Assignment

CE152

Spring 2020

WARNING AND ADVICE ABOUT POSSIBLE ACADEMIC OFFENCES

Your solutions should be your own unaided work. You can make use of any of the programs from the CE152 lecture notes, support classes and the lab solutions. You may use any features from the Java JDK API including those not covered in CE152.

You must NOT use any third-party classes (e.g. classes that are not provided as part of the Java JDK download). If you use any other sources, you must clearly indicate this as comments in the program, and the extent of the reference must be clearly indicated. For more information, please see the University pages on plagiarism and the Academic Offences Procedures.

DO NOT COPY PROGRAM CODE FOR THIS ASSIGNMENT FROM ANOTHER STUDENT OR FROM THE INTERNET OR FROM ANY OTHER SOURCES. DO NOT LET OTHER STUDENTS COPY YOUR WORK.

Deadline: see Faser.

Submission

The assignment should be submitted via Faser. Your submission should comprise a single zip file containing the source code (i.e. the .java files) for all the classes that you have written as solutions to the assignment tasks. No other files should be included in the zip file.

The name of your zip file should include both your name and your registration number.

You will receive a mark of zero if you fail to submit your solutions by the deadline.

Demonstration

You will be required to demonstrate your solutions to the assignment tasks in ***Week 31 or 32*** (i.e. shortly after the submission deadline). See provisional dates, times and locations below:

Week	Day	Time	Lab
31	Monday	9am - 12am and 1pm - 4pm	Stem 4.2A+B
31	Tuesday	9am - 12am and 1pm - 4pm	Stem 4.2A+B
32	Monday	9am - 12am and 1pm - 4pm	CES Lab 4
32	Tuesday	9am - 12am and 1pm - 4pm	Stem 4.2A+B
32	Wednesday	9am - 12am and 1pm - 4pm	Stem 4.2A+B

Please ensure that you know which time slot you have been assigned (the emails will be sent out before the end of the Spring Term). If you notify the module supervisor at least on the day before your demonstration that you cannot make you will be assigned a new slot (must be in Week 32 the latest)

If you do not notify the module supervisor in advance and do not show up for your time slot you will receive a mark of zero.

Please check that your mark is recorded correctly after your demonstration.

Extenuating Circumstances

The standard extenuating circumstances procedures will apply for those who - for circumstances beyond their control - are prevented from submitting work before the deadline or from attending the lab demonstration. Please see the Undergraduate Students' Handbook for the University policies regarding these matters.

