A picture containing logo

Description automatically generated

**MEHRAN UNIVERSITY OF ENGINEERING AND TECHNOLOGY, JAMSHORO**

[**PROBLEM BASED LEARNING ASSIGMENT**](https://github.com/UzairHussain193/Earthquakes_Dataset_PBL)

**( 21SW – III )**

**DATA STRUCTURES AND ALGORITHMS**

**BY**

**UZAIR HUSSAIN SHAIKH 21SW085-III**

**SUBMITTED TO:**

**SIR MOHSIN MEMON**

**Tasks**

**Dataset**: <https://www.kaggle.com/datasets/usgs/earthquake-database>

**Step** **1**: Use <https://www.daniel-braun.com/technik/reverse-geocoding-library-for-java/> library to find out the city and country from the given coordinates and store them in yearly earthquake collection along with magnitude. (Collection of each year means 52 collections)

**Step 2**: Make a queue storing biggest (with highest magnitude) quake of each year with magnitude and country, starting from 1965 to 2016. (52 elements in the queue approx.).

**Step 3**: Make a stack from the collections, one for each country which stores earthquake and its magnitude in the order of the event (the most recent event on top).

**Step 4**: Make a linked list which saves the one most recent earthquake with magnitude and country name from each country (use the stack from step 3).

**Problem 1**: How to find the average number of earthquakes per year for each country and which country is most vulnerable to earthquakes (which country has the greatest number of earth quakes)?

**Problem 2**: Which are the biggest earthquakes from 2005 to 2015 and occurred and in which country (use step 2)?

**Problem 3**: How to determine the recent 5 earthquakes from each country?

**Problem 4**: How to find the most recent above 6 magnitude earthquakes (use step 4)

**Make the algorithm and code (GUI or command line for results) for the above-mentioned problems.**

**You can also find project of GitHub:**

* <https://github.com/UzairHussain193/Earthquakes_Dataset_PBL>

**Class to read and write CSV File:**

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import com.opencsv.CSVWriter;

import eu.bitm.NominatimReverseGeocoding.Address;

import eu.bitm.NominatimReverseGeocoding.NominatimReverseGeocodingJAPI;

        // Un Cleaned files address

class CSV{

    public static void main(String[] args) throws Exception{

    int i=9631;

    Address ab;

    NominatimReverseGeocodingJAPI n1= new NominatimReverseGeocodingJAPI();

    String csvFilePath = "E:\\2. Muet material\\3. 3rd Sem SW\\1. DSA\\4. DSA\_PBL\_SEM\_ASSIGNMENT\\PBL\_21SW085\\PBL\_21SW85\\database.csv";

    String line;

    String csvDelimiter = ","; // or

    String outputCsvFile = "Addresses.csv";

    try (BufferedReader br = new BufferedReader(new FileReader(csvFilePath));

             FileWriter writer = new FileWriter(outputCsvFile)) {

            String[] header = {"S.No,Date,Time,Country,Magnitude,Address"}; // Write the header row for the output CSV file

            writer.write(header[0]);

            writer.write(System.lineSeparator());

            br.readLine();

        while ((line = br.readLine()) != null && i>=9631) {

            String[] values = line.split(csvDelimiter);

                // Access the values by index (e.g., values[0] for the first column)

                double latitude = Double.parseDouble(values[2]);

                double longitude = Double.parseDouble(values[3]);

                // Do something with the column values

                ab=n1.getAdress(latitude,  longitude);

                System.out.print(ab);

                String add= ab.toString();

                if(add.equals("")){

                    continue;

                }

                String country= ab.getCountry();

                String city = ab.getCity();

                String state= ab.getState();

                writer.write(String.valueOf(i+1));

                writer.write(csvDelimiter);

                writer.write(values[0]);

                writer.write(csvDelimiter);

                writer.write(values[1]);

                writer.write(csvDelimiter);

                writer.write(country);

                writer.write(csvDelimiter);

                writer.write(city);

                writer.write(csvDelimiter);

                writer.write(state);

                writer.write(csvDelimiter);

                writer.write(values[8]);

                writer.write(csvDelimiter);

                writer.write(add);

                writer.write(System.lineSeparator());

                i++;

