

Earthquake Monitor

CSC8001_COURSEWORK_PT1_2019_YANLIN GAO

Author: YANLIN GAO

Student_ID: 190522788

Earthquake Monitoring

Introduction

The overall aim of “earthquake monitoring” program is to input and retrieve the information of earthquake and observatory. As shown in figure1, the program contains 4 major classes, 3 test classes and 1 “LoadData” class consists of test data.

To be specified, “Earthquake” class and “Observatory” class store detail of earthquake events (magnitude, position (latitude, longitude), year) (Code1) and seismological observatories (name, country, covered area, construct year and a list of earthquakes recorded) (Code2) and provide a series of methods to retrieve these values. Besides, “Monitoring” class holds information of a list of observatories and contains some operations to monitor the observatory and earthquake. Moreover, “MonitoringIO” class is main class that interact with application clients. Its main() method makes use of “Earthquake”, “Observatory”, “Monitoring” objects and creates a menu (Figure 2) to enable them input and monitor information of earthquake events and observatories. So the clients can not only add earthquake event to an observatory and add an new observatory station to the list, but also monitor the statistic and turn out the most important information, like which earthquake has largest magnitude, which observatory has largest average magnitude and how many earthquake with magnitude greater than the number users input (Figure 3). As for “LoadData” class, it stores some test data.

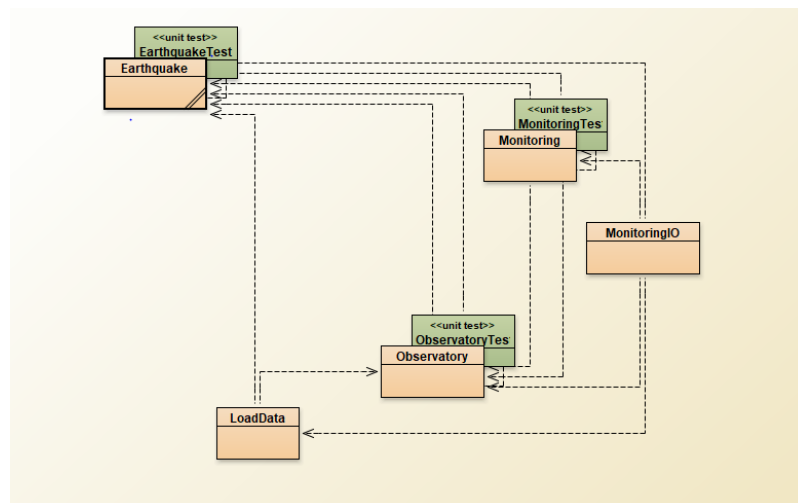


Figure1. The overview or Earthquake Monitoring project.

```
public Earthquake(double magnitude, double latitude, double longitude, int year)
{
    this.magnitude = magnitude;
    this.latitude = latitude;
    this.longitude = longitude;
    this.year = year;
    this.position = new double[2];
}
```

Code1. The Constructor of Earthquake class.

```
public Observatory(String name, String country, int year, double area,
ArrayList<Earthquake> earthquakes)
{
    this.observatoryName = name;
    this.startYear = year;
    this.country = country;
    this.area = area;
    this.earthquakes = earthquakes;
}
```

Code2. The Constructor of Observatory class.

```
#####MENU#####
##1.Enter observatory data##
##2.Enter earthquake data ##
##3.Monitor earthquakes  ##
##4.Exit                ##
#####
```

Please select one of the above options:

Figure2. The menu provided by “MonitoringIO” class.

