University of Toronto Mississauga

CSC207 Fall 2022

Design Document

AUTHORS

Jay Patel
Waleed Haddad
Aditya Kumar
Ryan Reza

GITHUB TEAM: **CSC207 Enjoyers**
GITHUB URL:
https://github.com/waleed-haddad/Data-Visualization—Climate-Change

November 13 2022

Contents

2 Project Identification

For this project, the group has decided to create a Data Visualizer that will display climate change data through an interactive world map. The focus will primarily be on displaying carbon emissions per capita for countries on an interactive world map, which will show each country's relative contributions toward global warming. It is important to spread awareness about this topic as both first-world and privileged countries, even Canada, comprise some of the major contributors to global warming. The goal is to display carbon emissions through coloured ranges and include a rank for each country, indicating where it lies in the countries with the most carbon emissions. Various modes, such as colour-blind and night mode, will also be implemented as accessibility options for the users.

This project is inspired by the Urban Tree Data Visualizer lab. The data visualizer explored in this project though expands the scope to an interactive world map which will enable the user to zoom and select a country. It will also allow the user to search countries by name with a search bar, among other features.

3 User Stories

Name	ID	Owner	Description	Implementation Details	Priority	Effort
Develop Country hashmap with Data [NEW]	1.0	Waleed	As a user, I want to have a hashmap which contains countries and their respective CO2 emissions, while also being able to access the countries and their various attributes.	Create Country object and store data from.csv files in attributes, create hashmap with Country objects as keys and Co2 emissions as attributes.	1	2
Select Country and Display Data [EDITED]	1.2	Aditya	As a user, I want to be able to easily select a country on the interactive world map so that I can see the respective data of the country including its emissions and rank in a list of all countries and their emission rates.	Use the map UI to highlight the country once selected by the user. Create an Observer pattern that handles such events and display the data accordingly to a display bar.	1	1
Search country to get CO_2 emissions	2.1	Aditya	As a user, I want a UI that allows me to search a country to find the CO_2 emissions for a country efficiently.	Create a UI so that users can type country names in a search bar which will output CO_2 emissions for the searched country.	1	1
Various Modes and Views	2.2	Jay	As a user, I want different options for the display of the world map (colorblind or dark mode).	Create a UI where users can switch views depending on the different categories provided by the API.	3	1
Color Countries Based on Carbon Emission Per Capita	2.3	Ryan	As a user, I want to see a world map with each country with a different shade of color based on carbon emission and its rank so that I can compare different countries.	Use the Iterator pattern to assign colors to each country and then integrate it into the WorldMapView of the Observer Pattern.	1	3

Name	ID	Owner	Description	Implementation Details	Priority	Effort
Undo and Return to Default	2.4	Waleed	As a user, I want the ability to undo any filters I apply to the map so that I can easily go back to what my map originally looked like. This includes reverting back to Default View.	Use the Commander design pattern to save commands given by the user into an Iterable data structure, so that these commands (filtering, zooming in/out) can be undone with a button.	2	3
Integrate Search and Undo	2.5	Aditya	As a user, I want to maintain the ability to use the undo button and unselect countries while using the search button as well.	Worked with the WorldMapView and Command pattern to make sure the according actions are triggered	2	2
Implement Undo Button	2.6	Waleed	As a user, I want the ability to press the undo button on the map UI to undo changes made to the default map.	Changed the undo button to be its own class, integrating it into the Observer pattern.	2	2
Displaying Map Legend	2.7	Jay	As a user, I want know the see the map legend based on the colours chosen for each view.	Imported a picture of the map legend based on the view of the map chosen.	2	2
Implement Accessibility User Interface	2.8	Aditya	As a user, I want the have buttons to switch between the different views of the application - Default view, Colour Blind mode, Night mode.	Integrated the accessibility buttons with the Observer patter.	2	2
Dark Mode	2.9	Jay	As a user, I want a UI that allows me to activate a night mode which recolors the map to reduce blue/bright light emissions.	Create a UI so that users can switch between the default view and a dark mode with a button press.	2	2
Color-blind Mode [NEW]	2.10	Jay	As a user, I want a UI that allows me to activate a colour-blind mode which uses a different hue of colours to ease the experience for those with visual impairments.	Create a UI so that users can switch between the default view and a colour-blind mode with a button press.	2	2