Introduction

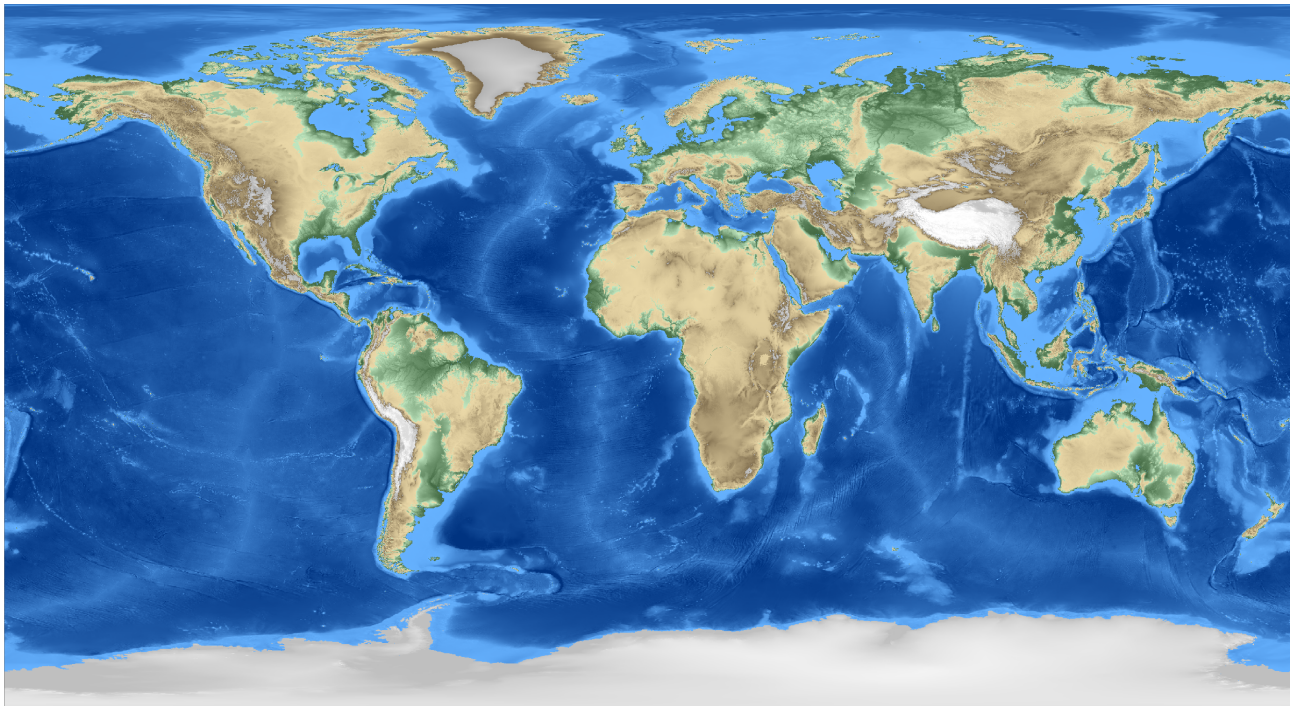

Figure 1: A potential rendering of the altitude data available on Moodle.

Dataset

The dataset on the Moodle (earth.xyz) is a downsampled version of the *ETOPO1 1 Arc-Minute Global Relief Model* (link) published in [1]. This *Global Relief Model* was published by the National Oceanic and Atmospheric Administration (NOAA). It is based on a combination of various global and regional datasets. There is a version that includes the ice sheets in Greenland and Antarctica and a version without. Our version contains the ice surface. Furthermore the original version is a file with a size of about 1 GB which has been downsampled to 61 MB for this assignment.

You will write code to read, process and visualise this data (see Figure 1 for an example of such a visualisation).

The file contains the altitude of the earth at over 2 million coordinates points specified in longitude and latitude (see Figure 2). See an example of three such coordinates below:

Longitude	Latitude	Altitude
323.500000000000	-78.6666666667000	86
323.666666667000	-78.6666666667000	124
323.833333333000	-78.6666666667000	177

In our data the longitude values range from 0 (at Greenwich) to <360 (almost again at Greenwich). The latitude values range from 90 (North Pole) to -90 (South Pole). Note the values are different in Figure 2 but the principle is the same.

