Criterion C – Development

Section Overview

Overview

CoronavirusAnalyzer (JFrame child) contains panes (JPanel children), in GridBagLayout.
Updated design has 3 columns:

- Left: ranking.
- Middle: map, scrollbar, play, stop buttons, settings radio buttons, country search.
- Right (specific to country search): country history graph and table.

Rankings, map, and graph achieved through SVG composition, drawn with Batik library. Data from CSV files and processed with library.

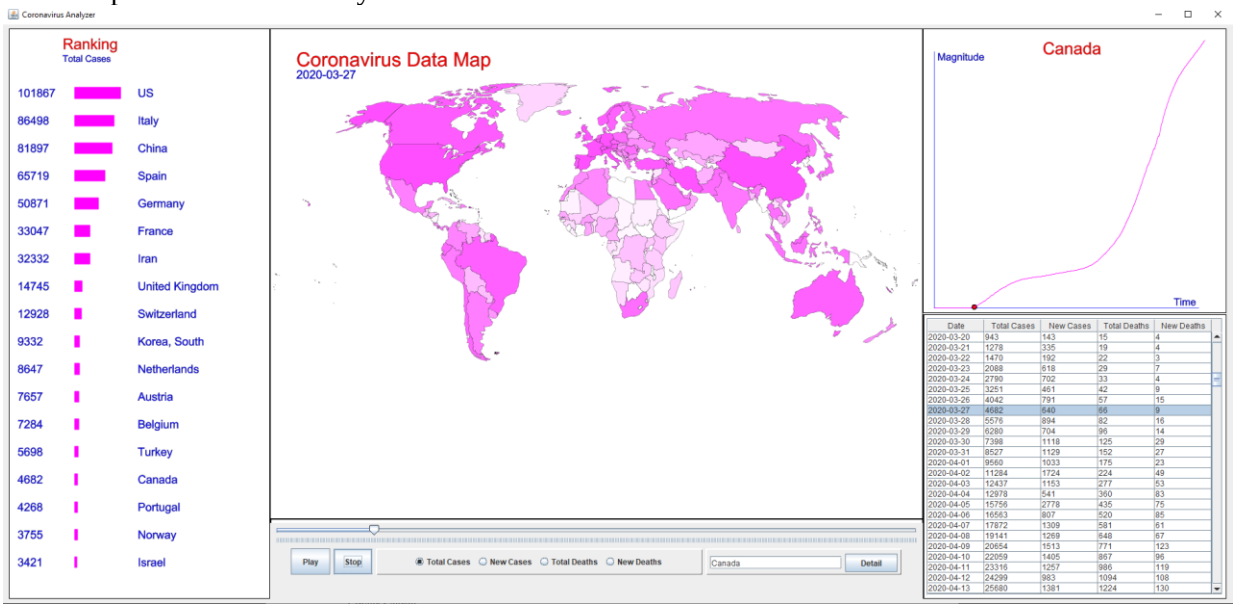

Figure C.1: updated graphical user interface (GUI). See Criterion B for initial GUI, Appendix A for why UI changed.

Libraries & Source Acknowledgements

JDK

GUI Components

```
import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.AffineTransform;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
```

Code C.1: sets up GUI.

```
import javax.swing.AbstractButton;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel
;
import javax.swing.WindowConstants;
```

Code C.2: for GUI.

Data Structures

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map.Entry;
import java.util.Map;
import java.util.TreeMap;
```

Code C.3: built-in data structures expedites development time.

Utilities

```
import java.net.URL;
import java.security.CodeSource;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService
;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
```

Code C.4: task helpers.

Input

```
import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.io.IOException;
```

Code C.5: helps take input.

SVG Document

```
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.svg.SVGDocument;
import org.w3c.dom.svg.SVGElement;
import org.w3c.dom.svg.SVGRect;
import org.w3c.dom.svg.SVGSVGElement
;
```

Code C.6: for SVG document reading, processing, making, drawing.

<table>
<tr><th colspan="2">External Libraries</th><th>Internal Libraries – Stack</th></tr>
<tr>
<td>SVG Composition
Jars (Apache, 2021)
batik-all-1.14.jar
xml-apis-ext-1.3.04.jar
xmlgraphics-commons-2.6.jar
<pre>import org.apache.batik.anim.dom.SAXSVGDocumentFactory;
import org.apache.batik.bridge.ViewBox;
import org.apache.batik.swing.JSVGCanvas;
import org.apache.batik.util.XMLResourceDescriptor;</pre>Code C.7: packages for SVG composition.</td>
<td>CSV File
Library for CSV file loading (Merritt, 2014).<pre>import au.com.bytecode.opencsv.CSVReader;</pre>Code C.8: CSV file loading package.</td>
<td>I decided to use my teacher’s stack instead of JDK’s because the linked list stack is clean and efficient.<pre>import Stack.LLStack;
import Stack.Stack;</pre>Code C.9: stack packages.</td>
</tr>
</table>

Covid-19 CSV Data Files	**World Map SVG**
Source updates daily (CSSEGISandData, n.d.).	Source (simplemaps, n.d.).

Data Structure & Abstract Data Type

See Appendix B and C for code omission denoted by ellipses (i.e., “…”).

List & ArrayList and Stack & LLStack

```
    private List<String> ranking = new ArrayList<String>();
    ...
/**
 * Update the ranking according to the data that is desired to be displayed.
 * @param covid
 * @return void
 */
public void updateDate(CAReadCSV covid)
 {
      // Stores already sorted list of values of a chosen statistic type (total case, new cases, etc.).
    this.ranking = covid.rankList;
    ...
    refresh();
   }

/**
 * Refresh the ranking.
 * @return void
 */
private void refresh()
{
    ...
    Stack reversed = new LLStack();
    // Extract highest numerical data value of top COUNTRIES (i.e., 20) countries.
    for(int j = Math.max(ranking.size()-COUNTRIES+1, 0); j < ranking.size(); j++)
        reversed.push(ranking.get(j)); // Stack allows the data to be used in correct order later.
    // Extract the highest value statistic to set anchor for bar width.
    int maxStat = Integer.parseInt(((String)((LLStack)reversed).peek()).split("\\.")[0].trim());

