WESTERN UNIVERSITY
CS 2212B
Introduction to Software Engineering

Project Description
Winter 2020-2021

1. Introduction

For this project you will specify, design, and implement a system that allows for *a)* retrieving demographic and other data for one selected country from the World Bank's data repository; *b)* processing, if required, the data using different types of analyses, and *c)* rendering the retrieved data or the processed data using appropriately selected visualization means such as bar charts, line graphs, scattered plots, and pie charts. Example data which can be retrieved for a given country from the World Bank data repository is *health expenditures per 1,000 people*, or *percentage of population attended primary school*.

In order to implement your system, you will need to retrieve data related *a)* to Environment and *b)* to Health.

Once the data are collected, they may have to be processed depending on the type of analysis you want to perform. For example, you may want to compute the ratio of hospital beds per 1,000 people with current health expenditures per capita, and then display the results on the UI. You will consider that initially you have at your disposal different types of graphs to visualise your data, and you can select every time which types of graphs can be used for your data to be displayed. Please note that your design should allow for adding new "types" of graphs (or as we refer to them *viewers*). For example, you may want to add histogram capabilities into your system, that is adding a new type of *viewer*. Your design should allow for new types of viewers to be added or removed without modifying the existing viewers or the functionality of the existing system.

The system will have a main User Interface (UI) where all selections of parameters and the rendering of the data are performed. An example layout of a possible main UI and its different parts is depicted in Figure 1 below.

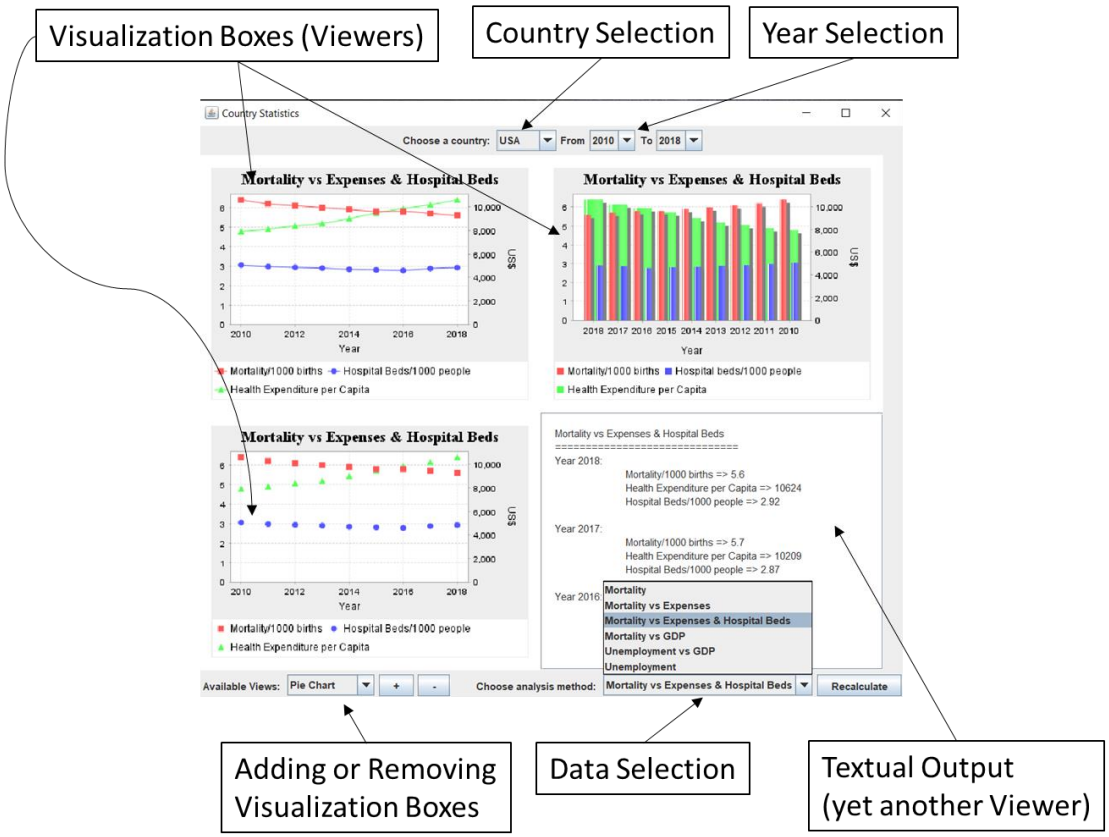

Figure 1. Sample Main UI for Word Bank Data Visualization System

2. Use Cases

The uses cases for your system are provided below:

UC1. The user logs into the system

When you start your system, the users are greeted with a login window or form, where they can input their username and password. If the combination of username and password is not correct or no such user exists in the system's database, a pop-up window or a notification in the form will notify them that there is an error with the provided credentials and the application will terminate. You can store the username-password pairs either in a database, or in a text file. If you use a simple text file to store the username/password combinations, the format can be plain text, JSON or XML. If the username-password combination is correct, then the main UI of the application is displayed.

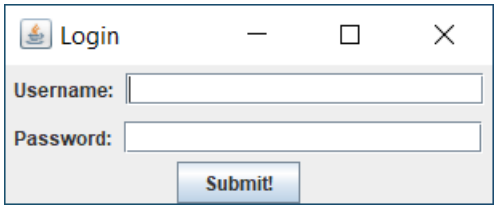

Figure 2. Sample Login panel

UC2. Selecting the analysis type to be performed

Once the user has selected the country for which they like to fetch, possibly process, and visualise data, they can then select the specific type of analysis they wish to perform. The choice is obtained from a drop-down menu of available types of analyses. You should implement 8 different types of analyses (2 per member). Your design and implementation must be modular. For example, the data required for each analysis type can be obtained in a modular way (i.e. by separate "reader" classes which read the specific data from the appropriate web site) and the analyses themselves have to be performed in a modular way, that is, each type of analysis has to be performed in its own class (i.e. as a strategy). Once an analysis type that is different from the previous selection is chosen, the list with the viewers for this analysis type is emptied. In case the user selects the same type, the list of viewers should remain intact. Figure 3 below depicts the scenario of a user selecting the type of analysis they want to perform. In Figure 3, six types of available analyses are shown: Mortality at birth (*one series data*), Mortality at birth vs Health Expenses per 1000 people (*two series data*), Mortality at birth vs Health Expenses per 1000 people vs. Hospital Beds per 1,000 people (*three series data*), Mortality at birth vs GDP (*two series data*), Percentage of Unemployment vs GDP (*two series data*), and Percentage of Unemployment (*one series data*).

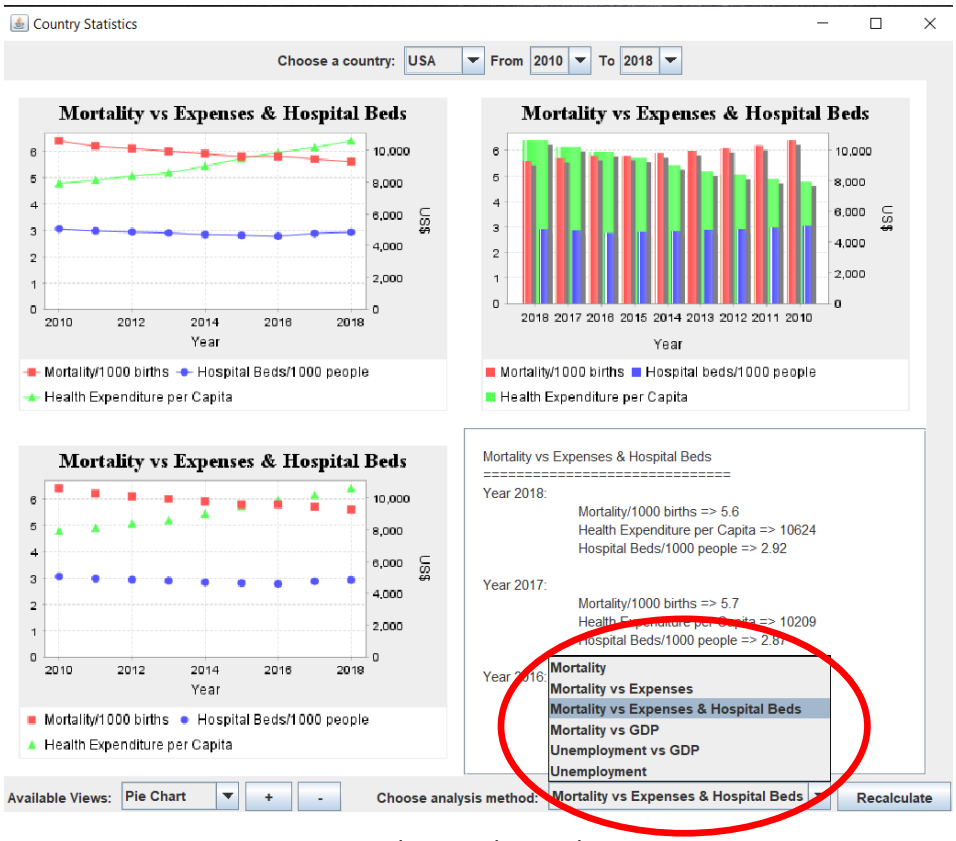

Figure 3. Selecting the Analysis Type.