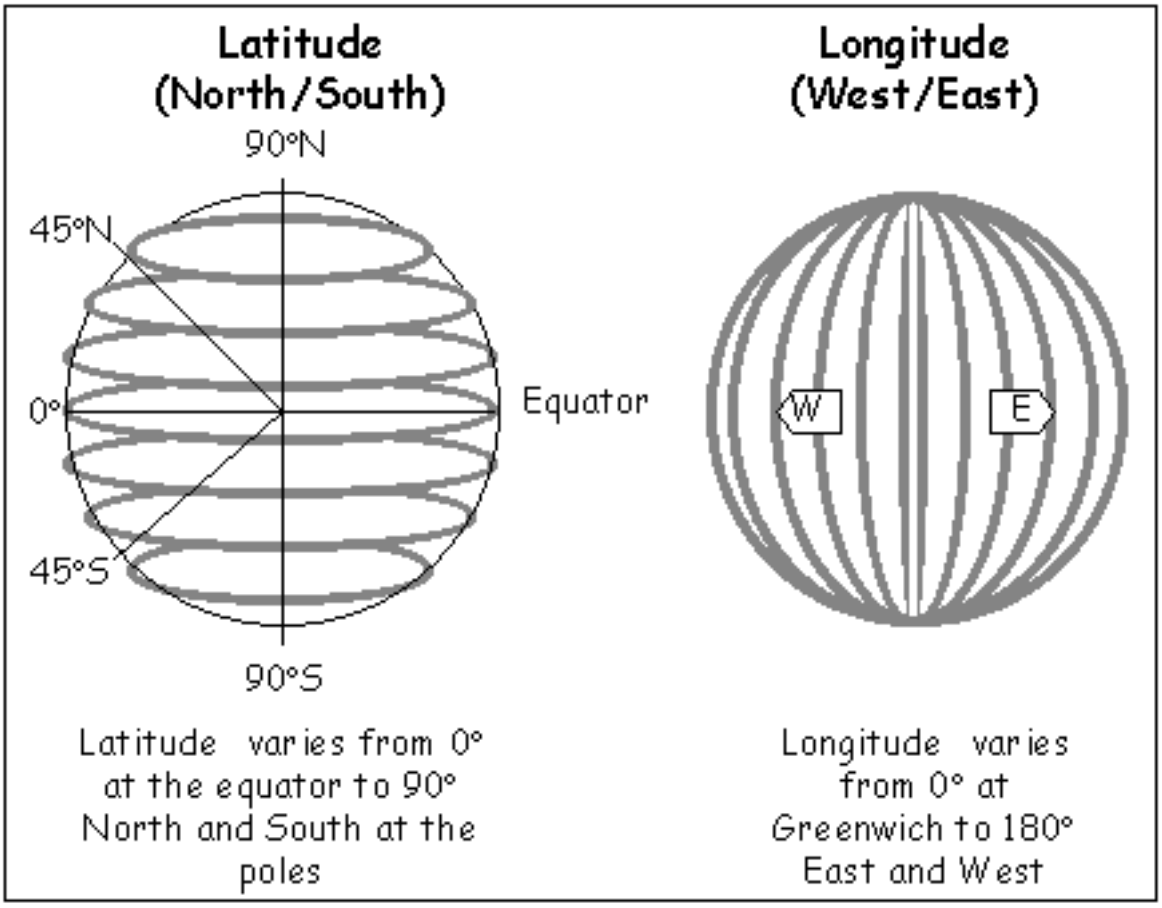

Figure 2: Latitude specifies the location going from north to south, longitude from east to west (Source: Wikipedia). Note that the range of values is different in our file, but the principle is the same.

Simulation of rising sea levels due to global warming

It is highly probable that industrial era human activity is affecting the global climate and causing average temperatures to rise (see Global Warming article on Wikipedia). It is expected that the global average temperatures will rise by between 2 and 4 degrees Celsius until the end of this century. The effects may include changes in the amount of rainfall, more extreme weather and also rising sea levels. The rise in sea levels could be caused by the melting ice in Greenland, Antarctica and glaciers at other locations of the planet. Over the long-term sea levels may rise by up to 6 meters. As part of this assignment you will simulate the effect of rising sea levels.

Start IntelliJ, open your previous project which should be called ce152 and add a new package called assignment. To do this, in the Project View, select the "src" folder. The click on "File -> New -> Package" and create a new package called assignment.

Download earth.xyz from the Moodle and place it in a directory you can access from your program.

Exercise 1 [20%]

This exercise is about reading the elevation data into an array. Create a class called Earth in a file called `Earth.java`. This file should be in the package `assignment`. See the variables and methods the class ***must*** have in Figure 3. You may create additional methods and variables of your choice. All variables and methods should be private unless access is required.