            }

            br.close();

        } catch (IllegalStateException e) {

            e.printStackTrace();

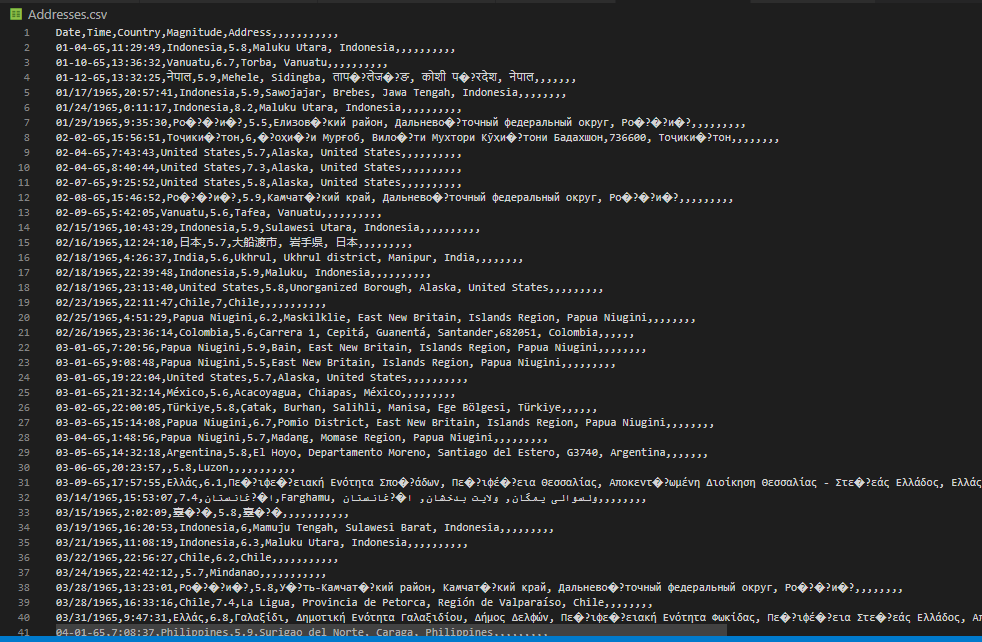
        }

        System.out.println(i);

    }

}

**Output:**

****

**Step 1:**

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

import com.opencsv.CSVReader;

import com.opencsv.exceptions.CsvValidationException;

class LinkedNode{           // node class use in collections

    LinkedNode next;

    int year;

    String country,state;

    float magnitude;

    Object date;

    Object time;

    LinkedNode(){

    }

    // constructor to add data

    public LinkedNode(int year, String country,String state, Object date, float magnitude, Object time){

        this.year=year;

        this.country=country;

        this.magnitude=magnitude;

        this.state=state;

        this.time=time;

        this.date=date;

    }

    //method to display element in nodes

    public String display(){

        return year +" , "+country + " , " + state+" , "+ date + " , " + magnitude + " , " + time;

    }

}

class Collections {

    ArrayList<LinkedNode> earthquake = new ArrayList<>();       // collection to store earthquakes yearly

    // arrays to store data from columns and parsing

    float[] highestmagnitudes;

    String[] country;

    int[] years;

    // contructor of collection class to initialize process by just calling

    Collections() throws IOException{

    int Year;

    int count=0;

    String filepath= "E:\\2. Muet material\\3. 3rd Sem SW\\1. DSA\\4. DSA\_PBL\_SEM\_ASSIGNMENT\\PBL\_21SW085\\PBL\_21SW85\\Cleaned\_Data.csv";

    String line;

    String splitBy = ",";

    String add;

    float magnitude;

    // Read the file and extract the year from the date in column 1

        try {

            BufferedReader br = new BufferedReader(new FileReader(filepath));

            br.readLine();      // reada header line

            while ((line = br.readLine()) != null) // returns a Boolean value

            {

                add="";