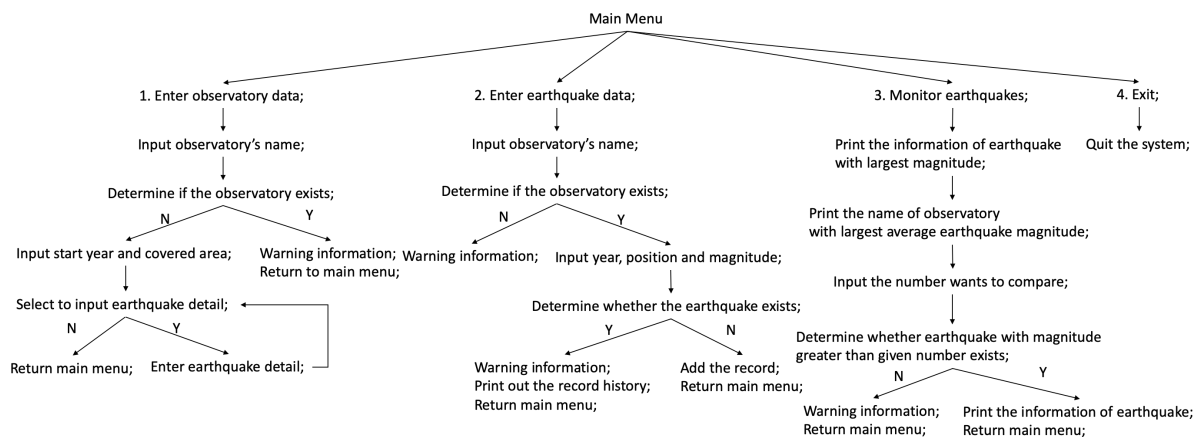


Figure3. The flow chart of “MonitoringIO” class.

Analyze

In program, I tried to use different kinds of iteration, like “for each loop”, “for loop”, and “while loop” (Code3). According to my experience, one huge edge of “for each loop” is that it is pretty easy to write. In other words, it has no need to worry about infinite iteration and the end of the iteration. However, it cannot break out at the middle of iteration, and it has to execute all elements in the collection, which is more time-consuming as it may does some useless implement, and takes more memory space when running some giant program. All in all, “for each loop” is appropriate choice when accessing all items and operating same thing in whole collection sequence, like picking up the maximum or minimum value or print all the items. As for “for loop”, it has similar pattern with “for each loop”. One distinct difference is that it can define “step”, which is better choice for a loop with the fix amount of iteration

times. It is common to use “for loop” when we want to execute same thing to each item of collection and get the index, such as Arrays, ArrayList, and HashTable. On the contrary, for “while loop”, it is much suitable for iteration that stops once met the condition. But it may lead to infinite loop if we forget to increment the counter variable.

A.

```
for (Earthquake e : earthquakes) {
    if (e.getMagnitude() > maxMagnitudeEarthquake.getMagnitude()) {
        maxMagnitudeEarthquake = e;
    }
}
```

B.

```
for (int i = 0; i < earthquakes.size(); i++){
    earthquake = earthquakes.get(i);
    if (earthquake.getMagnitude() > comparedMagnitude){
        resultEarthquake.add(earthquakes.get(i));
    }
}
```

C.

```
while (i < inputObservatory.size()){
    // Check whether the observatory exist or not;
    // If so, adding the detail of earthquake;
    if (inputObservatory.get(i).getName().equals(search)){
        newE = inputEarthquake();
        break;
    }
    else{
        i += 1;
    }
}
```

Code 3. Example of “for each loop”, “for loop” and “while loop”.

(A. Using “for each loop” to pick out the earthquake with largest magnitude; B. Using “for loop” to select earthquake in the arrayList with magnitude greater than given number; C. “while loop” to determine whether the observatory has already existed in the arrayList.)

