COVID-19 Tracking

Raneem Soufi

002-49-2884

1. Introduction

The COVID-19 Tracking app reports to the user information regarding the COVID-19 epidemic. The user can use this app as a fast and accurate way of checking how the virus is progressing in the country and in their state. On the “Country” tab, the app shows a graph of the number of increases in positive COVID cases per day since the start of the Coronavirus in the United States. The “Country” tab also shows the top 5 states with the highest number of increases in positive cases. The “State” tab has a list of every state in the United States. The user can select a state for the app to provide more information on the current status of the state. The popup shows: the name of the state, the date of the last report, the number in increase in positive cases, the number of increases in deaths, the number of people currently on a ventilator, and the number of people currently hospitalized.

1. Application Design
2. Country and State

Data in the app is divided into two classes: Country and State. The Country contains three linked lists: casesPerDay (records the number of cases per day since the start of the epidemic), dates (the associated date with each case in casesPerDay), and xValues (contain the xValues used in the graph – derived from the dates linked list). casesPerDay and xValues are used in order to create the graph showed on the main screen. State contains the same linked lists, as well as String name, int date, String dateText (the text displayed, derived from the date; example: int 10102020 becomes String “10/10/2020”), int deathIncrease (increase in the number of deaths), int hospitalizedCurrently (number of people currently hospitalized), int onVentilatorCurrently (number of people currently on ventilator), and int newCases (the number of increases in positive cases for that day). The number of positive tests include the ones reported by hospitals plus the estimate (to account for people who may not have been tested). For the user, the estimate + reported method of giving facts allows the user to more accurately know the current state of the country in regards to the pandemic.

Both Country and State implement Runnable and Serializable. Serializable allows the class to be turned into a byte stream. This allows the class to be passed between objects with no issues. Runnable gives the class a “run()” function. Runnable will be discussed later under “Multithreading - ThreadPoolExecutor.”

Country also has variables Handler handler, while State has variables Handler handler and Context context. The use of the two will be discussed in “Multithreading - Handler.” Context is the current state of the application (checking permissions, getting files, starting activities) and the system itself (such as the battery, power level, wallpaper, etc.).

1. Volley

The COVID-19 Tracking app uses API from “The COVID Tracking Project” in order to get the information used in the app. In order to retrieve this information, the app uses Volley API in order to make JSON requests. Volley makes a request to “The COVID Tracking Project” and receives a JSONObject or JSONArray in response. The JSONObject/JSONArray is then parsed, and the needed information is taken and saved in the app.

In the Country class, only a JSONArray is retrieved (A JSONArray is formatted using brackets at the start and end) in the method getJSONArrayFromURL(String urlString). This array contains data for every single day since the start of the pandemic. Since the response is so long, a BufferedReader is used in order to retrieve the entire JSONArray. Otherwise, the app may not receive the entire JSONArray, and the app would crash. A BufferedReader works by reading the array in increments and adding each increment to a string. The string is then turned into a JSONArray. After the JSONArray is received, it is parsed for “positiveIncrease” and “dates”, which are appended to the linked lists “casesPerDay” and “dates” respectively. After all the data is parsed, the function createGraph() is called, which turns the dates in “dates” into x-values on a graph. These values are saved into the linked list “xValues.”

The State class makes two JSON requests: first to a JSONObject (A JSONObject is formatted using curly brackets at the start and end), and then to a JSONArray. The JSONObject is received after making a JSONObjectRequest. The JSONObject contains all the information for the most recent COVID-19 report. Only information regarding that one day is received. After it is received, it is parsed for “date” (saved in the “date” variable), “deathIncrease” (saved in the “deathIncrease” variable), “hospitalizedCurrently” (saved in the “hospitalizedCurrently” variable), “onVentilatorCurrently” (saved in the “onVentilatorCurrently” variable), and “positiveIncrease” (saved in the variable “newCases”). Once that is complete, the app makes another request for a JSONArray, similar to the one in the Country class. The only difference between the two is that the Country class uses HttpURLConnection and BufferedReader in order to retrieve the response, while State saves the response directly into a JSONArray using JSONArrayRequest. After the information is received, the JSONArray is parsed the same way as it is in Country.

In total, 101 GET requests are made by the app in order to get all the information: two from each state, and 1 for the entire country. Because of the high load of content needed, the app uses multithreading in order to quickly get that content to the user.

1. Multithreading - ThreadPoolExecutor

If the app were to work sequentially, the user may be waiting a long period of time before the app is fully loaded. To make the app load faster to enhance user experience, multithreading is used to make multiple requests at the same time.

ThreadPoolExecutor creates a thread pool based on the number of requests and the developer specified max number of threads. For this app, the max number of threads is the number of cores available for use. After the thread pool is created, each Country and State is created. Country accepts a Handler as a parameter in its initializer, while State accepts String n (the name of the state, in initial form), Context con (the current context), and Handler handler. After the Country or State is created, calling ThreadPoolExecutor.execute(Runnable runnable) will add the Runnable (whatever variable was just created) to the queue. When a thread becomes available, a runnable is taken from the queue and executed. Once all the threads have run, calling shutdown() will free those allocated threads to be used for other functions.

Both State and Country implement the “Runnable” class. Inside both State and Country is a method called “run()”. When the runnable is put in a thread, the run() method is executed. Inside the run() function for both State and Country are the calls needed to get all the information for each state and the entire country – Volley is used in order to make and parse these requests.

Because all cores are being used, the app responds slowly to user interaction until shutdown() is called. An improvement on this app in order to better combine multithreading with user interaction would be to test what the most efficient number of threads are for each device that could possibly run the app.

1. Multithreading – Handler

UI in the app can only be changed in the Main Thread. Attempting to change the UI will result in an “Application Not Responding” error and crash the app. In order to update the UI after each Runnable is done running, a Handler is used in order to communicate information back to the Main Thread. In this app, two different handlers have been made. The first handler, called “myHandler,” handles messages from the State runnables. Before it does anything, it waits for all 50 State runnables to finish. It does this by comparing the value of two variables: ‘currentlyLoaded’ and ‘stateCount’. stateCount is the number of states minus one , while currentlyLoaded is the number of states that have completed loading. When the two are equal (i.e. all states are done loading), the handler updates the UI to show the top 5 states with the largest number in the increase of cases. To do this, it searches all the states to see which 5 have the highest number of cases. It then sorts the array from lowest to highest number of cases. If fragment\_main is the current fragment, the UI is updated without any user interaction to show the new information. The second handler, “myHandler2”, handles messages from the Country runnable. When it receives a message, it updates the graph on fragment\_main to show all the retrieved information. The handlers send a message at the end of each run() method.