    for(int j = 0; !reversed.isEmpty(); j++)
    {
        String [] elements = ((String)reversed.pop()).split("\\.");
        String stats = elements[0].trim();
        String country = elements[1];
        ...
    }
    ...
}
```

Code C.10: implementation of the ArrayList and LLStack abstract data types.

ArrayList<String> ranking

- Purpose: store full sorted rankings transferred from CAReadCSV.
- Justification:
  - I assigned an ArrayList instance to the List interface to provide extensibility and more general and broader options.
  - Ensure compatibility when receiving assignment of rankList in CAReadCSV.

- I wish to obtain values by index, so ArrayList would be more efficient than any Set interface implementation.
  - Why not TreeSet?
    - Rankings is already sorted by numerical value. TreeSet would sort by string, thus, bungling the correct order.

LLStack reversed

- Purpose: extract the top 20 rankings.
- Justification:
  - I declared the interface memory address and instantiated the LLStack class to allow extensibility and modification ease.
  - Values in "ranking" are in increasing order, so the top 20 covid stats is the last 20 in the ArrayList. Thus, the last 20 entries must be reversed to display the most severe countries in correct order.
  - Perfect stack implementation because of the last-in-first-out nature.

CAQueue & CALinkedListQueue & CAQueueNode & Interface Implementation

Class	Description
CAQueue	Queue interface.
CALinkedListQueue	Linked list queue implementing CAQueue.
CAQueueNode	Node class for linked list queue class.

Table C.1: class overview.

```
/**
 * Interface for a custom queue.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public interface CAQueue
{
    /**
     * Checks if the queue is empty.
     * @return Whether the queue is empty
     */
    public boolean isEmpty();

    /**
     * Dequeue an item from the queue.
     * @return The item that was dequeued from the queue
     */
    public Object dequeue();

    /**
     * Enqueue an item to queue.
     * @param item Item to be enqueued
     * @return Whether enqueue was successful
     */
    public boolean enqueue(Object item);
}
```

Code C.11: CAQueue interface.

Justification:

- Structure of interface implementation allows further developed by future users.
  - Easy creation of another queue type, e.g., ArrayList queue.

```
/**
 * Implementing the interface CAQueue to create a linked list queue.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CALinkedListQueue implements CAQueue
{
    private CAQueueNode front;
    private CAQueueNode rear;

    /**
     * Set default values.
     */
    public CALinkedListQueue()
    {
        front = null;
```

```
        rear = null;
    }

    /**
     * Checks if the linked list queue is empty.
     * @return void
     */
    public boolean isEmpty()
    {
        return front == null;
    }

    /**
     * Dequeue an item from the linked list queue.
     * @return The item that was dequeued.
     */
    public Object dequeue()
    {
        if(!isEmpty())
        {
            Object value = front.getValue();
            if(front.getNext() == null)
                rear = null;
            front = front.getNext();
            return value;
        }
        return null;
    }

    /**
     * Enqueue an item to the linked list queue.
     * @param item The item to be enqueued
     * @return Whether enqueue was successful
     */
    public boolean enqueue(Object item)
    {
        CAQueueNode newItem = new CAQueueNode(item);
        if(isEmpty())
            front = newItem;
        else
            rear.setNext(newItem);
        rear = newItem;

        return true;
    }
}
```

Code C.12: implementation of the CAQueue interface with a linked list queue, CALinkedListQueue.

Features:

- First-in-first-out implementation by creating and breaking links between nodes.
    - Enqueue at linked-list rear.
    - Dequeue at linked-list front.

```
/**
 * Node class for the CALinkedListQueue class.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CAQueueNode
{
    private Object value;
    private CAQueueNode next;

    /**
     * Sets inital values.
     * @param value The initial value of the node
     */
    public CAQueueNode(Object value)
    {
        this.value = value;
        this.next = null;
    }

    /**
     * Set the value at a node.
     * @param value The value to assign
     * @return void
     */
    public void setValue(Object value) {
        this.value = value;
    }

    /**
     * Set the next node link.
     * @param next The next node
     * @return void
     */
    public void setNext(CAQueueNode next) {
```

```
        this.next = next;
    }

    /**
     * Get the value at the node.
     * @return The value at the node
     */
    public Object getValue() {
        return this.value;
    }

    /**
     * Get the next node.
     * @return The next node
     */
    public CAQueueNode getNext() {
        return this.next;
    }
}
```

Code C.13: the node class that is responsible for the nodes in the linked list queue. CALinkedListQueue is composed of nodes because it is a linked list implementation of a queue.

CALinkedListQueue Application

```
    /**
     * Loads individual CSV day data files into TreeMap "data". Process "data" to setup the calculation for new cases and new deaths.
     * @return void
     */
    public void loadDataFiles()
    {
        CALinkedListQueue filenames = getFilenames(); // List of filenames for the covid-19 statistics.
        while(!filenames.isEmpty()) // Loop through each filename to process the data in each file.
            loadFileFromURL((String)filenames.dequeue()); // Call method to process and store data in CSV file into the TreeMap "data".
        ...
        }
```

Code C.14: “driver code” to start the data loading.

```
    /**
     * Get the flienames of the CSV files by looking in the jar.
     *
     * @return A custom linked list queue which contains the flienames
     */
    private CALinkedListQueue getFilenames()
    {
        /*
        Create an instance of the CALinkedListQueue class to store the filenames of all CSV files that contain covid-19 statistics.
        Two examples below shows the CSV filenames that are stored in the CALinkedListQueue instance.
            "data/csse_covid_19_daily_reports/01-01-2021.csv"
            "data/csse_covid_19_daily_reports/01-02-2021.csv"
        */
        CALinkedListQueue filenames = new CALinkedListQueue();

        // Find all the covid-19 CSV data files in the JAR file and add them to the linked list queue.
        try {
            CodeSource src = getClass().getProtectionDomain().getCodeSource(); // How to get the path of a running JAR file? // https://stack
overflow.com/questions/320542/how-to-get-the-path-of-a-running-jar-file
            if (src != null)
            {
                // How to list the files inside a JAR file? // https://stackoverflow.com/questions/1429172/how-to-list-the-files-inside-a-
jar-file
                URL jar = src.getLocation();
                ZipInputStream zip = new ZipInputStream(jar.openStream());
                while(true) {
                    ZipEntry e = zip.getNextEntry();
                    if (e == null)
                        break;
                    String name = e.getName();
                    if (!name.startsWith("data/csse_covid_19_daily_reports/"))
                        continue;
                    if (!name.endsWith(".csv"))
                        continue;
                    filenames.enqueue(name); // Add the filename to the linked list queue.
                }
            }
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
        return filenames; // Return the linked list queue of filenames with the covid-19 data.
    }
```

Code C.15: filenames are enqueued to the linked list queue. The method returns a CALinkedListQueue that stores the CSV filenames.

Additional Techniques:

- Obtaining the path of a running jar file (Chaves, Zarkonnen, Blarzek, & peterh, 2008).
- Listing files in a jar file (Ryz, erickson, 030, & V, 2009).

- Justification:
  - Data should be accessible when packaged in a jar.
- Class dependency between CALinkedListQueue and CAReadCSV.

