SUPER
SWEET
ANDROID
TIME

Level 2 – Section 1

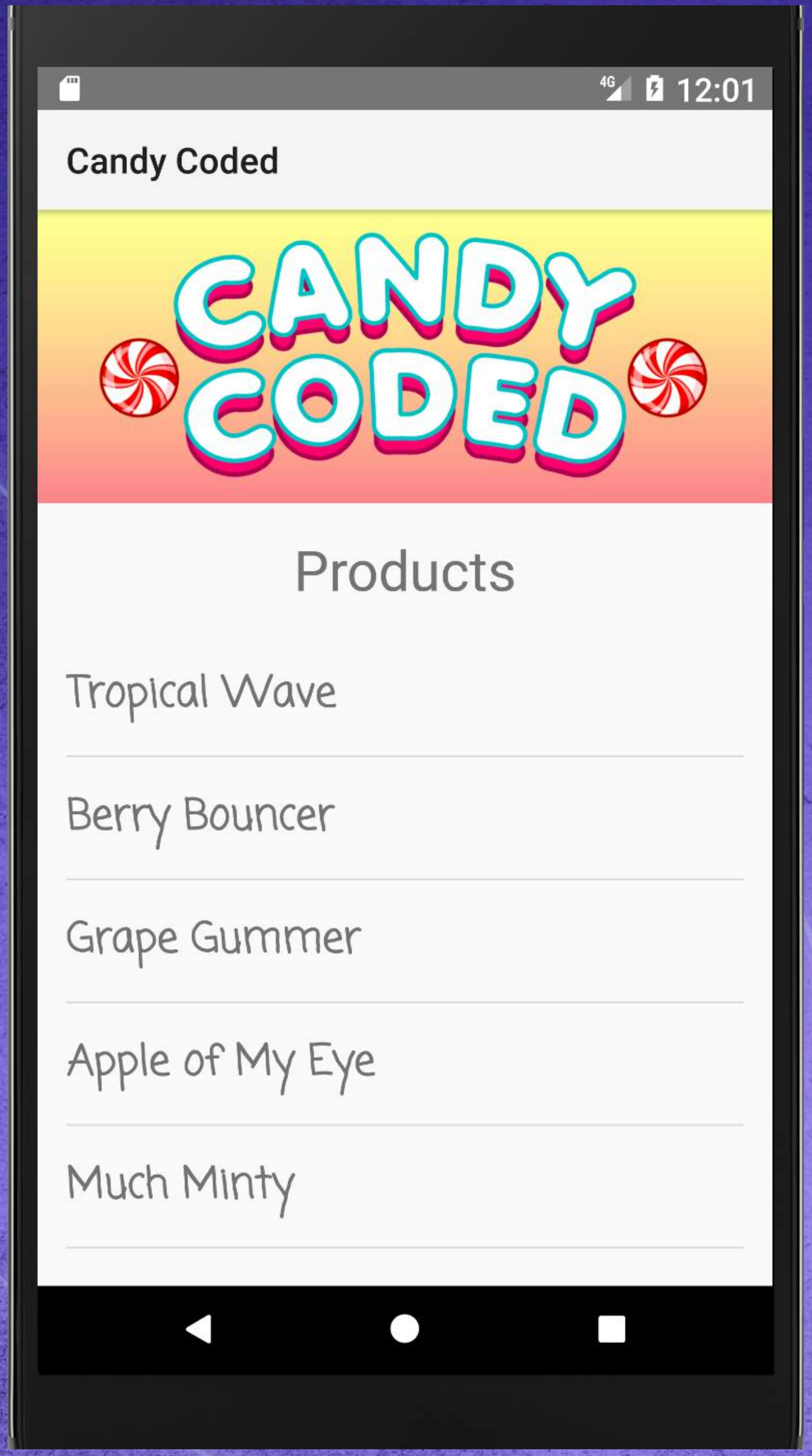
Dynamic Data

HTTP Requests with AsyncHttpClient

SUPER SWEET ANDROID TIME

Fetching Our Candy Data Dynamically

It doesn't make sense to hard-code our store's list of candy in the app. If the candy changes, the user would have to update their app for the list to update.