Every other data type can be converted to string by using “toString()” method, especially when printing a variable that is not a primitive type. For example, when we use “System.out.println() to print an array or an ArrayList out directly, it would display the storage location other than the content. When I tried to print out the content of variable Earthquake, I used “System.out.println(e)”, then it printed in the form of “classname@HashCode”_in_Hexadeciaml_form^[1](Figure 4A); Then I changed to try “System.out.println(e.toString())”, then it printed the same thing(Figure 4B), which was out of my expectation and confusing, since when printing an array, using Arrays.toString() method returns a string contains all the items in the Array. After searching^[2], I decided to override the toString() method so that Earthquake variable can be show properly(Code 4). Finally, this strategy works. It turned out the content of the Earthquake object (Figure 4C). In order to make the output result more clearly, concisely and aesthetically, all these values has restricted format, like year is expressed by four digits; magnitude values and position coordinate retain two decimal places.

A.

```
System.out.println(e);
Earthquake@46985003
Earthquake@502ca3c8
Earthquake@2c05c27b
Earthquake@56ad6af8
Earthquake@2c264fcf
Earthquake@28ba1a00
Earthquake@2a2e0376
Earthquake@17e95661
Earthquake@9af3b64
```

B.

```
System.out.println(e.toString());
Earthquake@6937bb0c
Earthquake@21aa9778
Earthquake@c4824d9
Earthquake@3a6e6d7c
Earthquake@7219086c
Earthquake@262232f8
Earthquake@6b01a409
Earthquake@224d3198
Earthquake@18ad17d5
```

C.

Year	Magnitude	Position
2010	7.00	[19.07, -72.24]
2004	9.20	[-1.40, 115.42]
2008	7.90	[39.55, 116.23]
2005	7.60	[33.41, 73.03]
2003	6.60	[35.41, 51.25]
2001	9.10	[35.41, 139.46]
2001	7.70	[28.36, 77.72]
2015	7.80	[28.10, 84.15]
2018	6.40	[28.10, 84.15]

Figure4. Results of different attempts to retrieve Earthquake content.

(A. Result of “System.out.println(e)”; B. Result of “System.out.println(e.toString())”; C. Result of “System.out.println(e.toString())” after overriding toString() method.

```
public String toString()
{
    String y = Integer.toString(year);
    if (y.length() < 4){
        for (int i = 0; i < 4 - y.length(); i++){
            y = "0" + y;
        }
    }
    String strDouble = String.format("%.2f", 2.00023);
    return y + " " + String.format("%.2f", magnitude) + " [" + String.format("%.2f", latitude)
    + ", " + String.format("%.2f", longitude) + "];"
}
```

Code 4. Overriding toString() method.

There are three different ways to read console input from user, such as bufferedReader, Scanner and console class in java^[3]. All of them have advantages and disadvantages. One reason made me choose to use scanner class is that it's a helpful strategy to accept inputs with various datatypes without parsing. For example, using “.nextLine()” method to accept String object; Using “.nextInt()” method to accept Integer object; Using “.nextDouble()” to accept double. While for other console input, they can only accept “String” type, then using “.parseInt()” or “.parseDouble()” or some other method to convert to other datatype; Basically, scanner class is an articulate method to take input.

In the body of “MonitoringIO” class, combination of “try...catch...finally” and “throw... exception” method make sure the received value from client is in correct format and concept. When the application user provides bad input, the program would throw an

“IllegalArgumentException”, print out the error message to notify users there is something wrong and display an instruct message to inform them to input in right way. Besides, nesting multiple “try...catch” expression enable program respond alternatively to various wrong inputs. For instance, in option2, clients are allowed to input an earthquake event to a seismological station that has already existed in the observatory list (Figure 5A). If the observatory hasn’t been recorded yet, then user would be informed to record the observatory information first (Figure 5B). All details of the earthquake have their scales and static format, and once the client break the rule, the program would remind them of correct ones (Figure 5C-F), and keep clients inputting until they input correctly. What’s more, the program would avoid the situation that single earthquake is recorded more than once (Figure 5G).

Similarly, when users provide information of observatory by choosing option1, they can’t input observatory that has already recorded (Figure 6A). Moreover, after adding information of new observatory, program provides two options to add earthquake events observed by this observatory. For one, clients can add a series of earthquake events directly by continuously choosing “Y” (Figure 6B); The other one is that saving the observatory with empty earthquake list, then adding them by choosing option2 next time (Figure 6C).

