

Android Application Development Lab 2

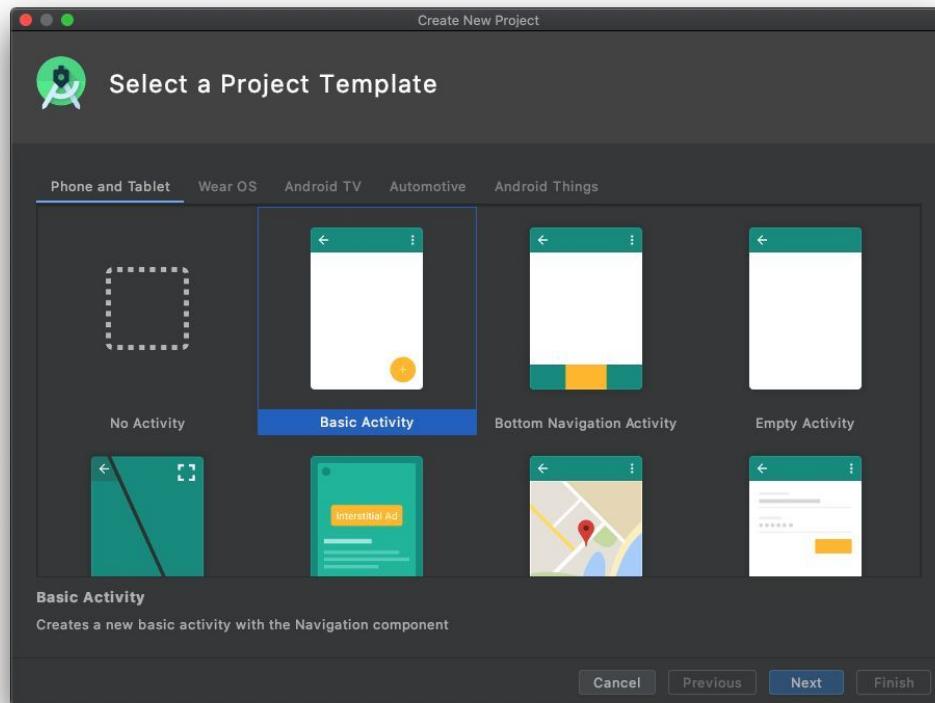
Human-Computer Interaction, AUEB
Εαρινό εξάμηνο 2024-2025

Lab Assistant: Sofia Eleftheriou

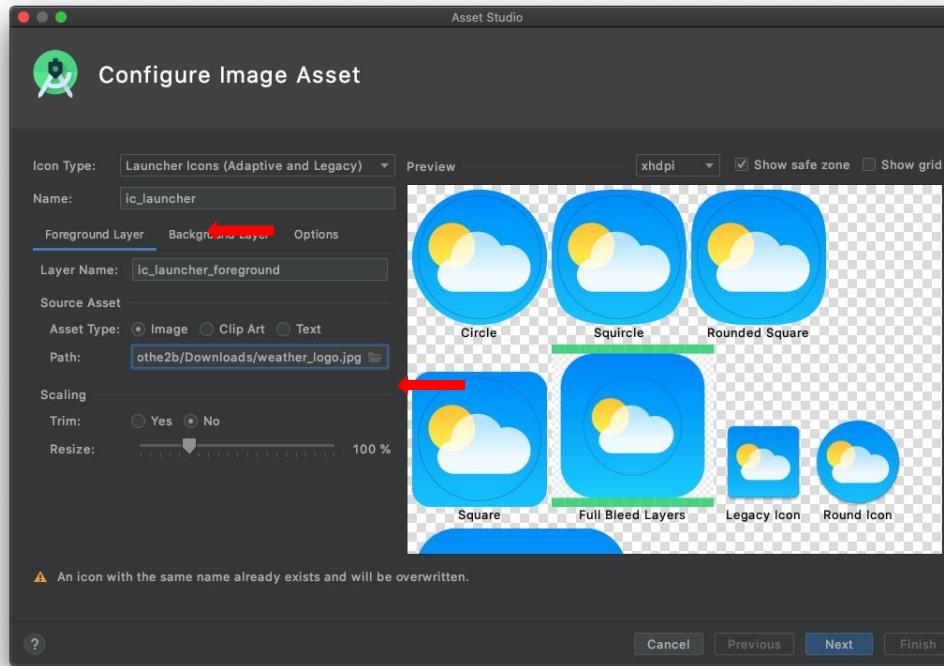
Android Development Advanced Use

- Catch up with Lab 1
- Refactor Project components
 - Rename Activity
 - Make property publicly available
- Use Public Forecast API
 - OpenWeatherMap API
 - REST call to API - JSON response
 - Create Asynchronous Task to call API
 - Add new Menu Option to refresh information
 - Update Business Logic to use the Asynchronous task - Involve the Refresh Option
 - Demonstrate updates

Catch up with Lab 1



Create new project with Basic Activity



Change Application Icon

The screenshot shows the Android Studio interface with two XML files open:

- `fragment_second.xml`: Contains a `ConstraintLayout` with a child `ListView`. The `ListView` has attributes: `layout_width="match_parent"`, `layout_height="match_parent"`, and `android:id="@+id/listView_forecast"`.
- `list_item_forecast.xml`: A single item layout for the `ListView`, which contains a `TextView` with the text "Weather forecast".

The preview window on the right shows a white screen next to a teal-colored `Weather forecast` card.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondFragment">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/listView_forecast" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Add ListView to layout

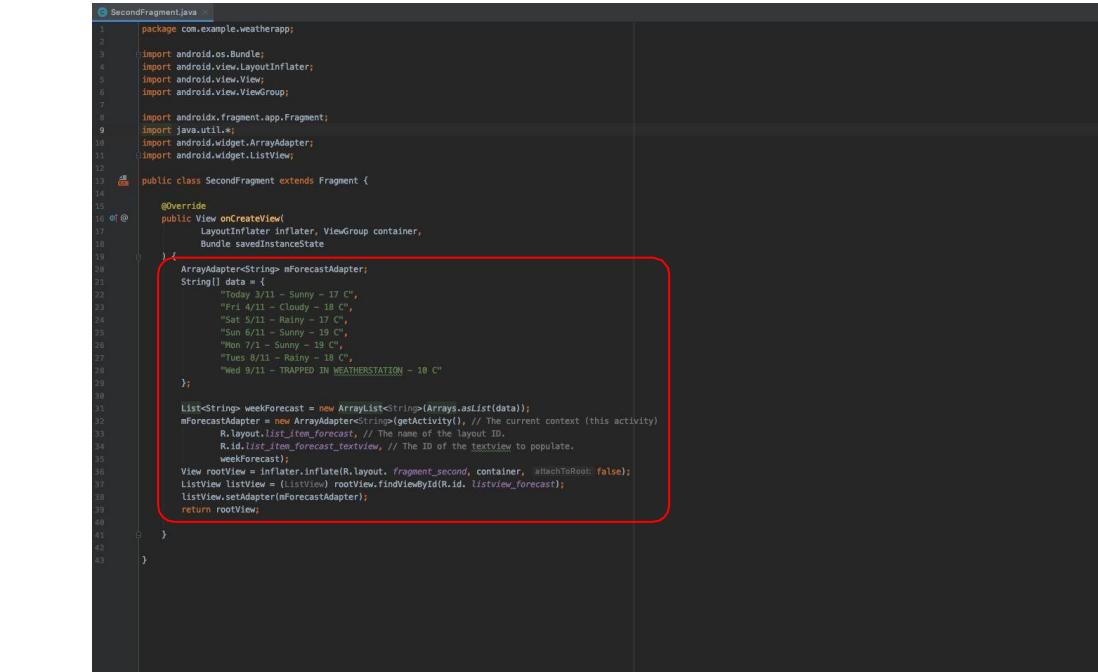
The screenshot shows the Android Studio interface with two XML files open:

- `fragment_second.xml`: Contains a single `ConstraintLayout` with a white background.
- `list_item_forecast.xml`: Contains a `ConstraintLayout` with a teal background. Inside it is a `TextView` with the ID `@+id/list_item_forecast_textview`.