Candy Coded

Products

Tropical Wave

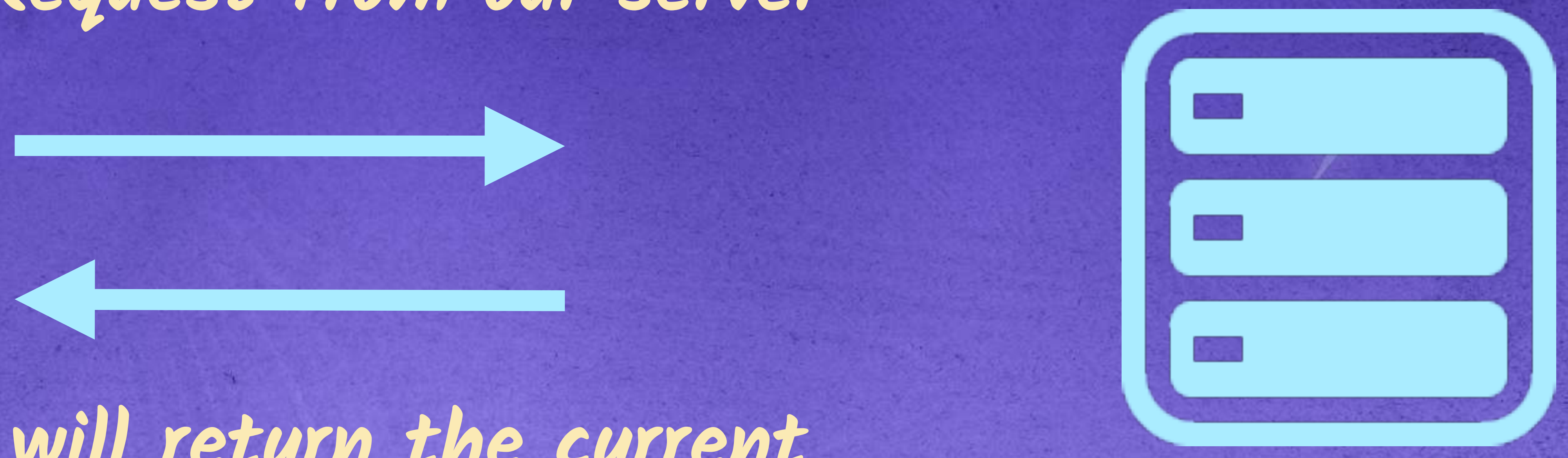
Berry Bouncer

Grape Gummer

Apple of My Eye

Much Minty

Instead, we want to fetch the latest candy data with an HTTP Request from our server



Which will return the current candies as a JSON response

Understanding Threads

When an application is launched, a thread of execution is created for the program.

Main Thread

Draws the layout to the screen

Creates the ArrayList of candy names

Create the ArrayAdapter

Populates the ListView with candy from the adapter

...

Creates Item Click Listener

A program thread runs commands sequentially

The exception is callback methods, like our `onItemClicked()` *method that gets called when a click triggers it*

Note: In Android, the main thread handles drawing to the screen and user interaction, so it's also called the UI thread

Understanding Threads

When an application is launched, a thread of execution is created for the program.

Main Thread

Draws the layout to the screen

Creates the ArrayList of candy names

Create the ArrayAdapter

Populates the ListView with candy from the adapter

🐌

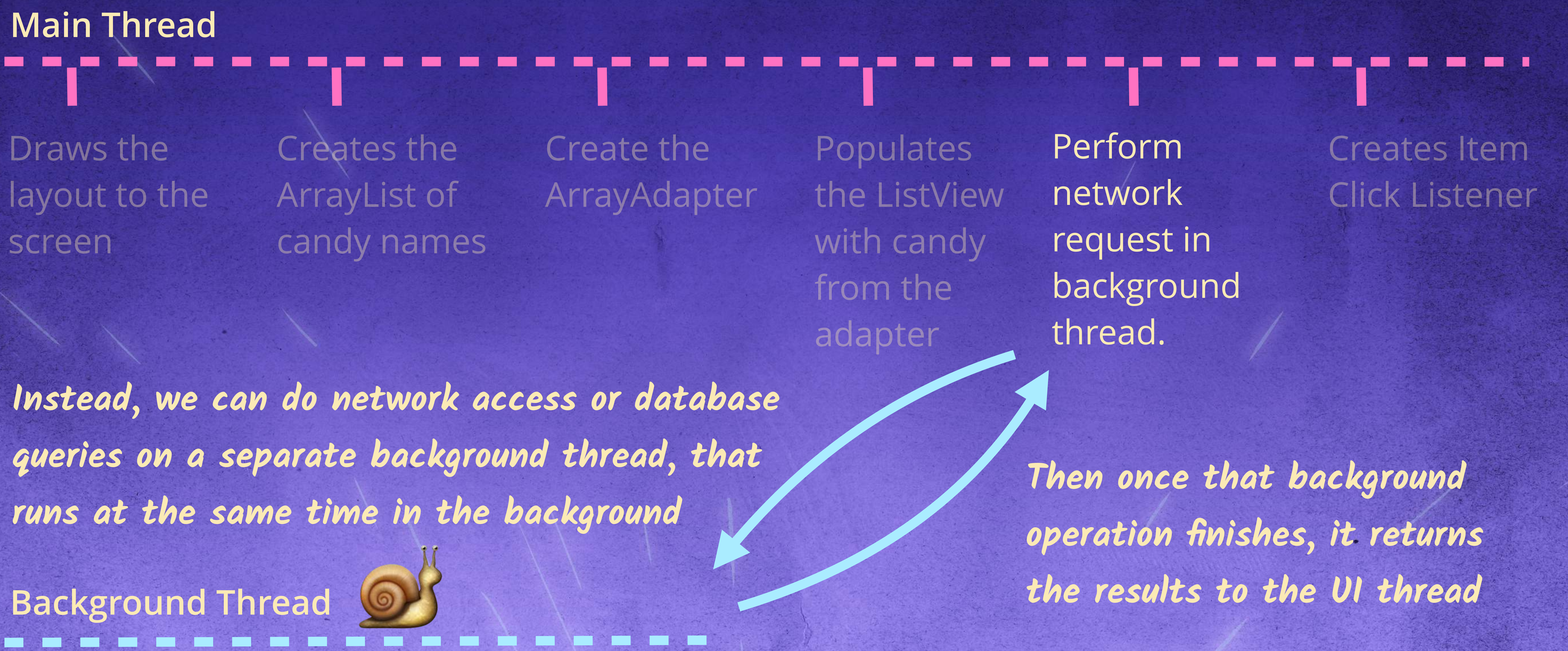
Slow operation here.

Creates Item Click Listener

If we also do time consuming operations, like network access or database queries, on the same thread, the whole UI would be blocked until those finish!

SUPER SWEET ANDROID TIME

Using a Separate Background Thread



Main Thread

Draws the layout to the screen

Creates the ArrayList of candy names

Create the ArrayAdapter

Populates the ListView with candy from the adapter

Perform network request in background thread.

