
SUPER
SWEET
ANDROID
TIME

Level 4 – Section 1

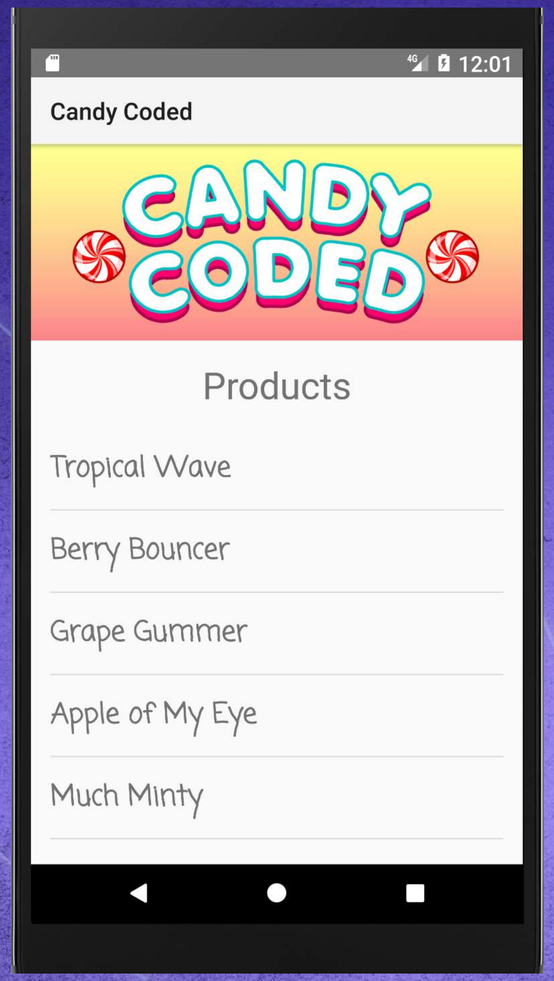
Storing Data

Using a SQLite Database

SUPER SWEET ANDROID TIME

Storing Candy Data in Our App

Why do we want to store Candy data in our app? Can't we always just do a network request?



Sometimes you want data offline

You can store app data on your device and read/write without an Internet connection

SUPER SWEET ANDROID TIME

Different Ways of Storing Data in Android Apps

A SharedPreferences **Object or a** SQLiteDatabase **are common ways to store data in Android apps.**

Shared Preferences

Stores small amounts of data in key/value pairs

: *"Location"*

Value: "Orlando"

SQLite Database

Stores structured data like our Candy data

Name	Price	Description	Image
Much Minty	4.50	This peppermint ...	...
So Fresh	5.50	The wintergreen ...	...
Uni-Pop	9.99	The sugary magic...	...
...			

Database Operations

To use a database we need to write some SQL commands to create a table and eventually perform operations like inserting, reading and deleting.

Name	Price	Description	Image
Much Minty	4.50	This peppermint ...	...
So Fresh	5.50	The wintergreen ...	...
Uni-Pop	9.99	The sugary magic...	...
...			

We can write SQL for creating a table in a Contract class

We'll cover all of the database concepts you need for this course
If you want learn more database concepts, check out our Try SQL Course!

SUPER SWEET ANDROID TIME

Defining a Contract Class

Just like defining a SQL database's organization in a schema, a Contract class is the place to define your database's structure in an Android app