In the Design tab, there are two views displayed side-by-side. The left view is white, and the right view is teal. The bottom navigation bar shows the following icons: Component Tree, Attributes, and a magnifying glass icon.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/list_item_forecast_textview"
        android:layout_width="match_parent" android:layout_height="match_parent"
        android:minHeight="?android:attr/listPreferredItemHeight"
        android:gravity="center_vertical"
        android:id="@+id/list_item_forecast_textview">
    </TextView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

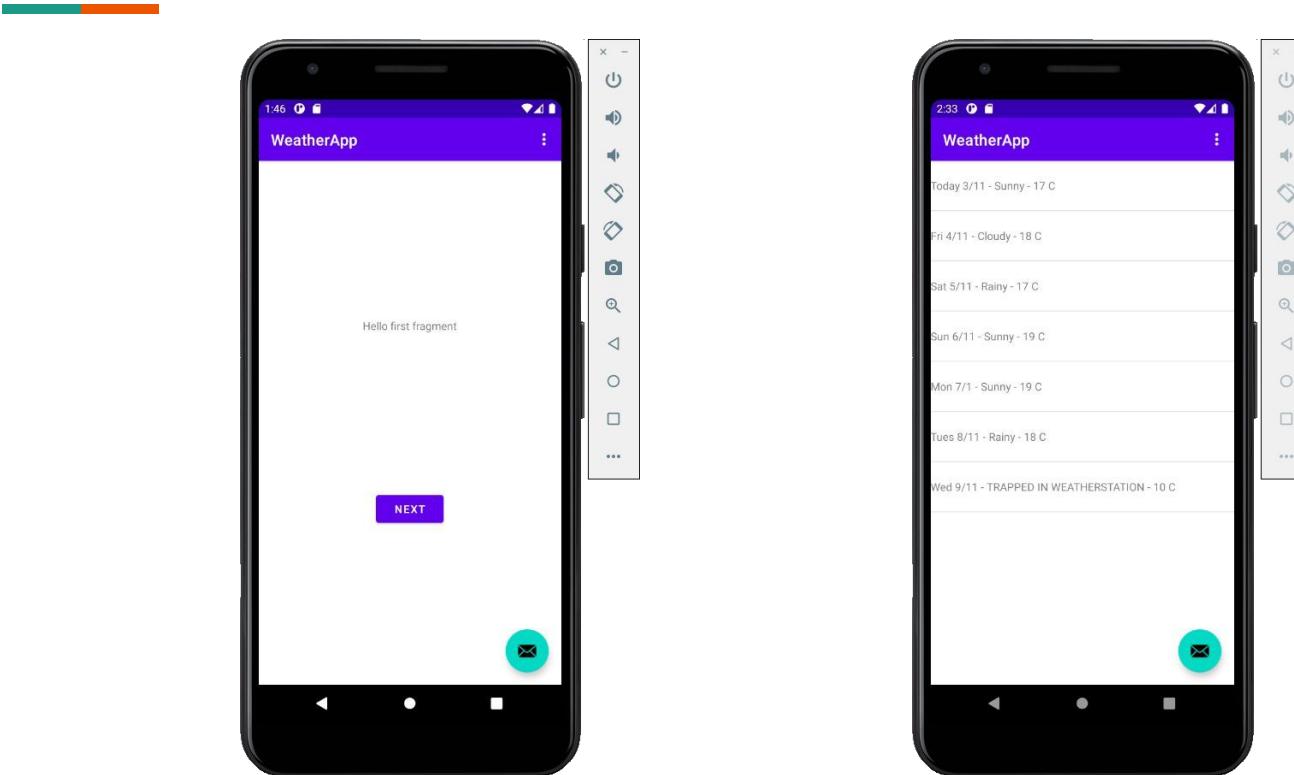
Add new layout with `TextView` for list items



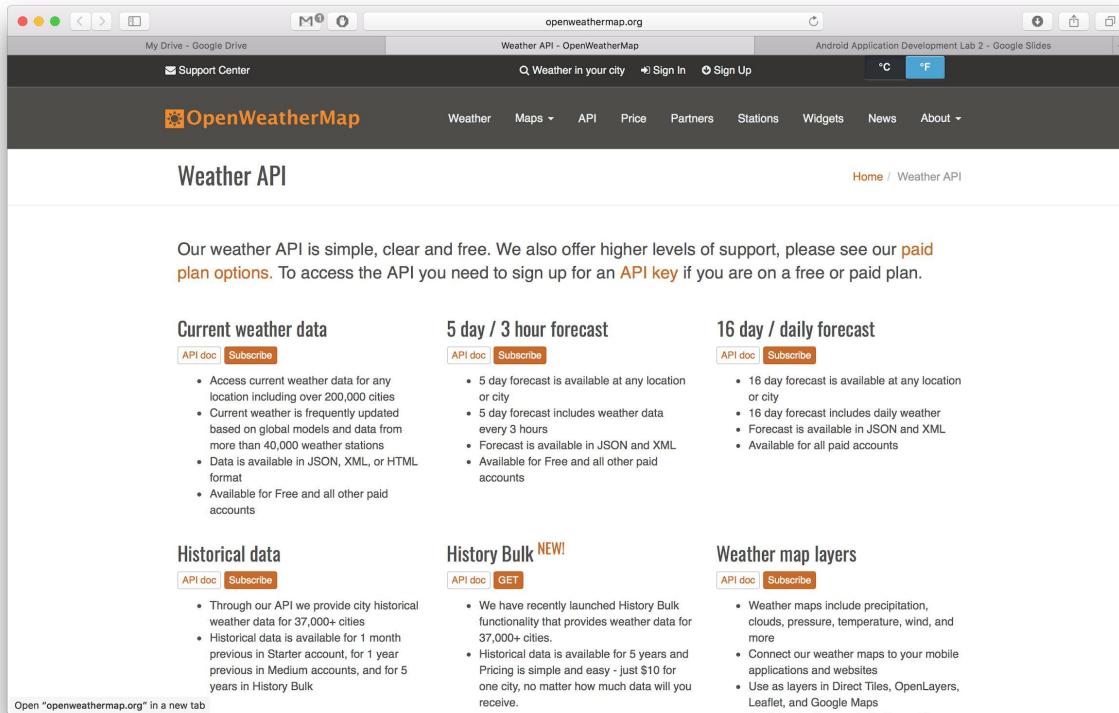
```
SecondFragment.java
```

```
1 package com.example.weatherapp;
2
3 import android.os.Bundle;
4 import android.view.LayoutInflater;
5 import android.view.View;
6 import android.view.ViewGroup;
7
8 import androidx.fragment.app.Fragment;
9 import java.util.*;
10 import android.widget.ArrayAdapter;
11 import android.widget.ListView;
12
13 public class SecondFragment extends Fragment {
14
15     @Override
16     public View onCreateView(
17             LayoutInflater inflater, ViewGroup container,
18             Bundle savedInstanceState
19     ) {
20         ArrayAdapter<String> mForecastAdapter;
21         String[] data = {
22             "Today 3/11 - Sunny - 17 °C",
23             "Fri 4/11 - Cloudy - 18 °C",
24             "Sat 5/11 - Rainy - 17 °C",
25             "Sun 6/11 - Sunny - 19 °C",
26             "Mon 7/11 - Sunny - 19 °C",
27             "Tue 8/11 - Rainy - 18 °C",
28             "Wed 9/11 - TRAPPED IN WEATHERSTATION - 10 °C"
29         };
30
31         List<String> weekForecast = new ArrayList<String>(Arrays.asList(data));
32         mForecastAdapter = new ArrayAdapter<String>(getActivity(), // The current context (this activity)
33             R.layout.list_item_forecast, // The name of the layout ID.
34             R.id.list_item_forecast_textview // The ID of the TextView to populate.
35         );
36         weekForecast.setAdapter(mForecastAdapter);
37
38         View rootView = inflater.inflate(R.layout.fragment_second, container, false);
39         ListView listView = (ListView) rootView.findViewById(R.id.listView_forecast);
40         listView.setAdapter(mForecastAdapter);
41         return rootView;
42     }
43 }
```

New business logic / Populate list with mock (fake) data



OpenWeatherMap API



The screenshot shows the OpenWeatherMap Weather API page. At the top, there's a navigation bar with links for Support Center, Weather, Maps, API, Price, Partners, Stations, Widgets, News, and About. Below the navigation is a search bar with placeholder text "Weather in your city". To the right of the search bar are "Sign In" and "Sign Up" buttons, and temperature units "°C" and "°F". The main content area has a dark header "OpenWeatherMap" and a sub-header "Weather API". A breadcrumb trail "Home / Weather API" is visible. The page content is organized into several sections:

- Current weather data**: Includes "API doc" and "Subscribe" buttons. A list of features:
 - Access current weather data for any location including over 200,000 cities
 - Current weather is frequently updated based on global models and data from more than 40,000 weather stations
 - Data is available in JSON, XML, or HTML format
 - Available for Free and all other paid accounts
- 5 day / 3 hour forecast**: Includes "API doc" and "Subscribe" buttons. A list of features:
 - 5 day forecast is available at any location or city
 - 5 day forecast includes weather data every 3 hours
 - Forecast is available in JSON and XML
 - Available for Free and all other paid accounts
- 16 day / daily forecast**: Includes "API doc" and "Subscribe" buttons. A list of features:
 - 16 day forecast is available at any location or city
 - 16 day forecast includes daily weather
 - Forecast is available in JSON and XML
 - Available for all paid accounts
- Historical data**: Includes "API doc" and "Subscribe" buttons. A list of features:
 - Through our API we provide city historical weather data for 37,000+ cities
 - Historical data is available for 1 month previous in Starter account, for 1 year previous in Medium accounts, and for 5 years in History Bulk
- History Bulk NEW!**: Includes "API doc" and "GET" buttons. A list of features:
 - We have recently launched History Bulk functionality that provides weather data for 37,000+ cities.
 - Historical data is available for 5 years and Pricing is simple and easy - just \$10 for one city, no matter how much data you will receive.
- Weather map layers**: Includes "API doc" and "Subscribe" buttons. A list of features:
 - Weather maps include precipitation, clouds, pressure, temperature, wind, and more
 - Connect our weather maps to your mobile applications and websites
 - Use as layers in Direct Tiles, OpenLayers, Leaflet, and Google Maps

At the bottom left, there's a link "Open "openweathermap.org" in a new tab".

