
SUPER
SWEET
ANDROID
TIME

Level 5 – Section 1

Querying a SQLite Database

Cursors & Cursor Adapters

SUPER SWEET ANDROID TIME

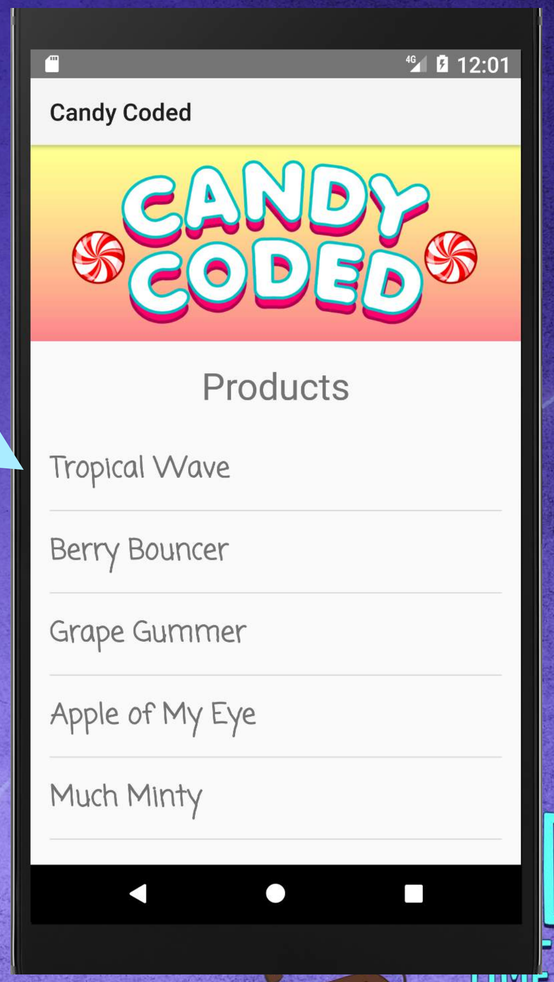
Now We Have Data in Our Database

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

Candy Database
Candy Table

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

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



Using Raw SQL to Get All of the Candies

We can get values from the table with Select.

Name	Price	Description	Image
So Fresh	5.50	The wintergreen ...	...
Uni-Pop	9.99	The sugary magic...	...

```
SELECT * FROM candy
```

```
+----+----------+-----+---+-------+
| Name      | Price | Desc | Image |
+----------+-------+------+-------+
| So Fresh | 4.50  | …    |  …    |
| Uni-Pop  | 5.50  | …    |  …    |
```

*SELECT * returns all of the columns and will return all of the candy rows which is what we want*

Querying Our SQLite Database

We can query a SQLiteDatabase object with the rawQuery() method like below:

```
Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
```

A `Cursor` *exposes the results from a query*

`db` *is our* `SQLiteDatabase`

We pass our SQL query as a `String` *to the* `rawQuery()` *method*

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

Our query will return all of the candy rows and we can navigate the resulting rows with our `Cursor`

SUPER SWEET ANDROID TIME

How a Cursor Works

Cursors store query result records in rows and have methods to access and iterate through the records.

```
Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
```

Cursor's initial position (i.e. -1)

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

How a Cursor Works

Cursors store query result records in rows and have methods to access and iterate through the records.

```
Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
```