1. Main Activity

Main Activity is where all the code is run, and where all the different components of the app come together. It contains the variables Country c (to store the information for the entire country), State[] stateList (array of size 50, to store the information for each state in alphabetical order), int stateCount (the number of states), int currentlyLoaded (the number of states that have finished loading), int[][] topStates (contains the index of the state and the number of cases for it for the top 5 states with the most number of cases), int activeFragmet (the current fragment shown on the screen, 0 being fragment\_main and 1 being fragment\_states), int NUMBER\_OF\_CORES (the number of available processors), int KEEP\_ALIVE\_TIME (the amount of time to keep a thread alive), and TimeUnit KEEP\_ALIVE\_TIME\_UNIT (the time unit used for KEEP\_ALIVE\_TIME, in this case milliseconds).

onCreate() is called when the Activity is created and is automatically ran by the app. The first thing it does is create the two handlers that will be used to communicate between the Main Thread and background threads. The two handlers are discussed in “Multithreading – Handlers.” After handlers are created, the ThreadPoolExecutor is created. It has a pool size of the number of cores that are available, and a keep alive time of 1000 milliseconds. It also has a queue to execute all the different runnables. The queue is empty at first. Then, ‘c’ is set to a new Country variable using ‘myHandler2’, and added to the queue. Then, all 50 states are created (with the current context of the Activity and using ‘myHandler’) and added to the queue. When all the threads are finished, shutdown() is called on the ThreadPoolExecutor to release the threads in the thread pool.

After all the thread setup is handled, the main fragment is added on top of background fragment. The current active fragment is set to 0. The ‘onTabSelectedListener’ is also set up, allowing for the fragments to switch if the user selects a different tab.

1. MySingleton

MySingleton is a class that contains a Singleton and a RequestQueue. This is used by Volley in order to help manage JSON requests.

1. User Interface
2. Fragments

The app uses 4 fragments in order to work. Fragments are views that are used in order to create larger views in the app. The four fragments are: fragment\_background, fragment\_main, fragment\_states, and states\_popup. All the setup in each fragment is done in onCreateView(), which runs automatically when the view is created by the app.

Fragment\_background is simply an empty fragment. It contains no subviews and has no background color. It is used in activity\_main as a base fragment. Other fragments are added on top of the background fragment and replaced as necessary. It acts similar as a plate does to a meal: it simply holds the more important part that is actually used. The corresponding Java file for fragment\_backgorund is BackgroundFragment.

Fragment\_main appears when the user selects the “Country” tab. It is also the first one that appears when the app is opened. It contains a Graph view in order to display the data in a Country variable, as well as a TableLayout in order to display the top 5 states with the highest number of COVID cases. The TableLayout is created in the Java file, rather than the XML file. The XML file for fragment\_main contains no information regarding the rows. In the java file, MainFragment.java, the rows and the information in them are added. This allows for the rows to contain different values (for example, one day the table can say California has the most cases, the next day Arizona), as well as for the table to be updated by the app (such as when all the data is done loading and the UI needs to update). The corresponding Java file for fragment\_main is MainFragment.

Fragment\_states appears when the user selects the “States” tab. It contains a TableLayout, where each row is a state, organized in alphabetical order. Just like in Fragment\_main, the TableLayout has its rows added programmatically. The Java file, StatesFragment, creates a row for each state. It also creates an “onClickListener” for each row. When a row is selected, the onClickListener has a pop up appear on the screen, with all the information regarding that specific state.

State\_popup appears on the screen when the user selects a row from fragment\_states. It contains a card view, which creates a small card that takes up a portion of the screen, rather than the entire screen. The background of the CardView is transparent black, to signal to the user that the previous screen cannot be interacted with, and that the new fragment is in focus. Inside the CardView is a Graph, similar to the one in Fragment\_main, but displays data for the specific state rather than for the entire country as a whole. The variables contained in the State class is also displayed on the screen. When the user wants to exit the popup, they simply have to tap on the screen. Doing so will dismiss the popup and return the focus to the previous fragment (fragment\_states in this case). The corresponding Java file for fragment\_states is StatesFragment.

1. TabLayout

TabLayout is used in order to allow for multiple fragments to be shown on the screen. When the user selects the “Country” tab, fragment\_main is shown. When the user selects the “States” tab, fragment\_states is shown. The fragments are changed using an addOnTabSelectedListener, which allows the developer to change what happens when a tab is selected, unselected, or released. For this app, fragments are changed and replaced when a tab is selected. The TabLayout is a part of MainActivity, rather than in its own fragment. However, fragment\_background is housed within the TabLayout. This is how the app knows where to place the knew fragments when they need to be added – it takes the dimensions from fragment\_background housed inside TabLayout. Currently, the app does nothing when a tab is unselected or released.

1. Graph (a custom view that extends view)

Graph is a custom class that extends view. It has 5 variables: xValues (a Linked List to contain the x values of the graph), cases (a Linked List to contain the y values of the graph), maxCases (an int which contains the max number of cases on the graph to scale the y-axis to the height of the graph), numberOfDays (an int which contains the number of days the graph looks at to scale the x-axis to the width of the graph), width (an int, width of the graph on the screen), and height (an int, height of the graph on the screen). It accepts the current context and attribute set as parameters for its initializer, as part of extending a view. It contains a function, “addValues,” which accepts two integer linked lists as parameters: xValues and yValues. These two linked lists are taken from either a state or a country’s “xValues” and “casesPerDay” linked lists. These lists are then copied over into the Graph. If the Graph’s “xValues” and “cases” variables are null, the Graph will display a message that says “Loading…”. Otherwise, it will display a line graph containing each point from the two variables. The graph is drawn in the onDraw() function, which allows the developer to draw custom shapes and objects onto the view. The graph is drawn by scaling the number of cases to the y axis, and the number of days to the x axis. Then, a line is drawn to each point, with (0, 0) being in the bottom left corner. The classes Canvas and Paint are used to draw this graph.

Graph contains a method “onSizeChange()” which gets the width and height of the view. This is necessary as when the Graph is first created, the height and width are set to 0. The height and width can only be retrieved after the view is done loading.

Within the fragments themselves, in order to reset the graph, invalidate() must be called. This way, the Handlers can update what is displayed on the screen. Otherwise, no new information would be able to be added to the graph.