Rank Countries Based on Carbon Emission Per Capita	2.11	Ryan	As a user, I want to see rank of the country based on carbon emissions when selected from the world map so that I can compare different countries.	Use the Iterator pattern to assign ranks to each country and then integrate it into the WorldMapView of the Observer Pattern.	1	3
Text Sizing [SCRAPPED]	3.2	Ryan	As a user, I want different options for text sizes to choose the size that works best with my eyesight.	Create a UI so that users can change the text size on the screen using the Slider class inherited from JavaFx.	1	1
Initial Map Setup [SCRAPPED]	3.3	Ryan	As a user, I want the option to initially set all my accessibility features so that I can set colours in a way that would favour my preferences.	Use WorldMapView to create a map according to specific information input by the user	3	1
Ranked List of Countries [SCRAPPED]	4.2	Waleed	As a user, I want to see a ranked list of countries based on CO2 emissions, so that I can see the major contributors.	Creating a UI with a drop-down menu that shows a list of countries in a ranked fashion.	3	2

4 Software Design

4.1 Design Pattern 1: Strategy Pattern

Overview: This pattern will be used to implement User Story 2.2 and accessibility features (Various Displays/Modes).

UML Diagram: Refer to Figure 1, below.

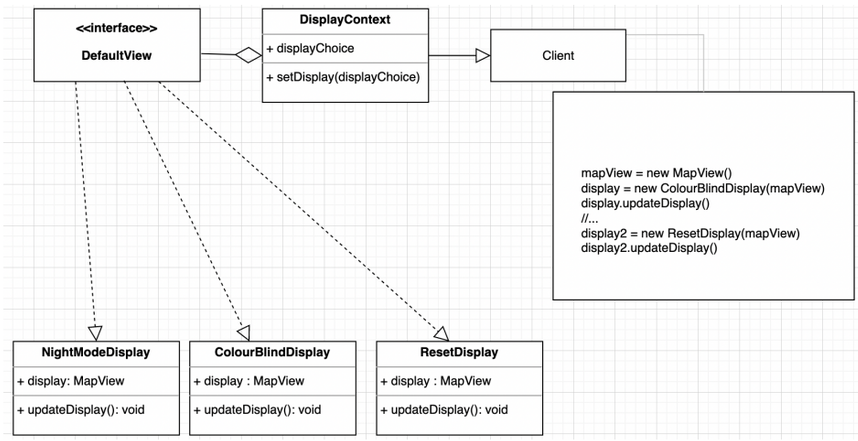

Figure 1: The Strategy Pattern

Implementation Details: The UML diagram outlines these main components:

1. **DefaultView**: The DefaultView interface, which includes one method: updateDisplay.
2. **ResetDisplay, ColourBlindDisplay, NightModeDisplay**: ResetDisplay, ColourBlindDisplay, NightModeDisplay which are the ConcreteStrategies that implement the DefaultView interface
3. **DisplayContext**: DisplayContext which includes one method: setDisplay.
4. **Client**: The Client sends the actions to the DisplayContext in order to update the display.

The setDisplay method in the DisplayContext updates all the required attributes of the current display (default if no display is currently shown) and switches to that display. The Client is responsible for creating the correct display and through the DisplayContext will show the display depending on the Display. ColourBlindDisplay changes the way the colour is shown by implementing the updateDisplay method and changes to a colour that accommodates all types of colour blindness. Additionally, NightModeDisplay will also implement the updateDisplay method in order to provide a more suitable experience for the user towards later times in the day. ResetDisplay will implement the updateDisplay method in order to reset the current display back to the default which is done through the setColour method from the World class.

4.2 Design Pattern 2: Observer Pattern

Overview: This pattern will be used to implement the Graphical User Interface.

UML Diagram: Refer to Figure 2, below.