After calling

`cursor.moveToNext()`

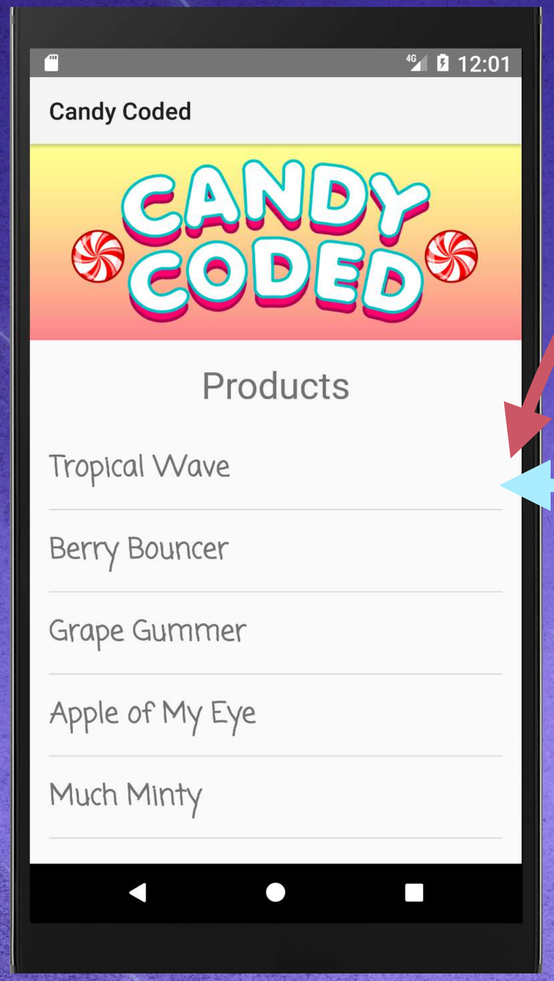
Name	Price	Description	Image
Much Minty	4.50	This peppermint ...	...
So Fresh	5.50	The wintergreen ...	...
Uni-Pop	9.99	The sugary magic...	...

```
int index = cursor.getColumnIndexOrThrow("name");
String candyName = cursor.getString(index);
```

`candyName` *is now "Much Minty"*

How Do We Get Our Candy Names in Our

Instead of using an ArrayAdapter to populate our ListView, we can use a CursorAdapter that gets data directly from our Database.