Candy Contract

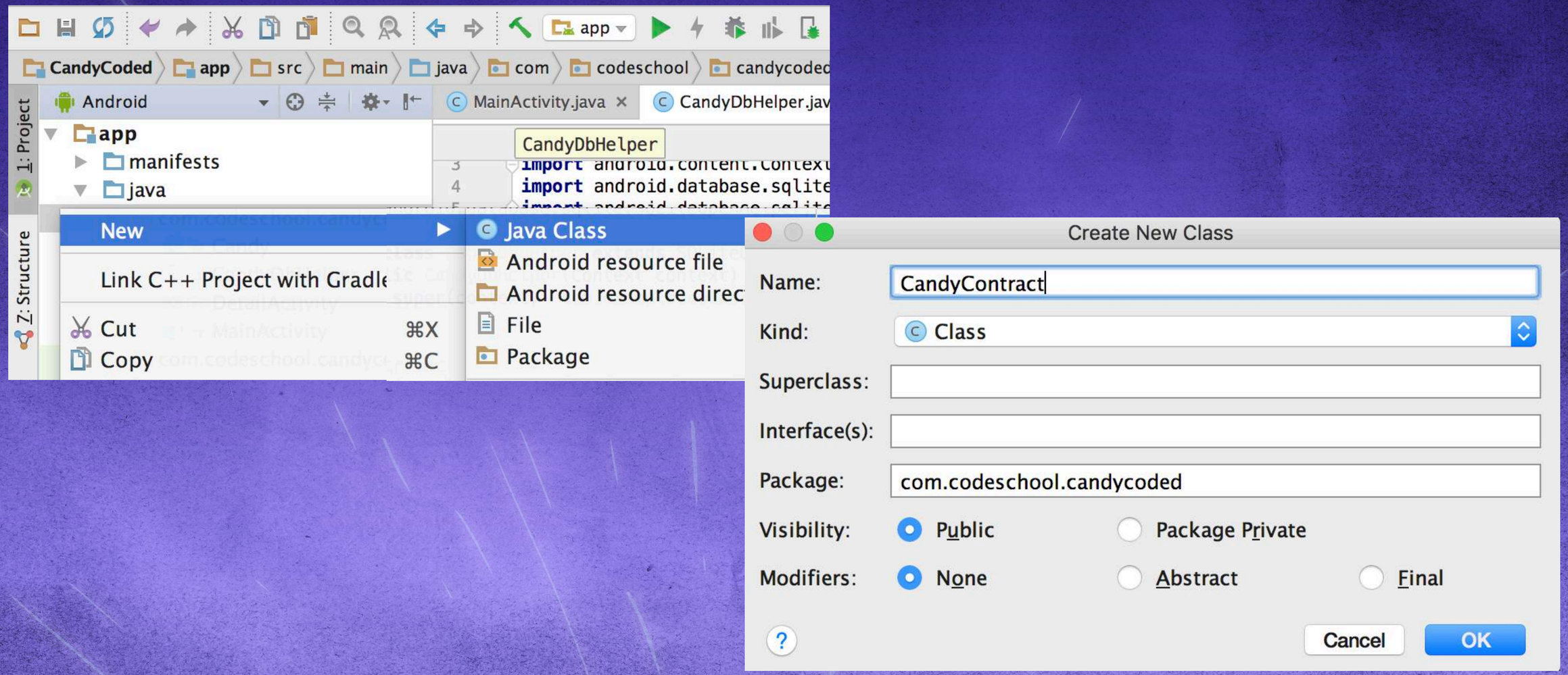
We are going to create a `Contract` class to define constants like the table and column names.

It can also act like a greeting card for other apps to know what data is provided.

SUPER SWEET ANDROID TIME

Creating our CandyContract Class

We will create a new Java class called CandyContract.



The CandyContract Class

We will define our database name, database version, table name, and column names in our CandyContract class.

CandyContract.java

```
public class CandyContract {
    public static final String DB_NAME = "candycoded.db";
    public static final int DB_VERSION = 1;

    public static class CandyEntry {

        Then create an inner class for each table that lists its columns.

    }
}
```

A good rule is to put definitions that are global to your whole database in the root level of the class

The CandyContract Class

By implementing the `BaseColumns` *Interface, your inner class can inherit a primary key field called _ID that we'll need later*

CandyContract.java

```
public class CandyContract {
    public static final String DB_NAME = "candycoded.db";
    public static final int DB_VERSION = 1;

    public static class CandyEntry implements BaseColumns {
        public static final String TABLE_NAME = "candy";
        public static final String COLUMN_NAME_NAME = "name";
        public static final String COLUMN_NAME_PRICE = "price";
        public static final String COLUMN_NAME_DESC = "description";
        public static final String COLUMN_NAME_IMAGE = "image";
    }
}
```