```
/**
 * Read the contents of the CSV files and process it.
 * @param filename The current file to read and process
 * @return void
 */
private void loadFileFromURL(String filename) // Process the data in each CSV file.
{
    /*
    The first three entries of the "data/csse_covid_19_daily_reports/01-01-2021.csv" file is shown below. Other CSV files are in the same format.
    FIPS,Admin2,Province_State,Country_Region,Last_Update       ,Lat     ,Long_    ,Confirmed,Deaths,Recovered,Active,Combined_Key,Incident_Rate     ,Case_Fatality_Ratio
        ,     ,            ,Afghanistan   ,2021-01-02 05:22:33,33.93911,67.709953,51526    ,2191  ,41727    ,0     ,Afghanistan ,0.0              ,4.2522221790940495
        ,     ,            ,Albania       ,2021-01-02 05:22:33,41.1533 ,20.1683  ,58316    ,1181  ,33634    ,23501 ,Albania     ,2026.4090624782818,2.025173194320598
    */

    // Extracting the year, month, and day from the CSV filename, e.g., "data/csse_covid_19_daily_reports/01-01-2021.csv".
    ...
    // Reformat the order of date to year, month, day.
    ...
    try
    {
        ...
        ... // Load the CSV file.
        ... // Store all the values read from the CVS file into a list of string arrays.

        // Obtain the index of the column headings.
        ...
        ... // Looping through the list rows.
            ... // First row contains the headings rather than statistical values.
                ... // Loop through the string array, i.e., columns of first row.
                ...

            // All code in this loop below, runs from second row to last, thus, they are statistical values.
            // Obtain the statistical value based on row (from loop index) and column (based on the i=0 loop column headings).
            ...

            // Adjust special cases.
            ...

            // Store values.
            ...
                ... // Used to sum up statistics for entire countries.
            ...
                // If there is already an entry for a country, add the existing statistical values as sum.
                ...
        // One complete day of statistics for each country is added to "data". This method is called for daily CSV files.
        ...
    }
    catch(Exception ex)
    {
        ex.printStackTrace();
    }
}
```

Code C.16: dequeued filenames are processed. See CAReadCSV.java in Appendix B.

Purpose:

- CALinkedListQueue stores filenames in the queue, then dequeuing one by one to process each file individually.

Justification:

- Queue:
  - Organization:
    - Dequeue maintains enqueue order, thus, a filename print statement would notify any errors, allowing us to easily discover which CSV file caused the error. Try-catch tells the error location in the code but not the filename. Maintaining order allows ease in finding the erroneous file.
  - Memory efficiency:
    - Unneeded files are dequeued, and garbage collected by Java once processed.
- Linked-List:
  - Dynamic:
    - Linked-list does not limit the queue size.
      - Number of CSV data files changes daily with new covid updates.
      - Array requires a file count, creating inefficiency in implementation.

- Memory efficiency: Java garbage collects unneeded broken links. Array cannot do this.
- I only need enqueue and dequeue features, i.e., searching is not required:
  - LinkedList outperforms ArrayList because removing item in ArrayList requires shifting the entire ArrayList, which is inefficient compared to deleting a single link.

Additional Techniques:

- Exception handling technique in file loading with try-catch.
  - Justification: prevent program crashing if file cannot be accessed.

Map & TreeMap Structure and Map.Entry & RecordItem Techniques

```
    // TreeMap "data" consists of the date, country, and statistical value.
    public Map<String, TreeMap<String, RecordItem>> data = new TreeMap<String, TreeMap<String, RecordItem>>();

    ...

    /**
     * Loads individual CSV day data files into TreeMap "data". Process "data" to setup the calculation for new cases and new deaths.
     *
     * @return void
     */
    public void loadDataFiles()
    {
        CALinkedListQueue filenames = getFilenames(); // List of filenames for the covid-19 statistics.
        while(!filenames.isEmpty()) // Loop through each filename to process the data in each file.
            loadFileFromURL((String)filenames.dequeue()); // Call method to process and store data in CSV file into the TreeMap "data".

        TreeMap<String, RecordItem> previous = new TreeMap<String, RecordItem>();

        /*
        Loops through the TreeMap "data" to calculate the statistics for new cases and
        new deaths through subtraction of adjacent total cases and total deaths data respectively.
        */
        Iterator<Map.Entry<String, TreeMap<String, RecordItem>>> i = data.entrySet().iterator(); // The day is the key in "data".
        while(i.hasNext())
        {
            Map.Entry<String, TreeMap<String, RecordItem>> m = i.next();

            TreeMap<String, RecordItem> tm = m.getValue(); // The value is a TreeMap with country as key and statistics as value.

            // Loop through the statistics of each country at a specific day.
            Iterator<Map.Entry<String, RecordItem>> j = tm.entrySet().iterator();
            while(j.hasNext()){
                Map.Entry<String, RecordItem> n = j.next();

                String country = n.getKey(); // Country is the key of the TreeMap that is the value of the "data" TreeMap.
                RecordItem v = n.getValue(); // RecordItem contains the statistics of a country at a day.

                RecordItem pv = previous.get(country);
                int pd = 0; // Previous total deaths.
                int pc = 0; // Previous total cases.
                if(pv != null) // If statistics for a country is not the first entry, i.e., not the first day of statistics.
                {
                    // Save the previous day information.
                    pc = pv.totalCases;
                    pd = pv.totalDeaths;

                    // Set the curent day as the previous day, for the processing of the next day information.
                    pv.totalCases = v.totalCases;
                    pv.totalDeaths = v.totalDeaths;
                }
                else // If it is the first entry for a country at a day, then there are no previous data, so add data that will become the pr
evious values for the next day.
                    previous.put(country, new RecordItem(v.totalCases, v.totalDeaths));