UC3. Selecting a country to fetch and visualize data for

The users can use a list of countries to select the country for which they want to fetch (and optionally

process) data. For this use case, the users can select a country from a drop-down menu. The drop-down menu has a list of all the countries in the world. Your system must have the capability of restricting the countries for which the chosen type of analysis is permitted. Therefore, once a country is selected, the system will check whether this country is in the list of countries for which data can be fetched. You decide for which countries you would like to be able to fetch and visualise data for. If analysis and visualization is allowed for the selected country, then the data fetching can proceed, otherwise a message is displayed that data fetching and or processing is not available for the selected country. For example, you could have a text file that contains types of analysis and the countries for which the analysis should not be available. You can choose whatever format you like for the text file (e.g. json, config, plain text, etc.).

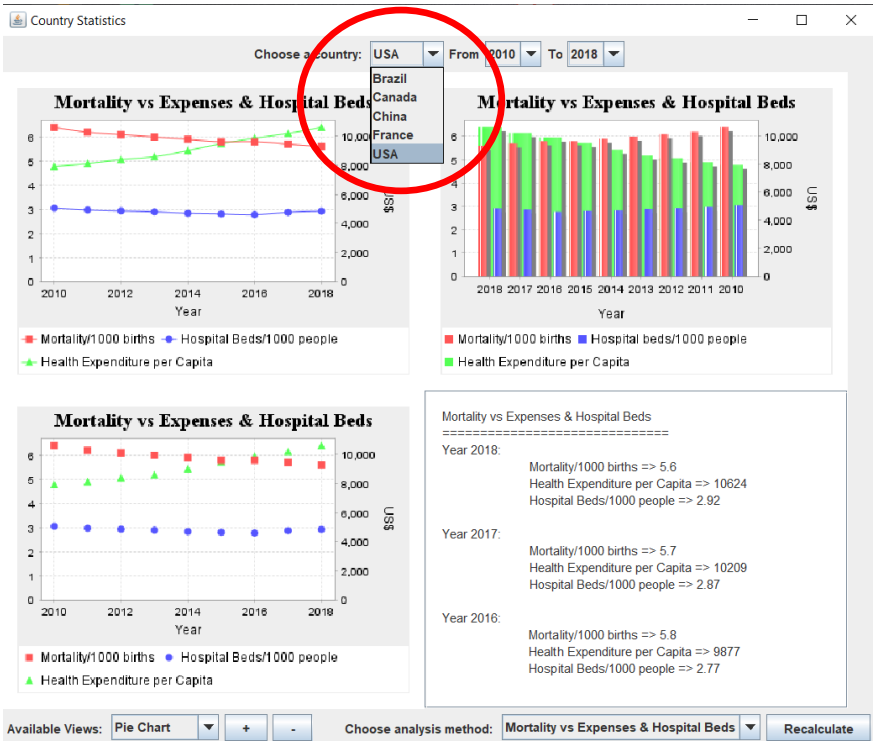

Figure 4. Selecting a country from the list of countries to fetch and visualise data for

UC4. Selecting the years for which the analysis type is to be performed

Once the user has selected the type of analysis, the user can then select the start and end year of the data to be fetched (see Figure 5). Each analysis type can be associated with a period for which data for the analysis is available. The system must check whether the start and end year are valid selections for the selected type of analysis. If the years selected are not valid for the chosen type of analysis, a message is displayed, and the user is prompted to select different start and end years.

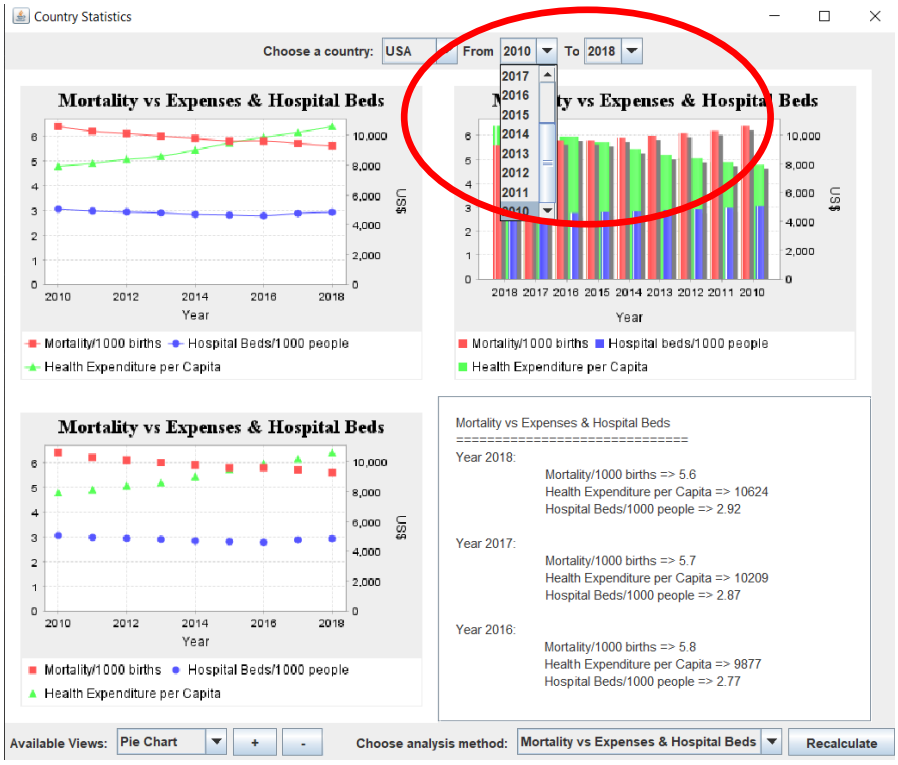

Figure 5. Selecting the Years to fetch data for.

UC5 and UC6. Adding and Removing visualization graphs to display the obtained/computed data

The user can add new or remove existing graphs (we refer to then as *viewers*). Those graphs are used to visualise the obtained or computed data (see Figure 6). For this use case, the user can select a *viewer* from a drop-down menu of *viewers* and then select the "add" (or "+") or the "remove" (or the "-") button to add or remove the viewer from the list of the viewers that are responsible for visualising the data. The system should perform checks when adding or removing viewers. More specifically, make sure viewers to be added are compatible with the analysis chosen and viewers to be removed are already in the list of viewers. In either case, if the criteria are not met, the user should receive a message.

As mentioned in UC3, every time a different analysis is selected, the list of viewers is emptied. That is, the list of viewers needs to be initially populated when the system is first run and subsequently repopulated every time a *new type of analysis* is chosen. The list remains intact if the user only changes the country and/or the start and end years.

Please note that it is not a correct design to create all possible graphs and then decide which ones to display. The correct design is to select and attach the different types of *viewers* for each analysis you run before you fetch, optionally analyse, the data. The selected viewers will then be the only ones to be notified once the data become available for visualization. As mentioned before, not all types of graphs are appropriate for all types of analyses. For example, pie-charts are good for data that sum up to 100%. Checking when adding viewers ensures that only the appropriate viewers are utilized.

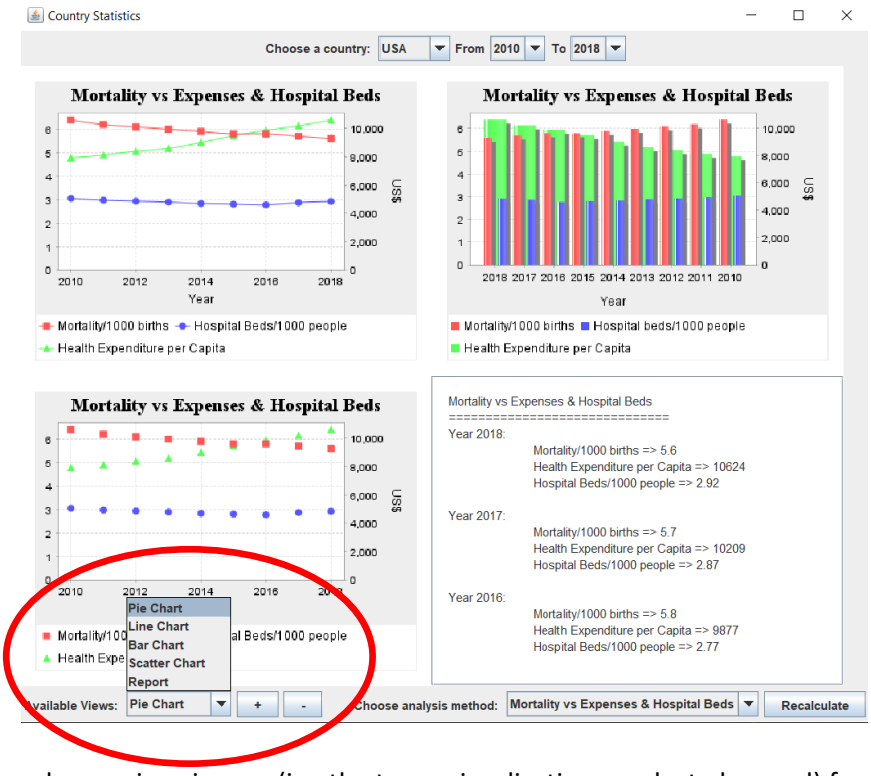

Figure 6. Adding and removing viewers (i.e. the types visualization graphs to be used) for a selected type of analysis

UC7. Performing the analysis

Once the user has selected the country, the type of analysis they want to perform, the years, and the viewers they want to use in order to display the results, then they can initiate the analysis by pressing the "Recalculate" button. The pressing of the "Recalculate" button will perform the following:

a) Get the country that has been chosen by the user.

b) Get the start and end years for the analysis chosen by the user.

c) Get the analysis type to be performed as chosen by the user.

d) Initiate the algorithm (we refer to is as *strategy*) corresponding to the chosen type of analysis selected (see the **Strategy design pattern**), and supply the information acquired in *steps a-c*.