A.

```
#####MENU#####
##1.Enter observatory data##
##2.Enter earthquake data ##
##3.Monitor earthquakes   ##
##4.Exit                  ##
#####

Please select one of the above options:
2
#####Input the detail of earthquake#####
Please input the name of observatory that you want to add earthquake record.
Willy Bob
please Input magnitude
6.83
Please input the position coordinate(latitude, longitude)
56.37,37.98
please Input the year of the event
2018
Congradulation!You've input the details of an earthquake event successfully
#####
```

B.

```
#####Input the detail of earthquake#####
Please input the name of observatory that you want to add earthquake record.
somewhere
The observatory "somewhere" doesn't in the list.
Please record the observatory first.
```

C.

please Input magnitude

-6

-6.0 is a wrong input.
Magnitude must within [1, 12].
Please check the magnitude and enter again

D.

Please input the position coordinate(latitude, longitude)

35.41

35.41 is wrong input
position coordinate must be input by (latitude, longitude)
Please check the position coordinate.

E.

Please input the position coordinate(latitude, longitude)

-233,328

[-233, 328] is wrong input
Both latitude and longitude should be in the scale [-180, 180].
Please check the coordinate enter again.

F.

please Input the year of the event

2010

2010 is a incorrect input.
Observatory Willy Bob was constructed in 2015.
So year should be in the scale of [2015, 2019].
Please check the year.

G.

#####Input the detail of earthquake#####

Please input the name of observatory that you want to add earthquake record.

Willy Bob

please Input magnitude

6.4

Please input the position coordinate(latitude, longitude)

28.10,84.15

please Input the year of the event

2018

Earthquake "2018 6.40 [28.10,84.15]" has already recorded by Observatory Isla Barro Colorado.
Please check the detail!

Figure5. Examples of correct input earthquake and some normal error inputs.

(A. The correct format of recording earthquake event; B. The warning message of adding earthquake to observatory that doesn't exist; C. The warning message of inputting magnitude out of scale; D. The error message of providing position in wrong format; E. The warning message of inputting latitude and longitude out of scale; F. The warning message of providing year before the construction year or after current year; G. The warning message of adding an earthquake event that has been recorded.)

A.

Please select one of the above options:

1

#####Input the detail of Observatory#####

please Input observatory's name

Willy Bob

The Willy Bob has already existed.
Please add the earthquake event directly.

B.

```

Please select one of the above options:
1
#####Input the detail of Observatory#####
please Input observatory's name
newObservatory
please Input the country the observatory located
somewhere
please input the start year of observatory
2019
please input the area covered by the observatory(Unit: km^2)
1000
Would you like to add the detail of events recorded at the observatory? Y/N
Y
please Input magnitude
8.0
Please input the position coordinate(latitude, longitude)
56.2,36.7
please Input the year of the event
2019
Would you like to add the detail of events recorded at the observatory? Y/N
Y
please Input magnitude


```

C.

```

Would you like to add the detail of events recorded at the observatory? Y/N
N
You've input the information of newObservatory successfully!
Please select one of the above options:
2
#####Input the detail of earthquake#####
Please input the name of observatory that you want to add earthquake record.
newObservatory
please Input magnitude
6
Please input the position coordinate(latitude, longitude)
67,37.9
please Input the year of the event
2019
Congradulation!You've input the details of an earthquake event successfully!

```



D.

```

Would you like to add the detail of events recorded at the observatory? Y/N
y
Would you like to add the detail of events recorded at the observatory? Y/N
n
Would you like to add the detail of events recorded at the observatory? Y/N
ajs

```

Figure6. Process of adding observatory information.

(A. Warning message of adding duplicated observatory; B. Adding earthquake record by choosing “Y”; C. Choosing “N” and adding earthquake event by choosing option2; D. Other options neither “Y” nor “N” are not acceptable.)