<http://openweathermap.org>

The screenshot shows a web browser window with the URL openweathermap.org in the address bar. The page title is "How to start - OpenWeatherMap". The main content area has a dark header with the "OpenWeatherMap" logo and navigation links for Weather, Maps, API, Price, Partners, Stations, Widgets, News, and About. Below the header, there's a section titled "How to start" with the sub-section "How to get API key (APPID)". A "Sign up" button is visible. The main text in this section states: "To get access to weather API you need an API key whatever account you choose from Free to Enterprise." There's also a "How to use API key in API call" section with a "Description:" paragraph and a link to "http://api.openweathermap.org/data/2.5/forecast?id=524901&APPID=(APIKEY)". The footer contains a "How to get accurate API response" link.

How to start

How to get API key (APPID)

[Sign up](#) to get unique API key on your account page

How to use API key in API call

Description:

To get access to weather API you need an API key whatever account you chose from Free to Enterprise.

Activation of an API key for Free and Startup plans takes 10 minutes. For other tariff plans it is 10 to 60 minutes.

We keep right to not to process API requests without API key.

API call:

[http://api.openweathermap.org/data/2.5/forecast?id=524901&APPID=\(APIKEY\)](http://api.openweathermap.org/data/2.5/forecast?id=524901&APPID=(APIKEY))

Parameters:

APPID {APIKEY} is your unique API key

Example of API call:

<http://api.openweathermap.org/data/2.5/forecast?id=524901&APPID=1111111111>

How to get accurate API response

Register & Get API key

REST Call to API - JSON response



- **What we would like to know?**

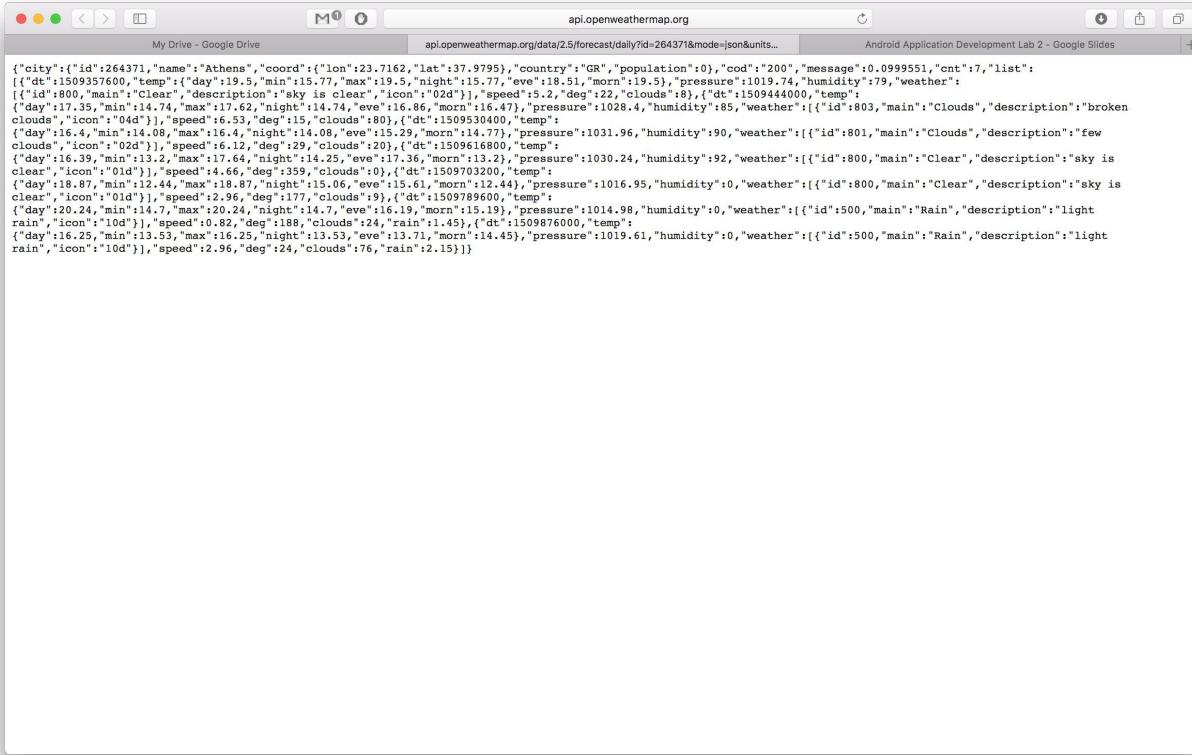
We would like to have the weather forecast for 1 week

<http://api.openweathermap.org/data/2.5/forecast/dail>
y

- **Which parameter we need to specify?**

- City id (Athens - 264371)
- Format (JSON)
- Unit: (Metric system)
- APPID: 612ce9a4c7726a3a0a00a69b84b9a01a

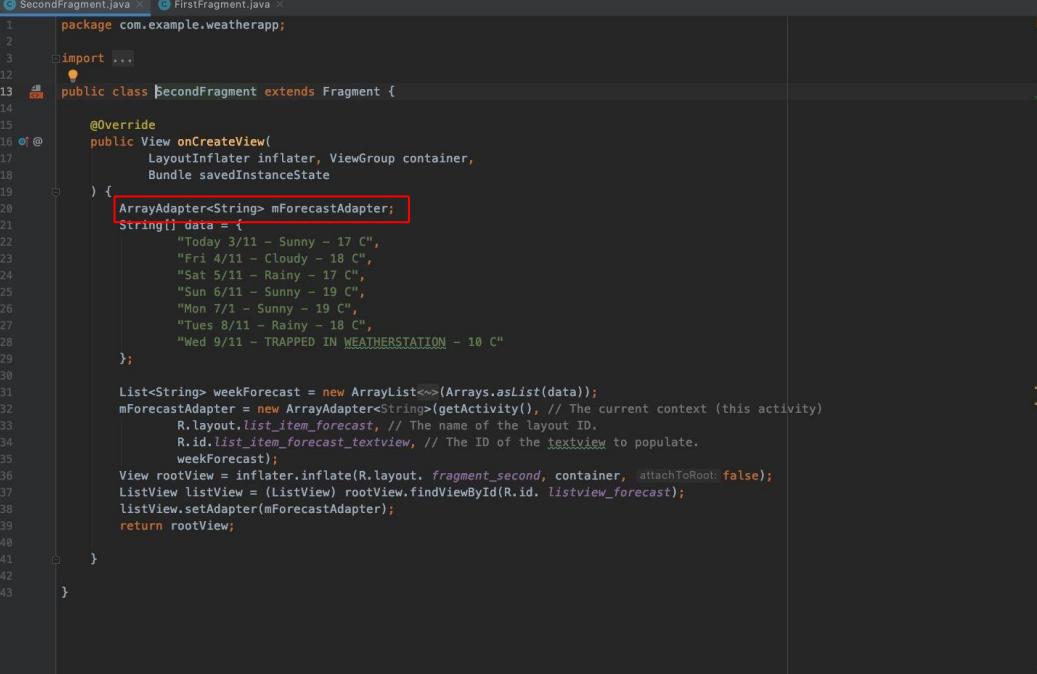
<http://api.openweathermap.org/data/2.5/forecast/daily?id=264371&mode=json&units=metric&cnt=7&APPID=612ce9a4c7726a3a0a00a69b84b9a01a>



The screenshot shows a Mac OS X window titled "api.openweathermap.org" with the URL "api.openweathermap.org/data/2.5/forecast/daily?id=264371&mode=json&units=metric". The content of the window is a large JSON object representing a weather forecast for Athens, Greece.