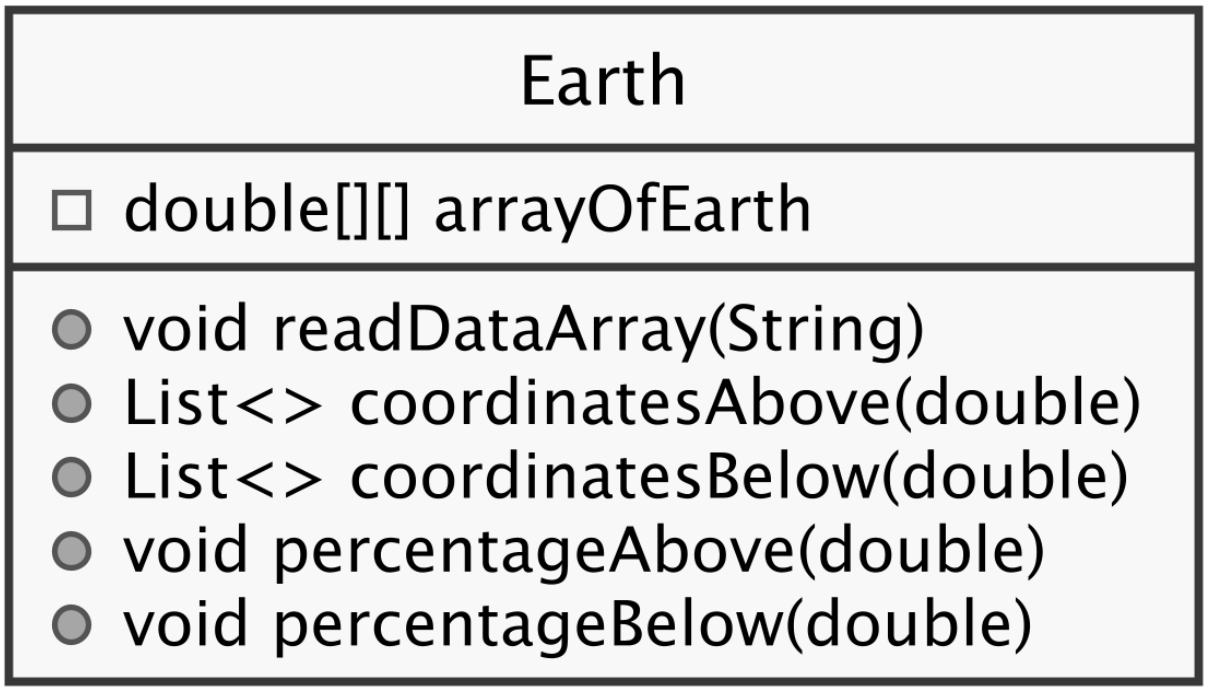

Figure 3: Class Earth you should create for Exercise 1. ***IMPORTANT:*** This diagram only shows the variables and methods the class must have. It may be convenient to create additional variables and methods.

Part A: Reading data into an array [10%]

Add a method

```
public void readDataArray(String fileName){...}
```

to your `Earth` class. Replace the "`...`" with your code. This method should store the data in a private two-dimensional array of type double:

```
private double[][] arrayOfEarth
```

The first dimension should be the coordinate number, that means a line from the file. Each line should contain longitude, latitude and altitude (in this sequence). Thus the first dimension should have a size of approximately 2 million, the second of 3.

Part B: Altitude of coordinates [5%]

Add an additional four methods to your class:

```
public List<...> coordinatesAbove(double altitude){...}
public List<...> coordinatesBelow(double altitude){...}
public void percentageAbove(double altitude){...}
public void percentageBelow(double altitude){...}
```

Replace the "..." with your code. Remember to also choose an appropriate type for the lists. The lists should contain all the coordinates below/above the specified altitude. The `percentageAbove`/`percentageBelow` methods should print the percentage of coordinates above or below the specified altitude to the command line with a precision of one value after the decimal point. All the methods in this part must use the data in `arrayOfEarth`.

Part C: Reading from System.in [5%]

Add code anywhere in your program that will prompt the user to enter an altitude (in meters or another unit of measurement if you prefer) on the command line and print the percentage of coordinates *above* this altitude. This prompt should query the user for new altitudes until the user enters "quit". You should verify that the input is a valid altitude. An example of the output could look like this:

```
Please enter an altitude:
-2000
Proportion of coordinates above -2000.0 meters: 47.7%
Please enter an altitude:
30ddd
Invalid altitude. Please enter an altitude or "quit" to end program.
Please enter an altitude:
quit
Bye!
```

Exercise 2 [25%]

In this exercise you will read the same data into a `Map` data structure. Add the methods specified in Figure 4 to your `Earth` class. Note that the diagram only shows the new variables/methods.

Part A: Reading the data into a Map [10%]

Write a method that reads the altitude data into a map data structure:

```
public void readDataMap(String fileName){...}
```

Replace "..." with your code. The data should be stored in private variable of type map:

```
private Map<...> mapOfEarth;
```

Choose an appropriate implementation of `Map` and replace "..." with an identifier for the elements in the map.

Part B: Random map [5%]

Write a method to generate a random map that will also be stored in `mapOfEarth`. You can use this method instead of reading the data from the file should you not have the time to complete reading the file. You may generate completely random data (generating a random altitude for each coordinate). Attempting to generate realistic landmasses will not lead to higher marks. The method should receive a resolution that will define the number of coordinates. For example if the