                // Call the current day RecordItem to calculate its new cases and new deaths based on the previous day statistics.
                v.setPrevious(pc, pd);
            }
        }
    }
```

Code C.17: shows one aspect of the nested TreeMap, TreeMap<String, TreeMap<String, RecordItem>> "data". loadDataFiles() loops through the nest with nested while loops. Map iteration performed with Iterator and Map.Entry. RecordItem stores, total cases, total deaths, new cases, new deaths. See Appendix B.

Purpose of "data" is responsible for storing:

- CSV file information: country, date (from filename), and total cases, deaths, which are sums of all country regional values in the CSV files.
- New cases, deaths, are calculated within RecordItem by setPrevious(int c, int d).

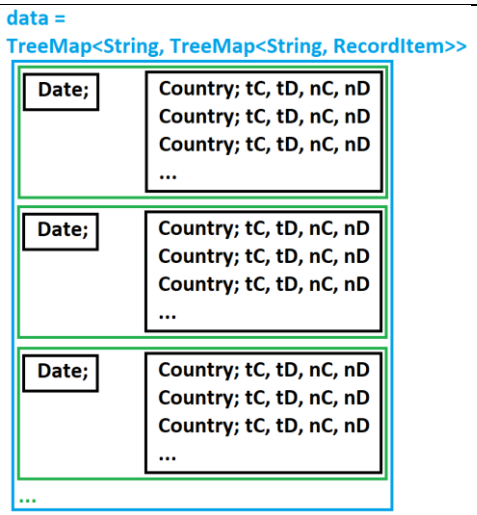

Figure C.2: visualization of the "data" showing date as the outer TreeMap key, country as the inner TreeMap key, and RecordItem containing total cases (tC), total deaths (tD), new cases (nC), and new deaths (nD).

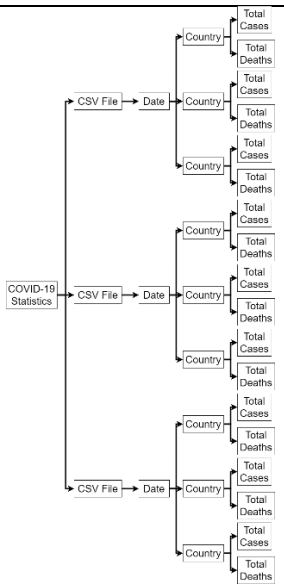

Figure C.3: visualization of the hierarchical structure of the coronavirus statistics from the CSV files.

Nested TreeMap Justification:

- Recall purpose: store data read from CSV files.
- Organization:
  - See Figure C.3. Maintains the organized hierarchical structure of Covid-19 data. Each date has a country and two statistical values and two processed values (RecordItem).
    - RecordItem justification: more organized than other structures like array.
    - TreeMap lists countries in alphabetic order since key sorting is automatic.
    - TreeMap sorts dates. I also reformatted CSV filename from MM-DD-YYYY to YYYY-MM-DD for sorting.
- Dynamic (user-friendly, memory efficient):
  - Its static data structure counterpart, 3D array is fixed in size at compile time.
    - Adding CSV files to update statistics changes file count.
      - TreeMap eliminates the need to update a constant value that specifies the number of files in a 3D array (i.e., String[][][] data = new String[FILES][MAX_COUNTRIES][4];).
    - Number of countries in each CSV file vary depending on day.
      - Thus, checking number of countries in every CSV file is inefficient.

3D array surpasses a nested TreeMap in ease of iterating and accessing values. Nevertheless, I iterated the nested TreeMap by manipulating Set.entrySet(), Map.Entry and Iterator. Code C.17 procedure is below.

- Iterator "i" takes value of "data" using entrySet().
- While i.hasNext().
  - Set Map.Entry to next value in iterator, i.next().
    - Map.Entry Key: date String.
    - Map.Entry Value: TreeMap of country, RecordItem.
      - Iterator "j" takes value of this TreeMap.
      - While j.hasNext().
        - Set Map.Entry with j.next().
          - Map.Entry Key: country.
          - Map.Entry Value: RecordItem.
            - Access RecordItem values for tC, tD, nC, nD.

CABinarySearchTree & CARankNode

Class overview in table below.

Class	Description
CABinarySearchTree	Implementation of a custom binary search tree.
CARankNode	Node class used by CABinarySearchTree

Table C.2: class overview.

```
 private CABinarySearchTree rankOfConfirmed; // Stores country and rank.
 ...

/**
 * Updates ranking based on day and presentation type.
 * @param nDay The current day
 * @param presentationType The statistical type
 * @return void
 */
public void updateRankOfConfirmed(int nDay, int presentationType)
{
    ...
    rankOfConfirmed = new CABinarySearchTree();
    rankList = new ArrayList<String>(); // Format: number.country.

    ... // Set the current entry to the last day entry.
    ... // If the parameter day is a valid day.
        ... // Loop through TreeMap "data" while having an index.
            ... // Linear search for the current day corresponding to the parameter current day.
                ... // Set the entry to the one specified by the parameter.

    ... // TreeMap "daily" has key country and value statistics for a single day.
            ... // Store the statistics value corresponding to a data type together with the country name separated by a period.

    ... // Quick sort the values of current chosen statistic type (total confirmed case, new cases, etc.).

    for(int j = 0; j < rankList.size(); j++) {
        String s = rankList.get(j).substring(rankList.get(j).indexOf(".")+1);
        // Insert the countries and their rank to binary search tree.
        rankOfConfirmed.insert(s, j);
    }
}

...

/**
 * Gets the data for the desired country.
 * @param country The desired country
 * @param cData Contains the statistical value and rank
 * @return Whether retrieval was successful
 */
public boolean getCountryData(String country, int [] cData)
{
    String name = getCountryName(country);
    ...
    if(value == null || rankOfConfirmed == null)
        return false;

    CARankNode node = (CARankNode)rankOfConfirmed.search(name);

    if(node == null)
        return false;

    Integer rank = node.getRank();

    cData[0] = value.totalCases;
    cData[1] = rank;

    return true;
}
```

Code C.18: see Appendix B for full code.

Justification

- By looping through rankList, the country name can be extracted. This is after the values has already been sorted. Looping in that chronology yields the ordered list of countries. Thus, the index is indicative of the ranking.
    - Passing these values to insert(String country, int rank) creates a new node and adds it to the binary search tree. Quick sort sorts the values. Binary search tree sorts the countries. The ranking is retrieved in the search method which implements a binary search (see later section).

- Binary Tree:
    - Efficient searching. Other data structures such as implementations of Maps or Lists would be inconvenient for ranking search by country because node creation is simpler than storing a concatenated string with a period delimiter.

Additional Techniques

- Linear search.
- NullPointerException prevention by checking for null values.

Algorithms

Binary Search

```
/**
 * Implements a Binary Search Tree to organize country and data ranking.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CABinarySearchTree
{
   private CARankNode rootPointer;

   /**
    * Sets initial value of root pointer.
    */

   /**
    * Insert a new CARankNode into the binary search tree based on country and ranking.
    *
    * @param country The name of the country
    * @param rank The rank of the country
    * @return Whether insert is succesful
    */
   ...