                String[] parts = line.split(splitBy); // use comma as separator

                magnitude= Float.parseFloat(parts[4]);  // reads magnitude column

                Year = Integer.parseInt(parts[0].substring(6));   //reads year from date column

                // add data in collection

                if(earthquake.size()!=0){

                    for(int i=0;i<earthquake.size();i++){

                        if(Year==(earthquake.get(i).year)){

                            LinkedNode temp = earthquake.get(i);

                            while(temp.next!=null){

                                temp=temp.next;

                            }

                            temp.next= new LinkedNode(Year, parts[2], parts[3], parts[0], magnitude, parts[0]);

                            continue;

                        }

                        else if(i==earthquake.size()-1){

                            LinkedNode temp= new LinkedNode(Year, parts[2], parts[3], parts[0], magnitude, parts[0]);

                            earthquake.add(temp);

                            break;

                        }

                    }

                }

                else if(earthquake.size()==0){

                    LinkedNode temp = new LinkedNode(Year, parts[2], parts[3], parts[0], magnitude, parts[0]);

                    earthquake.add(temp);

                }

            }

        br.close();

        count++;

        } catch (FileNotFoundException e) {

            System.out.println("File not found: ");

            e.printStackTrace();

        }

    }

    void display\_Collections(){

        try {

            Collections a = new Collections();

        } catch (Exception e) {

            // TODO: handle exception

        }

        int Year;

        Scanner s=new Scanner(System.in);

        System.out.print("Enter Year :  ");

        Year=s.nextInt();

        LinkedNode a = earthquake.get(Year-1965);

        int x=1;

        while(a!=null){

            System.out.println(x + " : " + a.time + " : " + a.year + " : " + a.country + " : " + a.state + " : " + a.magnitude);

            a.display();

            a=a.next;

            x++;

        }

    }

    void display\_All\_Collections(){

        for(int i=0;i<52;i++){     //  by using line 107, 108 we can see each year collection with thier count

            LinkedNode n=earthquake.get(i);

            int x=1;

            while(n!=null){

                System.out.println(x + " : " + n.time + " : " + n.year + " : " + n.country + " : " + n.state + " : " + n.magnitude);

                n.display();

                n=n.next;

                x++;

            }

        }

    }

    public int menu(){

        Scanner s = new Scanner(System.in);

        System.out.println("\n\tPBL Tasks ");

        System.out.println("1. Step 1 (Yearly Collections of Earthquakes.) ");

        System.out.println("2. Step 2 (Yearly Queue of Earthquakes.) ");

        System.out.println("3. Step 3 (Country Stacks of Earthquakes.) ");

        System.out.println("4. Step 4 (Recent Earthquake of each year.) ");

        System.out.println("5. Problem 1 (Most Vulnerable Country.) ");

        System.out.println("6: Problem 2 (Biggest Earthquakes from 2005 to 2015.) ");

        System.out.println("7: Problem 3 (Recent 5 Earthquakes.) ");

        System.out.println("8: Problem 4 (Most recent Earthquakes with magnitude above 6.) ");

        System.out.println("0. Exit");

        System.out.print("\nEnter Your Choice: ");

        int Choice=s.nextInt();

        return Choice;

    }

    void highest() {

        int j=0;

        highestmagnitudes = new float[52]; // Array to store highest magnitudes

        country = new String[52];

        years = new int[52];

        for(int i = 1965; i <=2016; i++) {

            highestmagnitudes[i-1965] = 0; // Initialize to smallest possible float value

        }

        // Loop through each year and find the highest magnitude for that year

        for(int i = 0; i < earthquake.size(); i++) {

            LinkedNode n = earthquake.get(i);

            while(n != null) {

                float high=highestmagnitudes[n.year-1965];

                if(n.magnitude > highestmagnitudes[n.year - 1965]) { // Update highest magnitude for the year

                    highestmagnitudes[n.year - 1965] = Math.max(n.magnitude, high);

                    country[i]=(String)n.country;

                    years[i]= n.year;

                }

                n = n.next;

            }

        }

        for(int i = 0; i < 52; i++,j++) {

            LinkedNode n = earthquake.get(i);