Creates Item Click Listener

Instead, we can do network access or database queries on a separate background thread, that runs at the same time in the background

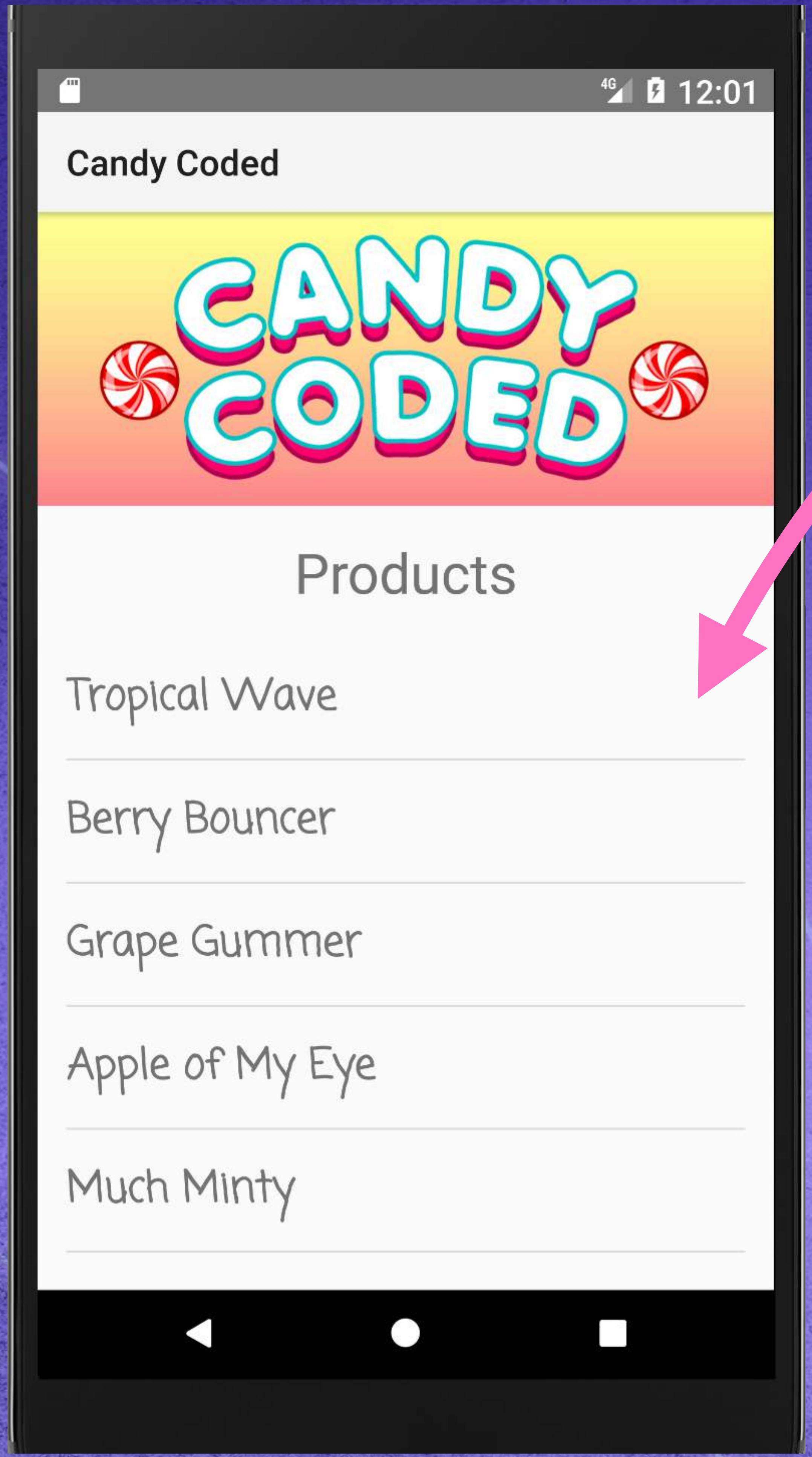
Then once that background operation finishes, it returns the results to the UI thread