   /**
    * Look for the country in the binary search tree and return the CARankNode.
    * @param country Name of country to be found
    * @return The CARankNode of the country or null if country is not in the tree.
    */
   public CARankNode search (String country)
   {
      // No tree exists.
      if (rootPointer == null)
         return null;

      CARankNode temp = rootPointer;
      while(temp != null)
      {
         // Country found.
         if(country.equals(temp.getCountry()))
            return temp;
         // Search deeper in the tree.
         temp = (country.compareTo(temp.getCountry()) < 0)?temp.getLeft():temp.getRight();
      }
      return null; // Noting is found.
   }
}
```

Code C.19: binary search in binary search tree.

Application.

- Searching for the country to obtain the ranking.

Justification.

- Quick, O(log n) time complexity.
- Convenient to implement with an existing binary search tree.

Quick Sort & Recursion

```
    /**
     * Performs quicksort to sort the ranking.
     * @param data The data to sort
     * @param start The starting position/index
     * @param end The ending position
     * @return void
     */
    public void quickSort(List<String> data, int start, int end)
    {
        if(start < end)
        {
            // Find partition.
```

```
            String pivot = data.get(end);
            int i = start-1;

            for(int j = start; j < end; j++)
            {
                if(data.get(j).compareTo(pivot) <= 0)
                {
                    i++;

                    String swapTemp = data.get(i);
                    data.set(i, data.get(j));
                    data.set(j, swapTemp);
                }
            }
            String swapTemp = data.get(i+1);
            data.set(i+1, data.get(end));
            data.set(end, swapTemp);

            // Continue sorting subsections.
            quickSort(data, start, i);
            quickSort(data, i+2, end);
        }
    }
```

Code C.20: implementation of quick sort to sort the rankList, which is a list of strings of "number.country" where number is the statistical value.

Justification

- Efficient, O(N log N) average time complexity.
- Will not cause significant delays to program execution compared to other $O(N^2)$ sorting algorithms.

OOP

Some OOP concepts are shown in previous sections (e.g., interface implementation, class dependency, class association).

Encapsulation

```
/**
 * Node class for CABinarySearchTree.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CARankNode
{
   private String country;
   private int rank;
   private CARankNode left, right;

   /**
    * Set initial valueS.
    * @param country The current country choice
    * @param rank The rank of the country
    */
   public CARankNode(String country, int rank)
   {
      this.country = country;
      this.rank = rank;
      left = right = null;
   }

   /**
    * Set the left CARankNode value.
    * @param left The left node
    * @return void
    */
   public void setLeft(CARankNode left) {
      this.left = left;
   }

   /**
    * Set the right CARankNode value.
    * @param left The right node
    * @return void
    */
   public void setRight(CARankNode right) {
      this.right = right;
   }

   /**
    * Get the left CARankNode value.
    * @return The left CARankNode
    */
```

```
    public CARankNode getLeft() {
        return left;
    }

    /**
     * Get the right CARankNode value.
     * @return The right CARankNode
     */
    public CARankNode getRight() {
        return right;
    }

    /**
     * Return the current node country.
     * @return The country
     */
    public String getCountry() {
        return country;
    }

    /**
     * Return the current node rank.
     * @return The rank
     */
    public int getRank() {
        return rank;
    }
}
```

Code C.21: sample manifestation of encapsulation in CARankNode.java.
Justification:

- Wrapping private variables and public accessor mutator methods in the same class to prevent direct access to node values.

Dynamic Polymorphism – Method Overriding

```
            // Instantiations of the JButton class.
            JButton buttonPlay = new JButton("Play");
            JButton buttonStop = new JButton("Stop");

            // Set button size and associate an action to a button press.
            buttonPlay.setPreferredSize(new Dimension(60, 40));
            buttonStop.setPreferredSize(new Dimension(60, 40));
            buttonPlay.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    scrollBar.play();
                }
            });
            buttonStop.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    scrollBar.stop();
                }
            });
```

Code C.22: sample dynamic polymorphism in CoronavirusAnalyser.java when adding action listener to play and stop buttons.
Justification:

- Allows the play and stop buttons have custom functionality (i.e., calling custom play and stop methods).

Static Polymorphism – Method Overloading

```
    /**
     * Initiates the update of the graph.
     * @param covid Has access to CSV data
     * @param country The current country to display the graph for
     * @return void
     */
    public void updateDate(CAReadCSV covid, String country)
    {
        covid.refreshHistory((TreeMap<String, Integer>)history, country);
        this.country = country;
        refresh();
    }

    /**
     * Update based on new day.
     * @param covid Has access to CSV data
     * @return void
     */
    public void updateDate(CAReadCSV covid)
    {
```

```
        currentDay = covid.currentEntry.getKey();
        refresh();
    }

    /**
     * Updates the graph by composing the SVG.
     *
     * @return void
     */
    private void refresh()
    {
        ...
    }
```

Code C.23: static polymorphism in CAHistoryPane.java. Note there is additional OOP technique used here, which is association relationship between CAHistoryPane and CAReadCSV.

```
    /**
     * Updates the graph and table in the right panel that is specific to a country, when new country is requested.
     *
     * @return void
     */
    private void updateDetail()
    {
        dataTable.updateDate(map.covid, tfCountry.getText());
        history.updateDate(map.covid, tfCountry.getText());
    }
```

Code C.24: sample use of the two parameter updateDate() when updating the historical graph of a country for a date.

```
    /**
     * Set up slider, which represents the timeline, and add it to the JPanel.
     */
    public CAScrollPane()
    {
        // Set up the slider and add a ChangeListener to it to respond to changes.
        timeline = new JSlider(); // Create a horizontal slider to represent the timeline.
        timeline.setValue(1); // Set the initial slider position at 1.
        timeline.setPreferredSize(new Dimension(1000, 50));
        timeline.setPaintTicks(true);
        timeline.setPaintLabels(true);
        timeline.setMajorTickSpacing(1);
        timeline.addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent evt) {
                JSlider source = (JSlider)evt.getSource();
                if (!source.getValueIsAdjusting()) {
                    int value = (int)source.getValue();
                    map.updateDate(value);
                    dataTable.updateDate(map.covid);
                    history.updateDate(map.covid);
                    ranking.updateDate(map.covid);
                }
            }
        });

        // Add the slider to the JPanel.
        this.add(timeline);
    }
```

Code C.25: calling the single parameter updateDate of the CAHistoryPane when the slider value changes to indicate a change in date.

Justification:

- Organization:
    - Use the same updateDate() method name with different functionalities, both methods updates based on date. Achieved through distinct number of parameters.

Inheritance

```
/**
 * Main class to run the Coronavirus Analyzer program.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CoronavirusAnalyzer extends JFrame
{
    ...
    /**
     * Setup and add components the frame of the Coronavirus Analyzer program.
     *
     * @return void
     */
```