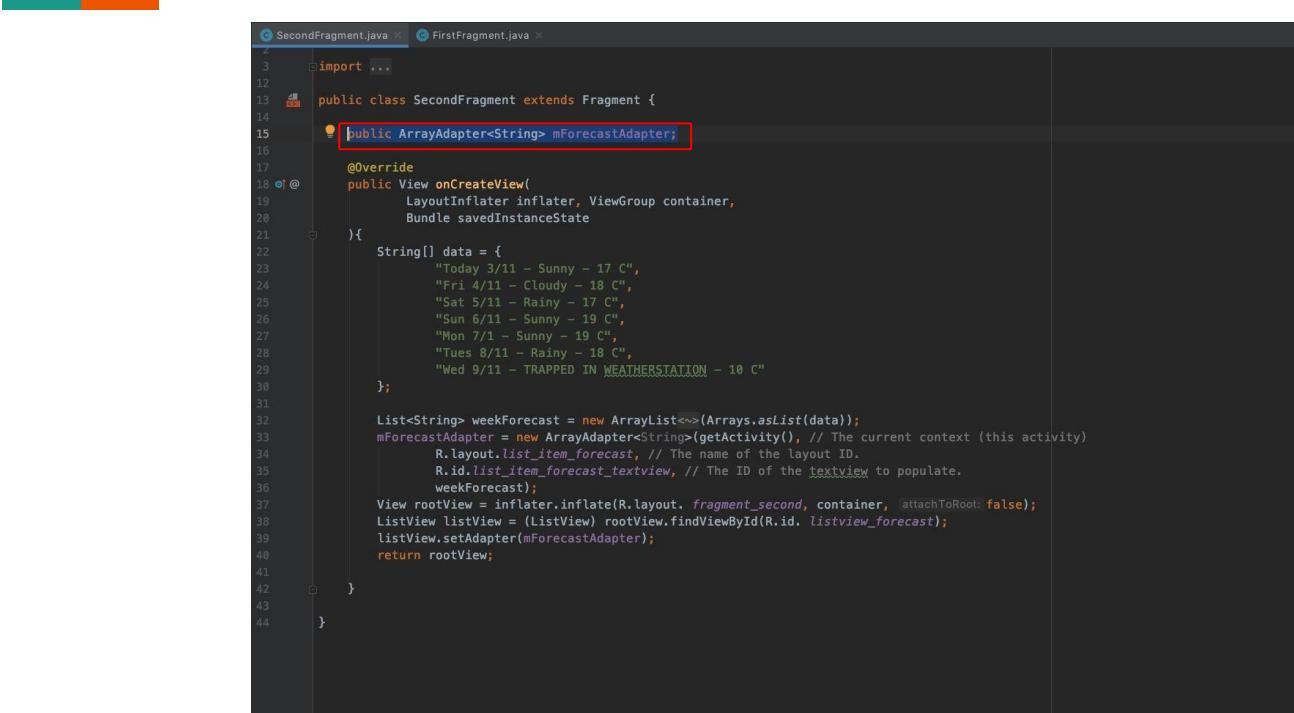
```
{
  "city": {
    "id": 264371,
    "name": "Athens",
    "coord": {
      "lon": 23.7162,
      "lat": 37.9795
    },
    "country": "GR",
    "population": 0,
    "cod": 200,
    "message": 0.0999551,
    "cnt": 7,
    "list": [
      {
        "dt": 1509357600,
        "temp": {
          "day": 19.5,
          "min": 18.77,
          "max": 19.5,
          "night": 15.77,
          "eve": 18.51
        },
        "morn": 19.5,
        "pressure": 1019.74,
        "humidity": 79,
        "weather": [
          {
            "id": 800,
            "main": "Clear",
            "description": "sky is clear",
            "icon": "04d"
          }
        ],
        "deg": 22,
        "clouds": 0
      },
      {
        "dt": 1509444000,
        "temp": {
          "day": 17.35,
          "min": 14.74,
          "max": 17.62,
          "night": 14.74,
          "eve": 16.86
        },
        "morn": 16.47,
        "pressure": 1028.4,
        "humidity": 85,
        "weather": [
          {
            "id": 803,
            "main": "Clouds",
            "description": "broken clouds",
            "icon": "04d"
          }
        ],
        "deg": 16.53,
        "clouds": 80
      },
      {
        "dt": 1509530400,
        "temp": {
          "day": 16.4,
          "min": 14.08,
          "max": 16.4,
          "night": 14.08,
          "eve": 15.29
        },
        "morn": 14.77,
        "pressure": 1031.96,
        "humidity": 90,
        "weather": [
          {
            "id": 801,
            "main": "Clouds",
            "description": "few clouds",
            "icon": "02d"
          }
        ],
        "deg": 16.12,
        "clouds": 20
      },
      {
        "dt": 1509616800,
        "temp": {
          "day": 16.39,
          "min": 13.2,
          "max": 17.64,
          "night": 14.25,
          "eve": 17.36
        },
        "morn": 13.2,
        "pressure": 1030.24,
        "humidity": 92,
        "weather": [
          {
            "id": 800,
            "main": "Clear",
            "description": "sky is clear",
            "icon": "01d"
          }
        ],
        "deg": 16.53,
        "clouds": 0
      },
      {
        "dt": 1509703200,
        "temp": {
          "day": 18.87,
          "min": 12.44,
          "max": 18.87,
          "night": 15.06,
          "eve": 15.61
        },
        "morn": 12.44,
        "pressure": 1016.95,
        "humidity": 0,
        "weather": [
          {
            "id": 800,
            "main": "Clear",
            "description": "sky is clear",
            "icon": "01d"
          }
        ],
        "deg": 17.77,
        "clouds": 9
      },
      {
        "dt": 1509789600,
        "temp": {
          "day": 20.24,
          "min": 14.7,
          "max": 20.24,
          "night": 14.7,
          "eve": 16.19
        },
        "morn": 15.19,
        "pressure": 1014.98,
        "humidity": 0,
        "weather": [
          {
            "id": 500,
            "main": "Rain",
            "description": "light rain",
            "icon": "10d"
          }
        ],
        "deg": 18.82,
        "clouds": 24,
        "rain": 1.45
      },
      {
        "dt": 1509876000,
        "temp": {
          "day": 16.25,
          "min": 13.53,
          "max": 16.25,
          "night": 13.53,
          "eve": 13.71
        },
        "morn": 14.45,
        "pressure": 1019.61,
        "humidity": 0,
        "weather": [
          {
            "id": 500,
            "main": "Rain",
            "description": "light rain",
            "icon": "10d"
          }
        ],
        "deg": 2.96,
        "clouds": 76,
        "rain": 2.15
      }
    ]
  }
}
```

Refactor Project components



```
1 package com.example.weatherapp;
2
3 import ...
4
5 public class SecondFragment extends Fragment {
6
7     @Override
8     public View onCreateView(
9             LayoutInflater inflater, ViewGroup container,
10            Bundle savedInstanceState
11    ) {
12        ArrayAdapter<String> mForecastAdapter;
13        String[] data = {
14            "Today 3/11 - Sunny - 17 C",
15            "Fri 4/11 - Cloudy - 18 C",
16            "Sat 5/11 - Rainy - 17 C",
17            "Sun 6/11 - Sunny - 19 C",
18            "Mon 7/11 - Sunny - 19 C",
19            "Tues 8/11 - Rainy - 18 C",
20            "Wed 9/11 - TRAPPED IN WEATHERSTATION - 10 C"
21        };
22
23        List<String> weekForecast = new ArrayList<>(Arrays.asList(data));
24        mForecastAdapter = new ArrayAdapter<String>(getActivity(), // The current context (this activity)
25            R.layout.list_item_forecast, // The name of the layout ID.
26            R.id.list_item_forecast_textview, // The ID of the textview to populate.
27            weekForecast);
28
29        View rootView = inflater.inflate(R.layout.fragment_second, container, false);
30        ListView listView = (ListView) rootView.findViewById(R.id.listView_forecast);
31        listView.setAdapter(mForecastAdapter);
32
33        return rootView;
34    }
35
36 }
37 }
```