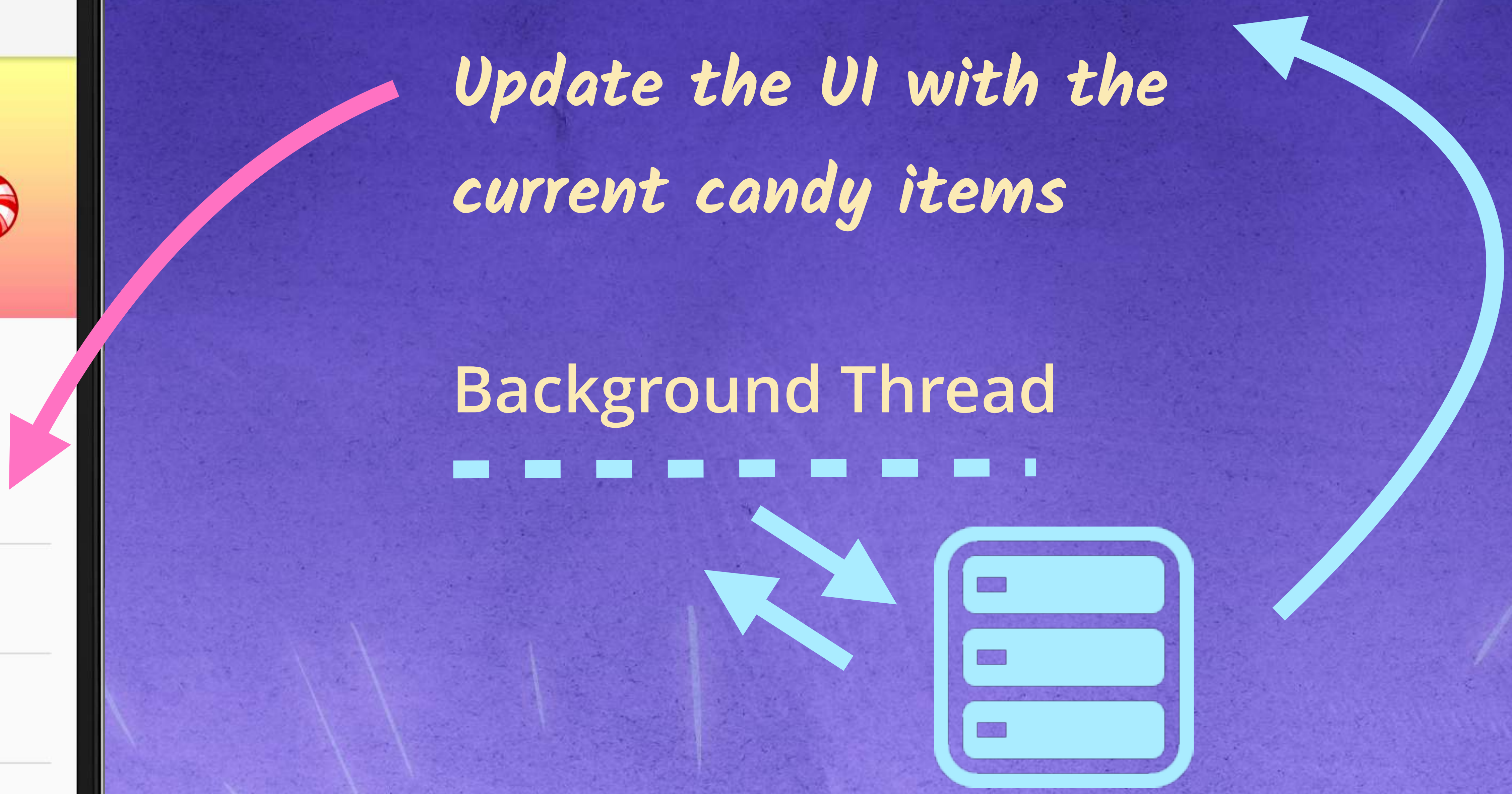
Background Thread 🐌

This way the UI continues running smoothly! ✓

Use AsyncHttpClient for Http Requests in the Background

Instead of writing the boilerplate code to create an Http Request in the background thread, the external Library Android Asynchronous Http Client has a class AsyncHttpClient that can do this for us.




Main UI Thread
Update the UI with the current candy items
Background Thread
Once AsyncHttpClient gets the result, it returns the current candy items to the UI thread
AsyncHttpClient can do the HTTP Request in the background to get the current Candy items

How to Use an External Library

To use an external library, like the Android Asynchronous Http Client **library, we need to add it as a dependency in our** build.gradle **file.**

build.gradle (app)

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 25 …
    defaultConfig {
        applicationId "com.codeschool.candycoded"
        minSdkVersion 10
        targetSdkVersion 25 …
    } …
}
dependencies {
    …
    compile 'com.android.support:appcompat-v7:25.3.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.loopj.android:android-async-http:1.4.9'
}
```

Add a reference to the remote repository for the external Library to our project by adding to the bottom of our dependencies.

Using the AsyncHttpClient Library

To use the AsyncHttpClient class to make an http request, we can create a new AsyncHttpClient object and make a request with the get() method.

MainActivity.java

```
import com.loopj.android.http.*
…
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        …
        AsyncHttpClient client = new AsyncHttpClient();
        client.get(…);

    }
}
```

Android Studio should import this automatically, but if it doesn't we need this line

Create our AsyncHttpClient object

The get() method will do our request, but it has a lot in it so we'll cover that next

Using the AsyncHttpClient Library

MainActivity.java

```
AsyncHttpClient client = new AsyncHttpClient();
client.get(
        "https://....herokuapp.com/main/api",
        new TextHttpResponseHandler() {

            A TextHttpResponseHandler needs
            to implement 2 callback methods -
            onSuccess() and onFailure()

        });
```

The URL:

go.codeschool.com/CandyAPI

A Response Handler object

Using the AsyncHttpClient Library

MainActivity.java

```
AsyncHttpClient client = new AsyncHttpClient();
client.get(
        "https://....herokuapp.com/main/api",
        new TextHttpResponseHandler() {
```

Inside the `onSuccess()` *callback method we'll just log the JSON response for now as a debug message with* `Log.d()`