```
    private void run()
    {
        ...
        // This section sets up the position of panels on the frame.
        {
            // An overall dashboard will contain all panels in a GridBagLayout. Overall, panels are catagorized in three columns.
            JPanel dashboard = new JPanel();
            dashboard.setLayout(new GridBagLayout());

            // The left column will contain the ranking panel.
            JPanel left  = new JPanel(new GridLayout(1, 1, 2, 2));
            left.add(ranking);

            // The right column will contain the graph and table of data with equal dimensions.
            JPanel right = new JPanel(new GridLayout(2, 1, 2, 2));
            right.add(history);
            right.add(dataTable);

            // Instantiate the GridBagConstraints class to customize the layout.
            GridBagConstraints gbc = new GridBagConstraints();
            gbc.fill = GridBagConstraints.VERTICAL;

            // The middle colum will contain the map, timeline, and the button control/search field with custom dimensions.
            gbc.gridx = 1; gbc.gridy = 0; dashboard.add(map, gbc);
            gbc.gridx = 1; gbc.gridy = 1; dashboard.add(scrollBar, gbc);
            gbc.gridx = 1; gbc.gridy = 2; dashboard.add(bottom, gbc);

            // Add the left and right columns to the dashboard. These columns take up equal vertical space as three rows in the middle column
.
            gbc.gridheight = 3;
            gbc.gridx = 2; gbc.gridy = 0; dashboard.add(right, gbc);
            gbc.gridx = 0; gbc.gridy = 0; dashboard.add(left, gbc);

            // Add the dashboard containing all panels to the frame.
            frame.add(dashboard);
        }

        // This section will sets up frame settings.
        {
            setSize(WINDOW_WIDTH, WINDOW_HEIGHT); // Set window size.
            setLocationRelativeTo(null); // Center frame in screen.
            setVisible(true); // Allow frame to be visible.
            setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); // End program when close button is pressed.
        }
        ...
    }

    /**
     * Calls the constructor method of the superclass JFrame to create a frame labelled "Coronavirus Analyzer".
     */
    public CoronavirusAnalyzer()
    {
        super("Coronavirus Analyzer");
    }

    /**
     * Starts the Coronavirus Analyzer program.
     * @param args Command-line arguments
     * @return void
     */
    public static void main(String[] args)
    {
        frame = new CoronavirusAnalyzer();
        frame.run();
    }
}
```

Code C.26: The CoronavirusAnalyzer class inherits the JFrame class which allows CoronavirusAnalyzer to call JFrame methods. Note additional techniques includes implementation of a GridBagLayout.

Justification for JFrame child:

- Ease:
    - No need to instantiate additional JFrame's.
    - Easily add and call JFrame methods (e.g., super()).
- Organization:
    - Clear hierarchical structure as JFrame encompasses the panes (JPanel children).

Aggregation & Map Colour

```
/**
 * Loads and processes the SVG world map file and composes SVG map when requested.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CAMapPane extends JPanel
{
```

```
    ...
    public CAReadCSV covid = new CAReadCSV();
    ...

    /**
     * Get the number of CSV files which corresponds to the number of days.
     * @return The number of CSV files
     */
    public int getCovidNumberOfDays() {
        return covid.data.size();
    }

    /**
     * Sets up the panel and laods the CSV files and SVG file.
     * @return void
     */
    public void createComponents()
    {
        ...
        covid.loadDataFiles();  // Load and process all CSV files.
        try {
            loadDocument(); // Load map data from SVG.
        } catch(IOException ex) {ex.printStackTrace();}

        updateDate(-1);
    }

    /**
     * Refreshes the map.
     * @return void
     */
    private void refresh()
    {
        covid.updateRankOfConfirmed(currentDay, presentationType);
        int[] cData = new int[2]; // Stores statistics value and rank.
            ...
            int j = 0xff; // Rank colour.
            if(covid.getCountryData(cn, cData)) // If successful retrieval of statistic value and rank.
            {
                j = cData[1]; // cData[1] is the country rank.
                if(j > 0xff) // If rank is greater than 255.
                    j = 0xff; // Set rank colour to 255.
                j = 0xff - j; // Rank colour is darker if ranking is larger.
            }
            strSvg += "<path fill=\"#"+ (String.format("%02X%02X%02X", 0xff,j,0xff)) +"\" "; // Add colour.
        ...
    }
    ...
}
```

Code C.27: manifestation of class aggregation between CAMapPane and CAReadCSV through CAReadCSV covid.

Justification for choice of aggregation:

- The “covid” instance of CAReadCSV is a liaison between the CSV data and all the other classes. E.g., “covid” is used in CAMapPane to:
    - Get number of files by accessing the size of the nested TreeMap “data”.
    - Load and process all CSV files.
    - Update ranking based on current day and statistical type.
        - Use the statistical value and rank to colour the map.

Additional Techniques (colour):

- Map colours determined by the country ranking; higher rankings have darker shades.
    - Achieved through a linear relationship between 0xff (max single RGB value) and rank. Transitioning from (0xff, 0, 0xff) to (0xff, 0xff, 0xff) creates the gradient from dark to light for fuchsia.
    - Justification:
        - Effective and easy; creates a gradient between countries and clearly shows the severity of each country.

Composition

```
/**
 * Class to organize SVG data.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
class SVGPath
{
    public String id = "";
```

```
    public String name = new String();
    public String cls = new String();
    public String d = new String();
}

/**
 * Stores extracted SVG data.
 * @author Vincent Zhang
 * @since 2021-03-24
 */
public class CASVGData
{
    private ArrayList<SVGPath> mPaths = new ArrayList<SVGPath>();

    /**
     * Add a new path.
     * @return Index for new entry
     */
    public int addPath()
    {
        mPaths.add(new SVGPath());
        return mPaths.size()-1;
    }

    /**
     * Set the id.
     * @param i The current position
     * @param id The id value
     * @return void
     */
    public void setId(int i, String id) {
        mPaths.get(i).id = id;
    }

    /**
     * Set the d value.
     * @param i Current position
     * @param d D value
     * @return void
     */
    public void setD(int i, String d) {
        mPaths.get(i).d = d;
    }

    /**
     * Set the class value.
     * @param i Current position
     * @param cls Class value
     * @return void
     */
    public void setClass(int i, String cls) {
        mPaths.get(i).cls = cls;
    }

    /**
     * Set the name value.
     * @param i Current position
     * @param name Name value
     * @return void
     */
    public void setName(int i, String name) {
        mPaths.get(i).name = name;
    }

    /**
     * Get total number of paths.
     * @return The number of paths
     */
    public int getSize() {
        return mPaths.size();
    }