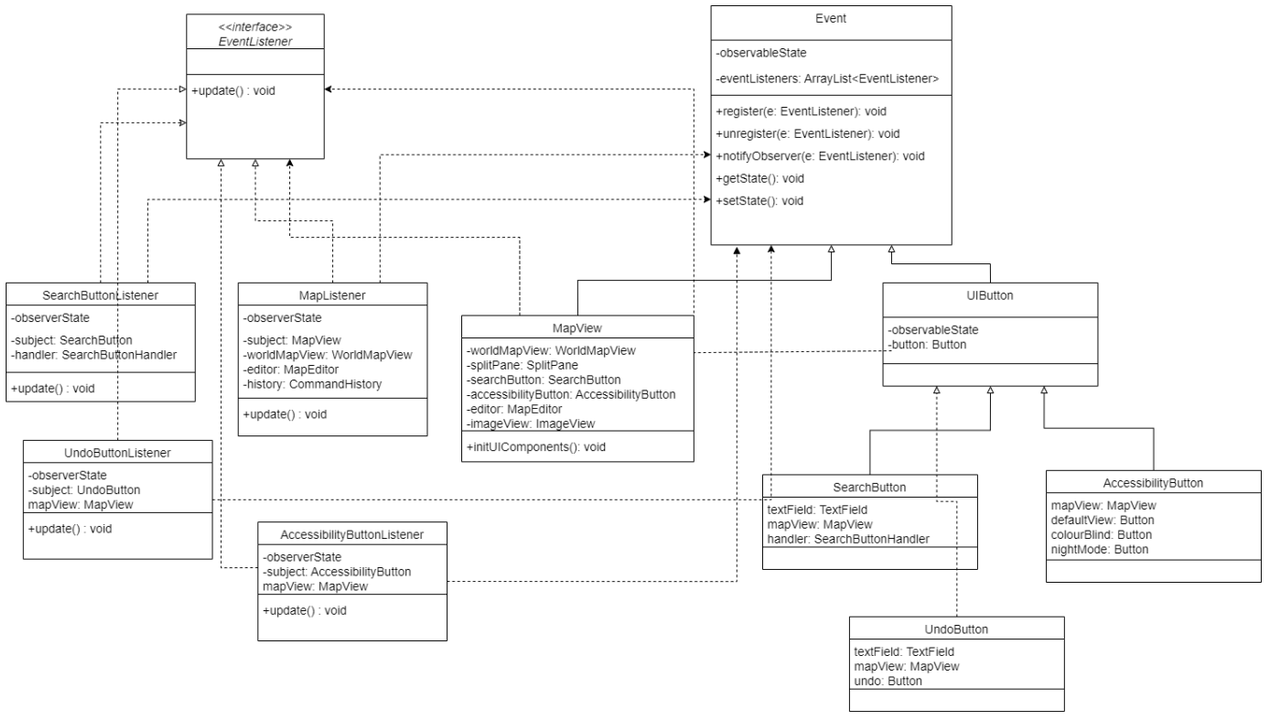

Figure 2: The Observer Pattern

Implementation Details: The UML diagram outlines these main components:

1. EventListener, which is the Observer in the pattern.
2. SearchButtonListener, AccessibilityButtonListener, UndoButtonListener and MapListener which are the ConcreteObserver type from the pattern and implement EventListener.
3. Event, which is the Observable in the pattern and is an abstract class.
4. MapView and UIButton which are the ConcreteObservable type from the pattern and extend Event.
5. SearchButton, UndoButton and AccessibilityButton which inherit the UIButton class.

EventListener

The SearchButtonListener, UndoButtonListener and AccessibilityButtonListener listens for the search button, undo button and accessibility buttons to be pressed and calls the update function accordingly. The MapListener listens for clicks on the interactive world map and waits for countries to be selected, calling the update function accordingly.

Event

The Event has the register, unregister, notifyObserver, getState, and setState functions which register and unregister listeners to the Event and notifies the observer accordingly. The MapView and UIButton represent the events on the interactive world map and buttons respectively. The MapView has the additional method of initUIComponents which initialize the attributes of MapView. This method was made to allow for sanity testing to occur as testing cannot occur with UI components. The UIButton class is the parent class of SearchButton, UndoButton and AccessibilityButton. The SearchButton represents the ability for the user to search a country and retrieve data accordingly. The AccessibilityButton represents the ability to change the views of the map from default to colour blind mode to night mode. The UndoButton represents the ability to undo the selection of the country last selected.

4.3 Design Pattern 3: Command Pattern

Overview: This pattern will be used to implement User Story 3.1 (Undo/Return to Default).

UML Diagram: Refer to Figure 3, below.