```
            @Override
            public void onSuccess(int statusCode, Header[] headers,
                                              String response) {
                Log.d("AsyncHttpClient", "response = " + response);
            }
        });
```

Using the AsyncHttpClient Library

MainActivity.java

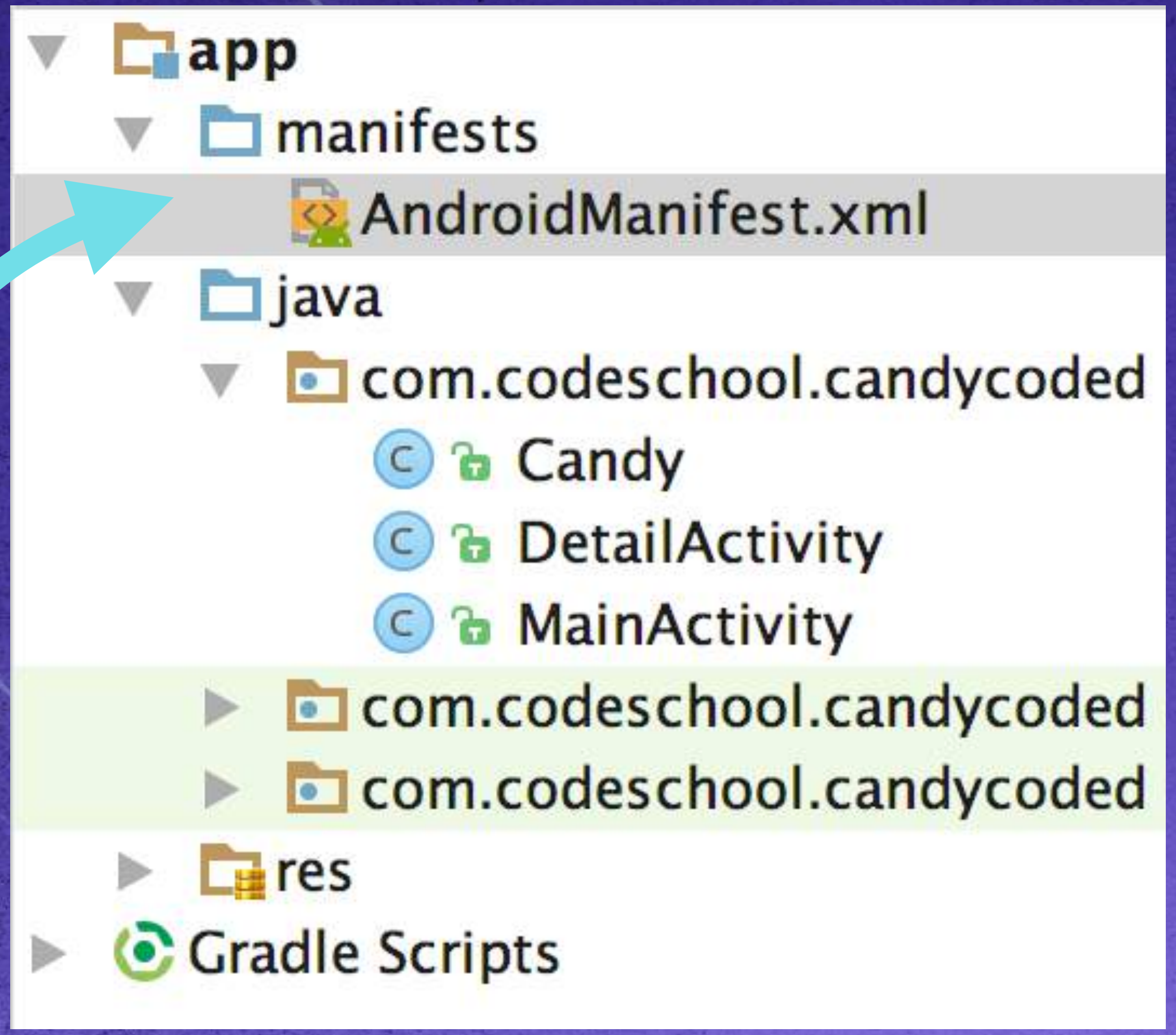
```
AsyncHttpClient client = new AsyncHttpClient();
client.get(
        "https://....herokuapp.com/main/api",
        new TextHttpResponseHandler() {
            @Override
            public void onFailure(int statusCode, Header[] headers,
                                  String response, Throwable throwable) {
                Log.e("AsyncHttpClient", "response = " + response);
            }

            @Override
            public void onSuccess(int statusCode, Header[] headers,
                                  String response) {
                Log.d("AsyncHttpClient", "response = " + response);
            }
        });
```

For a failure we'll log the same thing but with `Log.e()` *for errors*

Allowing Internet Access in the Manifest

The manifest file provides essential information about your app to the Android system.



Every application must have an `AndroidManifest.xml` *file (with precisely that name) in its root directory*

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="…"
    package="com.codeschool.candycoded">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.Light">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name=
                 "android.intent.action.MAIN" />
                  ...
```

Allowing Internet Access in the Manifest

We'll add the Internet permission to the top of the Manifest file.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="…"
    package="com.codeschool.candycoded">

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

    <application ...
</manifest>
```

`<uses-permission>` *requests a permission that the app needs.*

Permissions are granted by the user while the app is running

Screencast: Logging Debug Messages

SUPER SWEET ANDROID TIME

Level 2 – Section 2

Dynamic Data

Parsing JSON

SUPER SWEET ANDROID TIME

How to Parse a Simple JSON String

The JSON the Candy API returns has more information, but let’s start with a simple example that only has the name of 1 candy.

```
String response =  "[
                     {
                         "name": "Tropical Wave",
                     }
                    ]"
```

A **JSONArray** *is inside* `[]`

A **JSONObject** *is inside* `{}`

Each value has a key

```
JSONArray candyArray = new JSONArray(candyJsonStr);
JSONObject jsonObj = candyArray.getJSONObject(0);
String candyName = jsonObj.getString("name");
```

After this code runs, `candyName` *is* `"Tropical Wave"`

SUPER SWEET ANDROID TIME

The Actual JSON Response Has More Information

The JSON the Candy API actually returns looks like this.

A JSONArray is inside [ ]

A JSONObject is inside { }

Keys for each property