1. activity\_main

activity\_main is an XML file linked with the Java file MainActivity. In a ConstraintLayout, it contains a TabLayout which is used in order to format the tabs on the screen. The TabLayout is constrained to the top, left, and right edges of the screen. The height of the layout is automatically set to wrap the contents of the layout. It contains two premade tabs: Country and State. Below the TabLayout is a fragment. The default fragment is fragment\_background, which contains no views. It is constrained to the left, right, and bottom edges of the screen. The top is constrained to the TabLayout. The fragment allows for the size of each fragment to automatically adjust to the screen when added or removed.

1. Screenshots

Main Fragment

Before all the data is loaded: After all the data is loaded:

A screen shot of a computer

Description automatically generated A screen shot of a smart phone

Description automatically generated

|  |
| --- |
| The UI in Fragment Main is updated by the Handler. Once the Handler identifies that all the data is loaded in, it updates the UI. The Graph and the list of the top 5 States are loaded at two different times by two different handlers. This allows the code to be more readable (each handler does 1 task), as well as allows the user to know the app is loading the information and that the app is not frozen.  The top part of fragment\_main is the graph. Before the Country variable is completely loaded in, it will display ‘loading’ on the screen. Once it is complete, a message is sent using the handler, and the UI is updated.  The bottom part of fragment\_main is the top 5 states with the highest number of cases. In order to calculate this, all the states need to have finished loading in. Then, in the handler for the states, the states with the highest number of cases are saved in an array, and the UI is updated to display their information. The information is displayed using a TableLayout. Rows are added with all the needed information once the handler indicates all the states have loaded in and identifies the top 5 states.  On the top of the screen are the two tabs. The “Country” tab is currently selected and displaying fragment\_main. If the user selects the other tab, “States,” the screen changes to the next screenshot. |

State Fragment: PopUp Fragment, after a state has been

pressed:

A screen shot of a computer

Description automatically generated Screen of a cell phone

Description automatically generated

|  |
| --- |
| The screenshot on the left is fragment\_states after it is loaded in on the app. There are 50 rows, one for each state in the country. Each row is selectable. Each row also contains the information regarding that state. The user can also vertically scroll through the table. When a state is pressed, a new instance of PopUpState is created with the state selected as a variable of that instance.  PopUpState contains a graph at the top with the number of increase in cases per day since the start of the pandemic for that state. The information below the graph set using a TableLayout. If the information is not yet available, the table should only display “Loading…”. The same is true for the graph. However, because the app using multithreading, the information is often loaded before the user has a chance to select a state. In order to exit the popup, the user simply has to tap anywhere on the screen. The instance of PopUpState is dismissed. |

Code: (Github: <https://github.com/rsoufi1/COVID19-tracking-project> )

Country Class:

package com.example.osproject2;  
  
import android.content.Context;  
import android.os.Handler;  
import android.util.Log;  
  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.Serializable;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.util.LinkedList;  
  
public class Country implements Runnable, Serializable {  
 //Variables  
 private LinkedList<Integer> casesPerDay;  
 private LinkedList<Integer> dates;  
 private LinkedList<Integer> xValues;  
 private Handler handler;  
  
 //Initializers  
 public Country(Handler handle){  
 this.casesPerDay = new LinkedList<>();  
 this.dates = new LinkedList<>();  
 this.xValues = new LinkedList<>();  
 this.handler = handle;  
 }  
  
 public Country(){  
 this.casesPerDay = new LinkedList<>();  
 this.dates = new LinkedList<>();  
 this.xValues = new LinkedList<>();  
 }  
  
  
 @Override public void run() {  
 // Do some work that takes 50 milliseconds  
 try {  
 Thread.*sleep*(50);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 try {  
 getInfo();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void getInfo() throws IOException, JSONException {  
 String url2 = "https://api.covidtracking.com/v1/us/daily.json";  
 JSONArray a = *getJSONArrayFromURL*(url2);  
 addGraph(a);  
 }  
  
 public void addGraph(JSONArray response){  
 int length = response.length();  
 //Initialize the rest of the casesPerDay  
 for(int i = 0; i < length; i++){  
 try{  
 JSONObject o = response.getJSONObject(i);  
 JSONArray names = o.names();  
 int j = 0;  
 boolean found = false;  
 boolean foundDate = false;  
 while(j < names.length() && (!found || !foundDate)){  
 if(names.getString(j).equals("positiveIncrease")){  
 this.casesPerDay.addFirst(o.getInt(names.getString(j)));  
 found = true;  
 }  
 if(names.getString(j).equals("date")){  
 this.dates.addFirst(o.getInt(names.getString(j)));  
 foundDate = true;  
 }  
 j++;  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 createGraph();  
  
 Log.*d*("addGraph", "done");  
 //Log.d("results", toString());  
 this.handler.sendEmptyMessage(0);  
 }  
  
 @Override public String toString(){  
 String s = "";  
 s+= "cases per day and xValue: ";  
 for(int i = 0; i < this.casesPerDay.size(); i++){  
 s+= "day: " + this.xValues.get(i) + " cases: " + casesPerDay.get(i) + ", \n";  
 }  
 return s;  
 }  
  
 public void createGraph(){  
 int firstDay = this.dates.getFirst();  
 int lastDay = this.dates.getLast();  
  
 //Calculate the first day in number of days  
 int dayFirst = firstDay%100;  
 int monthFirst = (firstDay%10000 - dayFirst)/100;  
 int yearFirst = firstDay/10000;  
 firstDay = getDays(yearFirst, monthFirst, dayFirst);  
  
 //Calculate each day into the number of day it is in the year  
 for(int i = 0; i < this.dates.size(); i++){  
 int date = this.dates.get(i);  
 int day = date%100;  
 int month = (date%10000 - day)/100;  
 int year = date/10000;  
 int totalDays = getDays(year, month, day);  
 int xValue = totalDays - firstDay + 1;  
 xValues.add(xValue);  
 }  
 }  
  
 public int getDays(int year, int month, int day){  
 int currentMonth = 1;  
 int numberOfDays = 0;  
 while(currentMonth < month){  
 if(currentMonth == 1) numberOfDays+= 31; //Janurary  
 else if(currentMonth == 2){ //Feburary  
 if(year%4 == 0) numberOfDays+= 29;//leap year  
 else numberOfDays+=28; //normal year  
 } else if(currentMonth == 3) numberOfDays+= 31; //March  
 else if(currentMonth == 4) numberOfDays+= 30; //April  
 else if(currentMonth == 5) numberOfDays+= 31; //May  
 else if(currentMonth == 6) numberOfDays+= 30; //June  
 else if(currentMonth == 7) numberOfDays+= 31; //July  
 else if(currentMonth == 8) numberOfDays+= 31; //August  
 else if(currentMonth == 9) numberOfDays+= 30; //September  
 else if(currentMonth == 10) numberOfDays+= 31; //October  
 else if(currentMonth == 11) numberOfDays+= 30; //November  
 else if(currentMonth == 12) numberOfDays+= 31; //December  
  
 currentMonth++;  
 }  
 numberOfDays+= day;  
 return numberOfDays;  
 }  
  
 public LinkedList<Integer> getCases(){return casesPerDay;}  
 public LinkedList<Integer> getxValues(){return xValues;}  
  
 public static JSONArray getJSONArrayFromURL(String urlString) throws IOException, JSONException {  
 HttpURLConnection urlConnection = null;  
 URL url = new URL(urlString);  
 urlConnection = (HttpURLConnection) url.openConnection();  
 urlConnection.setRequestMethod("GET");  
 urlConnection.setReadTimeout(10000 /\* milliseconds \*/ );  
 urlConnection.setConnectTimeout(15000 /\* milliseconds \*/ );  
 urlConnection.setDoOutput(true);  
 urlConnection.connect();  
  
 BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream()));  
 StringBuilder sb = new StringBuilder();  
  
 String line;  
 while ((line = br.readLine()) != null) {  
 sb.append(line + "\n");  
 }  
 br.close();  
  
 String jsonString = sb.toString();  
  
 return new JSONArray(jsonString);  
 }  
}

State Class:

package com.example.osproject2;  
  
import android.content.Context;  
import android.os.Handler;  
import android.util.Log;  
  
import com.android.volley.Request;  
import com.android.volley.Response;  
import com.android.volley.VolleyError;  
import com.android.volley.toolbox.JsonArrayRequest;  
import com.android.volley.toolbox.JsonObjectRequest;  
  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.io.Serializable;  
import java.util.LinkedList;  
  
public class State implements Runnable, Serializable {  
 //Variables  
 private String name;  
 private int date;  
 private String dateText;  
 private int deathIncrease;  
 private int hospitalizedCurrently;  
 private int onVentilatorCurrently;  
 private int newCases;  
 private LinkedList<Integer> casesPerDay;  
 private LinkedList<Integer> dates;  
 private LinkedList<Integer> xValues;  
 private Context context;  
 private Handler handler;  
  
 //  
 public State(Context con, Handler handle){  
 this.name = "not yet filled";  
 this.date = -1;  
 this.deathIncrease = -1;  
 this.hospitalizedCurrently = -1;  
 this.onVentilatorCurrently = -1;  
 this.newCases = -1;  
 this.casesPerDay = new LinkedList<>();  
 this.dates = new LinkedList<>();  
 this.xValues = new LinkedList<>();  
 this.context = con.getApplicationContext();  
 this.handler = handle;  
 }  
  
 public State(String n, Context con, Handler handle){  
 this.name = n;  
 this.date = -1;  
 this.deathIncrease = -1;  
 this.hospitalizedCurrently = -1;  
 this.onVentilatorCurrently = -1;  
 this.newCases = -1;  
 this.casesPerDay = new LinkedList<>();  
 this.dates = new LinkedList<>();  
 this.xValues = new LinkedList<>();  
 this.context = con.getApplicationContext();  
 this.handler = handle;  
 }  
  
 @Override public void run() {  
 // Do some work that takes 50 milliseconds  
 try {  
 Thread.*sleep*(50);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 Log.*d*("myTag", "the thread is done: " + name);  
 getInfo();  
 }  
  
 public void addGraph(JSONArray response){  
 int length = response.length();  
 //Initialize the rest of the casesPerDay  
 for(int i = 0; i < length; i++){  
 try{  
 JSONObject o = response.getJSONObject(i);  
 JSONArray names = o.names();  
 int j = 0;  
 boolean found = false;  
 boolean foundDate = false;  
 while(j < names.length() && (!found || !foundDate)){  
 if(names.getString(j).equals("positiveIncrease")){  
 this.casesPerDay.addFirst(o.getInt(names.getString(j)));  
 found = true;  
 }  
 if(names.getString(j).equals("date")){  
 this.dates.addFirst(o.getInt(names.getString(j)));  
 foundDate = true;  
 }  
 j++;  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
  
 createGraph();  
  
 Log.d("addGraph", "done");  
 //Log.d("results", toString());  
 this.handler.sendEmptyMessage(0);  
 }  
  
 public void getInfo(){  
 String url = "https://api.covidtracking.com/v1/states/" + this.name + "/current.json";  
  
 JsonObjectRequest jsonObjectRequest = new JsonObjectRequest  
 (Request.Method.GET, url, null, new Response.Listener<JSONObject>() {  
 @Override  
 public void onResponse(JSONObject response) {  
 getFirstInfo(response);  
 Log.d("State", "created");  
 }  
 }, new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
 // TODO: Handle error  
 Log.d("error", error.getMessage());  
 }  
 });  
 // Access the RequestQueue through your singleton class.  
 MySingleton.getInstance(this.context).addToRequestQueue(jsonObjectRequest);  
  
 String url2 = "https://api.covidtracking.com/v1/states/" + this.name + "/daily.json";  
 JsonArrayRequest request2 = new JsonArrayRequest  
 (Request.Method.GET, url2, null, new Response.Listener<JSONArray>() {  
 @Override  
 public void onResponse(JSONArray response) {  
 addGraph(response);  
 Log.d("State", "created");  
  
 }  
 }, new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
 // TODO: Handle error  
 //error.getMessage()  
 Log.d("error", error.getMessage());  
 }  
 });  
  
 // Access the RequestQueue through your singleton class.  
 MySingleton.getInstance(this.context).addToRequestQueue(request2);  
 }  
  
 public void getFirstInfo(JSONObject response){  
 this.date = 0;  
 this.deathIncrease = 0;  
 this.hospitalizedCurrently = 0;  
 this.onVentilatorCurrently = 0;  
 this.newCases = 0;  
 this.casesPerDay = new LinkedList<>();  
  