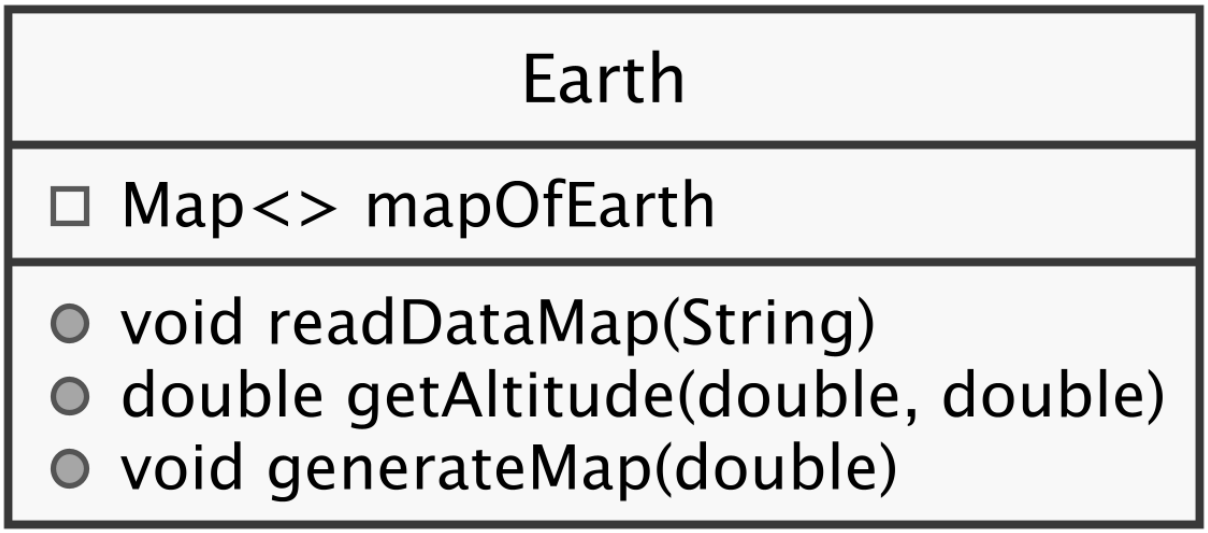

Figure 4: Methods for reading the elevation data into a Map data structure. Choose an appropriate type for the data in the map. An additional method should be provided to retrieve elevation data from the Map. ***IMPORTANT:*** This diagram shows only the methods added in Exercise 2.

resolution is 1 there will be 360 longitude * 180 latitude coordinates. If the resolution is 0.5 there will be 720 * 360, if the resolution is 2 there will 180 * 90 coordinates. The method should receive a double and return void:

```
public void generateMap(double resolution){...}
```

Replace "..." with your own code.

Part C: Retrieving altitude data from the map [5%]

Write a method that retrieves altitude data from the map data structure based on coordinates:

```
public double getAltitude(double longitude, double latitude){...}
```

Replace the "..." with your code. The double that is returned should contain the altitude at the specified position.

Part D: Read from System.in [5%]

Add code to your program that reads a longitude and latitude from `System.in`. You can integrate this into your code from 1C or add this functionality in a separate `main` method. The user should be prompted to enter two numbers. You should check if they are valid coordinates. Again this should run in an infinite loop until the user enters "quit". If the coordinates are valid output these coordinates and the respective altitude. An example of the output could look like this:

```
Please enter a longitude (0-360) and latitude (-90-90):
30 45
The altitude at longitude 30.0 and latitude 45.0 is -37.0 meters.
Please enter either 1 or 2.
Please enter 1 for percentage 2 for altitude.
2
Please enter a longitude (0-360) and latitude (-90-90):
30dd
Please enter valid longitude/latitude or "quit" to end program.
Please enter 1 for percentage 2 for altitude.
2
Please enter a longitude (0-360) and latitude (-90-90):
quit
```

Bye!

Exercise 3 [25%]

In this part you will create a visualisation of the data. This visualisation can be based on random values if you did not complete any of the previous exercises. You can consider the longitude as your x-coordinate, and the latitude as your y-coordinate.

Part A: JFrame and JComponent [10%]

Create a `JFrame` with a `JComponent`. You can use a design of your choice and any source code from the lecture notes, support classes and the lab solutions.

In the `JComponent` visualise the altitude data at the corresponding coordinates from either the file or the randomly generated data. If you did not complete Exercise 2 you can generate random values between -5000 and +5000 in a grid of 360 by 180 pixels and draw those values onto the `JComponent`. The visualisation may be via grey scale.

There are two important aspects to the visualisation: the rate at which you sample the data in the file and the size of the visualisation you use to represent the altitude in your `JComponent`. If the size of the visualisation is too small gaps will appear and the colour of the background of your `JComponent` will show..