            System.out.println(years[i]  + " : " + country[i] + " : " + highestmagnitudes[i]);

        }

        System.out.println(j);

    }

    int size(){

        return earthquake.size();

    }

    LinkedNode get(int index){

        return earthquake.get(index);

    }

    public static void main(String[] args) throws IOException {

        Collections a = new Collections();

        // a.display\_All\_Collections();

        a.display\_Collections();

    }

}

**Algorithm of Step 1.**

1. In this step we have to create collections from csv file in which we store data from addresses using Nomination Reverse Geocoding Library.
2. Then we have to read that file and store country, city, magnitude and state from that files columns.
3. I have created linked list which store that data and that I have created array list of that linked list and then I have created three arrays of different data types which store data from column of csv while reading and helps in store that data in array list of linked list.
4. This all work done as the class’s object initialize and then collections will stored in array list.
5. For reading CSV file I used Buffer Reader and it iterate up to the size of array list and store in linked list and then add that linked list in array list.
6. In last I have method of display collections by asking year and then print all collections of that year.
7. I also have a method of menu for main class in for asking options what to do with this data set.

**Output:**

**Text

Description automatically generated**

**Step 2 & Problem 2:**

import java.io.IOException;

interface Queue{

    public void addQueue(int year, String country, float magnitude);

    public int size();

}

class LinkedQueue implements Queue{

    private DataNode head= new DataNode(0, null, 0);

    private int Queuesize;

    private static class DataNode{  // inner node class to add data

        int year;

        String country;

        float magnitude;

        DataNode prev=this;

        DataNode next=this;

        public DataNode(int year,String country,float magnitude){

            this.year=year;

            this.country=country;

            this.magnitude=magnitude;

        }

        // constructor to add data

        public DataNode(int year,String country,float magnitude,DataNode n, DataNode p){

            this.year=year;

            this.country=country;

            this.magnitude=magnitude;

            next=n;

            prev=p;

        }

        // method to display data

        public void display(){

            System.out.println( year +" : "+country + " : " + magnitude);

        }

    }

    LinkedQueue(){

    }

    public int size(){

        return Queuesize;

    }

    public void addQueue(int year, String country,float magnitude) {

        head.prev.next = new DataNode(year, country, magnitude, head, head.prev);

        head.prev = head.prev.next;

        ++Queuesize;

    }

    public boolean isEmpty(){

        return (head==null);

    }

    // returns higest all yearly

    void  highest() throws IOException {

        Collections earthquake = new Collections();

        LinkedQueue q=new LinkedQueue();

        int num=1;

        int year=0;

        float highest=0;

        String Country="";

        for(int i = 0; i < earthquake.size(); i++) {

            LinkedNode n = earthquake.get(i);

            highest=0;

            while(n != null) {

                if(n.magnitude > highest) { // Update highest magnitude for the year

                    highest=n.magnitude;

                    Country=n.country;

                    year=n.year;

                }

                n = n.next;

            }

            q.addQueue(year, Country, highest);

        }

        for(DataNode t=q.head.next;t!=q.head;t=t.next){

            System.out.print(num+ " : ");

            t.display();

            num++;

        }

    }

    // returns highest from 2005 to 2016

    void  highest2() throws IOException {

        LinkedQueue q=new LinkedQueue();

        Collections earthquake = new Collections();

        int num=1;

        int year=0;

        float highest=0;

        String Country="";

        for(int i = 0; i < earthquake.size(); i++) {

            LinkedNode n = earthquake.get(i);

            highest=0;

            while(n != null) {

                if(n.magnitude > highest) { // Update highest magnitude for the year

                    highest=n.magnitude;

                    Country=n.country;

                    year=n.year;

                }

                n = n.next;

            }

            q.addQueue(year, Country, highest);

        }

        for(DataNode t=q.head.next;t!=q.head;t=t.next){

            if(t.year>2004 && t.year<2016){

                System.out.print(num+ " : ");

                t.display();

                num++;

            }

        }

    }

    // method to display earthquake all yearly

    void display\_All() {

        try {

            highest();