`ArrayAdapter<String>` ← `{"Tropical Wave", "Berry Bouncer", ...}`

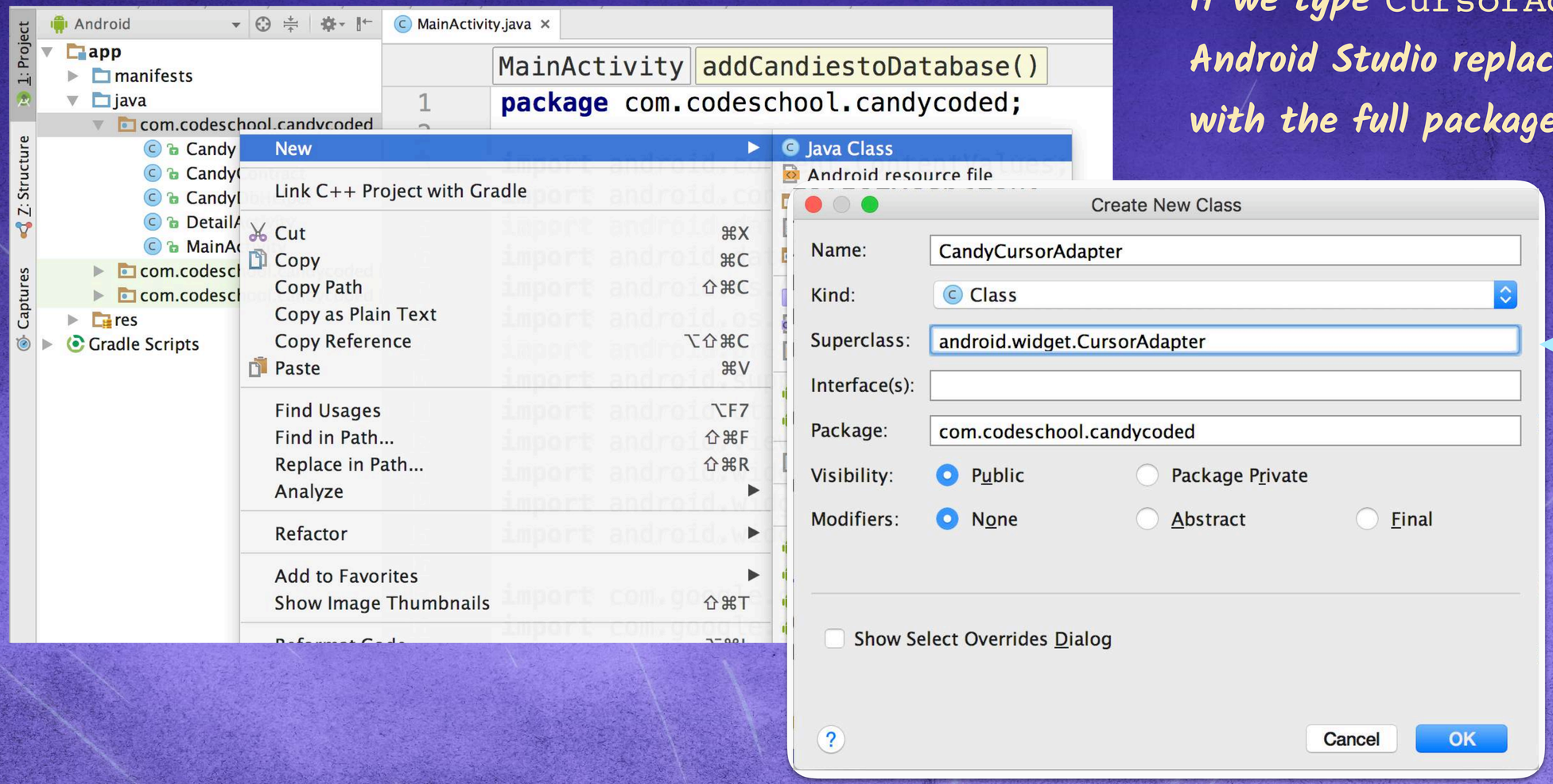
`CandyCursorAdapter` ←

Name	Price	Desc	Image
Much Minty	4.50	...	...
So Fresh	5.50	...	...
Uni-Pop	9.99	...	...

Creating a CursorAdapter Subclass

To use a CursorAdapter to populate our ListView we need to create a CursorAdapter Subclass, which we will name CandyCursorAdapter.

If we type `CursorAdapter`, *Android Studio replaces it with the full package name*



SUPER SWEET ANDROID TIME

The CandyCursorAdapter Class

Since CandyCursorAdapter extends CursorAdapter it has to have certain methods listed below.

In this class we need to:
(1) Make a constructor

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

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

CandyCursorAdapter.java

```
package com.codeschool.candycoded;
import …

public class CandyCursorAdapter extends CursorAdapter {

    public CandyCursorAdapter(Context context, Cursor c) {…}

    @Override
    public View newView(Context context, Cursor cursor,
                        ViewGroup parent) {…}

    @Override
    public void bindView(View view, Context context,
                         Cursor cursor) {…}
}
```

The CandyCursorAdapter Class

CandyCursorAdapter.java

```
…
public class CandyCursorAdapter extends CursorAdapter {

    public CandyCursorAdapter(Context context, Cursor c) {
        super(context, c, false);
    }

    @Override
    public View newView(Context context, Cursor cursor,
                        ViewGroup parent) {…}

    @Override
    public void bindView(View view, Context context,
                         Cursor cursor) {…}
}
```

The constructor will call the `super()` *method that needs the* `context`*, the* `Cursor`*, and* `false` *for re-query*

The CandyCursorAdapter Class

CandyCursorAdapter.java

```
…
public class CandyCursorAdapter extends CursorAdapter {

    public CandyCursorAdapter(Context context, Cursor c) {
        super(context, c, false);
    }

    @Override
    public View newView(Context context,
                        Cursor cursor, ViewGroup parent) {
        return LayoutInflater.from(context).inflate(
                R.layout.list_item_candy, parent, false);
    }

    @Override public void bindView(View view, Context context,
                        Cursor cursor) {…}
}
```

This method inflates the view and returns it so it can be displayed

We need to tell it to what layout to use: `list_item_candy` *layout we created in Try Android*

The CandyCursorAdapter Class

CandyCursorAdapter.java

```
…
public class CandyCursorAdapter extends CursorAdapter {
    public CandyCursorAdapter(Context context, Cursor c) {…}

    @Override public View newView(…) {…}

    @Override public void bindView(View view, Context context,
                                   Cursor cursor) {
        TextView textView = (TextView) view.findViewById(
                        Cursor cursor, R.id.text_view_candy);
        String candyName = cursor.getString(
                           cursor.getColumnIndexOrThrow("name"));
        textView.setText(candyName);

    }
}
```