Creating a Database Table with Raw SQL

The Create Table command creates a table in our database with the specified columns and data types.

```
CREATE TABLE candy
(
  _ID INTEGER PRIMARY KEY,
  Name TEXT,
  Price TEXT,
  Description TEXT,
  Image TEXT
)
```

This SQL command would generate this table structure

_ID	Name	Price	Description	Image

Adding a Statement to Create the Candy Table

Once we have defined how our database looks, we can create SQL statements that create and maintain the database and tables.

CandyContract.java

```
public class CandyContract {
    public static final String DB_NAME = "candycoded.db";
    public static final int DB_VERSION = 1;

    public static final String SQL_CREATE_ENTRIES =
        "CREATE TABLE " + CandyEntry.TABLE_NAME + " (" +
        CandyEntry._ID + " INTEGER PRIMARY KEY," +
        CandyEntry.COLUMN_NAME_NAME + " TEXT," +
        CandyEntry.COLUMN_NAME_PRICE + " TEXT," +
        CandyEntry.COLUMN_NAME_DESC + " TEXT," +
        CandyEntry.COLUMN_NAME_IMAGE + " TEXT)";

    public static class CandyEntry implements BaseColumns {
        ...
    }
}
```

Adding a Statement to Drop the Candy Table

Once we have defined how our database looks, we can create SQL statements that create and maintain the database and tables.

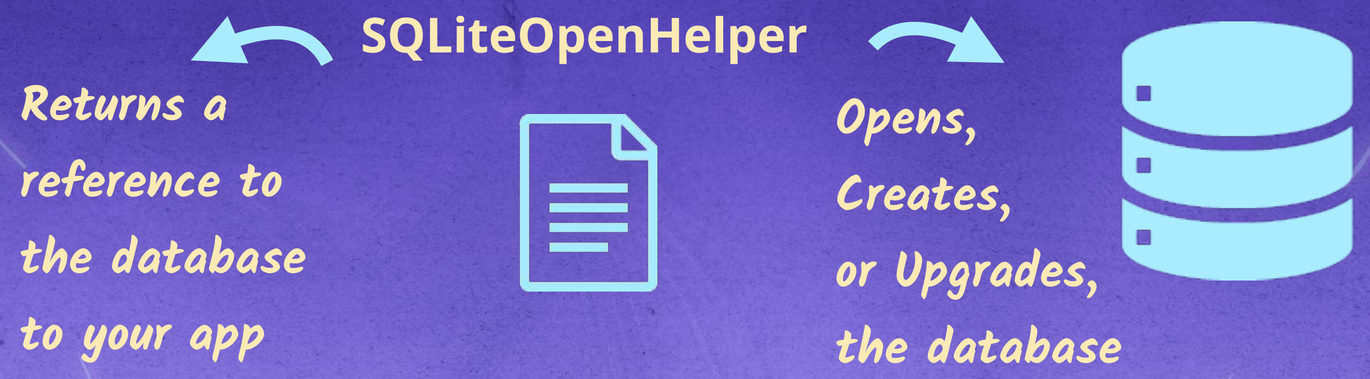
CandyContract.java

```
public class CandyContract {
    public static final String DB_NAME = "candycoded.db";
    public static final int DB_VERSION = 1;

    private static final String SQL_CREATE_ENTRIES = " ... ";

    private static final String SQL_DELETE_ENTRIES =
        "DROP TABLE IF EXISTS " + CandyEntry.TABLE_NAME;

    public static class CandyEntry implements BaseColumns {
        ...
    }
}
```

Now that we have our Schema, or Contract, set up we want to create our database...