 JSONArray r = response.names();  
  
 for(int i = 0; i < r.length(); i++){  
 try {  
 if(r.getString(i).equals("date")){  
 this.date = response.getInt(r.getString(i));  
 int day = this.date%100;  
 int month = (this.date%10000 - day)/100;  
 int year = this.date/10000;  
 this.dateText = month + "/" + day + "/" + year;  
 }  
 if(r.getString(i).equals("deathIncrease")){  
 this.deathIncrease = response.getInt(r.getString(i));  
 }  
 if(r.getString(i).equals("hospitalizedCurrently")){  
 this.hospitalizedCurrently = response.getInt(r.getString(i));  
 }  
 if(r.getString(i).equals("onVentilatorCurrently")){  
 this.onVentilatorCurrently = response.getInt(r.getString(i));  
 }  
 if(r.getString(i).equals("positiveIncrease")){  
 this.newCases = response.getInt(r.getString(i));  
 }  
 } catch (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 @Override public String toString(){  
 String s = "";  
 s+= "name: " + this.name + ", ";  
 s+= "date: " + this.date + ", ";  
 s+= "dateText: " + this.dateText + ", ";  
 s+= "deathIncrease: " + this.deathIncrease + ", ";  
 s+= "hospitalizedCurrently: " + this.hospitalizedCurrently + ", ";  
 s+= "onVentilatorCurrently: " + this.onVentilatorCurrently + ", ";  
 s+= "newCases: " + this.newCases;  
  
 s+= "cases per day: ";  
 for(int i = 0; i < this.casesPerDay.size(); i++){  
 s+= casesPerDay.get(i) + ", ";  
 }  
 return s;  
 }  
  
 public String getName(){return name;}  
  
 public void createGraph(){  
 int firstDay = this.dates.getFirst();  
 int lastDay = this.dates.getLast();  
  
 //Calculate the first day in number of days  
 int dayFirst = firstDay%100;  
 int monthFirst = (firstDay%10000 - dayFirst)/100;  
 int yearFirst = firstDay/10000;  
 firstDay = getDays(yearFirst, monthFirst, dayFirst);  
  
 //Calculate each day into the number of day it is in the year  
 for(int i = 0; i < this.dates.size(); i++){  
 int date = this.dates.get(i);  
 int day = date%100;  
 int month = (date%10000 - day)/100;  
 int year = date/10000;  
 int totalDays = getDays(year, month, day);  
 int xValue = totalDays - firstDay + 1;  
 xValues.add(xValue);  
 }  
 }  
  
 public int getDays(int year, int month, int day){  
 int currentMonth = 1;  
 int numberOfDays = 0;  
 while(currentMonth < month){  
 if(currentMonth == 1) numberOfDays+= 31; //Janurary  
 else if(currentMonth == 2){ //Feburary  
 if(year%4 == 0) numberOfDays+= 29;//leap year  
 else numberOfDays+=28; //normal year  
 } else if(currentMonth == 3) numberOfDays+= 31; //March  
 else if(currentMonth == 4) numberOfDays+= 30; //April  
 else if(currentMonth == 5) numberOfDays+= 31; //May  
 else if(currentMonth == 6) numberOfDays+= 30; //June  
 else if(currentMonth == 7) numberOfDays+= 31; //July  
 else if(currentMonth == 8) numberOfDays+= 31; //August  
 else if(currentMonth == 9) numberOfDays+= 30; //September  
 else if(currentMonth == 10) numberOfDays+= 31; //October  
 else if(currentMonth == 11) numberOfDays+= 30; //November  
 else if(currentMonth == 12) numberOfDays+= 31; //December  
  
 currentMonth++;  
 }  
 numberOfDays+= day;  
 return numberOfDays;  
 }  
  
 public LinkedList<Integer> getCases(){return casesPerDay;}  
 public LinkedList<Integer> getxValues(){return xValues;}  
 public String getDate(){return dateText;}  
  
 public int getDeathIncrease(){return deathIncrease;}  
 public int getHospitalizedCurrently(){return hospitalizedCurrently;}  
 public int getOnVentilatorCurrently(){return onVentilatorCurrently;}  
 public int getNewCases(){return newCases;}  
  
  
}

Graph Class:

package com.example.osproject2;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.CornerPathEffect;  
import android.graphics.Paint;  
import android.graphics.Path;  
import android.util.AttributeSet;  
import android.util.Log;  
import android.view.View;  
  
import java.util.LinkedList;  
  
public class Graph extends View {  
 LinkedList<Integer> xValues;  
 LinkedList<Integer> cases;  
 int maxCases;  
 int numberOfDays;  
 int width;  
 int height;  
  
 public Graph(Context context, AttributeSet attrs) {  
 super(context, attrs);  
 this.xValues = new LinkedList<>();  
 this.cases = new LinkedList<>();  
 this.maxCases = 0;  
 this.numberOfDays = 0;  
 this.width = 0;  
 this.height = 0;  
 }  
  
 @Override  
 protected void onSizeChanged(int w, int h, int oldw, int oldh) {  
 super.onSizeChanged(w, h, oldw, oldh);  
 Log.*d*("width", String.*valueOf*(w));  
 this.width = w;  
 this.height = h;  
 }  
  
 @Override  
 protected void onDraw(Canvas canvas){  
 super.onDraw(canvas);  
 //Testing: draw a circle lol  
 Paint paint=new Paint();  
 paint.setStrokeWidth(3);  
  
 int padding = 32;  
 if(numberOfDays != 0){  
 double scaleX = (double)(width - padding)/(double)numberOfDays;  
 double scaleY = (double)(height - padding\*3)/(double)maxCases;  
 Log.*d*("scales: ", String.*valueOf*(scaleX) + " " + String.*valueOf*(scaleY));  
  
 //Draw graph lines  
 paint.setColor(Color.*WHITE*);  
 paint.setAlpha(100);  
 //Major lines  
 int middle = height/2;  
 int min = height-16;  
 int max = 32;  
 canvas.drawLine(0, min, width, min, paint);  
 canvas.drawLine(0, max, width, max, paint);  
 canvas.drawLine(0, middle, width, middle, paint);  
  
 //minor lines  
 paint.setAlpha(55);  
 int middleToMin = (min+middle)/2;  
 int middleToMax = (max+middle)/2;  
 canvas.drawLine(0, middleToMin, width, middleToMin, paint);  
 canvas.drawLine(0, middleToMax, width, middleToMax, paint);  
  
 //Add y-axis  
 paint.setAlpha(255);  
 paint.setTextSize((float) 50.0);  
 paint.setTextAlign(Paint.Align.*LEFT*);  
 paint.setStrokeWidth(5);  
 canvas.drawText("0", 24, min, paint);  
 canvas.drawText(String.*valueOf*(maxCases/2), 24, middle+25, paint);  
 canvas.drawText(String.*valueOf*(maxCases), 24, max+25, paint);  
  