As mentioned above (Figure 3), Option3 is designed to provide essential data of the observatory and earthquake (Figure 7A). And it would update immediately when a new earthquake or a new observatory is recorded. For example, after add new earthquake record with magnitude 10.0 to “Willy Bob” observatory, both average magnitude of “Willy Bob” and earthquake with largest magnitude change. Obviously, list of earthquakes with magnitude

greater than 1.0 add the new member (Figure 7B). Correspondently, adding an observatory named “newObservatory” which observed earthquake with magnitude 9.8, then the observatory with largest average magnitude is replaced by it (Figure 7C).

A.

Please select one of the above options:

3

Please input the numeber you want to be compared

1

The largest average earthquake magnitude is 8.03, which is recorded by Willy Bob.

The largest earthquake ever is Earthquake happened in [-1.4, 115.42] with magnitude 9.2, 2004.

Earthquake with magnitude greater than 1.0:

Year	Magnitude	Position
2010	7.00	[19.07,-72.24]
2004	9.20	[-1.40,115.42]
2008	7.90	[39.55,116.23]
2005	7.60	[33.41,73.03]
2003	6.60	[35.41,51.25]
2001	9.10	[35.41,139.46]
2001	7.70	[28.36,77.72]
2015	7.80	[28.10,84.15]
2018	6.40	[28.10,84.15]

B.

Please input the numeber you want to be compared

1

The largest average earthquake magnitude is 8.53, which is recorded by Willy Bob.

The largest earthquake ever is Earthquake happened in [35.0, 26.0] with magnitude 10.0, 2019.

Earthquake with magnitude greater than 1.0:

Year	Magnitude	Position
2010	7.00	[19.07,-72.24]
2004	9.20	[-1.40,115.42]
2008	7.90	[39.55,116.23]
2019	10.00	[35.00,26.00]
2005	7.60	[33.41,73.03]
2003	6.60	[35.41,51.25]
2001	9.10	[35.41,139.46]
2001	7.70	[28.36,77.72]
2015	7.80	[28.10,84.15]
2018	6.40	[28.10,84.15]

C.

Please input the numeber you want to be compared

1

The largest average earthquake magnitude is 9.8, which is recorded by newObservatory.

The largest earthquake ever is Earthquake happened in [35.0, 26.0] with magnitude 10.0, 2019.

Earthquake with magnitude greater than 1.0:

Year	Magnitude	Position
2010	7.00	[19.07,-72.24]
2004	9.20	[-1.40,115.42]
2008	7.90	[39.55,116.23]
2019	10.00	[35.00,26.00]
2005	7.60	[33.41,73.03]
2003	6.60	[35.41,51.25]
2001	9.10	[35.41,139.46]
2001	7.70	[28.36,77.72]
2015	7.80	[28.10,84.15]
2018	6.40	[28.10,84.15]
2019	9.80	[45.00,78.00]

Figure 7. Results of Option3.

(A. The initial result of test data; B. Result after adding new earthquake event to observatory “Willy Bob”; C. Result after inputting new observatory.)

For testing part, I used three different strategies. Firstly, for all the “return” method, using JUnit framework with “assertEquals()” method to investigating if the program turns out the expected result (Figure8). However, if using JUnit to test “void” method, it would have an error and could not be compiled. Therefore, I made use of debugger, breakpoint and print statement “System.out.println()” to check the flow of “void” method. It is convenient to verify if the operations in the program work well or the values of variables and parameters are defined correctly or the data is collected as I expected. Besides, “LoadData” class contains detail of three observatories, and each of them records three earthquake events. With assistance of them, I walked through the whole program step by step and examined each branch to discover the potential logical errors.