    /**
     * Get id at position i.
     * @param i Current position
     * @return The id at the current position
     */
    public String getId(int i) {
        return mPaths.get(i).id;
    }

    /**
     * Get the name at position i.
     * @param i Current position
     * @return The name at the current position
     */
    public String getName(int i) {
        return mPaths.get(i).name;
    }

    /**
     * Get the d value at position i.
```

```
     * @param i The current position
     * @return The d value at the current position
     */
    public String getD(int i) {
        return mPaths.get(i).d;
    }

    /**
     * Get the class value at position i.
     * @param i Current position
     * @return The class value at the current position
     */
    public String getClass(int i) {
        return mPaths.get(i).cls;
    }
}
```

Code C.28: class composition between CASVGData and SVGPath.

Purpose

- CASVGData stores the data when adding a new path from the SVG world map.
- Allows retrieval of SVG map data when refreshing the map.
- As seen in the SVG file, four important path information are, id, name, class, and d.

Justification

- CASVGData contains one instance variable ArrayList of type SVGPath. I.e., CASVGData is composed of solely a list of SVGPath's.

Techniques

Use of Constants

```
    // Declare and initialize constants.
    private static final int WINDOW_WIDTH = 1900;
    private static final int WINDOW_HEIGHT = 924;
    private static final String LABEL_TOTAL_CASES = "Total Cases";
    private static final String LABEL_NEW_CASES = "New Cases";
    private static final String LABEL_TOTAL_DEATHS = "Total Deaths";
    private static final String LABEL_NEW_DEATHS = "New Deaths";
       ...
                // Instantiations of the JRadioButton class.
                JRadioButton rb1 = new JRadioButton(LABEL_TOTAL_CASES); // Stands for radio button 1.
                JRadioButton rb2 = new JRadioButton(LABEL_NEW_CASES);
                JRadioButton rb3 = new JRadioButton(LABEL_TOTAL_DEATHS);
                JRadioButton rb4 = new JRadioButton(LABEL_NEW_DEATHS);
      ...
    // Instantiate the ActionListener class to apply it to the radio buttons.
    ActionListener rbActionListener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent actionEvent) {
            AbstractButton src = (AbstractButton) actionEvent.getSource();
            if     (src.getText().equals(LABEL_TOTAL_CASES))  updatePresentationType(1);
            else if(src.getText().equals(LABEL_NEW_CASES))    updatePresentationType(2);
            else if(src.getText().equals(LABEL_TOTAL_DEATHS)) updatePresentationType(3);
            else if(src.getText().equals(LABEL_NEW_DEATHS))   updatePresentationType(4);
        }
    };
    rb1.addActionListener(rbActionListener);
    rb2.addActionListener(rbActionListener);
    rb3.addActionListener(rbActionListener);
    rb4.addActionListener(rbActionListener);
```

Code C.29: sample use of constants. Justification: better organization, easier changes in label name, and maintaining good coding style.

Lambda Expression

```
    // For the purpose of executing the task of scrolling the timeline repeatedly over a fixed interval.
    private final ScheduledExecutorService SCHEDULER = Executors.newScheduledThreadPool(1);
    private ScheduledFuture<?> handleOfPlay = null;
    ...

    /**
     * Begin the autoscroll if there is no autoscrolling present.
     * @return void
     */
    public void play()
    {
        if(handleOfPlay != null) // If the timeline is already playing then no need to add an overlapping auto scroller.
            return;
        // Lambda expression simplifies and clarifies the code in place for the @override run() method in Runnable. Also loop the timeline when end is reached.
        handleOfPlay = SCHEDULER.scheduleAtFixedRate (()->timeline.setValue((timeline.getValue()+1) % timeline.getMaximum()), 300, 300, TimeUnit.MILLISECONDS);
    }
```

Code C.30: use of lambda expression to facilitate the autoscrolling. Justification: simplify and clarify code. Function created without belonging to a class. More efficient use of space in the code writing.

Drawing with SVG

```
    /**
     * Refresh the ranking.
     * @return void
     */
    private void refresh()
    {
        // Compose the SVG for the image to be painted.
        String strSvg = new String();
        strSvg += "<svg baseprofile=\"tiny\" fill=\"#ececec\" stroke=\"black\" stroke-linecap=\"round\" stroke-linejoin=\"round\" stroke-width=\".2\" "
                + "version=\"1.2\" viewbox=\"0 0 " + PANEL_WIDTH + " " + PANEL_HEIGHT + "\"  "
                + "width=\"" + PANEL_WIDTH + "\"  height=\"" + PANEL_HEIGHT + "\" xmlns=\"http://www.w3.org/2000/svg\">\n";
        strSvg += "<rect x=\"0\" y=\"0\" width=\"" + PANEL_WIDTH + "\" height=\"" + PANEL_HEIGHT + "\" style=\"fill:rgb(255,255,255);stroke-width:3;stroke:white\" />";
        strSvg += "<text x=\"100\" y=\"40\" fill=\"red\" font-size=\"2.5em\">Ranking</text>\n";
        strSvg += "<text x=\"100\" y=\"65\" fill=\"blue\" font-size=\"1.5em\">"+rankingType+"</text>\n";

        Stack reversed = new LLStack();
        // Extract highest numerical data value of top COUNTRIES (i.e., 20) countries.
        for(int j = Math.max(ranking.size()-COUNTRIES+1, 0); j < ranking.size(); j++)
            reversed.push(ranking.get(j)); // Stack allows the data to be used in correct order later.
        // Extract the highest value statistic to set anchor for bar width.
        int maxStat = Integer.parseInt(((String)((LLStack)reversed).peek()).split("\\.")[0].trim());

        for(int j = 0; !reversed.isEmpty(); j++)
        {
            String [] elements = ((String)reversed.pop()).split("\\.");
            String stats = elements[0].trim();
            String country = elements[1];

            double barWidth = Math.max((double)Integer.parseInt(stats)/maxStat * BAR_WIDTH, 1); // If a bar is less than width 1, set it to width 1.