Part B: Add some colour [3%]

Add some colour to your visualisation. To get credit for this part you must use one colour gradient for altitudes below 0 and a different colour gradient for altitudes above 0. The colour should show a variation that depends on the altitude, that means 50 m above zero should be distinguishable from 3000 m above zero.

Have a look at (link) or Figure 1 for some inspiration on which colours you could use.

You may use as many different colour gradients as you wish but will receive full credit if you use at least two.

Part C: Centre the map [2%]

Centre the map on 0 longitude. This should then be in the centre of your `JComponent`. If you are using a random array because you did not complete Exercise 2 centre on the first column of your array.

Part D: Sea level [7%]

Add a method that will shift the sea level of your rendering. Ideally you will modify the altitudes in the data structure that is the basis for your visualisation. Iterating over maps is a bit tricky but it is possible to do so with for-each loops by accessing the so-called `entrySet`.

Part D: Read sea level from args[] [3%]

Add code to your program that reads the *simulated* sea level from args[]. You can set the parameter either through IntelliJ or using the command line. If you want to run your compiled IntelliJ code from the command line remember the binaries are stored in a folder called “out”. Set the sea level and visualise the altitude data accordingly (see Figure 5 and 6 for examples).

Exercise 4 [15%]

Create a new class `MapCoordinate` in a file called MapCoordinate.java. See Figure 7 for the corresponding class diagram.

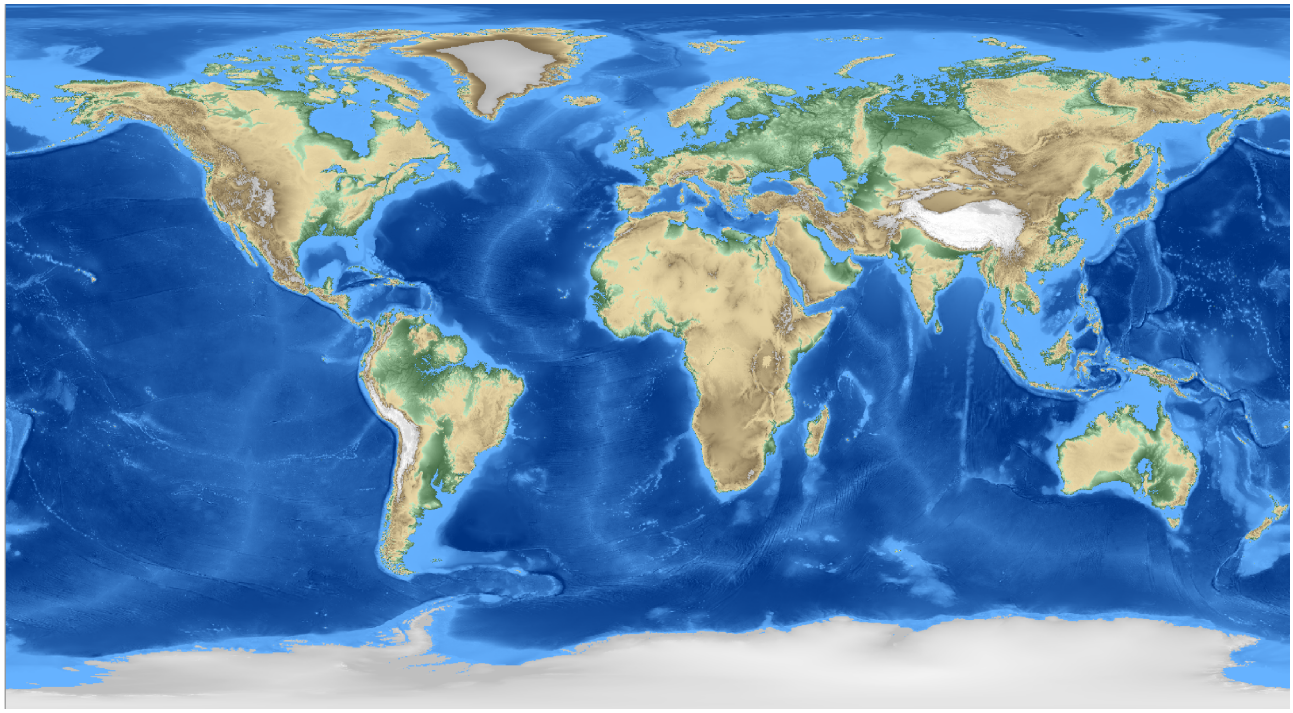

Figure 5: The Earth after sea levels rose by 50 m.