        } catch (Exception e) {

            // TODO: handle exception

        }

    }

    /// Problem # 02

    // method to display earthquake from 2005 to 2015

    public void displayFrom\_05\_15() {

        try {

            highest2();

        } catch (Exception e) {

            // TODO: handle exception

        }

    }

    public static void main(String[] args) {

        LinkedQueue q = new LinkedQueue();

        q.displayFrom\_05\_15();

    }

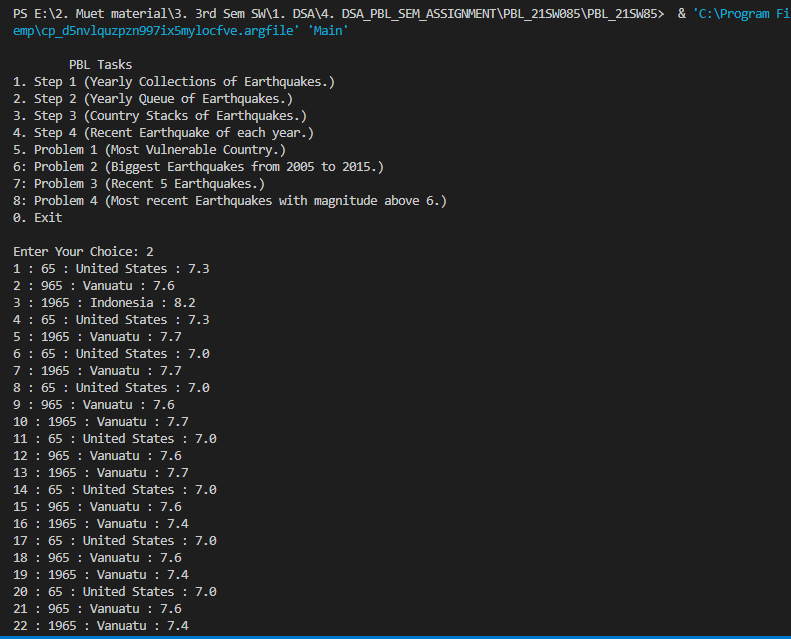
}

**Algorithm of Step 2 & Problem 2.**

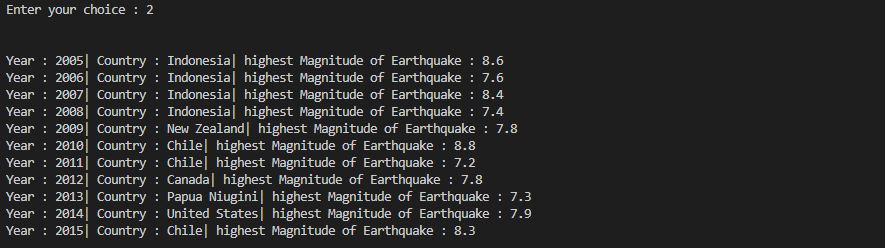
1. In this step I have create an interface of queue having methods addQueue and size.
2. Then I have created Linked Queue class which implements that interface.
3. Then I also created static class of Data Node which store data and also display method to display these data elements.
4. Then there is add Queue method which adds data in node and size increases for each addition.
5. Then there is highest method which calculates the highest magnitudes for each year and stores in node and then returns country, state and magnitude of highest one.
6. And there is also a method to display highest from each country.
7. Also there is problem 2 in which we have to calculate and display highest earthquakes from 2005 to 2015.

**Outputs:**

**step 2:**

****

**prob 2:**

****

**Step 3,4:**

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Scanner;

import java.util.Stack;

class StackNode{        // node to store stacks data

    int year;

    int size;

    String country;

    float magnitude;

    StackNode prev=null;

    StackNode next=null;

    StackNode top=null;

    StackNode(){

    }

    StackNode(int year,String country,float magnitude){     // constructor

        this.year=year;

        this.country=country;

        this.magnitude=magnitude;

    }

    StackNode(int year,float magnitude){

        this.year=year;

        this.magnitude=magnitude;