Once the ***strategy*** is initiated, it should in turn perform the following actions:

a) initiate the identification and retrieval of the individual pieces of data that are required for the type of analysis selected. For example, if we want to calculate the ratio of *GDP per person* for a given country for a given year(s), we need to collect two pieces of data; first the GDP for the given year(s), and second the population for a given year(s). These two read operations constitute a "workflow" (see the Façade design pattern). Note you will need different types of readers which populate and return different types depending on the type of data you are retrieving. The obtained data will then become available to the calling ***strategy***. The way to extract the data and the format of the data are shown in detail in Appendix I.A.

b) Trigger the computation of the specific analysis using all the required individual pieces of data which have been collected in *step a*. If no processing is needed (i.e. the data can be displayed as they have been retrieved), then the strategy just populates a *result* object which has the same data as the ones read. If processing *is* required (e.g. you need to compute a ratio), then the strategy does the computation and populates the *result* object with the computed value. We refer to the result object that holds the data to be displayed as the ***model*** or the ***subject*** (see the **Observer design pattern**). For example, as mentioned above, if we want to calculate the ratio of *GDP per person* for a given country for a given year(s), once these two pieces of data are collected, a division (i.e. GDP/population) is all that's required to compute the result for this desired type of analysis. If all the data required for the computation of the selected type of analysis is not available (e.g. values are missing), a message must displayed to the user indicating that the analysis is not possible. If *some* of the data are available (i.e. data are available only for some of the years selected) then the computation proceeds only for these years. If the analysis proceeds (even for some of the selected years) then the *result* object is populated so that the results can be displayed (see UC6).

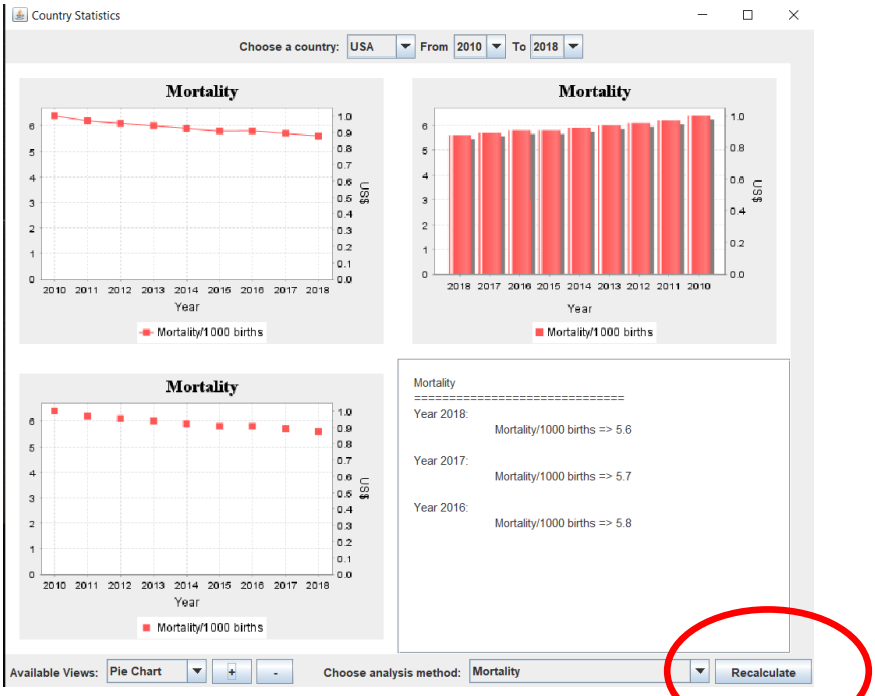

Figure 7. Triggering the analysis; that is a) fetching of data, b) performing any computations (if needed), and c) visualising the results by notifying the viewers selected

UC8. Displaying the results

Once the data have been fetched, and any required computation (if needed) is completed and results have been calculated, then the system renders the results on the selected viewers. The viewers should identify, based on the analysis, the number of series of data that need to be visualized. For some types of analysis only one series of data is produced (see Figure 8a), while other types of analysis yield two series (see Figure 8.b) or even three series of data (see Figure 8.c).

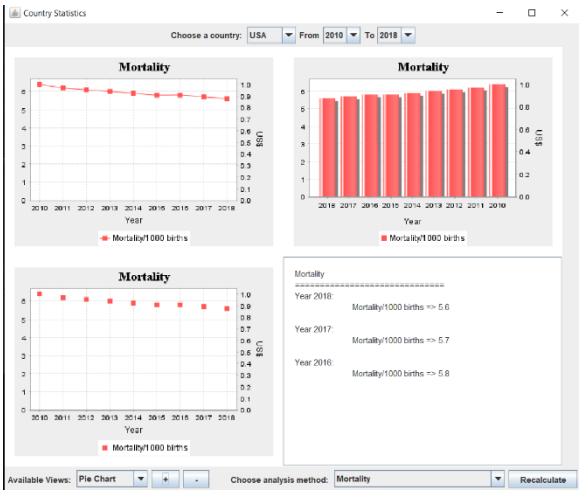

Figure 8.a.

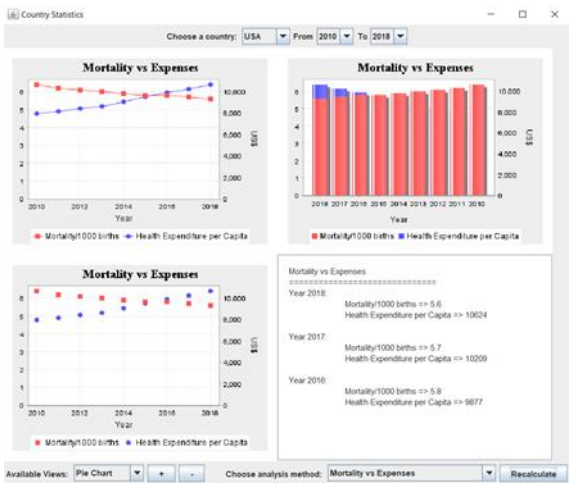

Figure 8.b.

One-series data visualization (Fig. 8a) and two-series data visualization (fig. 8.b)

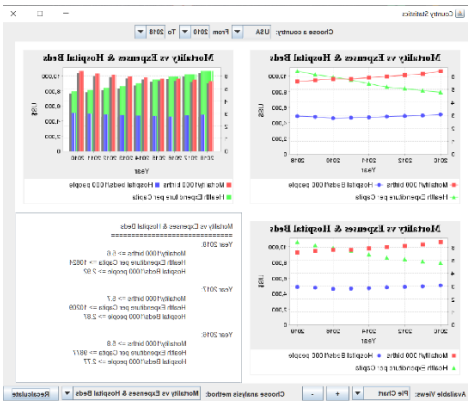

Figure 8.c.

Figure 8d

Three-series data visualization (Fig. 8c) and use of pie charts (Fig. 8.d)

For information on rendering the results, please see Section 5 “Rendering the Results” below and also Appendix I.C.

3. Suggested Types of Analyses

You are free to pick your own data (i.e. indicators) and types of analyses. Here we suggest a few based on the following categories (see the corresponding indicators on the right).

ENVIRONMENT

Total Population	SP.POP.TOTL
CO2 emissions (metric tons per capita)	EN.ATM.CO2E.PC
PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)	EN.ATM.PM25.MC.M3
Forest area (% of land area)	AG.LND.FRST.ZS

Energy use (kg of oil equivalent per capita)	EG.USE.PCAP.KG.OE
GDP per capita (current US$)	NY.GDP.PCAP.CD

HEALTH

Hospital beds (per 1,000 people)	SH.MED.BEDS.ZS
Government expenditure on education, total (% of GDP)	SE.XPD.TOTL.GD.ZS
Maternal mortality ratio (modeled estimate, per 100,000 live births)	SH.STA.MMRT
Current health expenditure per capita (current US$)	SH.XPD.CHEX.PC.CD
Current health expenditure (% of GDP)	SH.XPD.CHEX.GD.ZS
Mortality rate, infant (per 1,000 live births)	SP.DYN.IMRT.IN

These are just suggested indicators. Use the list of indicators in:
http://api.worldbank.org/v2/source/75/indicators and in
https://api.worldbank.org/v2/sources/39/indicators
to get additional types of data if you want to provide other types analyses. For your system you should have 8 different types of analyses (2 per group member) and be able to discuss/demonstrate the way to add in your system new analyses and types of visualisation graphs (i.e. how modular and extensible your design is).

The **suggested** types of analyses for your system are:

1. *CO2 emissions (metric tons per capita)* vs *Energy use (kg of oil equivalent per capita)* vs *PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)* (3-series graph)
2. *PM2.5 air pollution, mean annual exposure (micrograms per cubic meter)* vs *Forest area (% of land area)* (2-series graphs)
3. Ratio of *CO2 emissions (metric tons per capita)* and GDP per capita (current US$)
4. Average *Forest area (% of land area)* for the selected years (1-series graphs – you can use a pie-chart here)
5. Average of *Government expenditure on education, total (% of GDP)* for the selected years (1-series graphs – you can use a pie-chart here)
6. Ratio of *Hospital beds (per 1,000 people)* and *Current health expenditure (**per 1,000 people**)* (2-series graphs). Note the computations needed here.
7. *Current health expenditure **per capita** (current US$)* vs *Mortality rate, infant (per 1,000 live births)* (2-series graphs)
8. Ratio of *Government expenditure on education, total (% of GDP)* vs *Current health expenditure (% of GDP)*.