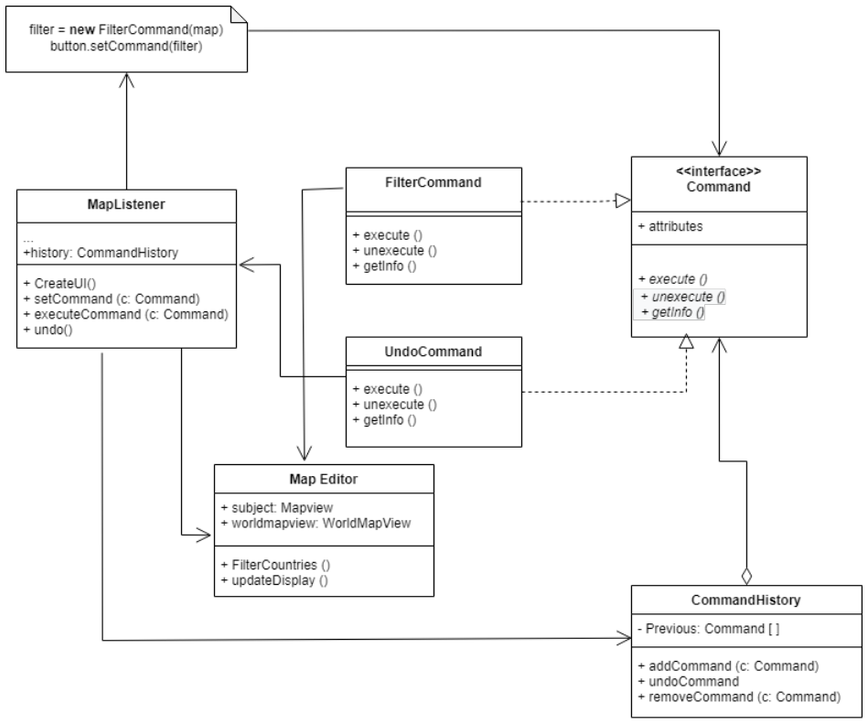

Figure 3: The Command Pattern

Implementation Details: The UML diagram outlines these main components:

1. **Map Listener:** This acts as the client which creates and configures concrete command objects.
2. **Command:** This is the command interface, which is implemented by all concrete command classes.
3. **FilterCommand, UndoCommand:** These concrete command classes implement the command interface.
4. **Map Editor:** This is the receiver class and contains methods for carrying out all of the commands.
5. **CommandHistory:** The CommandHistory class acts as the Iterable class to store commands.

Map Listener

The Map Listener also acts as the Invoker as it triggers commands rather than sending them directly to the Map Editor, which is the receiver. It passes all requests into the command's constructor.

Command

The command interface has an abstract execute method, which is implemented in the concrete command classes, and an abstract undo method. Since Command is simply an interface, it does not perform any of the commands itself, rather it acts as an outline upon which the concrete command classes are made.

FilterCommand, UndoCommand

Implement various functionality for filtering and working with the data visualization map. They do not perform any of the logical work, but rather pass the required parameters to the Map Editor, which is the receiver. They are associated with the Map Editor, except for the Undo Command, which is directly associated with the Invoker(MapListener) class.

Map Editor

The Map Editor has methods that get called depending on which concrete command sends in the parameters, and processes the logic accordingly. It is associated with the concrete command classes as well as the Invoker class.

CommandHistory

This class stores all of the commands called by the User through the Map Application, and it has an aggregation with the Command instance. Commands are stored in an ArrayList data structure called Previous[]. Whenever a concrete command is called, it is pushed into the list with addCommand(), and whenever the undo button is used, the command is popped out of the list with undoCommand().

4.4 Design Pattern 4: Iterator Pattern

Overview: This pattern will be used to implement ways to cycle through countries based on a certain category.
UML Diagram: Refer to Figure 2, below.