Using SQLiteOpenHelper to Manage Database Creation

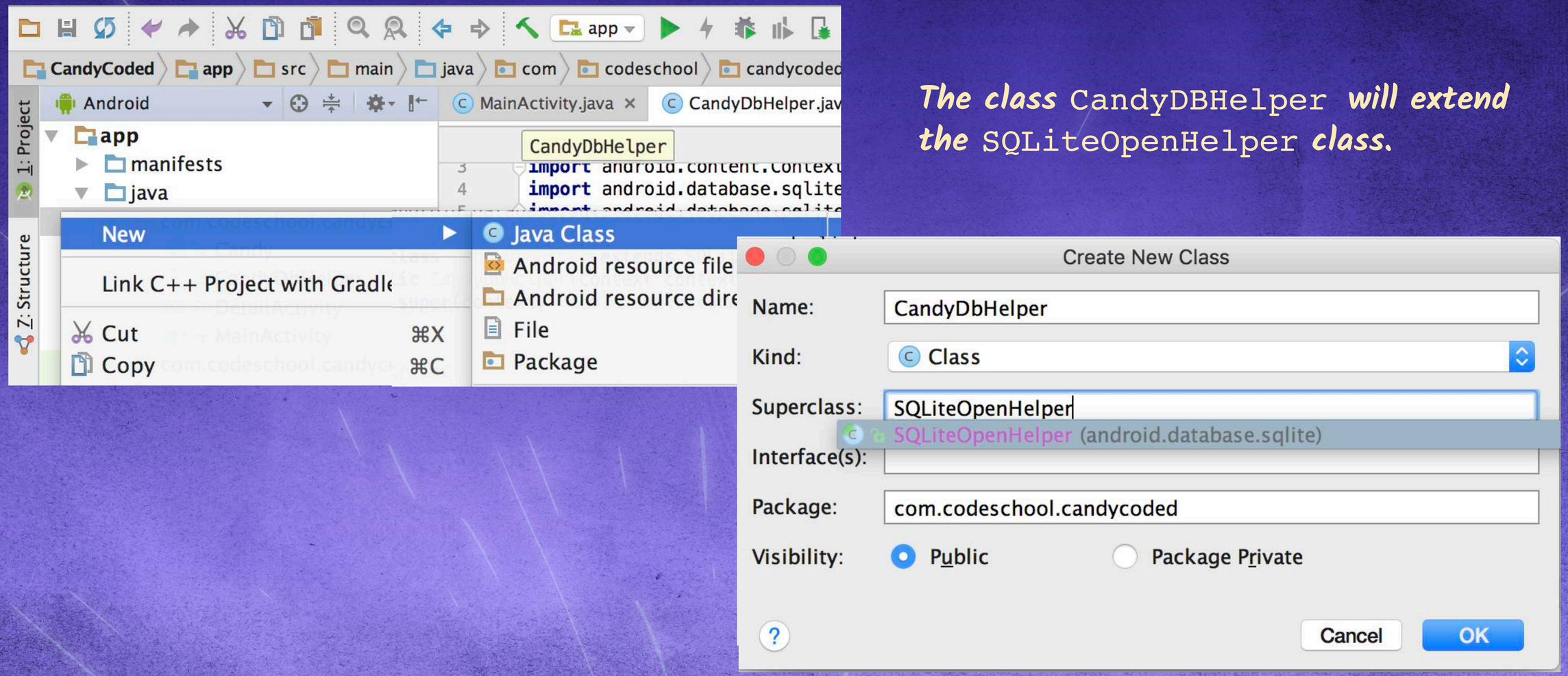
The SQLiteOpenHelper class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary.



SUPER SWEET ANDROID TIME

Creating Our Own SQLiteOpenHelper Subclass

To use the SQLiteOpenHelper class we need to create a new Java class called CandyDBHelper.



The class `CandyDBHelper` *will extend the* `SQLiteOpenHelper` *class.*

The Necessary Methods for the CandyDBHelper

The CandyDBHelper needs a constructor and needs to override the onCreate() and onUpgrade() methods.