Array Adapter should be public to be
amendable



```
SecondFragment.java x FirstFragment.java x

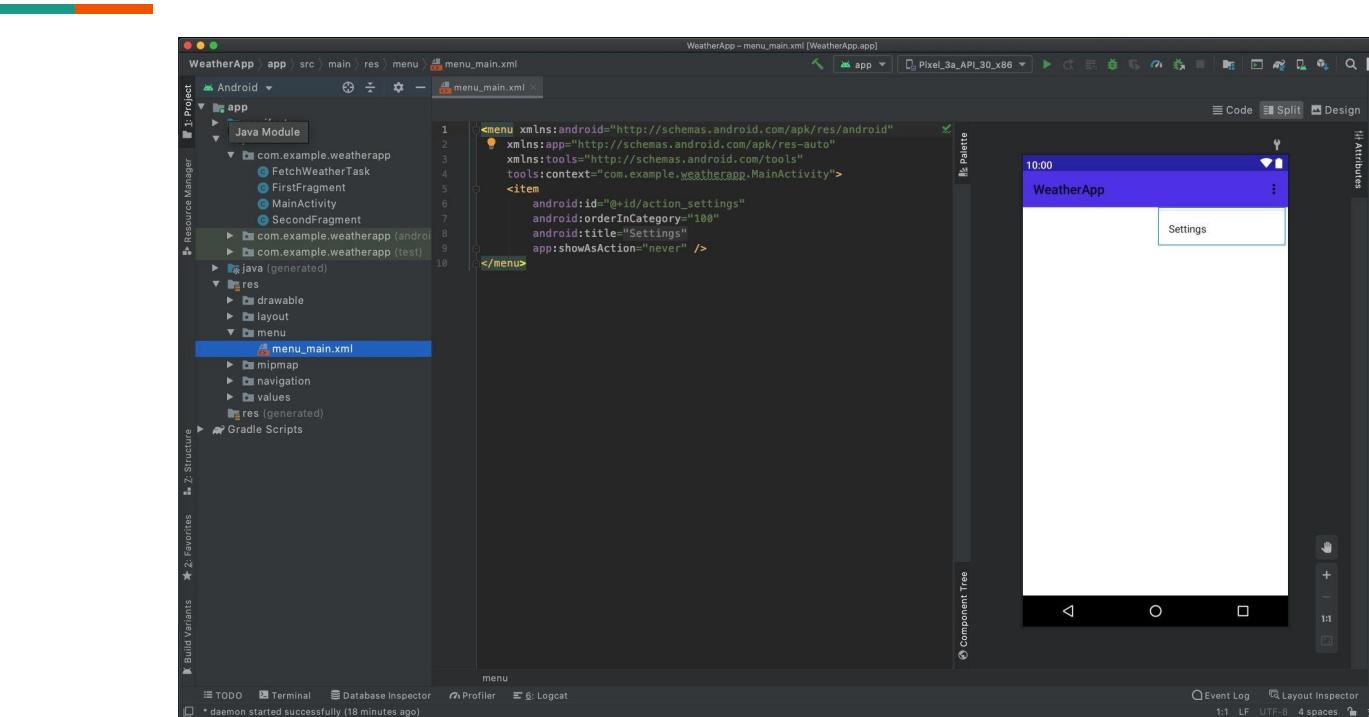
3 import ...
4
5 public class SecondFragment extends Fragment {
6
7     public ArrayAdapter<String> mForecastAdapter;
8
9     @Override
10    public View onCreateView(
11        LayoutInflater inflater, ViewGroup container,
12        Bundle savedInstanceState
13    ){
14        String[] data = {
15            "Today 3/11 - Sunny - 17 C",
16            "Fri 4/11 - Cloudy - 18 C",
17            "Sat 5/11 - Rainy - 17 C",
18            "Sun 6/11 - Sunny - 19 C",
19            "Mon 7/1 - Sunny - 19 C",
20            "Tues 8/11 - Rainy - 18 C",
21            "Wed 9/11 - TRAPPED IN WEATHERSTATION - 10 C"
22        };
23
24        List<String> weekForecast = new ArrayList<>(Arrays.asList(data));
25        mForecastAdapter = new ArrayAdapter<String>(getActivity(), // The current context (this activity)
26            R.layout.list_item_forecast, // The name of the layout ID.
27            R.id.list_item_forecast_textview, // The ID of the textview to populate.
28            weekForecast);
29        View rootView = inflater.inflate(R.layout.fragment_second, container, attachToRoot: false);
30        ListView listView = (ListView) rootView.findViewById(R.id.listView_forecast);
31        listView.setAdapter(mForecastAdapter);
32        return rootView;
33    }
34
35
36
37
38
39
40
41
42
43
44 }
```

Turn Array Adapter should be public to be amendable

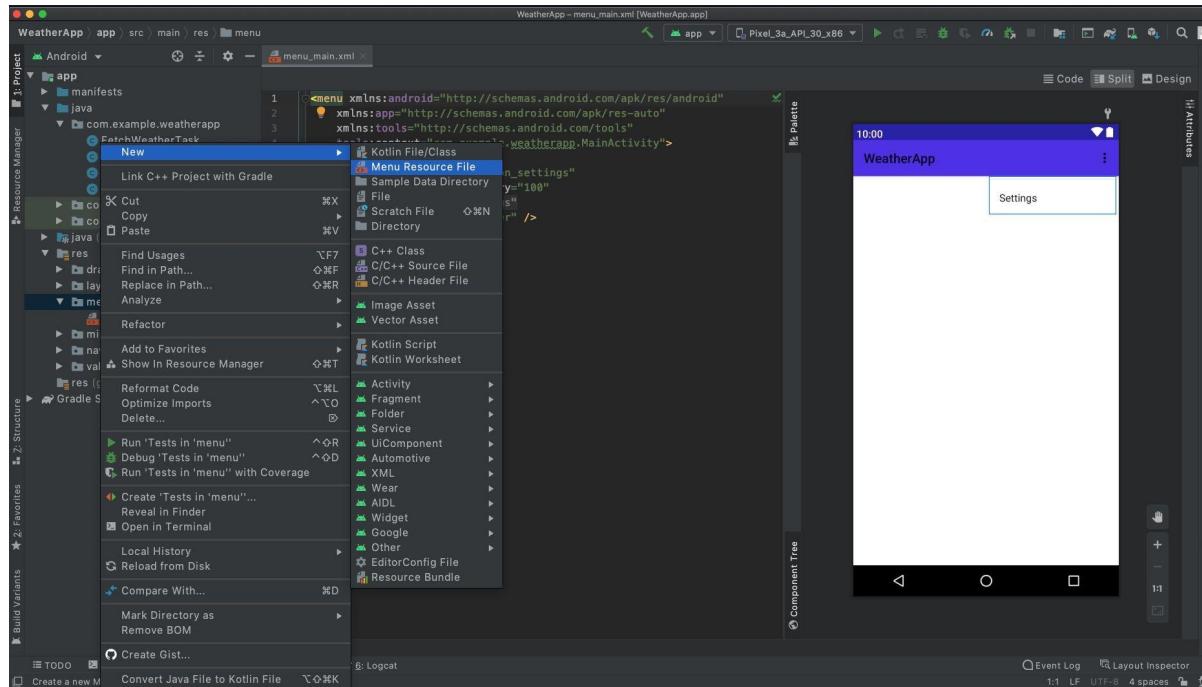


```
public static ArrayAdapter<String> mForecastAdapter;
```

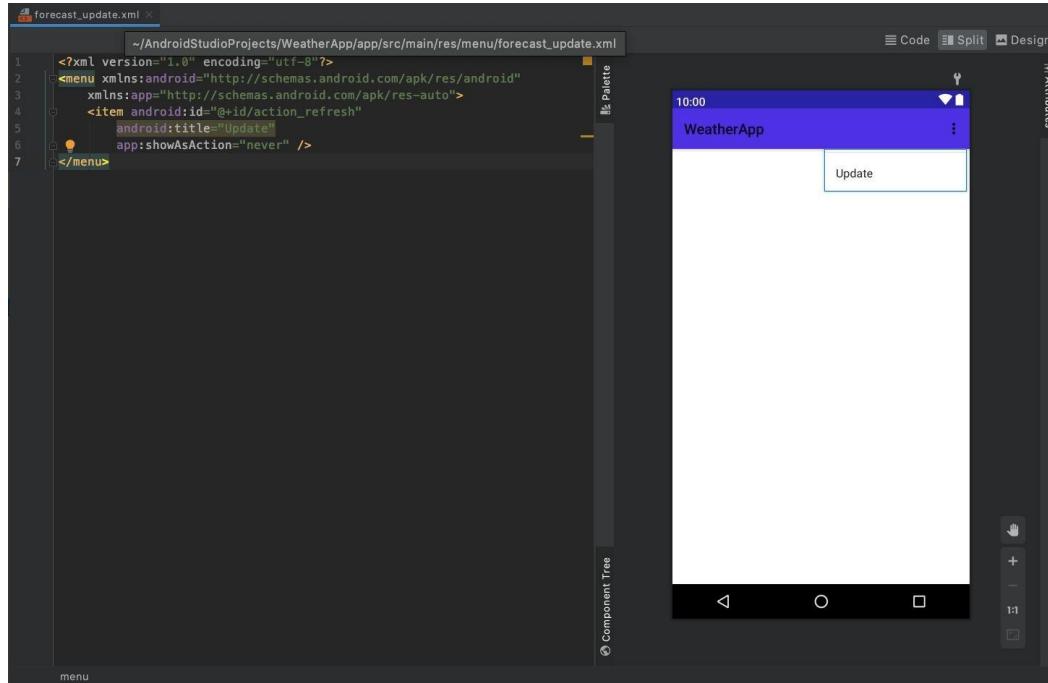
Create new Menu option to update Forecast



Current menu for first (home) fragment (Settings option)