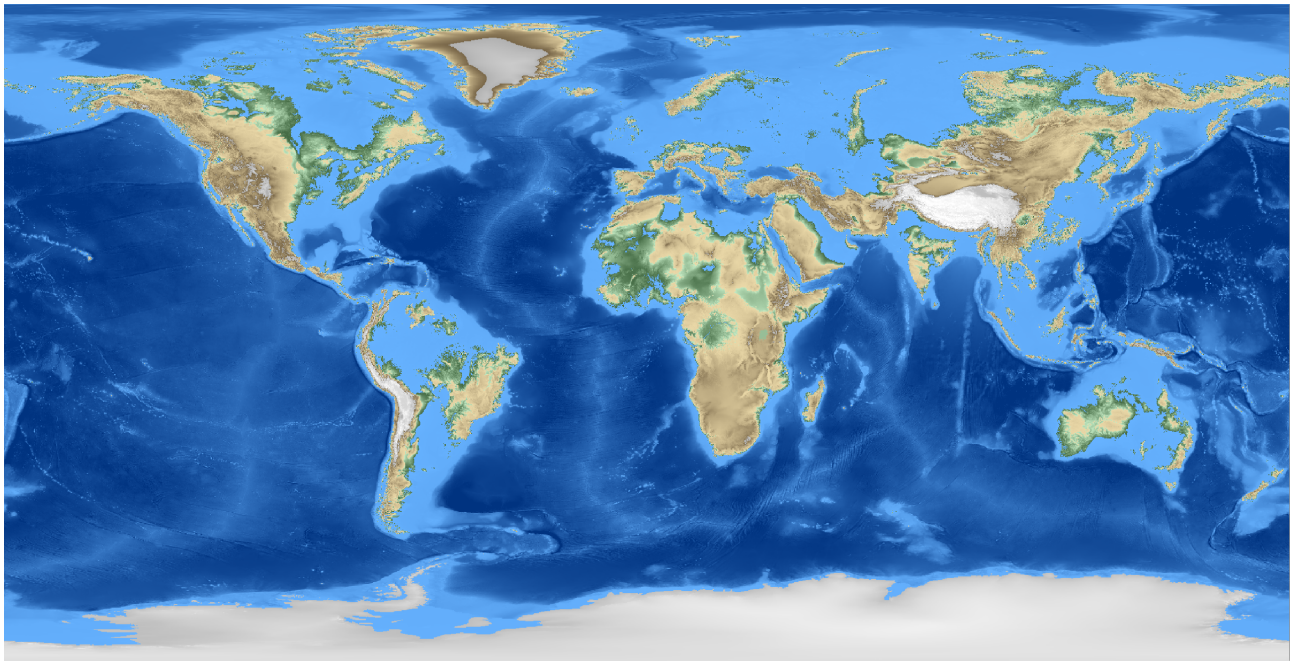

Figure 6: The Earth after sea levels rose by 250 m.

Part A: Immutable [5%]

The instance variables for latitude, longitude and altitude should be immutable and public. Design a corresponding constructor. You may add additional variables if you wish.

Part B: Distance [5%]

The class should have a method to calculate the distance to another `MapCoordinate` in km across the surface of the globe. Altitude must not be taken into account and you can assume the Earth is a smooth sphere for the purposes of this calculation.

Part C: Overriding [5%]

Furthermore the class should implement `Comparable` and override `equals` and `toString`. Equals should be consistent with the result of `compareTo()`. The comparison should be first according to altitude, then latitude, then longitude.

Exercise 5 [15%]

Use the `JFrame` from Exercise 3 if you completed it OR create a new `JFrame` with `JComponent` if you did not.

Part A: Mouse [10%]

Add a mouse listener to your graphical user interface (GUI). A left click should:

- add the coordinates of your map + true elevation to a `List<MapCoordinate>` OR if you did not complete Exercise 3+4 add a random double with the current screen coordinates to the list
- Print the clicked coordinates to the command line
- Print the distance to the previously clicked point (if there was one) to the command line
- Sort the list

A right click should:

- Delete the last coordinate that was added to the list (not the last one in the list)
- Print the deleted coordinate

Part B: Write [5%]

Add a method that will write the list of coordinates to a file. Start a new file every time the program is run (you may overwrite the existing file) but do not overwrite the file when you write a new set of coordinates.