```
[
  {
    "id": 1,
    "name": "Tropical Wave",
    "image": "https://s3.amazonaws.com/.../gumdrops-1.png",
    "price": "5.99",
    "description": "These tropical-flavored gummies …",
  },
  {
    "id": 2,
    "name": "Berry Bouncer",
    "image": "https://s3.amazonaws.com/.../gumdrops-2.png",
    "price": "4.99",
    "description": "Berry delicious! This …",
  },
  …
]
```

Since we have multiple properties for each Candy, we want a way to group these together…

Storing the Candy Data in a Candy Class

We want to group the candy properties - name, image, price and description - together as one object. We can do that by creating a Candy class.

JSON data for each Candy

```
{
    "id": 1,
    "name": "Tropical Wave",
    "image": "https://s3.amazonaws.com/.../gumdrops-1.png",
    "price": "5.99",
    "description": "These tropical-flavored gummies …",
}
```

Candy *class*

```
public class Candy {
    public int id;
    public String name;
    public String imageURL;
    public String price;
    public String description;
}
```

Let's map JSON data to a Java class

Screencast: How to Create a New Class

Using the GSON Library to Parse the JSON for Us

We could write Java code to loop over every JSONObject in the JSONArray and get each Candy's properties into this class, but the GSON Library will do this for us.

JSON data for each Candy

```
{
    "id": 1,
    "name": "Tropical Wave",
    "image": "https://s3.amazonaws.com/.../gumdrops-1.png",
    "price": "5.99",
    "description": "These tropical-flavored gummies …"
}
```

Candy class

```
public class Candy {
    public int id;
    public String name;
    public String image;
    public String price;
    public String description;
}
```

The GSON Library can automatically convert JSON objects into Candy objects for us!

For GSON to work without any other configuration, the property names in your class need to match up exactly with the keys in your JSON file.

Adding the GSON Library as a Dependency

To use the GSON Library, we need to add it as a dependency in our build.gradle file.

build.gradle (app)

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 25 …
    defaultConfig {
        applicationId "com.codeschool.candycoded"
        minSdkVersion 10
        targetSdkVersion 25 …
    } …
}
dependencies {
    …
    compile 'com.loopj.android:android-async-http:1.4.9'
    compile 'com.google.code.gson:gson:2.8.0'
}
```

Add a reference to the GSON Library to our project by adding to the bottom of our dependencies

How to Use GSON

The GSON Library converts JSON strings to Java objects. But note that this will only work if your class properties match the JSON keys, otherwise GSON requires customization.

```
String response = "{
                     "id": 1,
                     "name": "Tropical Wave",
                     "image": "https://.../gumdrops-1.png",
                     "price": "5.99",
                     "description": "These tropical-flavored gummies …"
                  }";

Gson gson = new GsonBuilder().create();
Candy candy = gson.fromJson(response, Candy.class);
```

After this code is run all of candy's properties are set:

`candy.name`	*is*	`"Tropical Wave"`
`candy.price`	*is*	`5.99`

etc.

The JSON string

The class we want to convert to

SUPER SWEET ANDROID TIME

How to Use GSON for an Array of Objects

```
String response = "[{"id": 1,
                     "name": "Tropical Wave",
                     "image": "https://.../gumdrops-1.png",
                     "price": "5.99",
                     "description": "These tropical-flavored gummies …"},

                    {"id": 2,
                     "name": "Tropical Wave",
                     "image": "https://.../gumdrops-1.png",
                     "price": "5.99",
                     "description": "These tropical-flavored gummies …”},
                    …
                   ]";

Gson gson = new GsonBuilder().create();
Candy[] candies = gson.fromJson(response, Candy[].class);
```

We can just specify we have an Array of Candy *objects*

Adding our GSON Code to Our AsyncHttpClient Request

We'll add our GSON code to create an Array of Candy objects after we get our JSON response.