    }

    public void push1(int year2, String country2, float magnitude2) {

        this.year=year2;

        this.country=country2;

        this.magnitude=magnitude2;

    }

    // displays elements in node

    public void display3() {

        System.out.println( year +" : "+ country+" : "+ magnitude);

    }

}

class Stacks{

    StackNode head;

    ArrayList<Stacks> countries= new ArrayList<>();         // arraylist of stacks to store country wise earthquakes

    int size;

    public void push2(int year,String Country, float magnitude) {

        StackNode temp = new StackNode( year, Country, magnitude);

        temp.next=head;

        head = temp;

        size++;

    }

    public void peek(){

        StackNode n=head;

        while(n!=null){

            System.out.println(n.country+ " : " + n.year + " : " + n.magnitude);

            n=n.next;

        }

    }

    boolean isEmpty(){

        return (head==null);

    }

    public void CountryStacks() throws IOException{

        int Year;

        String country;

        String filepath= "E:\\2. Muet material\\3. 3rd Sem SW\\1. DSA\\4. DSA\_PBL\_SEM\_ASSIGNMENT\\PBL\_21SW085\\PBL\_21SW85\\Cleaned\_Data.csv";

        String line;

        String splitBy = ",";

        String add;

        float magnitude;

        // Read the file and extract the year from the date in column 1

        try {

            BufferedReader br = new BufferedReader(new FileReader(filepath));

            br.readLine();      // reada header line

            while ((line = br.readLine()) != null) // returns a Boolean value

            {

                add="";

                String[] parts = line.split(splitBy); // use comma as separator

                magnitude= Float.parseFloat(parts[4]);

                Year = Integer.parseInt(parts[0].substring(6));

                country= parts[2];

                if(countries.size()!=0){

                    for(int i=0;i<countries.size();i++){

                        if(countries.get(i).head.country.equals(country)){

                            countries.get(i).push2(Year, country, magnitude);

                            break;

                        }

                        else if (i==countries.size()-1){

                            Stacks temp = new Stacks();

                            temp.push2(Year, country, magnitude);

                            countries.add(temp);

                            break;

                        }

                    }

                    continue;

                }

                else{

                    Stacks temp=new Stacks();

                    temp.push2(Year, country, magnitude);

                    countries.add(temp);

                    continue;

                }

            }

        br.close();

        } catch (FileNotFoundException e) {

            System.out.println("File not found: ");

            e.printStackTrace();

        }

    }

    // method to print stacks recent ones first but all

    void recent\_stacks(){

        try {

            CountryStacks();

        } catch (Exception e) {

            // TODO: handle exception

        }

        StackNode t=countries.get(0).head;

        StackNode r=new StackNode(t.year, t.country, t.magnitude);

        StackNode temp=r;

        for(int i=1;i<countries.size();i++){

            StackNode x=countries.get(i).head;

            temp.next=new StackNode(x.year, x.country, x.magnitude);

            temp=temp.next;

        }

        for(StackNode z=r;z!=null;z=z.next){

            z.display3();

        }

    }

    // method to print stacks yearly but recent one first

    void display\_Yearly\_Stacks(){

        int num=0;

        try {

            CountryStacks();

        } catch (Exception e) {

            // TODO: handle exception

        }

        StackNode t=countries.get(0).head;

        StackNode r=new StackNode(t.year, t.country, t.magnitude);

        StackNode temp=r;

        for(int i=1;i<countries.size();i++){

            StackNode x=countries.get(i).head;

            temp.next=new StackNode(x.year, x.country, x.magnitude);

            temp=temp.next;

        }

        int c=1;

        for(int i=0;i<countries.size();i++){

            StackNode a = countries.get(i).head;

            while(a!=null){

                if(c>150){

                    break;

                }

                System.out.print(c+". ");

                a.display3();

                a=a.next;

                c++;

            }

        }

    }

    public static void main(String[] args) {

        Stacks s = new Stacks();

        // s.display\_Yearly\_Stacks();

        // s.average\_earthquake();

        s.recent\_above\_6();