CandyContract.java

```
package com.codeschool.candycoded;

import android.database.sqlite.SQLiteOpenHelper;

public class CandyDbHelper extends SQLiteOpenHelper {
    public CandyDbHelper(Context context) {...}

    @Override
    public void onCreate(SQLiteDatabase db) {...}

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

In this class we need to:

(1) Make a constructor that calls the `super()` *method*

(2) Override the `onCreate()` *method*

(3) Override the `onUpgrade()` *method*

Adding a Constructor

Our constructor will call the super() method which calls the SQLiteOpenHelper's constructor.

CandyContract.java

```
...

public class CandyDbHelper extends SQLiteOpenHelper {
    public CandyDbHelper(Context context) {
        super(context,
              CandyContract.DB_NAME ,
              null,
              CandyContract.DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {...}

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

We pass in the context, the database name, and database version

(`null` is for the optional `CursorFactory` which we won't be using)

Adding the onCreate() Method

CandyContract.java

```
...
public class CandyDbHelper extends SQLiteOpenHelper {
    public CandyDbHelper(Context context) {
        super(…);


    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CandyContract.SQL_CREATE_ENTRIES);
    }

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

The `onCreate()` method executes the statement we created earlier to create our table with the columns we have defined

Adding the onUpgrade() Method

CandyContract.java

```
public class CandyDbHelper extends SQLiteOpenHelper {
    public CandyDbHelper(Context context) {
        super(…);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CandyContract.SQL_CREATE_ENTRIES);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
        int oldVersion, int newVersion) {...}
        db.execSQL(CandyContract.SQL_DELETE_ENTRIES);
        onCreate(db);
    }
}
```

To upgrade our database we will simply execute our statement to delete the table and then re-create our table

Now we have everything in place to begin using our database to store data!

Level 4 – Section 2

Storing Data

Adding Values to a SQLite Database

SUPER SWEET ANDROID TIME

Saving Candies to the Database

We want to save Candies to the database after we get the Candy Array from the API. We can do this in the onSuccess() callback method as part of the AsyncHttpClient request.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
   ...
    @Override protected void onCreate(Bundle savedInstanceState) {
      ...
        AsyncHttpClient client = new AsyncHttpClient();
        client.get(…
            new TextHttpResponseHandler() {
            @Override public void onSuccess(…) {
                Gson gson = new GsonBuilder().create();
                Candy[] candies = gson.fromJson(response, Candy[].class);
                adapter.clear();
                for(Candy candy : candies) {
                    adapter.add(candy.name);
                }

            }
        ...
```

Remember the code we wrote for our network request?

This is where we'll add candies to our database

Saving Candies to the Database

We will create a method addCandiesToDatabase() to save each Candy to the database.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
   ...
    @Override protected void onCreate(Bundle savedInstanceState) {
      ...
        AsyncHttpClient client = new AsyncHttpClient();
        client.get(…
            new TextHttpResponseHandler() {
            @Override public void onSuccess(…) {
                Gson gson = new GsonBuilder().create();
                Candy[] candies = gson.fromJson(response, Candy[].class);
                adapter.clear();
                for(Candy candy : candies) {
                    adapter.add(candy.name);
                }
                addCandiesToDatabase(candies);
            }
```

We'll call this method like this. But now we have to create the method!

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private Candy[] candies;
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    This is where we'll define our addCandiesToDatabase() method

}
```

We'll also add a `CandyDbHelper` *variable that we can access throughout the* `MainActivity` *class*

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private Candy[] candies;
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    public void addCandiesToDatabase(Candy[] candies){

    }
}
```

We need to pass in our array of Candy objects so that we can add each one to the database

The return type is void since we don't need to return anything we're just adding to the database

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);
   ...
    public void addCandiesToDatabase(Candy[] candies){
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();



    }
...
```

We'll store a reference to our database in a **SQLiteDatabase** *variable*

To access our database, we can get a writeable database from our **candyDbHelper**

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);
   ...
    public void addCandiesToDatabase(Candy[] candies){
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();

        for(Candy candy : candies) {

            }
    }
...
```

Loop that reads each candy in our array of candies

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);
   ...
    public void addCandiesToDatabase(Candy[] candies){
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();

        for(Candy candy : candies) {
            ContentValues values = new ContentValues();
```