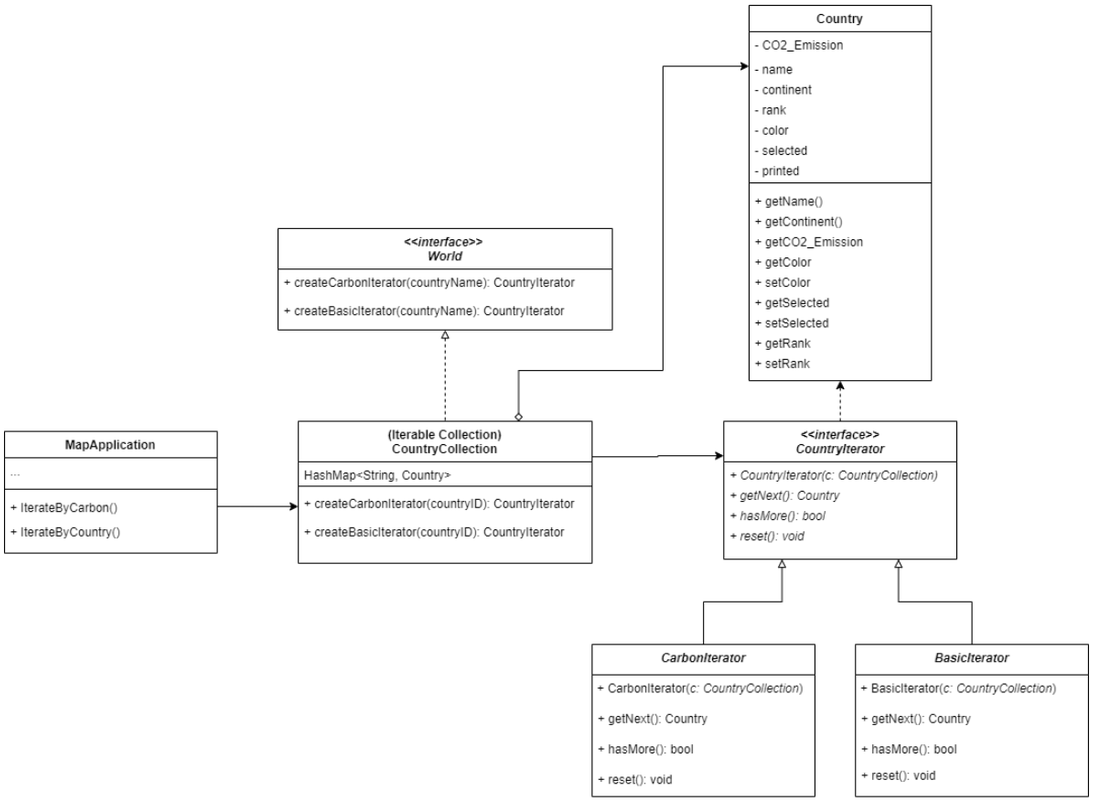

Figure 4: The Iterator Pattern

Implementation Details: The UML diagram outlines these main components:

1. **CountryCollection**: This is the iterable collection on which CountryIterators will act on.

2. **CountryIterator**: An interface working on the CountryCollection that sets a foundation for extended iterators like the *CarbonIterator* and *ClimateIterator*.

3. **MapApplication**: This application will make calls to iterate along a certain category.

The World is an interface designed to hold a collection of various countries. CountryCollection extends this interface and stores all countries in the world in a HashMap data structure. Country objects will be stored as dictionary values with the respective names used as keys. The goal of a CountryIterator is to ease traversal through the collection of country objects. BasicIterator and CarbonIterator are both implementations of the CountryIterator interface which each iterate differently. The former simply iterates country by country (one by one) whereas the latter iterates through the collection based on the countries' carbon emissions (highest to lowest). CarbonIterator will decide on which values fall under "getNext()" with the use of country methods like "getCO2_Emission()". The MapApplication will call methods that require these iterators to set a specific colour to a country, set its relative ranking of carbon emissions, or perform other tasks using the Country objects. Both iterators will also make heavy use of "hasMore()" to check whether a country is last in the collection or whether the HashMap can still be iterated through.

5 Expected Timeline

Timeline and Milestones

Week 1:

- Learn how to read climate data and retrieve the required information.
- Learn how to use World Map API and its functionality. Learn how to select individual countries and colour them.

Week 2:

- Integrate the carbon emission data onto the interactive map to colour countries according to the data.
- Implement a search bar to allow the user to search countries and display the information.

Week 3:

- Implement accessibility features including various view modes.
- Create functionality to save country data and also undo operations.
- Color countries based on a range of carbon emissions. Display a rank for each country also.

Week 4:

- Merge all code together.
- Revise the final product prior to submission.

The first sprint/week of the project timeline will be largely based on learning the various tools/libraries that will be used to build the bulk of the project. It will allow the group to gain a better understanding of which features are attainable and which ones may have to be scrapped. The second and third sprints/weeks will include writing most of the project's code; implementing various features and design patterns. The final week/sprint will be used to merge code from each group member together and tidy up the final product.