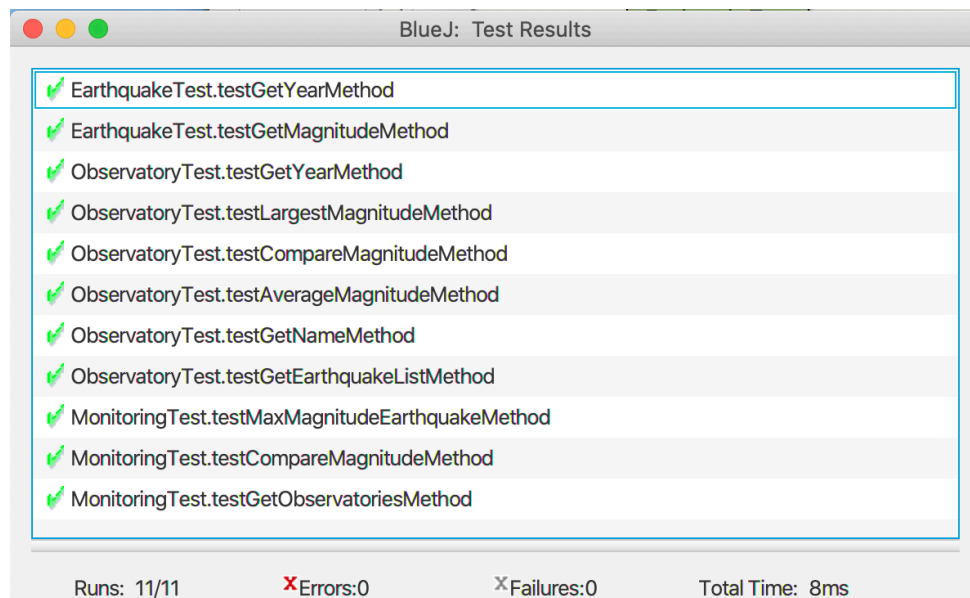


Figure 8. The test results of JUnit.

Discussion

In this program, I assumed ideally that every earthquake has unique magnitude and every observatory's average magnitude is different. However, in the real life, it is likely that there is more than one earthquake with same magnitude, and all of them are the largest magnitude earthquake. Also, two or more observatories may have same average earthquake magnitude which is the largest among all. Therefore, it is more appropriate to return all of these results other than single result. Besides, before inputting the information of earthquake, client should select an observatory in the list first; It would be better to check the fact that if the position of earthquake they added is covered by the detective area.

Basically, I used ArrayList to store the information of different observatories, and had “while loop” to operate the simple search to check if the observatory has already existed, which is to

avoid storing duplicated items. However, considering that scanning a giant list is quite time consuming, it is better option to use HashTable other than ArrayList, using observatories' name as key, and store other details in the value which is much more efficient and effective to use HashTable to insert, delete and find specified item than using an array or an ArrayList. In average condition, HashTable takes constant time $O(1)$ for inserting, deleting and searching element^[4]. Even in the worst case, it takes linear time $O(n)$, just as same as simple search. It is not uncommon to use HashTable check for the duplicates as it implements very fast.

This program interacts with users by simple console input and output. Also, it might be a bit clumsy and annoyed if users input something wrong constantly. It would be better to have graphic interface, where has elements like list or calendar to provide some options assist client to select by clicking instead of manually typing and some warning icons to inform errors, which would improve use experience.

Reference

- [1] David J. Barnes, Michael Kölling. *Objects first with java: a practical Introduction using BlueJ*. Boston: Pearson Education Inc. Print.
- [2] Bishal Kumar Dubey. *Object toString() Method in Java*. Retrieved from <https://www.geeksforgeeks.org/object-tostring-method-in-java/>.
- [3] DATAFLAIR TEAM. (2018, September 14). *How to Read Java Console Input | 3 Ways To Read Java Input*. Retrieved from <https://data-flair.training/blogs/read-java-console-input/>.
- [4] Aditya Y. Bhargava. *Grokking algorithms*. New York: Manning Publications Co. Print.