To insert values into the database we can use a `ContentValues` *object*

The `ContentValues` *class does some data validation and ensures data gets inserted in the correct format*

```
        }
    }
...
```

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);
   ...
    public void addCandiesToDatabase(Candy[] candies){
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();

        for(Candy candy : candies) {
            ContentValues values = new ContentValues();
            values.put(CandyEntry.COLUMN_NAME_NAME, candy.name);
```

We then put each column name and value into the `ContentValues` *object with the* `put()` *method*

```
        }
    }
...
```

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);
   ...
    public void addCandiesToDatabase(Candy[] candies){
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();

        for(Candy candy : candies) {
            ContentValues values = new ContentValues();
            values.put(CandyEntry.COLUMN_NAME_NAME, candy.name);
            values.put(CandyEntry.COLUMN_NAME_PRICE, candy.price);
            values.put(CandyEntry.COLUMN_NAME_DESC, candy.description);
            values.put(CandyEntry.COLUMN_NAME_IMAGE, candy.image);

            And we'll do this for the rest of our candy's properties
        }
    }
...
```

Saving Candies to the Database

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);
   ...
    public void addCandiesToDatabase(Candy[] candies){
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();

        for(Candy candy : candies) {
            ContentValues values = new ContentValues();
            values.put(CandyEntry.COLUMN_NAME_NAME, candy.name);
            values.put(CandyEntry.COLUMN_NAME_PRICE, candy.price);
            values.put(CandyEntry.COLUMN_NAME_DESC, candy.description);
            values.put(CandyEntry.COLUMN_NAME_IMAGE, candy.image);

            db.insert(CandyEntry.TABLE_NAME, null, values);
        }
    }
...
```

Finally we can use the `insert` *method to insert the row into our database*

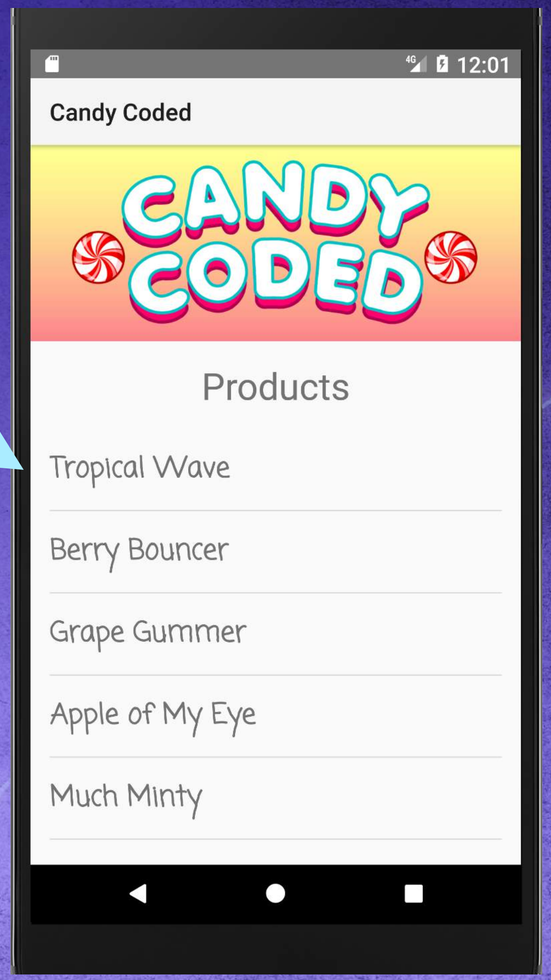
Now We Have Data in Our Database

Now that we have our candies in our database, the next step is to use them in our app.

Candy Database
Candy Table

_ID	Name	Price	Description	Image
1	Much Minty	4.50	This peppermint ...	...
2	So Fresh	5.50	The wintergreen ...	...
3	Uni-Pop	9.99	The sugary magic...	...
...	...	...	...	...

Next we need to query our database for candies and add them to our app




SUPER
SWEET
ANDROID
TIME