References

[1] Amante, C., & Eakins, B. W. (2009). ETOPO1 arc-minute global relief model: procedures, data sources and analysis.

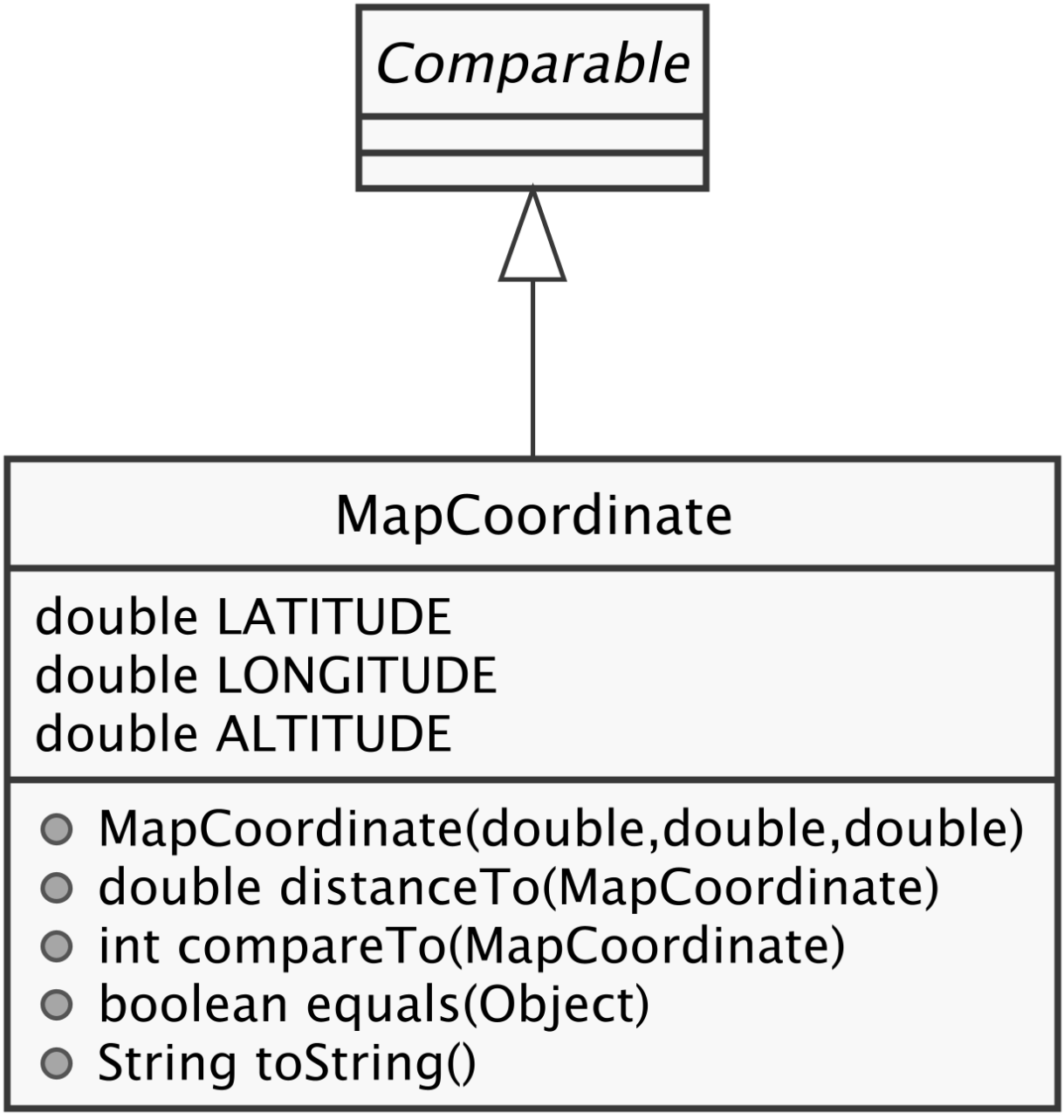

Figure 7: MapCoordinate class diagram. MapCoordinate should implement `Comparable` and override `equals` and `toString`. The coordinates should be immutable.