4. Data Acquisition

Your analyses and visualizations will require data to either be retrieved from one or more data pools from the World Bank's data base remote issuing http GET requests from your Java application. The different topics for which the World Bank provides data for can be found in https://datahelpdesk.worldbank.org/knowledgebase/articles/889392-about-the-indicators-api-documentation. The retrieved data can be in xml or json formats. The format of the World Bank's API (i.e. how to call the API and get a response) and the indicator IDs (i.e. what data to retrieve) are provide in Appendix I.A below. An example on ow to programmatically fetch the data is shown in Appendix I.B.

For example, an http request to:

http://api.worldbank.org/v2/country/can/indicator/SP.POP.TOTL?date=2000:2001&format=json

will provide in json format (see the "format" parameter in the URL above) the total population (see the indicator "SP.POP.TOTL" in the UR above) of Canada (see the "can" section of the URL above) in years 2000 and 2001 (see the "date" parameter in the UR above).

Sample Java code on how to issue programmatically from within your java program a call to a URL as the one shown above is provided in Appendix I.B below. This sample code is provided to you as a Maven Eclipse project. You can find it on the Supporting Resources section in OWL. Once you download the archive and unzip it, this sample code can be imported in Eclipse as a Maven Project (File→Import→Maven→Existing Maven Projects). Make sure after you import it, to right click on the project and select Maven → Update Project to import all necessary libraries.

Make sure that the design of your system is modular. An example of modularity is that you have different "reader" components for different data sources.

5. Rendering the Results

Once the necessary data have been fetched for the selected type of analysis (see the Façade design pattern), then the data need either to be directly displayed or processed. Notes on the UI are provided in Appendix I.C.

1. If the analysis type selected (i.e. the algorithm or as we refer to is the *strategy*) (see the Strategy design pattern) does not require any data processing, then the algorithm (i.e. the strategy) will not do any processing, and its job is to just pass the data to the model (i.e. the object that holds the data to be displayed) which in turn will notify the viewers (graphs) (see the Observer design pattern). An example of analysis type that requires processing is # 2 above (*PM2.5 air pollution, mean annual exposure (micrograms per cubic meter) vs Forest area (% of land area)*) where the data can directly be displayed after the model notifies the viewers.
2. If the analysis type selected (i.e. the algorithm or as we refer to it as *the strategy*) (see the Strategy design pattern) require some data processing, then the algorithm (i.e. the strategy) will do the necessary processing, and it will pass the result of this processing to the model (i.e. the object that holds the data to be displayed) which in turn will notify the viewers (graphs) (see the Observer design pattern). An example of analysis type that requires processing is #4 above (Average Forest area (% of land area) for the selected years) where you need to collect the forest area per year for all selected years and then compute the average, and #6 where a ratio must be computed.

6. Design and Implementation Notes

The design and implementation of your system have to exhibit the following properties:

1. The architecture must be modular and conform to one or more of the architectural styles you learn in class.
2. The architecture must exhibit the properties of high cohesion and low coupling. Here high cohesion means that packages are designed to contain classes that perform a specific functionality of the system (e.g. a package that contains all classes that perform the different analysis strategies).
3. Your code must correctly implement several design patterns. The design patterns which are applicable to this system are: *Proxy*, *Façade, Chain of Responsibility*, *Strategy*, *Observer*, *Singleton* and, *Factory Method*. For example, the Proxy design pattern can be used for login services, the Façade design pattern for coordinating activities between the UI and the backend services (e.g. retrieving data, rendering images, checking for proper country names etc.).
4. You may create your project as a *Maven* project or convert it afterwards (see https://crunchify.com/how-to-convert-existing-java-project-to-maven-in-eclipse/). This will allow you to automatically download and install in your project any external libraries. An example pom.xml file which is needed for such a Maven project can be found in the Supporting Resources section in OWL.
5. For your collaboration you are expected to use bitBucket as your source code repository where you will be regularly committing your code changes and any supporting documents. We will provide bitbucket accounts for each team. For communication with your fellow team members you are expected to use Microsoft Teams. You should provide a trace record of the code commits each one of you performed throughout the term.

7. Tools

You can use the following tools to proceed with the deliverables of your project:

Drawing UML Diagrams:

a. UMLet (https://www.umlet.com/) This tool has a very wide spectrum of UML symbols
b. MS-Visio (download the UML stencils) https://support.microsoft.com/en-us/office/uml- diagrams-in-visio-ca4e3ae9-d413-4c94-8a7a-38dac30cbed6?ui=en-us&rs=en-us&ad=us

Automatically compiling class diagrams from your source code (useful for acquiring a visual overview of your system in terms of your source code classes):

1. ObjectAid (https://www.objectaid.com/home). Outstanding tool. Read the short instructions how to use within Eclipse in the link provided above

8. Notes

- **An SRS and an SDD example for a sample project can be found on the Supporting Resources section on OWL. You can use another format of your choice if you want.**

Appendix I.A - The structure of the World Bank's API

The World Bank's data set provides a wealth of data form different repositories. The World Bank provides for most data sets a unified API using REST calls. The format of the REST-API is provided can be found at https://datahelpdesk.worldbank.org/knowledgebase/articles/898581. An example call is given below for extracting the information regarding the total population of Canada for the years 200 to 2001. The request uses the indicator "can" to denote that data for **Canada** are required, the Indicator "SP.POP.TOTL" to denote that the total population is required (for Canada), the *date* parameter to denote the range of years for which data are required (here 2000 and 2001), and the *format* parameter to denote the output format (here json). The call in this respect looks like: http://api.worldbank.org/v2/country/**can**/indicator/**SP.POP.TOTL**?date=2000:2001&format=json

For your project you can use the indicators (data) and the types of analyses provided in Section 3.

For your information, the list of all data categories which are available in World Banks' repository can be found in

https://datacatalog.worldbank.org/search?sort_by=field_wbddh_modified_date&sort_order=DESC&f%5B0%5D=field_license_wbddh%3A1335&q=search&page=0%2C0

You can select a specific category of data and then find the available indicators. For example, in:

http://api.worldbank.org/v2/source/75/indicators you can see the first indicator AG.LND.AGRI.ZS which refers to Agricultural land (% of land area).

In this respect, the call to the following URL will return the agricultural land in Canada as a percentage of the total land in years 2000 and 2001. Try the call to see the result. http://api.worldbank.org/v2/country/can/indicator/**AG.LND.AGRI.ZS**?date=2000:2001&format=json

Note the similarity of this call with the call above related to total population (i.e. http://api.worldbank.org/v2/country/**can**/indicator/**SP.POP.TOTL**?date=2000:2001&format=json). Here only the indicator is changing as we request data for the same country and years.

In this respect, the result of calling

http://api.worldbank.org/v2/country/can/indicator/**AG.LND.AGRI.ZS**?date=2000:2001&format=json

is:

```
[
 {"page":1,"pages":1,"per_page":50,"total":2,"sourceid":"2","lastupdated":"2020-12-16"},
 [
  {"indicator":{"id":"AG.LND.AGRI.ZS","value":"Agricultural land (% of land area)"},
   "country":{"id":"CA","value":"Canada"},
   "countryiso3code":"CAN",
   "date":"2001",
   "value":7.42309625216226,
   "unit":"",
   "obs_status":"",
   "decimal":1},
  {
   "indicator":{"id":"AG.LND.AGRI.ZS","value":"Agricultural land (% of land area)"},
   "country":{"id":"CA","value":"Canada"},
   "countryiso3code":"CAN",
   "date":"2000",
   "value":7.4353027598804,
   "unit":"",
   "obs_status":"",
   "decimal":1}
 ]
]
```


The result above shows that in 2001, 7. 4230962521622% of the land in Canada was agricultural land. Try that for 2016! The value is 6.89183824507808%. There are no data available after 2016. Try for 2019 and you will see that the *value* filed is “null”.

For another data category (e.g. *Health Nutrition And Population Statistics By Wealth Quintile* in https://datacatalog.worldbank.org/dataset/health-nutrition-and-population-statistics-wealth-quintile) the API file is located under the (Data&Resources tab ➔ API Documentation). This API file can be found on https://api.worldbank.org/v2/sources/39/indicators

The list of country codes according to the ISO standard can be found and downloaded on https://unstats.un.org/unsd/tradekb/Knowledgebase/Comtrade-Country-Code-and-Name

Appendix I.B – Using the API from your Java program

The request for retrieving in JSON format the total population of Canada for the years 200 to 2001 is

http://api.worldbank.org/v2/country/can/indicator/SP.POP.TOTL?date=2000:2001&format=json

(the Indicator is "SP.POP.TOTL" for total population and the country code "can" for Canada. The years selected are 2000 and 2001, and the output format be json.