 //Draw graph  
 paint.setColor(Color.*parseColor*("#52aeff"));  
 paint.setStrokeWidth(5);  
 paint.setStyle(Paint.Style.*STROKE*);  
 Path p = new Path();  
 Paint p2 = new Paint();  
 p2.setColor(Color.*parseColor*("#52aeff"));  
 p2.setStrokeWidth(5);  
 p2.setDither(true);  
 p2.setStrokeCap(Paint.Cap.*ROUND*);  
 p2.setStyle(Paint.Style.*STROKE*);  
 p2.setStrokeJoin(Paint.Join.*ROUND*);  
 p2.setPathEffect(new CornerPathEffect(30) );  
 p2.setAntiAlias(true);  
 p.moveTo(0, height - padding);  
 for(int i = 0; i < xValues.size(); i++){  
 int x = (int)(xValues.get(i)\*scaleX);  
 int y = height - (int)(padding+(cases.get(i)\*scaleY));  
 p.lineTo(x, y);  
 //Log.d("line", String.valueOf(x) + " " + String.valueOf(y));  
 }  
 canvas.drawPath(p, p2);  
  
 paint.setStyle(Paint.Style.*FILL*);  
  
 //Display current number of cases  
 paint.setTextSize(100);  
 paint.setAlpha(255);  
 paint.setColor(Color.*WHITE*);  
 canvas.drawText(String.*valueOf*(cases.getLast()) + " Cases", 50, 175, paint);  
 Log.*d*("hewwow", "uwu2");  
 } else{  
 paint.setStyle(Paint.Style.*FILL*);  
  
 //Display current number of cases  
 paint.setTextSize(100);  
 paint.setAlpha(255);  
 paint.setColor(Color.*WHITE*);  
 paint.setTextAlign(Paint.Align.*CENTER*);  
 canvas.drawText("Loading...", width/2, height/2, paint);  
 }  
  
  
 }  
  
 public void hewwow(){Log.*d*("hewwow", "uwu");}  
  
 public void addValues(LinkedList<Integer> xValue, LinkedList<Integer> yValue){  
 for(int i = 0; i < xValue.size(); i++){  
 this.xValues.add(xValue.get(i));  
 this.cases.add(yValue.get(i));  
  
 if(this.cases.getLast() > this.maxCases){  
 this.maxCases = this.cases.getLast();  
 }  
 }  
  
 this.numberOfDays = this.xValues.getLast();  
 Log.*d*("addValues", "done" + String.*valueOf*(maxCases));  
 }  
  
 @Override  
 public String toString(){  
 String s = "";  
 for(int i = 0; i < this.cases.size(); i++){  
 s += String.*valueOf*(this.cases.get(i)) + " ";  
 }  
  
 return s;  
 }  
}

Main Activity – onCreate():

BackgroundFragment f = (BackgroundFragment) this.getSupportFragmentManager().findFragmentById(R.id.*fragment*);  
  
//Get data  
Handler myHandler = new Handler() {  
 @Override  
 public void handleMessage (Message msg) {  
 if(currentLoaded == 49){  
 //Determine the top 5 states  
 //fill the array with the first 5 states in the list  
 for(int i = 0; i < 5; i++){  
 top5States[i] = new int[]{i, stateList[i].getNewCases()};  
 }  
  
 //Go through the rest of the states, and add/remove states as necessary  
 for(int i = 5; i < stateList.length; i++){  
 //go through the list  
 boolean greaterValue = false;  
 for(int j = 0; j < top5States.length; j++){  
 if(top5States[j][1] < stateList[i].getNewCases()){  
 greaterValue = true;  
 }  
 }  
  
 if(greaterValue){  
 //Select a victim state  
 int minStateIndex = 0;  
 int minStateValue = top5States[0][1];  
 for(int j = 1; j < top5States.length; j++){  
 if(top5States[j][1] < minStateValue){  
 minStateIndex = j;  
 minStateValue = top5States[j][1];  
 }  
 }  
 //Replace victim with current state  
 top5States[minStateIndex] = new int[]{i, stateList[i].getNewCases()};  
 }  
 }  
  
 //Sort list  
 for(int i = 1; i < top5States.length; i++){  
 int currentIndex = i;  
 int previous = i-1;  
 while(previous >=0 && top5States[currentIndex][1] < top5States[previous][1]){  
 //Swap previous and current  
 int tempIndex = top5States[currentIndex][0];  
 int tempValue = top5States[currentIndex][1];  
  
 top5States[currentIndex][0] = top5States[previous][0];  
 top5States[currentIndex][1] = top5States[previous][1];  
  
 top5States[previous][0] = tempIndex;  
 top5States[previous][1] = tempValue;  
  
 //decrement  
 currentIndex--;  
 previous--;  
 }  
 }  
  
 //Update the main page if necessary  
 if(activeFragment == 0){  
 // Reload current fragment  
 getSupportFragmentManager().popBackStack();  
 Bundle bundle = new Bundle();  
 bundle.putSerializable("Country", (Serializable) c);  
  
 if(currentLoaded == 49){  
 //Get top 5 states  
 State[] top5 = new State[5];  
 for(int i = 0; i < top5.length; i++){  
 top5[i] = stateList[top5States[i][0]];  
 }  
  
 bundle.putSerializable("States", top5);  
 }  
  
 MainFragment tabFragment = new MainFragment();  
 tabFragment.setArguments(bundle);  
 getSupportFragmentManager().beginTransaction().remove(f);  
 getSupportFragmentManager().beginTransaction().replace(R.id.*fragment*, tabFragment).addToBackStack(null).commit();  
 }  
 } else{  
 currentLoaded++;  
 }  
 }  
};  
  
Handler myHandler2 = new Handler() {  
 @Override  
 public void handleMessage (Message msg) {  
 if(activeFragment == 0){  
 // Reload current fragment  
 getSupportFragmentManager().popBackStack();  
 Bundle bundle = new Bundle();  
 bundle.putSerializable("Country", (Serializable) c);  
  
 if(currentLoaded == 49){  
 //Get top 5 states  
 State[] top5 = new State[5];  
 for(int i = 0; i < top5.length; i++){  
 top5[i] = stateList[top5States[i][0]];  
 }  
  
 bundle.putSerializable("States", top5);  
 }  
  
 MainFragment tabFragment = new MainFragment();  
 tabFragment.setArguments(bundle);  
 getSupportFragmentManager().beginTransaction().remove(f);  
 getSupportFragmentManager().beginTransaction().replace(R.id.*fragment*, tabFragment).addToBackStack(null).commit();  
 }  
 }  
};  
  