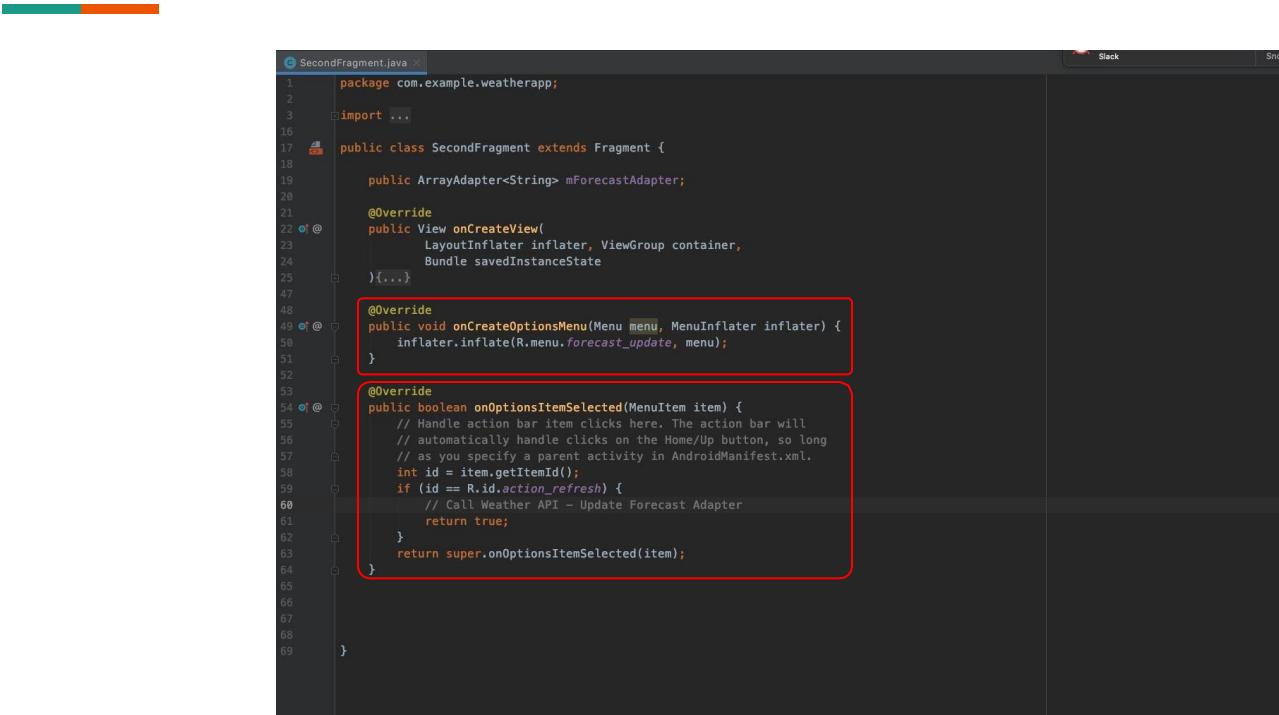


Create new menu for second (Forecast) fragment



Add menu option (Update Forecast)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_refresh"
          android:title="Update"
          app:showAsAction="never" />
</menu>
```



```
SecondFragment.java
1 package com.example.weatherapp;
2
3 import ...
4
5 public class SecondFragment extends Fragment {
6
7     @Override
8     public View onCreateView(
9             LayoutInflater inflater, ViewGroup container,
10            Bundle savedInstanceState
11    ) {...}
12
13    @Override
14    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
15        inflater.inflate(R.menu.forecast_update, menu);
16    }
17
18    @Override
19    public boolean onOptionsItemSelected(MenuItem item) {
20        // Handle action bar item clicks here. The action bar will
21        // automatically handle clicks on the Home/Up button, so long
22        // as you specify a parent activity in AndroidManifest.xml.
23        int id = item.getItemId();
24        if (id == R.id.action_refresh) {
25            // Call Weather API - Update Forecast Adapter
26            return true;
27        }
28        return super.onOptionsItemSelected(item);
29    }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69 }
```

Update business logic to support new menu

```
import android.view.MenuInflater;
import android.view.Menu;
import android.view.MenuItem;
import androidx.lifecycle.Lifecycle;

@Override
public void onViewCreated(@NonNull View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

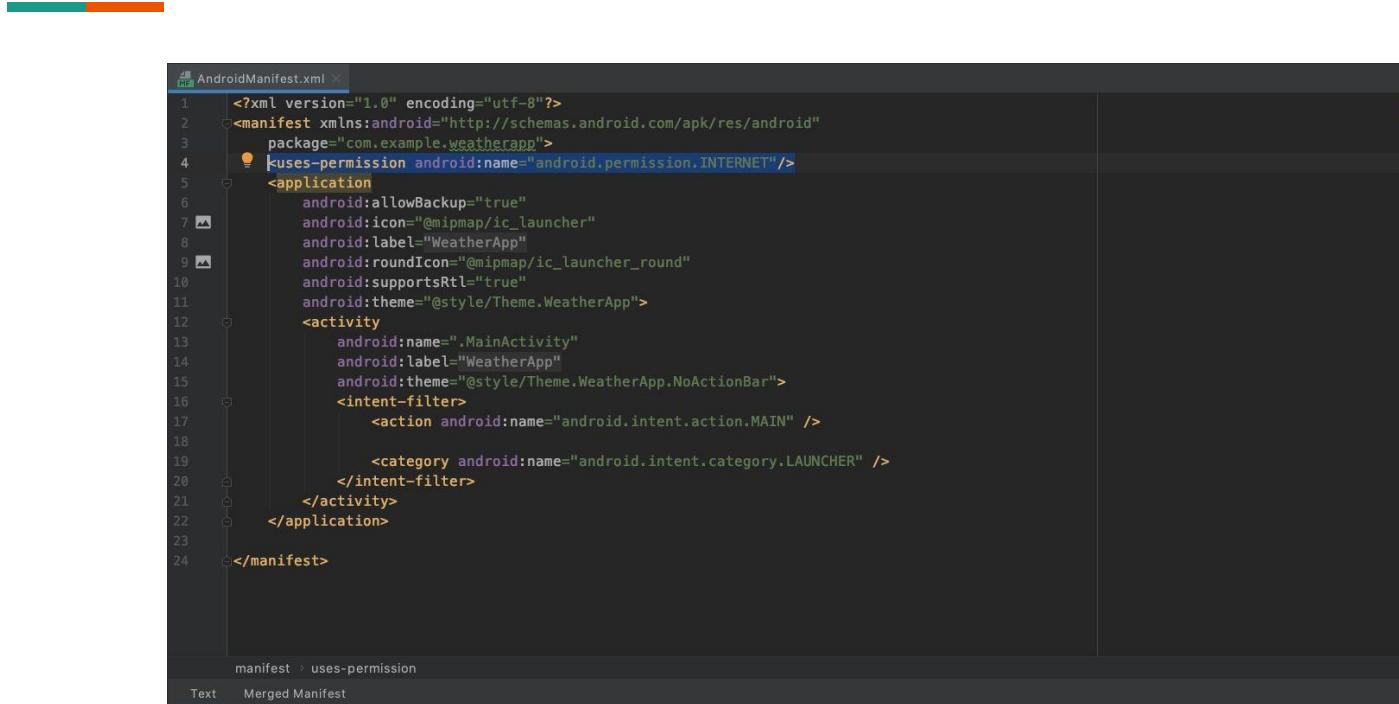
    // Get the MenuHost from the parent Activity
    MenuHost menuHost = requireActivity();

    // Add a new MenuProvider to handle the options menu in this fragment
    menuHost.addMenuProvider(new MenuProvider() {

        @Override
        public void onCreateMenu(@NonNull Menu menu, @NonNull MenuInflater menuInflater) {
            // Inflate the menu resource (R.menu.forecast_update) into the fragment's menu
            menuInflater.inflate(R.menu.forecast_update, menu);
        }