If you try the above URL you will get as a result the following Json which is an array of two objects. The second object is an array of objects, and each object has a field "value" with the desired value we want. Note a Json object is enclosed in **{}** and an array of objects in **[]**.


```
[
 {"page":1,"pages":1,"per_page":50,"total":2,"sourceid":"2","lastupdated":"2020-12-16"},
 [
  {"indicator":{"id":"SP.POP.TOTL","value":"Population, total"},
  "country":{"id":"CA","value":"Canada"},
  "countryiso3code":"CAN",
  "date":"2001",
  "value":31020902,
  "unit":"",
  "obs_status":"",
  "decimal":0
  },
  {"indicator":{"id":"SP.POP.TOTL","value":"Population, total"},
  "country":{"id":"CA","value":"Canada"},
  "countryiso3code":"CAN",
  "date":"2000",
  "value":30685730,
  "unit":"",
  "obs_status":"",
  "decimal":0
  }
 ]
]
```


The result shows that In 2001 Canada had a population of **31,020,902** and in 2000, **30,685,730** people.

You can find below the Java code that programmatically issues the request above, gets the result as a single line, gets the second object (the array of size two i.e. for years 2000 and 2001) and then for each object in the array gets the value of the "value" field. It is best to try is as a Maven project (File->New->Maven Project). The pom.xml file to fetch all the required libraries used in the code below looks like:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>httpTest</groupId>
 <artifactId>httpTest</artifactId>
 <version>0.0.1-SNAPSHOT</version>

 <repositories>
   <repository>
       <id>maven-repository.java.net</id>
       <name>Java.net Repository for Maven</name>
       <url>https://maven.java.net/content/repositories/releases/</url>
       <layout>default</layout>
    </repository>
  </repositories>
<dependencies>
   <dependency>
      <groupId>com.google.code.gson</groupId>
       <artifactId>gson</artifactId>
       <version>2.2.2</version>
     </dependency>
</dependencies>
</project>
```

And here is the sample Java code:

```
package httpTest;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;
import com.google.gson.JsonArray;
import com.google.gson.JsonParser;
```

```
public class GetData {
	public static void main(String[] args) {
		String urlString =
				String.format("http://api.worldbank.org/v2/country/%s/indicator/SP.
				POP.TOTL?date=2000:2001&format=json", "can");
		System.out.println(urlString);
		int populationForYear = 0;
		int cumulativePopulation = 0;
		try {
			URL url = new URL(urlString);
			HttpURLConnection conn = (HttpURLConnection) url.openConnection();
			conn.setRequestMethod("GET");
			conn.connect();
			int responsecode = conn.getResponseCode();
			// IF THE RESPONSE IS 200 OK GET THE LINE WITH THE RESULTS
			if (responsecode == 200) {
				String inline = "";
				Scanner sc = new Scanner(url.openStream());
				while (sc.hasNext()) {
					inline += sc.nextLine();
				}
				sc.close();
				// PROCESS THE JSON AS ONE LINE
				JsonArray jsonArray = new
						JsonParser().parse(inline).getAsJsonArray();
				int size = jsonArray.size();
				int sizeOfResults = jsonArray.get(1).getAsJsonArray().size();
				int year = 0;
				for (int i = 0; i < sizeOfResults; i++) {
					// GET FOR EACH ENTRY THE YEAR FROM THE “date” FIELD
					year =
					jsonArray.get(1).getAsJsonArray().get(i).getAsJsonObject().g
					et("date").getAsInt();
					// CHECK IF THERE IS A VALUE FOR THE POPULATION FOR A
					//  GIVEN YEAR
					if(jsonArray.get(1).getAsJsonArray().get(i).getAsJsonObject(
							).get("value").isJsonNull())
						populationForYear = 0;
					else
					  // GET THE POPULATION FOR THE GIVEN YEAR FROM THE
					  // “value” FIELD
						populationForYear =
						jsonArray.get(1).getAsJsonArray().get(i).getAsJsonObj
						ect().get("value").getAsInt();

					System.out.println("Population for : " + year + " is " +
							populationForYear);
					cumulativePopulation =
							cumulativePopulation + populationForYear;
				}
				System.out.println("The average population over the selected years
						is " + cumulativePopulation / sizeOfResults);
			}

		} catch (IOException e) {
			// TODO Auto-generated catch block e.printStackTrace();
		}