ThreadPoolExecutor mThreadPoolExecutor = new ThreadPoolExecutor(  
 *NUMBER\_OF\_CORES*, // Initial pool size  
 *NUMBER\_OF\_CORES*, // Max pool size  
 *KEEP\_ALIVE\_TIME*, // Time idle thread waits before terminating  
 *KEEP\_ALIVE\_TIME\_UNIT*, // Sets the Time Unit for KEEP\_ALIVE\_TIME  
 new LinkedBlockingDeque<Runnable>()); // Work Queue  
  
c = new Country(myHandler2);  
mThreadPoolExecutor.execute(c);  
  
currentLoaded = 0;  
stateList = new State[states.length];  
stateCount = stateList.length;  
  
for(int i = 0; i < stateList.length; i++){  
 stateList[i] = new State(states[i], this, myHandler);  
 mThreadPoolExecutor.execute(stateList[i]);  
}  
  
mThreadPoolExecutor.shutdown();  
  
TabLayout tabs = (TabLayout)findViewById(R.id.*tab\_layout*);  
  
//Add main fragment  
  
getSupportFragmentManager().popBackStack();  
Bundle bundle = new Bundle();  
MainFragment tabFragment = new MainFragment();  
tabFragment.setArguments(bundle);  
getSupportFragmentManager().beginTransaction().remove(f);  
getSupportFragmentManager().beginTransaction().replace(R.id.*fragment*, tabFragment).addToBackStack(null).commit();  
activeFragment = 0;  
  
  
tabs.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {  
  
 @Override  
 public void onTabSelected(TabLayout.Tab tab) {  
 if(tab.getPosition() == 0){  
 activeFragment = 0;  
 Log.*d*("tabClick", "0");  
 //getFragmentManager().popBackStackImmediate();  
 getSupportFragmentManager().popBackStack();  
 Bundle bundle = new Bundle();  
 bundle.putSerializable("Country", (Serializable) c);  
  
 if(currentLoaded == stateCount){  
 //Get top 5 states  
 State[] top5 = new State[5];  
 for(int i = 0; i < top5.length; i++){  
 top5[i] = stateList[top5States[i][0]];  
 }  
  
 bundle.putSerializable("States", top5);  
 }  
  
 MainFragment tabFragment = new MainFragment();  
 tabFragment.setArguments(bundle);  
 getSupportFragmentManager().beginTransaction().remove(f);  
 getSupportFragmentManager().beginTransaction().replace(R.id.*fragment*, tabFragment).addToBackStack(null).commit();  
 } else{  
 activeFragment = 1;  
 Log.*d*("tabClick", "1");  
 getSupportFragmentManager().popBackStack();  
 Bundle bundle = new Bundle();  
 bundle.putSerializable("States", (Serializable) stateList);  
 StatesFragment tabFragment = new StatesFragment();  
 tabFragment.setArguments(bundle);  
 getSupportFragmentManager().beginTransaction().remove(f);  
 getSupportFragmentManager().beginTransaction().replace(R.id.*fragment*, tabFragment).addToBackStack(null).commit();  
 }  
 }  
  
 @Override  
 public void onTabUnselected(TabLayout.Tab tab) {  
  
 }  
  
 @Override  
 public void onTabReselected(TabLayout.Tab tab) {  
  
 }  
});

MainFragment – onCreateView() :

// Inflate the layout for this fragment  
View view = inflater.inflate(R.layout.*fragment\_main*, container, false);  
  
//TextView textView = (TextView)view.findViewById(R.id.textView2);  
TableLayout tableLayout = (TableLayout)view.findViewById(R.id.*table*);  
tableLayout.setGravity(Gravity.*TOP*);  
tableLayout.setStretchAllColumns(true);  
Graph graph = (Graph)view.findViewById(R.id.*graph*);  
graph.hewwow();  
  
//Get country  
Bundle args = getArguments();  
c = (Country) args.getSerializable("Country");  
s = (State[]) args.getSerializable("States");  
if(c != null){  
 graph.addValues(c.getxValues(), c.getCases());  
}  
graph.refreshDrawableState();  
graph.invalidate();  
  
//Add rows  
if(s != null){  
 for(int i = s.length - 1; i >=0; i--){  
 TableRow tableRow = new TableRow(container.getContext());  
 TableRow.LayoutParams lp = new TableRow.LayoutParams(TableRow.LayoutParams.*MATCH\_PARENT*);  
 lp.width = TableRow.LayoutParams.*MATCH\_PARENT*;  
 tableRow.setGravity(Gravity.*TOP*);  
 tableRow.setLayoutParams(lp);  
  
 TextView stateName = new TextView(container.getContext());  
 stateName.setText(s[i].getName());  
 stateName.setGravity(Gravity.*LEFT*);  
 stateName.setTextSize(20);  
  
 TextView numCases = new TextView(container.getContext());  
 numCases.setText(String.*valueOf*(s[i].getNewCases()));  
 numCases.setGravity(Gravity.*RIGHT*);  
 numCases.setTextSize(20);  
  
 tableRow.addView(stateName);  
 tableRow.addView(numCases);  
 tableLayout.addView(tableRow);  
 }  
} else{  
 TableRow tableRow = new TableRow(container.getContext());  
 TableRow.LayoutParams lp = new TableRow.LayoutParams(TableRow.LayoutParams.*MATCH\_PARENT*);  
 lp.width = TableRow.LayoutParams.*MATCH\_PARENT*;  
 tableRow.setGravity(Gravity.*TOP*);  
 tableRow.setLayoutParams(lp);  
  
 TextView stateName = new TextView(container.getContext());  
 stateName.setText("Loading...");  
 stateName.setGravity(Gravity.*LEFT*);  
 stateName.setTextSize(20);  
  
 tableRow.addView(stateName);  
 tableLayout.addView(tableRow);  
}  
  
return view;

StateFragment – onCreateView:

// Inflate the layout for this fragment  
View view = inflater.inflate(R.layout.*fragment\_states*, container, false);  
  
TableLayout tableLayout = (TableLayout)view.findViewById(R.id.*table2*);  
tableLayout.setGravity(Gravity.*TOP*);  
tableLayout.setStretchAllColumns(true);  
  
//Get states  
Bundle args = getArguments();  
stateList = (State[]) args.getSerializable("States");  
  
//Add rows  
for(int i = 0; i < stateList.length; i++){  
 TableRow tableRow = new TableRow(container.getContext());  
 TableRow.LayoutParams lp = new TableRow.LayoutParams(TableRow.LayoutParams.*MATCH\_PARENT*);  
 lp.width = TableRow.LayoutParams.*MATCH\_PARENT*;  
 tableRow.setGravity(Gravity.*TOP*);  
 tableRow.setLayoutParams(lp);  
  
 TextView stateName = new TextView(container.getContext());  
 stateName.setText(stateList[i].getName());  
 stateName.setGravity(Gravity.*LEFT*);  
 stateName.setTextSize(24);  
  
 int finalI = i;  
 tableRow.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick( View v ) {  
  