        @Override
        public boolean onMenuItemSelected(@NonNull MenuItem item) {
            // Handle menu item clicks
            if (item.getItemId() == R.id.action_refresh) {
                FetchWeatherForecast weatherTask = new FetchWeatherForecast();
                weatherTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, "0000");
                return true; // Indicate that the event was handled
            }
            return false; // Let the system handle other menu actions
        }
    }, getViewLifecycleOwner(), Lifecycle.State.RESUMED); // Ensure the menu is only active when the fragment is visible
}
```

Add Internet permissions



The screenshot shows the AndroidManifest.xml file in the Android Studio code editor. The manifest file includes the following XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.weatherapp">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="WeatherApp"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.WeatherApp">
        <activity
            android:name=".MainActivity"
            android:label="WeatherApp"
            android:theme="@style/Theme.WeatherApp.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

The line `<uses-permission android:name="android.permission.INTERNET"/>` is highlighted with a yellow oval icon, indicating it is selected or being edited. The status bar at the bottom shows the path `manifest > uses-permission`. The bottom navigation bar has tabs for `Text` and `Merged Manifest`, with `Text` currently selected.

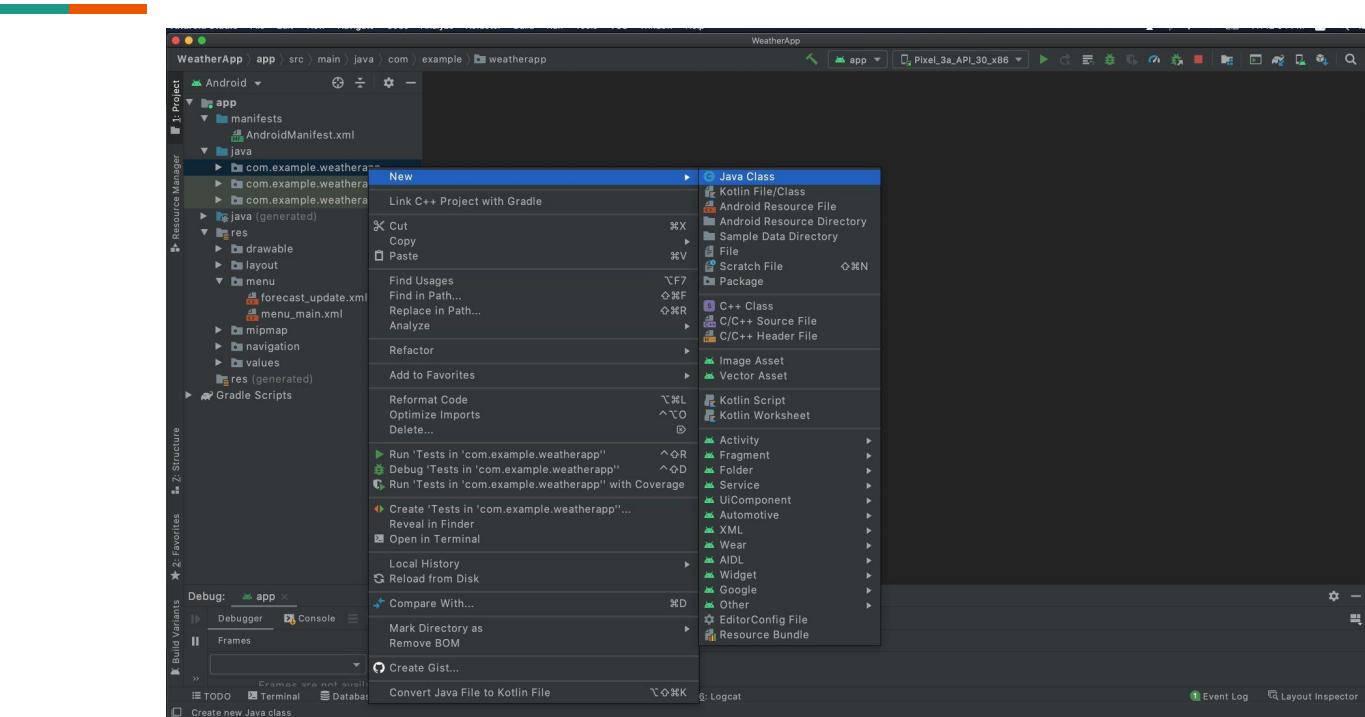
Add permission for internet in
manifest

Create Asynchronous Task to call API

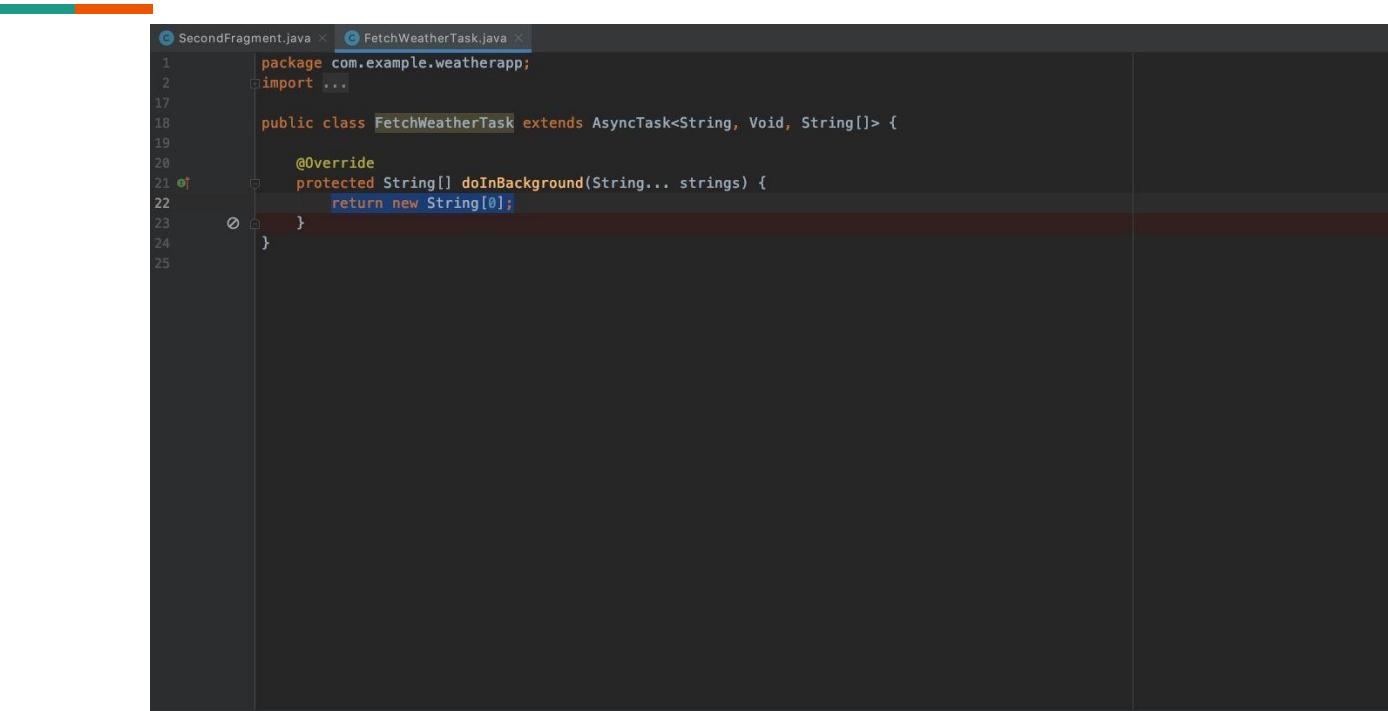


Next we need to apply a HTTP Request for weather forecast:

- Make HTTP Requests using the API URL
- Read HTTP Responses from the input stream in JSON format
- Clean up and Save the results
- Log any errors



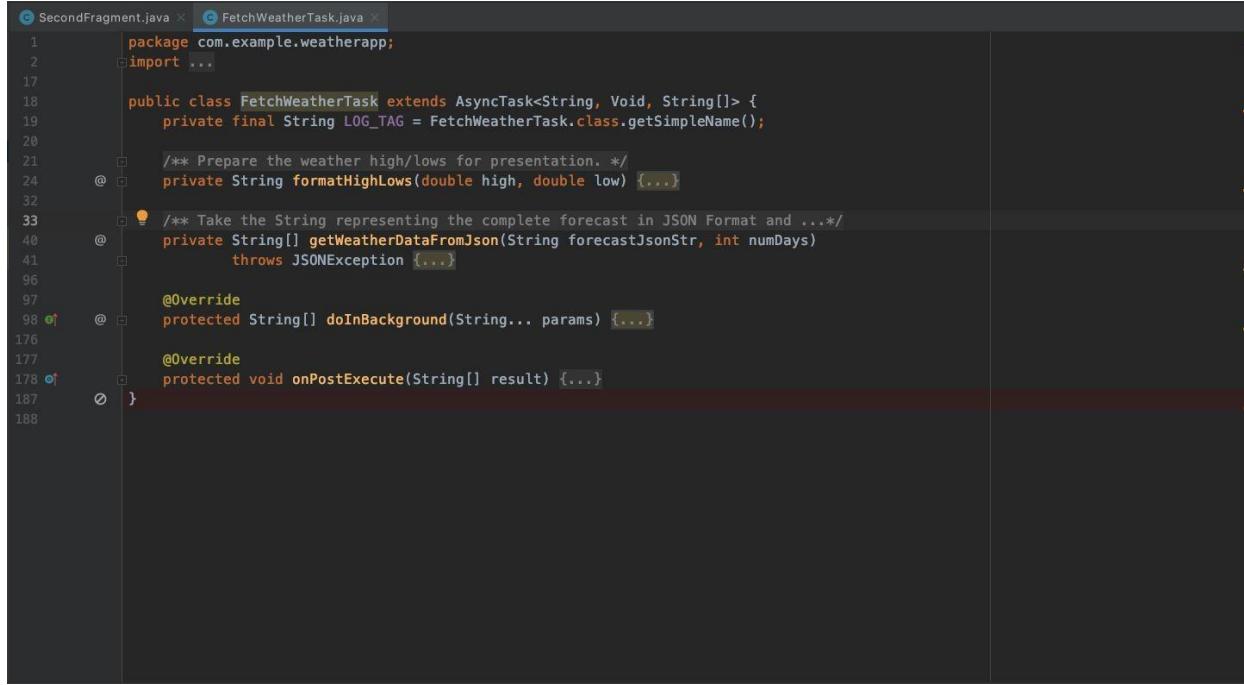
Create new Java class for Asynchronous Task



The screenshot shows a dark-themed IDE interface with two tabs at the top: "SecondFragment.java" and "FetchWeatherTask.java". The "FetchWeatherTask.java" tab is active, displaying the following code:

```
1 package com.example.weatherapp;
2 import ...
17
18 public class FetchWeatherTask extends AsyncTask<String, Void, String[]> {
19
20     @Override
21     protected String[] doInBackground(String... strings) {
22         return new String[0];
23     }
24 }
25
```

FetchWeatherTask extends AsynchTask

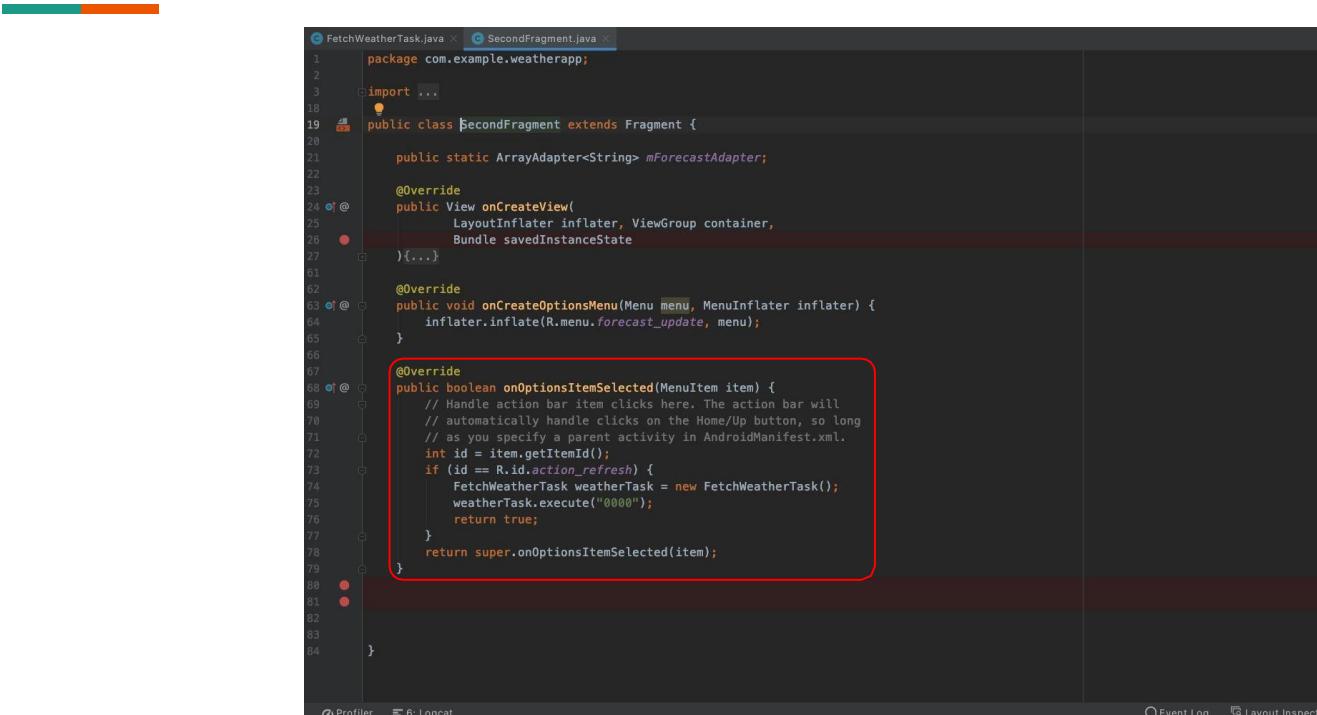


The screenshot shows a code editor with two tabs: "SecondFragment.java" and "FetchWeatherTask.java". The "FetchWeatherTask.java" tab is active, displaying the following Java code:

```
1 package com.example.weatherapp;
2 import ...
17
18 public class FetchWeatherTask extends AsyncTask<String, Void, String[]> {
19     private final String LOG_TAG = FetchWeatherTask.class.getSimpleName();
20
21     /** Prepare the weather high/lows for presentation. */
22     @Override
23     private String formatHighLows(double high, double low) {...}
24
25     /** Take the String representing the complete forecast in JSON Format and ...*/
26     @Override
27     private String[] getWeatherDataFromJson(String forecastJsonStr, int numDays)
28         throws JSONException {...}
29
30     @Override
31     protected String[] doInBackground(String... params) {...}
32
33     @Override
34     protected void onPostExecute(String[] result) {...}
35 }
```

Build business logic to fetch and preview
forecast

Call Asynchronous Task



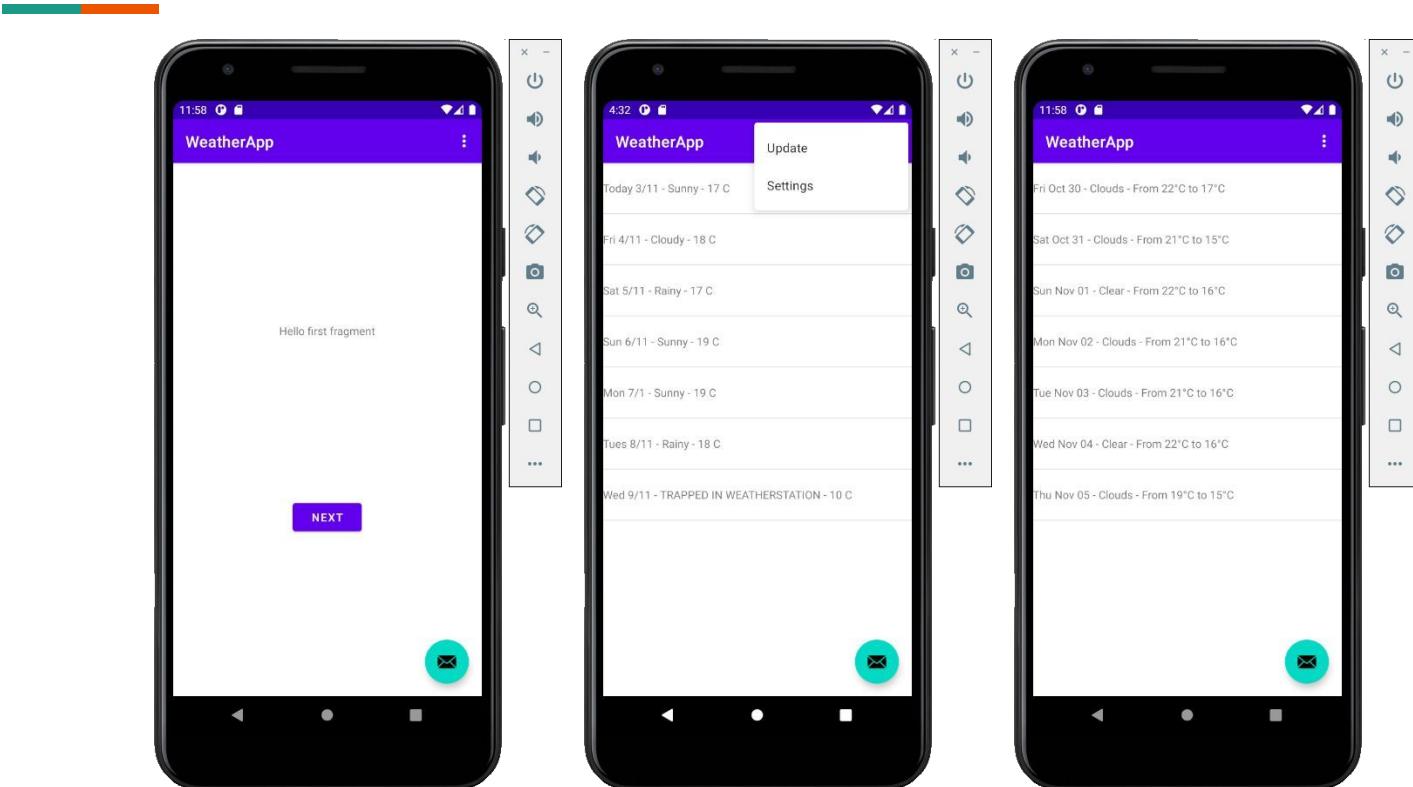
```
1 package com.example.weatherapp;
2
3 import ...
4
5 public class SecondFragment extends Fragment {
6
7     public static ArrayAdapter<String> mForecastAdapter;
8
9     @Override
10    public View onCreateView(
11        LayoutInflater inflater, ViewGroup container,
12        Bundle savedInstanceState
13    ) {...}
14
15    @Override
16    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
17        inflater.inflate(R.menu.forecast_update, menu);
18    }
19
20    @Override
21    public boolean onOptionsItemSelected(MenuItem item) {
22        // Handle action bar item clicks here. The action bar will
23        // automatically handle clicks on the Home/Up button, so long
24        // as you specify a parent activity in AndroidManifest.xml.
25        int id = item.getItemId();
26        if (id == R.id.action_refresh) {
27            FetchWeatherTask weatherTask = new FetchWeatherTask();
28            weatherTask.execute("0000");
29            return true;
30        }
31        return super.onOptionsItemSelected(item);
32    }
33
34 }
```

Build business logic to fetch and preview forecast

```
import android.os.AsyncTask;

@Override
public boolean onMenuItemSelected(@NonNull MenuItem item) {
    if (item.getItemId() == R.id.action_refresh) {
        FetchWeatherForecast weatherTask = new FetchWeatherForecast();
        weatherTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, "0000");
        return true;
    }
    return false;
}
```

Demonstrate Updates



Review of Lab 2

- Catch up with Lab 1
- Introduce OpenWeatherMap API
- Update code - Use Public Forecast API
 - Make property publicly available
 - REST call to API - JSON response
 - Create new menu option to update forecast
 - Add internet permission
 - Create Asynchronous Task to call API
 - Call Asynchronous Task
 - Demonstrate updates