		return;
	}

}
```

This code is available on the Supporting Resources section at OWL. You can import the sample code above as a Maven Eclipse project in your Eclipse IDE. You can find the code in the Supporting Resources section at OWL. Here is a quick summary (Fig. 9 – 14) how to load the project as a Maven project in Eclipse :

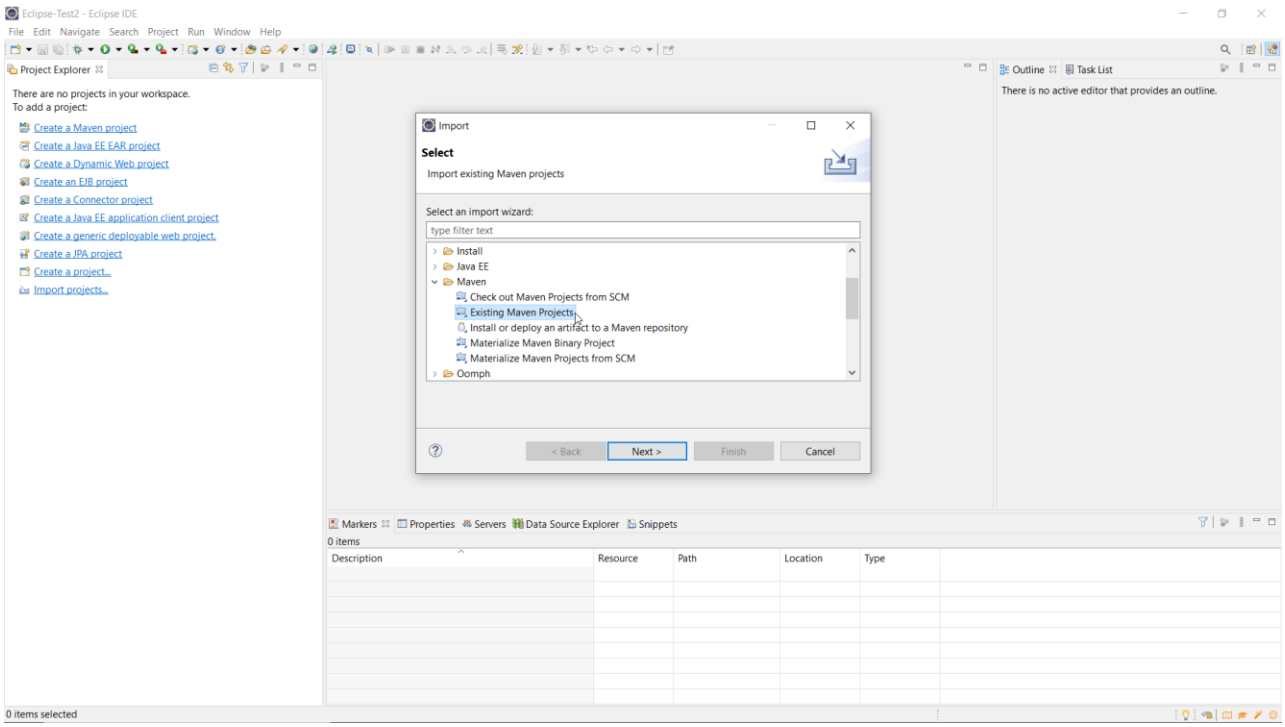

Figure 9.

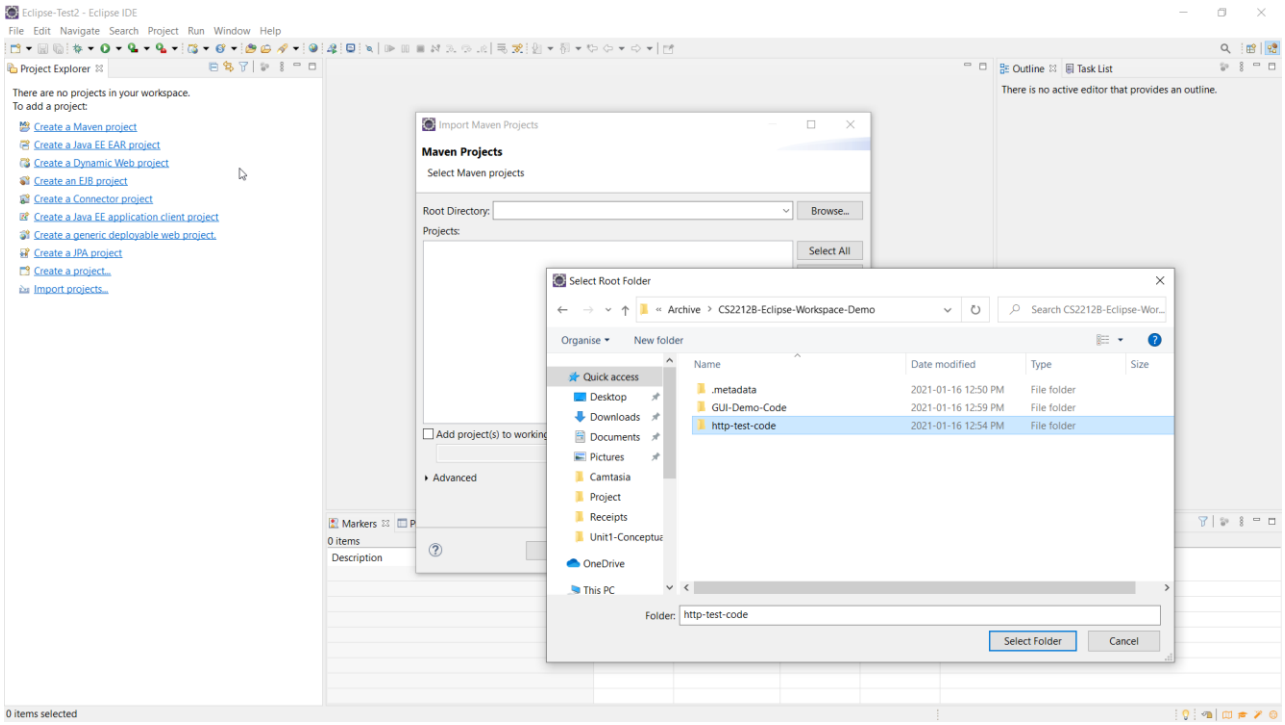

Figure 10.

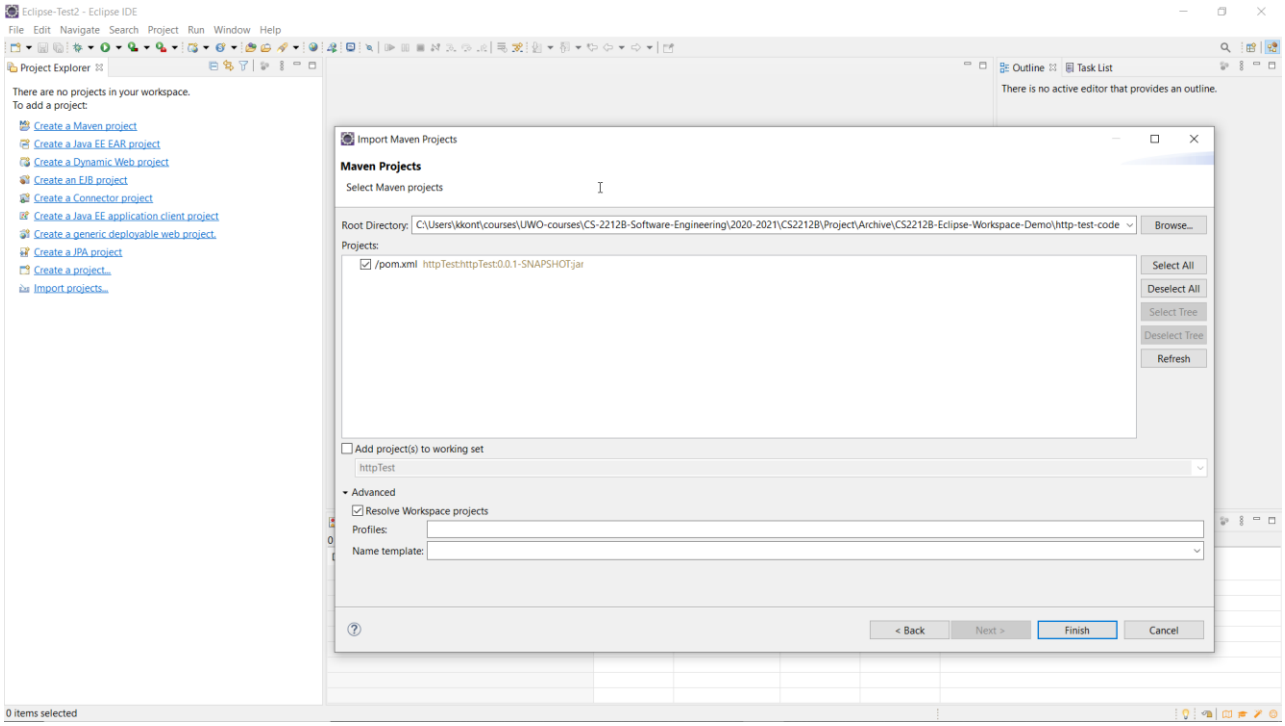

Figure 11.

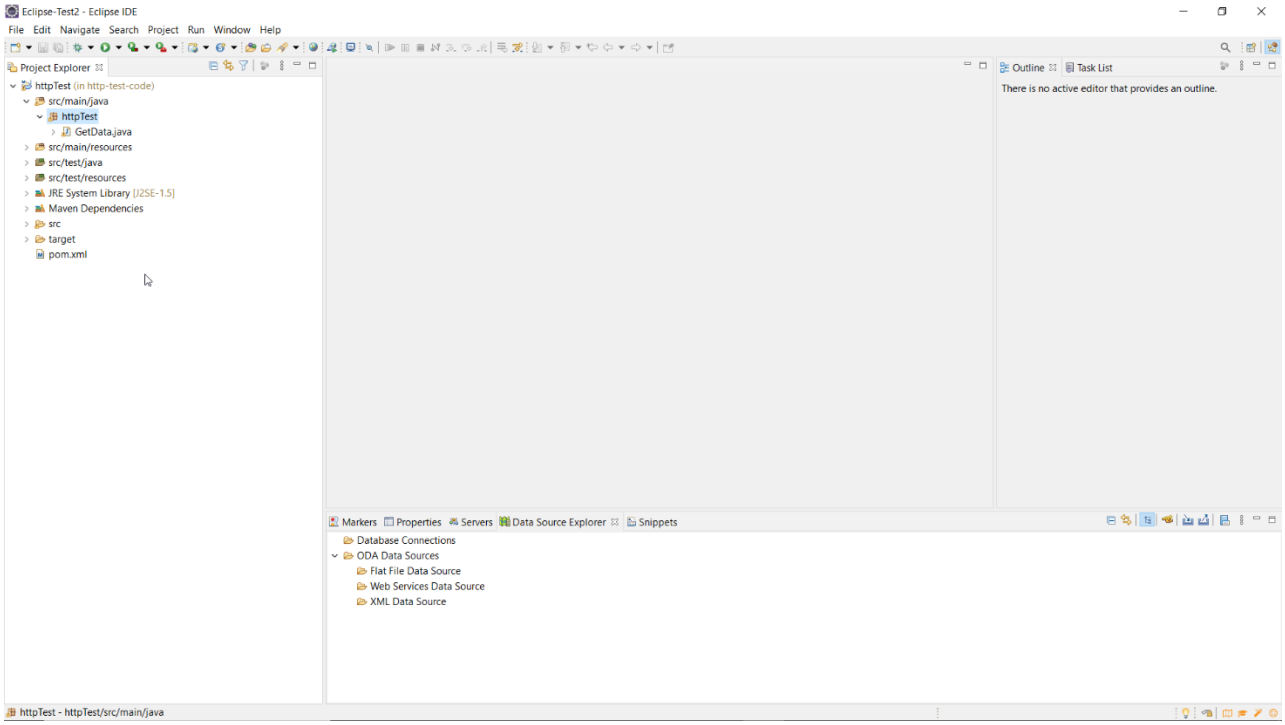

Figure 12.

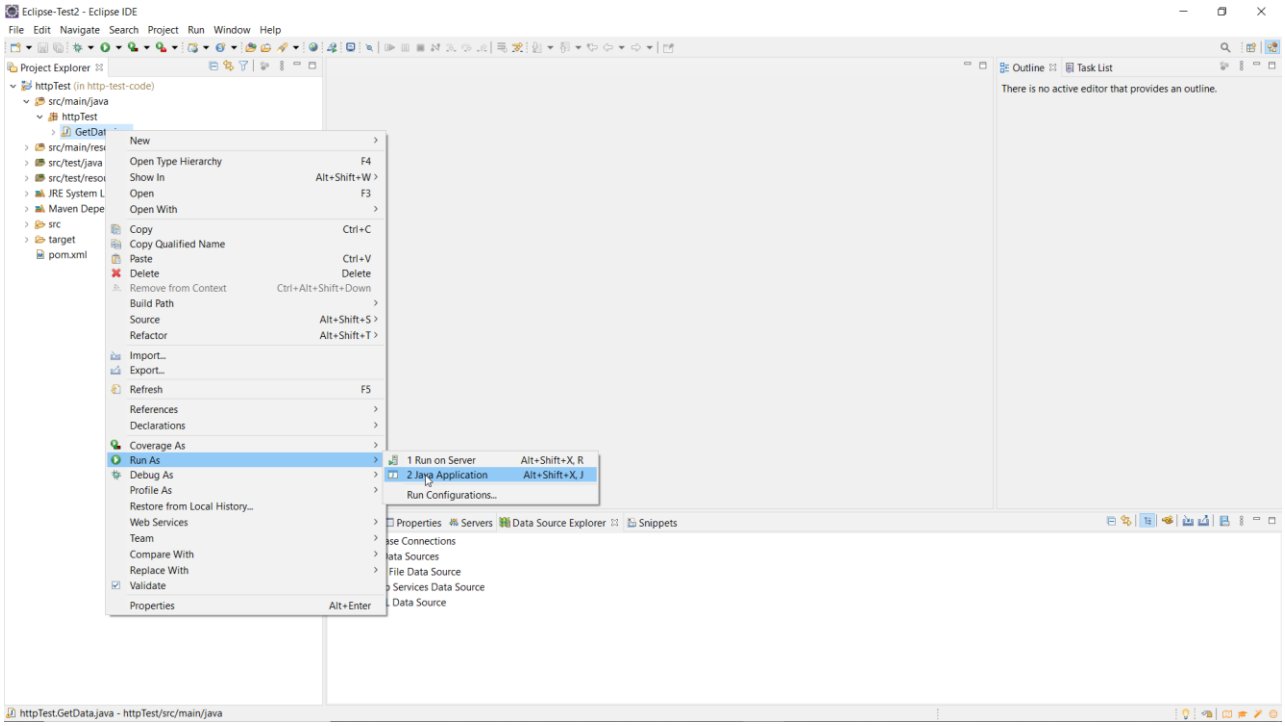

Figure 13.

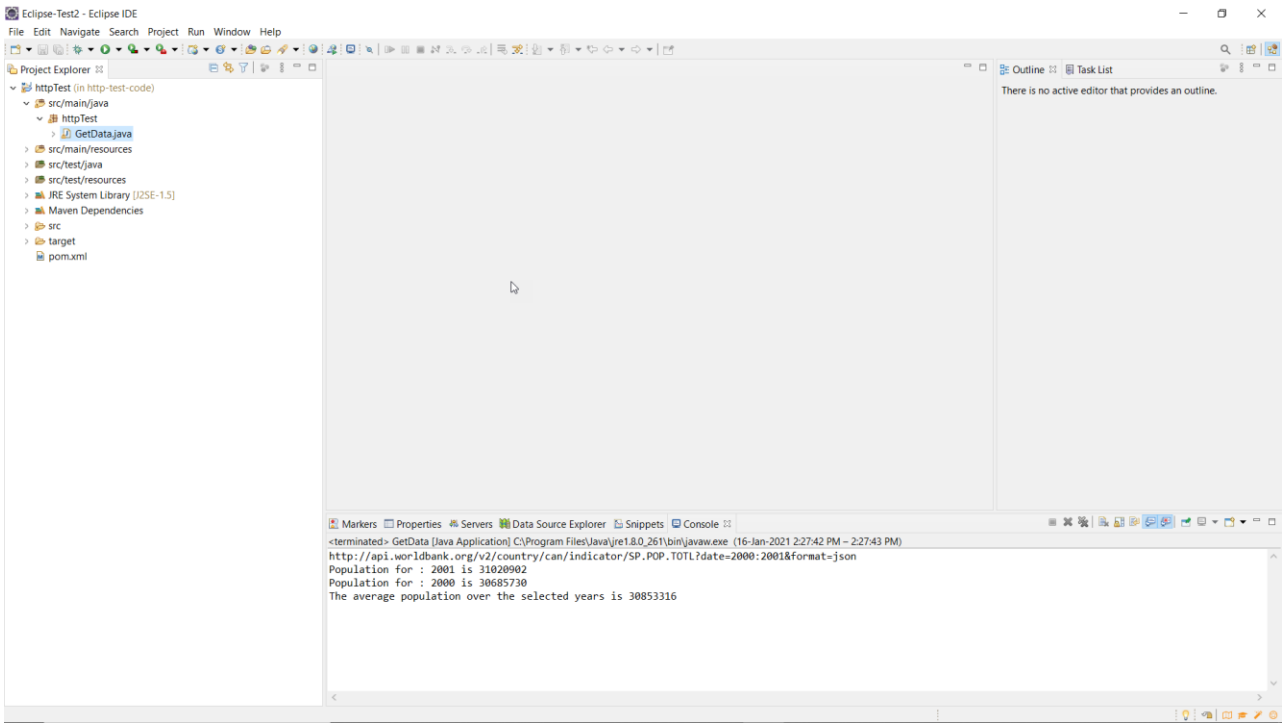

Figure 14.

Appendix I.C – Rendering the results