 PopUpState popUpClass = new PopUpState();  
 popUpClass.PopUpWindow(v, stateList[finalI]);  
 }  
 } );  
  
 tableRow.addView(stateName);  
 tableLayout.addView(tableRow);  
}  
  
return view;

PopUpState – PopUpWindow:

//Create a View object yourself through inflater  
 LayoutInflater inflater = (LayoutInflater) view.getContext().getSystemService(view.getContext().*LAYOUT\_INFLATER\_SERVICE*);  
 View popupView = inflater.inflate(R.layout.*state\_popup*, null);  
  
 //Specify the length and width through constants  
 int width = LinearLayout.LayoutParams.*MATCH\_PARENT*;  
 int height = LinearLayout.LayoutParams.*MATCH\_PARENT*;  
  
 //Make Inactive Items Outside Of PopupWindow  
 boolean focusable = true;  
  
 //Create a window with our parameters  
 final PopupWindow popupWindow = new PopupWindow(popupView, width, height, focusable);  
  
 //Set the location of the window on the screen  
 popupWindow.showAtLocation(view, Gravity.*CENTER*, 0, 0);  
  
 //Set up graph  
 Graph graph = (Graph)popupView.findViewById(R.id.*graph3*);  
 if(state.getxValues() != null){  
 graph.hewwow();  
 graph.addValues(state.getxValues(), state.getCases());  
 graph.refreshDrawableState();  
 graph.invalidate();  
  
 //Set up table  
 TableLayout tableLayout = (TableLayout)popupView.findViewById(R.id.*table\_popup*);  
 tableLayout.setGravity(Gravity.*TOP*);  
 tableLayout.setStretchAllColumns(true);  
  
 //Add rows  
 //Name of state  
 TableRow tableRow = new TableRow(popupView.getContext());  
 TableRow.LayoutParams lp = new TableRow.LayoutParams(TableRow.LayoutParams.*MATCH\_PARENT*);  
 tableRow.setGravity(Gravity.*TOP*);  
 tableRow.setLayoutParams(lp);  
  
 TextView text = new TextView(view.getContext());  
 text.setText("State: " + state.getName());  
 //Log.d("popUp", state.getName());  
 text.setPadding(0, 5, 0, 5);  
 text.setGravity(Gravity.*LEFT*);  
 text.setTextSize(20);  
  
 tableRow.addView(text);  
 tableLayout.addView(tableRow);  
  
 //Date  
 TableRow tableDate = new TableRow(popupView.getContext());  
 tableDate.setLayoutParams(lp);  
  
 TextView dateText = new TextView(view.getContext());  
 dateText.setText("Date: " + state.getDate());  
 dateText.setPadding(0, 5, 0, 5);  
 dateText.setGravity(Gravity.*LEFT*);  
 dateText.setTextSize(20);  
  
 tableDate.addView(dateText);  
 tableLayout.addView(tableDate);  
  
  
 //New cases  
 TableRow newCasesRow = new TableRow(popupView.getContext());  
 newCasesRow.setLayoutParams(lp);  
  
 TextView newCasesText = new TextView(view.getContext());  
 newCasesText.setText("New Cases: " + state.getNewCases());  
 newCasesText.setPadding(0, 5, 0, 5);  
 newCasesText.setGravity(Gravity.*LEFT*);  
 newCasesText.setTextSize(20);  
  
 newCasesRow.addView(newCasesText);  
 tableLayout.addView(newCasesRow);  
  
  
 //Death increase  
 TableRow deathIncreaseRow = new TableRow(popupView.getContext());  
 deathIncreaseRow.setLayoutParams(lp);  
  
 TextView deathIncreaseText = new TextView(view.getContext());  
 deathIncreaseText.setText("Death Increase: " + state.getDeathIncrease());  
 deathIncreaseText.setPadding(0, 5, 0, 5);  
 deathIncreaseText.setGravity(Gravity.*LEFT*);  
 deathIncreaseText.setTextSize(20);  
  
 deathIncreaseRow.addView(deathIncreaseText);  
 tableLayout.addView(deathIncreaseRow);  
  
 //Hospitalized currently  
 TableRow hospitalizedCurrentlyRow = new TableRow(popupView.getContext());  
 hospitalizedCurrentlyRow.setLayoutParams(lp);  
  
 TextView hospCurrentText = new TextView(view.getContext());  
 hospCurrentText.setText("Hospitalized Currently: " + state.getHospitalizedCurrently());  
 hospCurrentText.setPadding(0, 5, 0, 5);  
 hospCurrentText.setGravity(Gravity.*LEFT*);  
 hospCurrentText.setTextSize(20);  
  
 hospitalizedCurrentlyRow.addView(hospCurrentText);  
 tableLayout.addView(hospitalizedCurrentlyRow);  
  
 //on ventilator currently  
 TableRow onVentCurrently = new TableRow(popupView.getContext());  
 onVentCurrently.setLayoutParams(lp);  
  
 TextView onVentText = new TextView(view.getContext());  
 onVentText.setText("On Ventilator Currently: " + state.getOnVentilatorCurrently());  
 onVentText.setPadding(0, 5, 0, 5);  
 onVentText.setGravity(Gravity.*LEFT*);  
 onVentText.setTextSize(20);  
  
 onVentCurrently.addView(onVentText);  
 tableLayout.addView(onVentCurrently);  
 } else{  
 graph.refreshDrawableState();  
 graph.invalidate();  
  
 //Set up table  
 TableLayout tableLayout = (TableLayout)popupView.findViewById(R.id.*table\_popup*);  
 tableLayout.setGravity(Gravity.*TOP*);  
 tableLayout.setStretchAllColumns(true);  
  
 //Add rows  
 TableRow tableRow = new TableRow(popupView.getContext());  
 TableRow.LayoutParams lp = new TableRow.LayoutParams(TableRow.LayoutParams.*MATCH\_PARENT*);  
 tableRow.setGravity(Gravity.*TOP*);  
 tableRow.setLayoutParams(lp);  
  
 TextView text = new TextView(view.getContext());  
 text.setText("Loading...");  
 text.setPadding(0, 5, 0, 5);  
 text.setGravity(Gravity.*LEFT*);  
 text.setTextSize(20);  
  
 tableRow.addView(text);  
 tableLayout.addView(tableRow);  
 }  
  
  
 //Handler for clicking on the inactive zone of the window  
  
 popupView.setOnTouchListener(new View.OnTouchListener() {  
 @Override  
 public boolean onTouch(View v, MotionEvent event) {  
 //Close the window when clicked  
 popupWindow.dismiss();  
 return true;  
 }  
 });