MainActivity.java

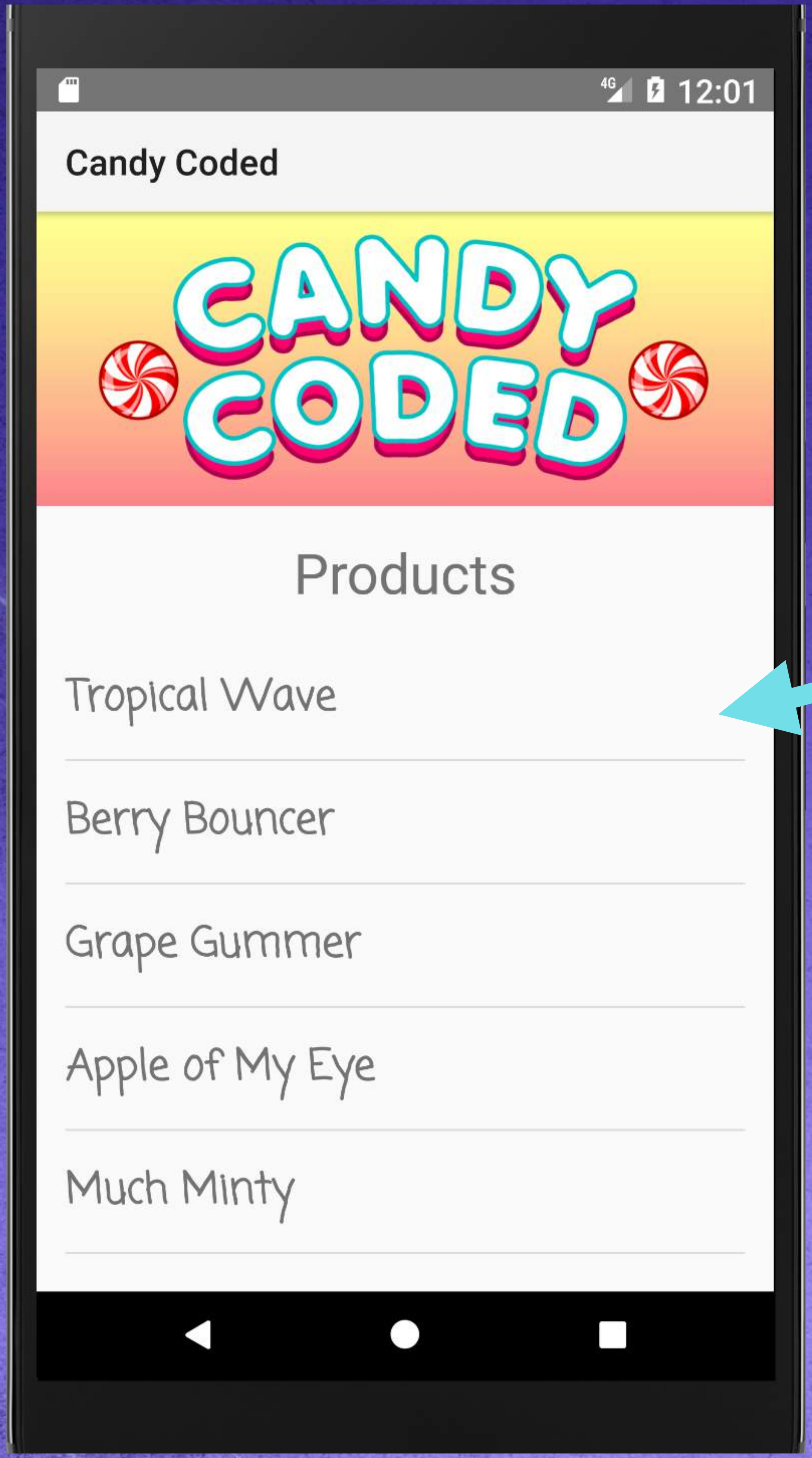
```
AsyncHttpClient client = new AsyncHttpClient();
client.get(
        “https://....herokuapp.com/main/api”,
        new TextHttpResponseHandler() {
            @Override public void onFailure(...) {}

            @Override public void onSuccess(int status, Header[] headers,
                                            String response) {
                Log.d("AsyncHttpClient", "response = " + response);
                Gson gson = new GsonBuilder().create();
                Candy[] candies = gson.fromJson(response, Candy[].class);
            }
        });
```

Now we have all our `Candy` *objects, let's update the adapter to use the* `Candy` *names.*

Adding the Candy Names to the Adapter

Now we have all our Candy objects, let's update the adapter to use the Candy names.



Tropical Wave
Berry Bouncer
Grape Gummer
Apple of My Eye
Much Minty

`candies`

We can put each `Candy`*'s name in our* `ListView` *by adding each* `candy.name` *to the* `Adapter`.

Adding our Candy Names to Our ListView's Adapter

MainActivity.java

```
AsyncHttpClient client = new AsyncHttpClient();
client.get(
        "https://....herokuapp.com/main/api",
        new TextHttpResponseHandler() {
            @Override public void onFailure(...) {}

            @Override public void onSuccess(int status, Header[] headers,
                                            String response) {
                Log.d("AsyncHttpClient", "response = " + response);
                Gson gson = new GsonBuilder().create();
                Candy[] candies = gson.fromJson(response, Candy[].class);
                adapter.clear();
                for(Candy candy : candies) {
                    adapter.add(candy.name);
                }
            }
        });
```

First clear what's in the `adapter` *now*

Then add each `candy`*'s name*

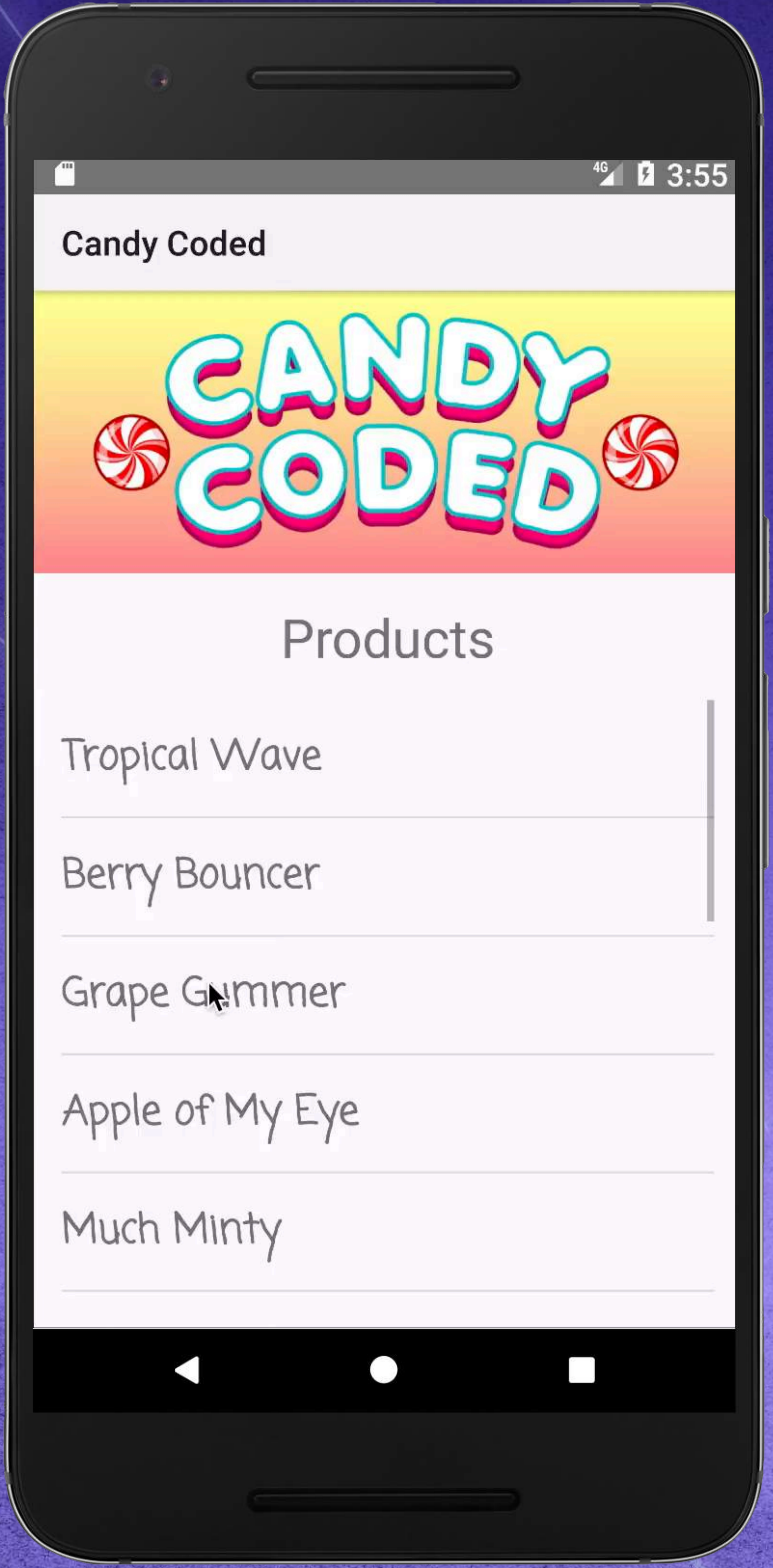
Screencast: The ListView Updating in the Background