The results can be rendered by any appropriate Java library of your choice. The library we suggest is **jfreechart**. We have also uploaded on OWL, a sample Java project which renders the UI presented above using some "static" hardcoded data for demonstration purposes only, so that you can see the use of the **jfreechart** toolkit.

Please note: The provided code is only to demonstrate how to use the **jfreechart** toolkit. **THIS SAMPLE CODE PROVIDED DOES NOT REFLECT THE DESIGN WHICH IS REQUIRED AND EXPECTED FROM YOUR IMPLEMENTATION.** For example, the different graphs have to implemented as different *viewers*, each time appropriately attached to the *model* (i.e. the data or "the subject") as a collection of *observers*. Every time the *model* (i.e. the data or "the subject") gets new values, then the attached to it *viewers* (i.e. observers) have to be *notified*.

You can import the sample code provided as a Maven Project. The steps (Fig. 15 – 19) are provided below.

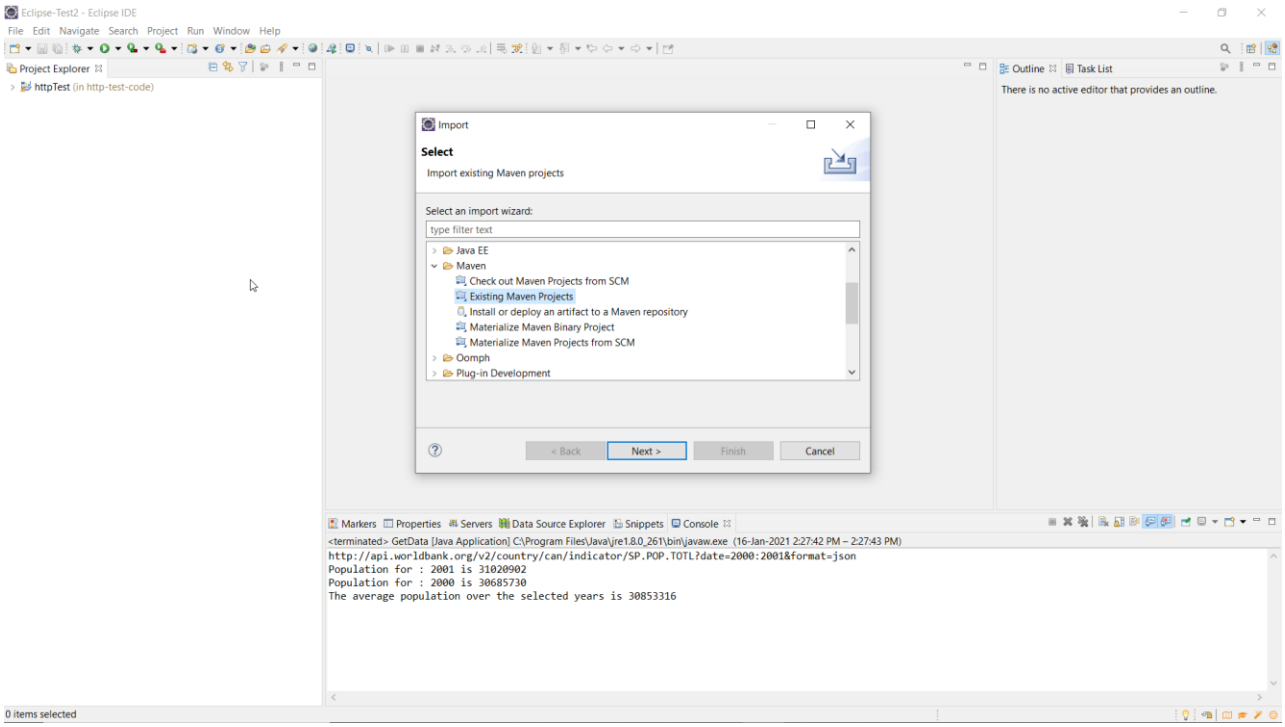

Figure 15.

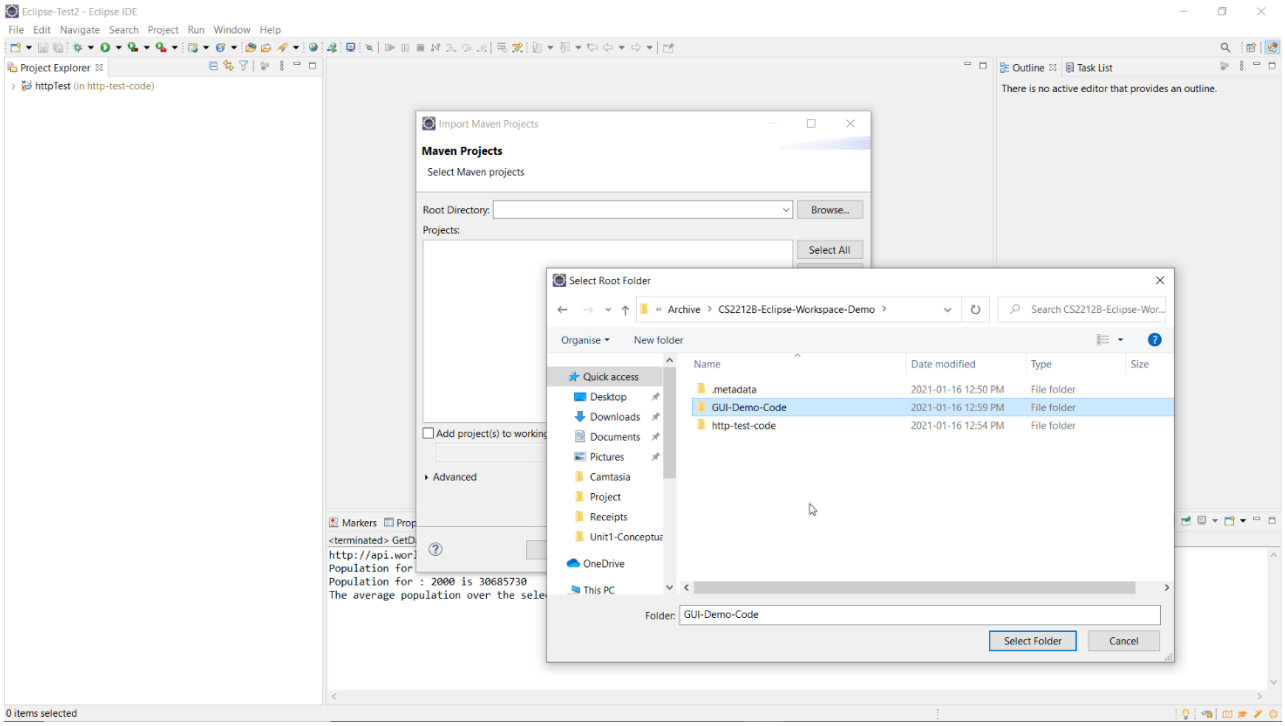

Figure 16.

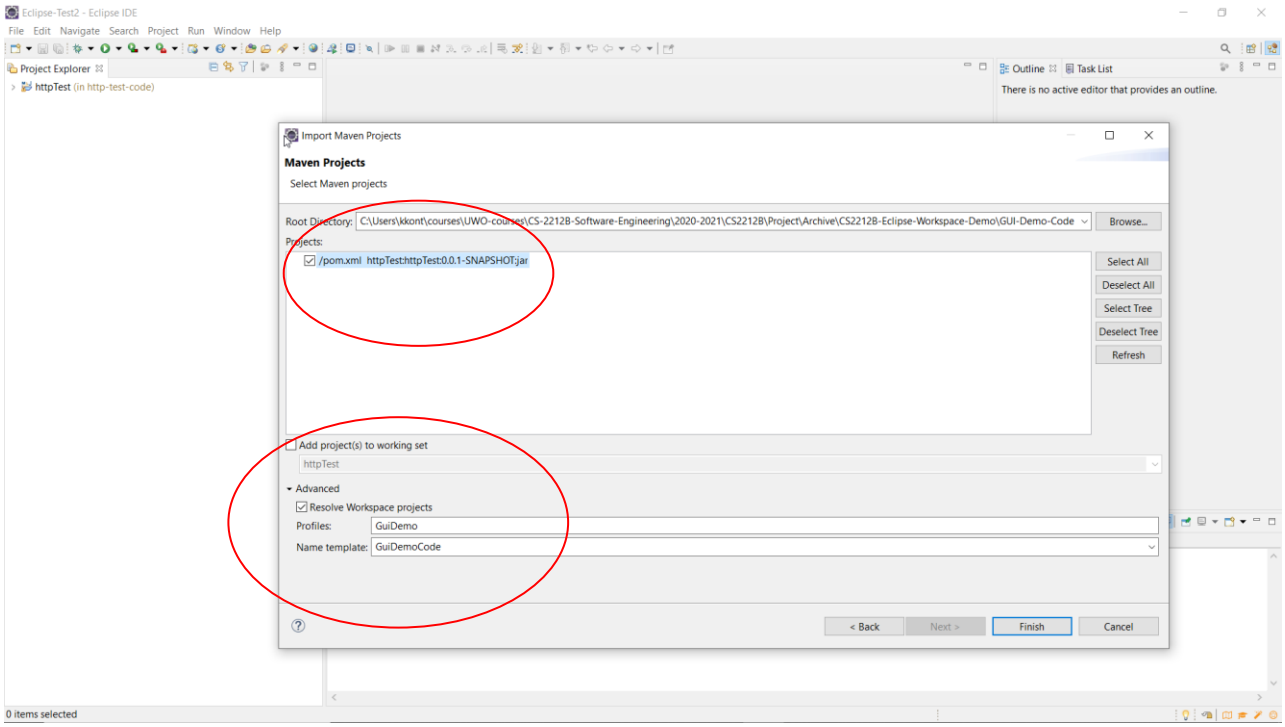

Figure 17.

In this step make sure you expand the Advanced Tab and fill in the values GuiDemo, GuiDemoCode.

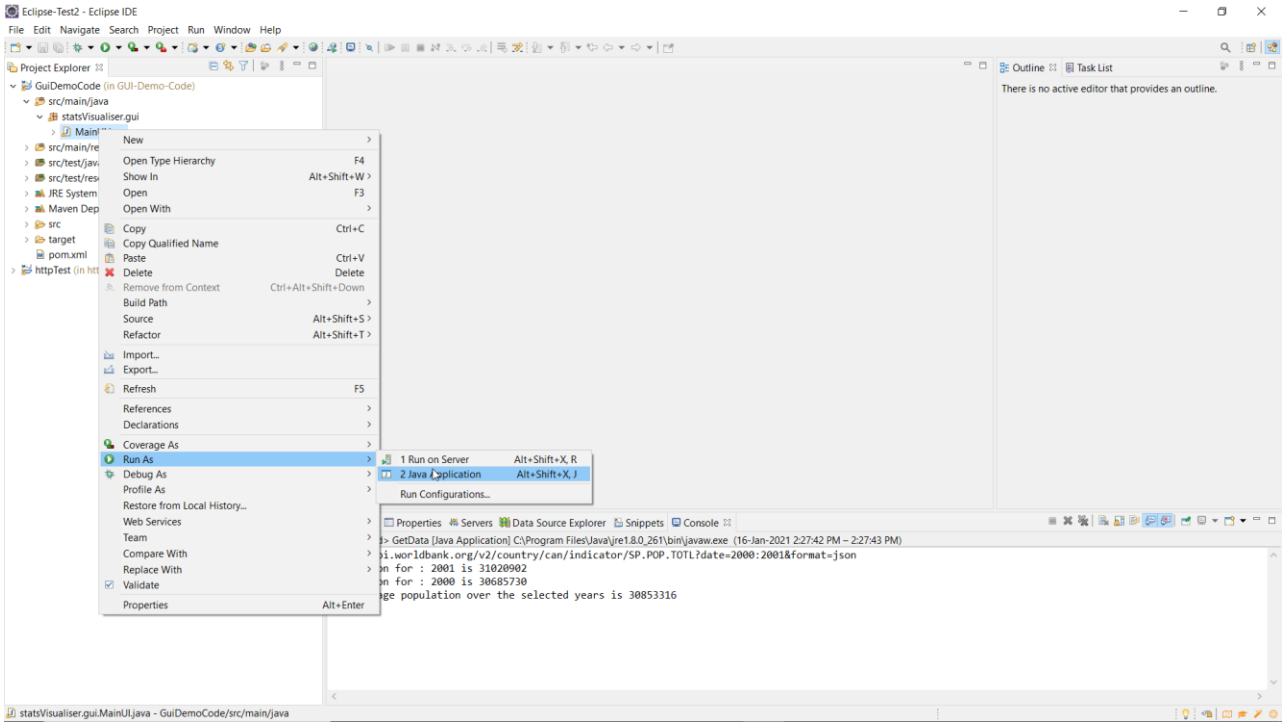

Figure 18.

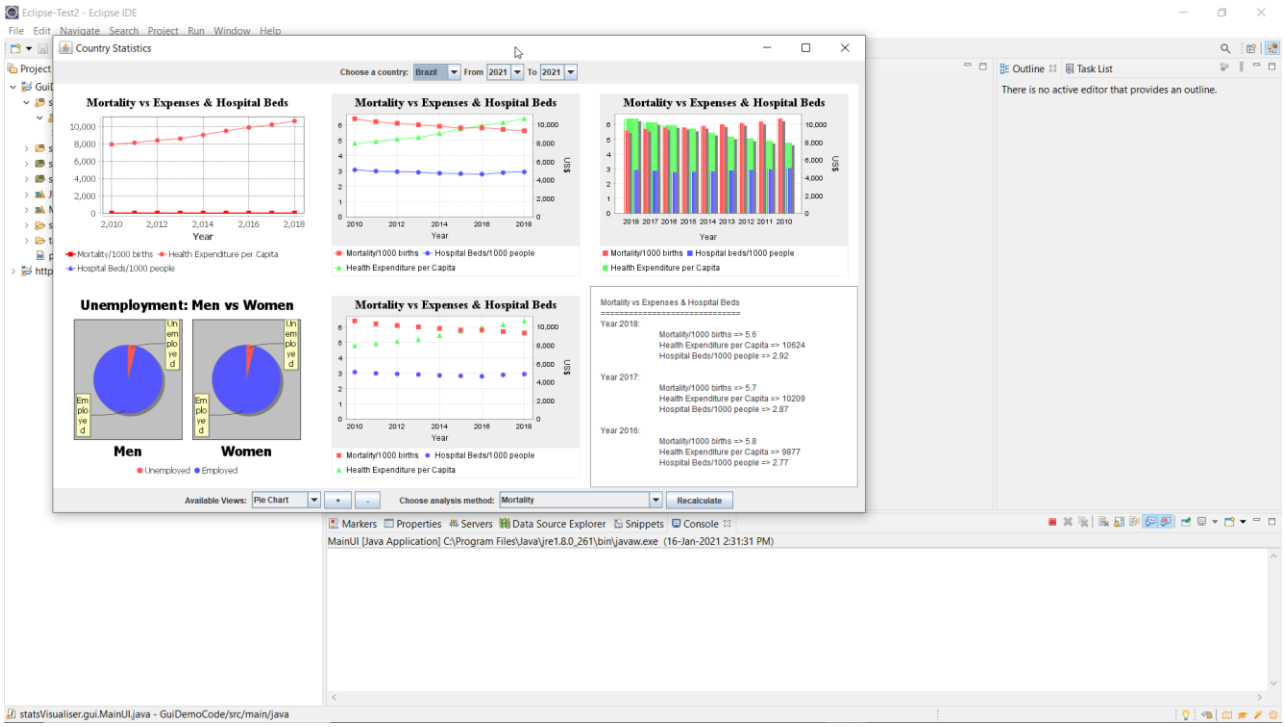

Figure 19.