    }

}

**Problem 1:**

    // method to vulnerablitlity of coutnries

    void average\_earthquake(){

        float count=0;

        String country\_vulnerable = "";

        double result=0;

        try {

            CountryStacks();

        } catch (Exception e) {

            // TODO: handle exception

        }

        StackNode t=countries.get(0).head;

        StackNode r=new StackNode(t.year, t.country, t.magnitude);

        StackNode temp=r;

        // for(int i=1;i<countries.size();i++){

        //     StackNode x=countries.get(i).head;

        //     // x.display3();

        //     temp.next=new StackNode(x.year, x.country, x.magnitude);

        //     temp=temp.next;

        // }

        for(int i=0;i<countries.size();i++){

            count=0;

            // System.out.println(countries.get(i).size);

            StackNode a = countries.get(i).head;

            country\_vulnerable=a.country;

            a=a.next;

            count=countries.get(i).size;

            if(a==null){

                break;

            }

            result=(count/52);

            System.out.println(country\_vulnerable  + " : "+count + " : " + result );

        }

    }

**Problem 3:**

// method to print 5 recent stacks of each country

    void display\_recent\_5(){

        int num=1;

        try {

            CountryStacks();

        } catch (Exception e) {

            // TODO: handle exception

        }

        StackNode t=countries.get(0).head;

        StackNode r=new StackNode(t.year, t.country, t.magnitude);

        StackNode temp=r;

        for(int i=1;i<countries.size();i++){

            StackNode x=countries.get(i).head;

            // x.display3();

            temp.next=new StackNode(x.year, x.country, x.magnitude);

            temp=temp.next;

        }

        for(int i=0;i<countries.size();i++){

            StackNode a = countries.get(i).head;

            for(num=0;num<6;num++){

                a.display3();

                a=a.next;

                if(a==null){

                   break;

                }

            }

        }

    }

**Problem 4:**

    // method to print stacks recent but having magnitude above 6

    void recent\_above\_6(){

        int num=1;

        try {

            CountryStacks();

        } catch (Exception e) {

            // TODO: handle exception

        }

        StackNode t=countries.get(0).head;

        StackNode r=new StackNode(t.year, t.country, t.magnitude);

        StackNode temp=r;

        for(int i=1;i<countries.size();i++){

            StackNode x=countries.get(i).head;

            temp.next=new StackNode(x.year, x.country, x.magnitude);

            temp=temp.next;

        }

        System.out.println("\n\tPrinting magnitudes from country stacks having magnitude above 6.\n");

        for(StackNode z=r;z!=null;z=z.next){

            if(z.magnitude>6 && z.next!=null){

                System.out.print(num+ " : ");

                z.display3();

                num++;