In this method we set the elements of our view

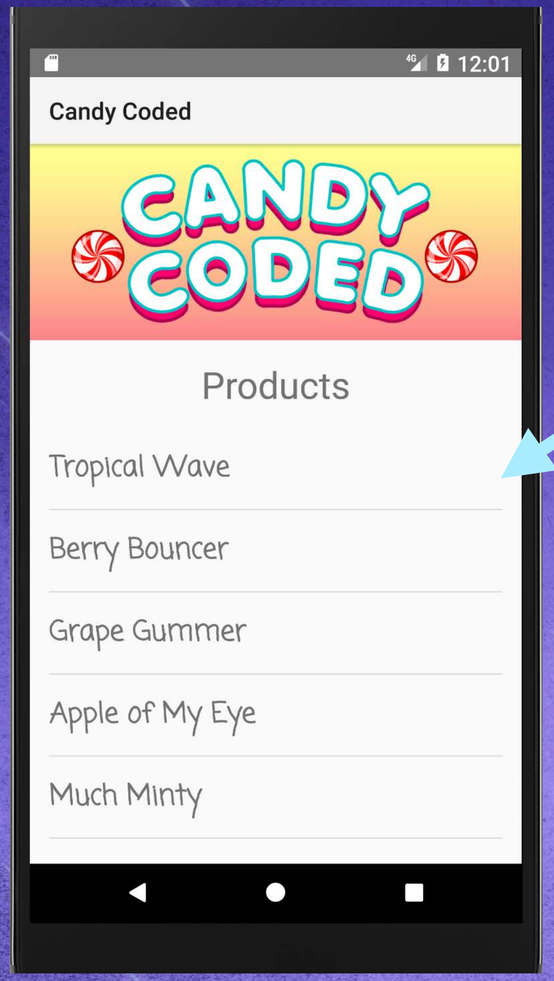
(1) We need to get `TextView` *to fill*

(2) Then we get the column in our database to fill it with

(3) Then we use the `setText()` *method to connect the two*

Using the CandyCursorAdapter to Put Candy in Our ListView

Now we can use our CursorAdapter to get data directly from our Database into our ListView.



`CandyCursorAdapter`

Name	Price	Desc	Image
Much Minty	4.50	...	...
So Fresh	5.50	...	...
Uni-Pop	9.99	...	...

To use our new `CandyCursorAdapter` *we need to refactor the code in* `MainActivity.java` *a bit*

SUPER SWEET ANDROID TIME

Refactoring Our Code to Use the CandyCursorAdapter

MainActivity.java

```
…
public class MainActivity extends AppCompatActivity {
    …

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        …
        final ArrayList<String> candyList = new ArrayList<String>();
        candy_list.add("Tropical Wave");
        …
        final ArrayAdapter<String> adapter = new ArrayAdapter<String>(…);
        …
    }
}
```

We can remove the code that creates the `ArrayList` *and the* `ArrayAdapter` *since we'll be using our* `Database` *and* `CandyCursorAdapter` *instead.*

Refactoring Our Code to Use the CandyCursorAdapter

MainActivity.java

```
…
public class MainActivity extends AppCompatActivity {
    private CandyCursorAdapter adapter;
    private CandyDbHelper candyDbHelper = new CandyDbHelper(this);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        …
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();
        Cursor cursor = db.rawQuery("SELECT * FROM candy", null);

        adapter = new CandyCursorAdapter(this, cursor);
        …
        listView.setAdapter(adapter);
        …
    }
}
```

We will first query for all candies

Then we can create a `CandyCursorAdapter` *from the results in the cursor*

Refactoring Our Code to Use the CandyCursorAdapter

MainActivity.java

```
...
    AsyncHttpClient client = new AsyncHttpClient();
    client.get(
        "https://....herokuapp.com/main/api",
        new TextHttpResponseHandler() {
            …
            @Override public void onSuccess(int status, Header[] headers,
                                            String response) {
                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 can also remove the code to update the ArrayAdapter *after our request*

Refactoring Our Code to Use the CandyCursorAdapter

MainActivity.java