            // Draw statistics value, bar, and country name.
            strSvg += "<text x=\"" + STATISTIC_ALIGN + "\" y=\"" + (TOP_MARGIN+BAR_SCALE*(j+1)) + "\" fill=\"blue\" font-size=\"6mm\">" + stats + "</text>\n";
            strSvg += "<rect x=\"" + BAR_ALIGN       + "\" y=\"" + (TOP_MARGIN+(BAR_SCALE*(j+1))-
18.0) + "\" width=\"" + barWidth + "\" height=\""+BAR_HEIGHT+"\" style=\"fill:rgb(255,0,255);stroke-width:3;stroke:rgb(255,0,255)\" />";
            strSvg += "<text x=\"" + COUNTRY_ALIGN    + "\" y=\"" + (TOP_MARGIN+BAR_SCALE*(j+1)) + "\" fill=\"blue\" font-size=\"6mm\">" + country + "</text>\n";
        }
        strSvg += "</svg>";

        // Generate the SVG document from String strSvg.
        SVGDocument doc = null;
        try {
            doc = (new SAXSVGDocumentFactory(XMLResourceDescriptor.getXMLParserClassName())).createSVGDocument(null, new ByteArrayInputStream(strSvg.getBytes("UTF-8")));
        }
        catch (Exception ex) {ex.printStackTrace();}

        // Pass the document to the canvas for Batik to draw.
        svgCanvas.setDocumentState(JSVGCanvas.ALWAYS_DYNAMIC);
        svgCanvas.setDocument(doc);
    }
```

Code C.31: simply specify parameters for a tag. E.g., a text tag, needs x-, y- coordinate, color, size, and words. I also draw the relative bars, which is a rectangle where I specify the x-, y- coordinate, etc. Justification: clean, efficient, easy implementation. Constants are used to calculate bar length ratios

Table Scrolling

```
    /**
     * Scrolls and highlights table to current day entry.
     *
     * @param row The current day row
     * @return void
     */
    private void scrollToRow(int row)
    {
        JScrollBar bar = scrollPane.getVerticalScrollBar();
        int max = bar.getMaximum();
        int min = bar.getMinimum();
        int ext = bar.getVisibleAmount(); // Height of scrollbar.
        int val = bar.getValue(); // Top of scrollbar.
        int rows = table.getRowCount();

        if  (rows * (val    -min) >= (row+1) * (max - min)   // When current row is above view window.
          || rows * (val+ext-min) <=  row    * (max - min))  // When current row is below view window.
            bar.setValue(row * (max - min) / rows + min);    // Reset the current view window.

        table.clearSelection();
        table.addRowSelectionInterval(row, row);
        this.repaint();
    }
```

Code C.32: original creative technique for scrolling to the correct line in the table through ratios.

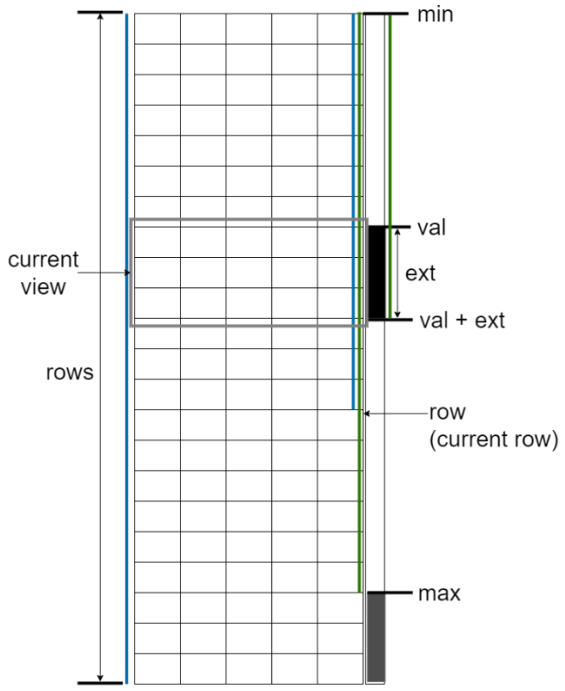

Figure C.4: diagram of the table model.

Explanation:
See second if statement in Code C.32.

$$Equation\ 1: \frac{val + ext - min}{(max - min)} \leq \frac{row}{rows}$$

When true, row is below the current view. Similar mechanism when row is above current view. Scroll to the following location by setting "val".

$$Equation\ 2: \frac{val - min}{(max - min)} = \frac{row}{rows}$$

Scroll to the position where the percentage of the max min range at val is the same as the percentage of the current row to the number of rows. Rearranging:

$$Equation\ 3: val = \frac{row * (max - min)}{rows} + min$$

(1141 words)

Summary

Read if desired. Not part of word count. Non-exhaustive list of concepts discussed only in Criterion C. Purpose: help jog memory (in addition to Section Overview) when grading this Criterion.

Library Implementation

- GUI Components
- Data Structures
- Utilities
- Input
- SVG Document
- SVG Composition
- CSV File Loading
- Stack, LLStack

Data Structures & Abstract Data Type

- List, ArrayList
- Stack, LLStack
- CAQueue, CALinkedListQueue, CAQueueNode
- Map, TreeMap
- TreeMap<String, TreeMap<String, RecordItem>>
- CABinarySearchTree & CARankNode
- RecordItem

Extensibility

- Class implementation

- Constants
- Comments
    - Javadoc
- Coding style

Algorithms

- Quick sort
- Recursion
- Binary search

Techniques

- Obtain running jar path.
- List files in a jar.
- Exception handelling (e.g. NullPointerException prevention).
- Map.Entry, Iterator
- Load, read, process CSV files.
- GridBagLayout usage.
- RGB hexadecimal colour gradient manipulation.
- Lambda expression.
- Drawing with SVG.
- Table scaling.

Understanding of Concepts

- Java garbage collection
- Static, dynamic, abstract data type
    - 3D array.
- Concatenation and delimiters.

OOP

- Interface, implementation
- Dependency
- Association
- Encapsulation
- Dynamic polymorphism (method overriding)
- Static polymorphism (method overloading)
- Inheritance
- Aggregation
- Composition

Works Cited

Apache. (2021, January 12). Batik-all JAR files with all dependencies. Retrieved from https://jar-download.com/artifacts/org.apache.xmlgraphics/batik-all

Chaves, T., & Zarkonnen. (2008, November 26). How to get the path of a running JAR file? (Blarzek & Peterh, Eds.). Retrieved from https://stackoverflow.com/questions/320542/how-to-get-the-path-of-a-running-jar-file

CSSEGISandData. (n.d.). CSSEGISandData/COVID-19. Retrieved from https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_daily_reports

Merritt, A. (2014, August 13). Au.com.bytecode JAR files with all dependencies. Retrieved from https://jar-download.com/artifacts/au.com.bytecode

Ryz, O., & Erickson. (2009, September 15). How to list the files inside a JAR file? (030 & D. V, Eds.). Retrieved from https://stackoverflow.com/questions/1429172/how-to-list-the-files-inside-a-jar-file

Simplemaps. (n.d.). Free blank world map in svg - resources. Retrieved from https://simplemaps.com/resources/svg-world