            }

        }

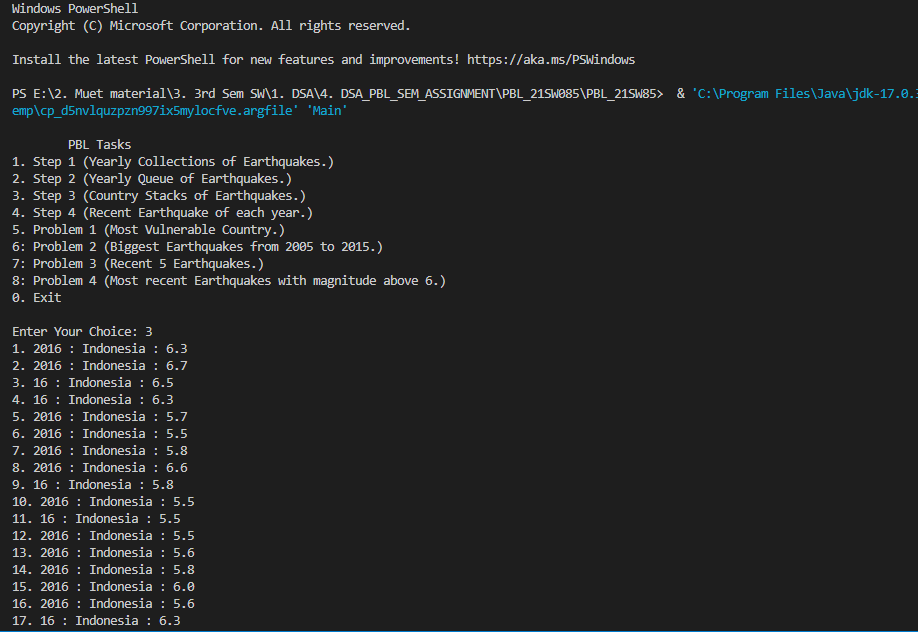
    }

**Algorithm of Step 3,4 & Problem 1,3,4.**

1. Created a class Stack Node with instance variables year, country, magnitude, prev, next, and top
2. Created constructors and methods for push1, display3, and other required functionalities.
3. Create a class Stacks with instance variables head, countries, and size.
4. Created constructors and methods for push2, peek, is Empty, and other required functionalities.
5. Created a method Country Stacks to read a CSV file and store earthquakes country-wise in an array list of stacks.
6. In Country Stacks method, initialize the required variables and read the file using Buffered Reader and File Reader
7. Read the file line by line using a while loop and split each line by comma separator.
8. Store the magnitude and year as a float and integer respectively
9. Extract the country from the line and check if the country is already in the list of countries.
10. If the country is present, add the earthquake details to the that particular stack
11. If the country is not present, create a new stack and add the earthquake details.
12. Created a method recent stacks to print all earthquakes country-wise, with the most recent earthquake printed first
13. In recent stacks method, call Country Stacks method to initialize the countries array list of stacks
14. Initialize a new stack node as the head of the first stack and create a temporary stack node temp to traverse through the list.
15. Print the earthquakes from the new list using display3 method.
16. Create a method display\_Yearly\_Stacks to print all earthquakes year-wise, with the most recent earthquake printed first
17. In display\_Yearly\_Stacks method, call CountryStacks method to initialize the countries array list of stacks
18. Print the earthquakes from the new list in yearly order using display3 method
19. Create a method recent\_above\_6 to print all earthquakes with magnitude above 6 country-wise, with the most recent earthquake printed first
20. In recent\_above\_6 method, call CountryStacks method to initialize the countries array list of stacks
21. Traverse through the countries array list and add the earthquakes with magnitude above 6 to the new list
22. Print the earthquakes from the new list using display3 method.

**Outputs:**

**Step 3:**

****

**Step 4:**

**Text

Description automatically generated**

**Problem 1:**

**Text

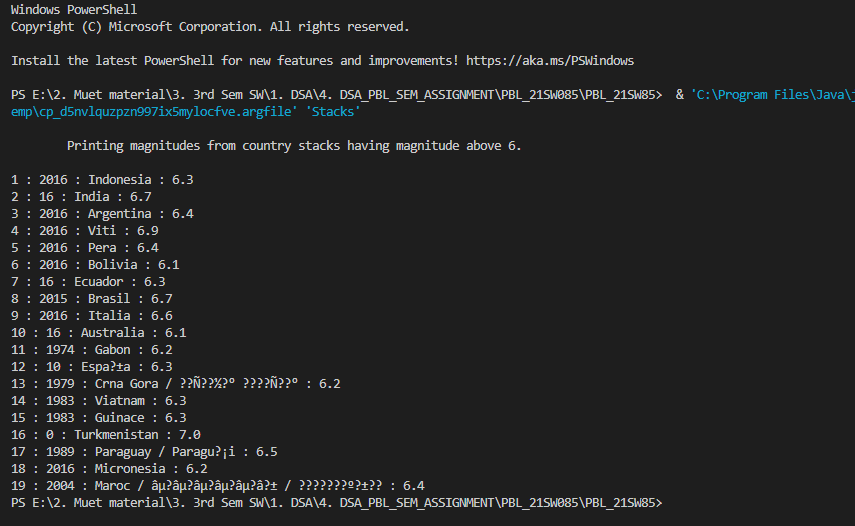
Description automatically generated**

**Problem 3:**

**Text

Description automatically generated**

**Problem 4:**

****

**The End!**