Level 2 – Section 3

Dynamic Data

Passing Dynamic Data to the Detail Activity

Pass Live Data to the Detail Activity



Right now our Detail Activity is only showing the candy name

We also want to display the image, price and description we got from the Candy API

To do that we'll also need to pass the image, price and description from the array of Candy objects and send it to the Detail Activity

We Want to Pass the Candy Values with the DetailActivity Intent

We also want to pass the Candy's image URL, price, and description.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private Candy[] candies;
    …
    @Override protected void onCreate(Bundle savedInstanceState) {
        …
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener(){
            @Override
            public void onItemClick(AdapterView<?> …) {
                Intent detailIntent = new Intent(this, DetailActivity.class);
                detailIntent.putExtra("candy_name", candy_list.get(i));

                startActivity(detailIntent);
            }
        });
    …
```

We also want to use the Candy's array throughout the entire class, so we'll declare it here.

This is where we want to pass the Candy's image URL, price, and description to the Detail Activity

Passing the Candy Values to the DetailActivity Intent

We can pass the Candy's image URL, price, and description with the putExtra() method.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private Candy[] candies;
    …
    @Override protected void onCreate(Bundle savedInstanceState) {
        …
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener(){
            @Override
            public void onItemClick(AdapterView<?> …) {
                Intent detailIntent = new Intent(this, DetailActivity.class);
                detailIntent.putExtra("candy_name", candies[i].name);
                detailIntent.putExtra("candy_image", candies[i].imageURL);
                detailIntent.putExtra("candy_price", candies[i].price);
                detailIntent.putExtra("candy_desc", candies[i].description);
                startActivity(detailIntent);
            }
        });
    …
```

Getting the Data from the Intent in DetailActivity

Back in the DetailActivity class, we can get the image, price, and description after we displayed the candy name in the textView and then display them.

DetailActivity.java

```
...
    Intent intent = DetailActivity.this.getIntent();
    String candyName = "";

    if (intent.hasExtra("candy_name")) {
        candyName = intent.getStringExtra("candy_name");
    }

    TextView textView = (TextView)this.findViewById(R.id.name_text_view);
    textView.setText(candyName);
    This is where we can display the image, price and description
...
```

First, let's Log the image URL, price and description to make sure they're coming in correctly

Logging the DetailActivity Intent's Data

After we get our values with getStringExtra(), we'll log all of the values to make sure we got them correctly before we worry about displaying them in the layout.

DetailActivity.java

```
...
    String candyImage = intent.getStringExtra("candy_image");
    String candyPrice = intent.getStringExtra("candy_price");
    String candyDesc = intent.getStringExtra("candy_desc");

    Log.d("DetailActivity", "Intent data: " + candyImage + ", " +
                  candyPrice + ", " + candyDesc);
...
```

We can get our passed in values from the `Intent` *with the same* `getStringExtra()` *method we used for the name*

Then we'll log these properties with `Log.d()` *to make sure they look correct*

Logging the DetailActivity Intent's Data

We also want to add some checks to make sure these values actually exist to prevent errors.

DetailActivity.java

```
...
    String candyImage = "";
    if (intent.hasExtra("candy_image"))
        candyImage = intent.getStringExtra("candy_image");

    String candyPrice = 0;
    if (intent.hasExtra("candy_price"))
        candyPrice = intent.getStringExtra("candy_price");

    String candyDesc = "";
    if (intent.hasExtra("candy_desc"))
         candyDesc = intent.getStringExtra("candy_desc");

    Log.d("DetailActivity", "Intent data: " + candyImage + ", " +
                            candyPrice + ", " + candyDesc);
...
```

Screencast: Logging the DetailActivity Intent's Data

SUPER
SWEET
ANDROID
TIME