```
...
    AsyncHttpClient client = new AsyncHttpClient();
    client.get(
        "https://....herokuapp.com/main/api",
        new TextHttpResponseHandler() {
            …
            @Override public void onSuccess(int status, Header[] headers,
                                            String response) {
                Gson gson = new GsonBuilder().create();
                Candy[] candies = gson.fromJson(response, Candy[].class);
                addCandiesToDatabase(candies);

                SQLiteDatabase db = candyDbHelper.getWritableDatabase();
                Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
                adapter.changeCursor(cursor);
            }
        });
...
```

Updating our CursorAdapter *with the latest Database entries*

Refactoring How We Pass Data to the DetailActivity

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    …
    @Override protected void onCreate(Bundle savedInstanceState) {
        …
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();
        Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
        adapter = new CandyCursorAdapter(this, cursor);

        listView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
            @Override public void onItemClick(
                    AdapterView<?> adapterView, View view, int i, long l) {
                Intent detailIntent = new Intent(this, DetailActivity.class);
                detailIntent.putExtra("position", i);
                startActivity(detailIntent);
            }
        });
    …
```

Instead of passing each candy property to the Detail Activity, we can pass just i and then query the database in the Detail Activity

Level 5 – Section 2

Querying a SQLite Database

Getting Data in the Detail Activity

SUPER SWEET ANDROID TIME

Refactoring the Detail Activity to Query Our Database

Instead of passing each of these values (the `candy name, price, image, and description`*) individually to the* `DetailActivity`*, we are passing the* `id` *to use in a database query*

SUPER SWEET ANDROID TIME

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
…
public class DetailActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        Intent intent = DetailActivity.this.getIntent();

        String candyName = "";
        if (intent != null && intent.hasExtra("candy_name")) {
            candyName = intent.getStringExtra("candy_name");
        }
        TextView textViewName = (TextView)this.findViewById(
                                    R.id.text_view_name);
        textViewName.setText(candyName);
        ...
```

Before we were passing in each value to display, but now we need to query the database instead

So we can remove this code that looked up the extra properties for the name, description, price and image

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
…
public class DetailActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        Intent intent = DetailActivity.this.getIntent();

        if (intent.hasExtra("position")) {

            First we'll make sure our intent has the
            extra "position" value

        }
```

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
…
public class DetailActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        Intent intent = DetailActivity.this.getIntent();

        if (intent.hasExtra("position")) {
            int position = intent.getIntExtra("position", 0);

        }
```

We can then get the position value, which is row that was selected in the **ListView**

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
...
    if (intent.hasExtra("position")) {
        int position = intent.getIntExtra("position", 0);
        CandyDbHelper candyDbHelper = new CandyDbHelper(this);
        SQLiteDatabase db = candyDbHelper.getWritableDatabase();
        Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
        cursor.moveToPosition(position);
```

We can query all candies and then move our cursor to the selected position

```
    }
...
```

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
...
    if (intent.hasExtra("position")) {
        int position = intent.getIntExtra("position", 0);
        ...
        Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
        cursor.moveToPosition(position);

        int columnIndex = cursor.getColumnIndexOrThrow(
                          CandyEntry.COLUMN_NAME_NAME);
```

We want the value in the name column so we can display the candy's name

```
    }
...
```

We can use the `getColumnIndexOrThrow()` *to get the index of our name column. The "*`orThrow`*" means it will throw an error if it's not there*

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
...
    if (intent.hasExtra("position")) {
        int position = intent.getIntExtra("position", 0);
        ...
        Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
        cursor.moveToPosition(position);

        int columnIndex = cursor.getColumnIndexOrThrow(
                          CandyEntry.COLUMN_NAME_NAME);
        String candyName = cursor.getString(columnIndex);

        Then we can get the candy's name with the
        getString() method and pass the columnIndex
    }
...
```

Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
...
    if (intent.hasExtra("position")) {
        int position = intent.getIntExtra("position", 0);
        ...
        Cursor cursor = db.rawQuery("SELECT * FROM candy", null);
        cursor.moveToPosition(position);
        String candyName  = cursor.getString(cursor.getColumnIndexOrThrow(
                            CandyEntry.COLUMN_NAME_NAME));
        String candyPrice = cursor.getString(cursor.getColumnIndexOrThrow(
                            CandyEntry.COLUMN_NAME_PRICE));
        String candyImage = cursor.getString(cursor.getColumnIndexOrThrow(
                            CandyEntry.COLUMN_NAME_IMAGE));
        String candyDesc  = cursor.getString(cursor.getColumnIndexOrThrow(
                            CandyEntry.COLUMN_NAME_DESC));
```

We want to do the same thing for the price, image, and description values

```
    }
...
```

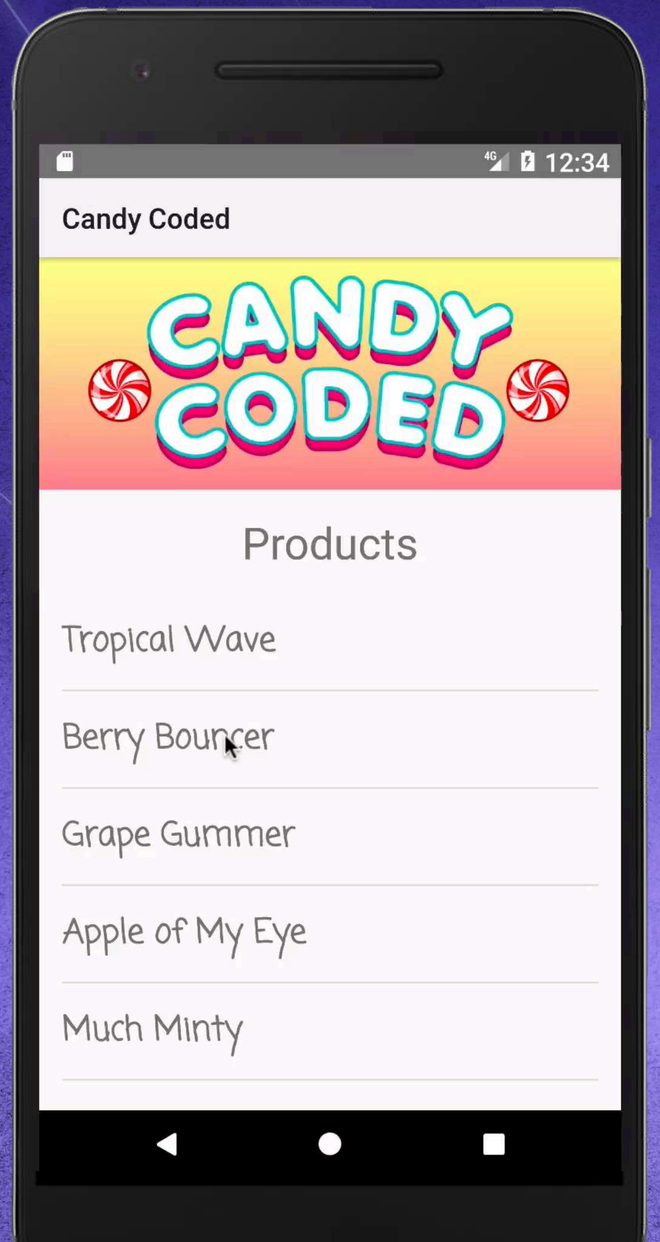
Refactoring the DetailActivity to Query the Database

DetailActivity.java

```
...
    if (intent.hasExtra("position")) {
        ...

        ((TextView) DetailActivity.this.findViewById(R.id.text_view_name))
                                        .setText(candyName);

        ((TextView) DetailActivity.this.findViewById(R.id.text_view_price))
                                        .setText(candyPrice);

        ((TextView) DetailActivity.this.findViewById(R.id.text_view_desc))
                                        .setText(candyDesc);

        Picasso.with(DetailActivity.this).load(candyImage).into(
        (ImageView) DetailActivity.this.findViewById(R.id.image_view_candy));
    }
...
```

This code still works like before because our name, price, description, and image variable names haven't changed

The Detail Activity Working with Our Database Data



Now if we run our app we can see our Detail Activity working with the data queried from our database!

SUPER SWEET ANDROID TIME


SUPER
SWEET
ANDROID
TIME