirate ship's crew to the database"*.

Now we want our crew to be displayed when we reopen the Ship. At the moment this doesn't happen because we didn't load the crew in the editShipActivity onCreate method. Take a look at that method, and before the crew is passed to the Ship member variable write some code to rebuild all the Pirates. To do this first we need to get all pirate IDs from the records in the CrewMember table that have the same Ship ID as our ship. Then for each pirate ID we need to get the information about the Pirates themselves from the Pirate table so that we can rebuild the Pirate objects and add them to the Ship member variable's crew.

First build the query to get the pirate IDs for crew members of this particular ship.

```
String[] crewProjection = {
        PirateDBContract.CrewMemberEntry.COLUMN_NAME_PIRATE_ID,
        PirateDBContract.CrewMemberEntry.COLUMN_NAME_SHIP_ID
};
selection = PirateDBContract.CrewMemberEntry.COLUMN_NAME_SHIP_ID + " = ?";

c = db.query(
        PirateDBContract.CrewMemberEntry.TABLE_NAME,
        crewProjection,
        selection,
        selectionArgs,
        null,
        null,
        null
);
```

Use the cursor object to extract each of those IDs and store them in an ArrayList.

```
c.moveToFirst();

ArrayList<String> crewIds = new ArrayList<>();

while(c.moveToNext())
{
    crewIds.add(c.getString(c.getColumnIndexOrThrow(PirateDBContract.CrewMemberEntry.COLUMN_NAME_PIRATE_ID)));
}

c.close();
```

Then create a new query to get the information of each pirate using the IDs in the ArrayList

```
String[] pirateProjection = {
        PirateDBContract.PirateEntry.COLUMN_NAME_NAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME,
        PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY
};

selection = PirateDBContract.PirateEntry._ID + " = ?";
for(int i = 0; i < crewIds.size(); i++) {

    selectionArgs[0] = String.valueOf(crewIds.get(i));

    c = db.query(
            PirateDBContract.PirateEntry.TABLE_NAME,
            pirateProjection,
            selection,
            selectionArgs,
            null,
            null,
            null
    );

    c.moveToFirst();
    String pirateName = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NAME));
    String nickname = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NICKNAME));
    String nationality = c.getString(c.getColumnIndexOrThrow(PirateDBContract.PirateEntry.COLUMN_NAME_NATIONALITY));
    crew.add(new Pirate(pirateName, nickname, nationality, Integer.parseInt(crewIds.get(i))));
    c.close();
}
```

And finally close the database, create our new Ship member variable including our new crew, use the crew to create a pirateAdapter and set that to the list view.

```
db.close();

mShip = new Ship(shipKey, Integer.parseInt(shipKey), crew);

PirateAdapter adapter = new PirateAdapter(this, android.R.layout.simple_list_item_1, crew);

ListView pirateListView = (ListView)findViewById(R.id.CrewListView);
pirateListView.setAdapter(adapter);
```

If it works commit to SVN with an appropriate log message like *"Completed lab 6 task 14 Loaded the pirate ship's crew from the database"*.

Summary

In this lab we looked briefly at how to parse JSON. Next we used and SQLite database to store and retrieve all our data, reusing several user interfaces to both add and edit Pirates and Ships.