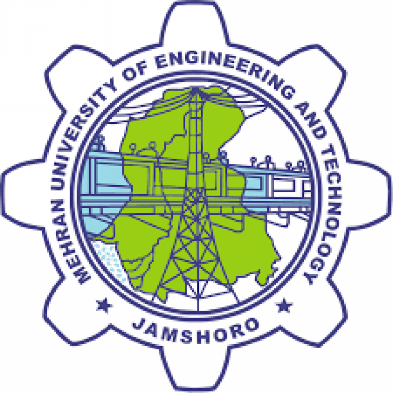
3/14/2023



**Problem-Based Assignment (21SW-III)**

**Of**

**Data Structure and Algorithms**

|  |  |
| --- | --- |
| **Student Name** | **Jawad Ali** |
| **Roll Number** | **21SW138** |
| **Section #** | **03** |

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 most 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).

**Algorithm**

The class has four constructors, which can be used to create an object of the class. The first constructor takes four parameters - country, city, magnitude, and date, and initializes the member variables accordingly. The second constructor takes three parameters - magnitude, date, and country, and initializes the member variables accordingly. The third constructor takes three parameters - type, magnitude, and date, and initializes the member variables accordingly. The fourth constructor takes no parameters and initializes the member variables to their default values.

The class also has five getter methods - "getCity()", "getCountry()", "getMagnitude()", "getType()", and "getDate()", which can be used to get the values of the member variables.

The class also has an overridden "toString()" method, which returns a string representation of the earthquake object. It returns a string containing the date, magnitude, and country of the earthquake

Overall, the "EarthQuake" class provides a simple model for representing earthquakes and their properties.

*public class* EarthQuake {  
 *private* String country,city,type,date;  
 *private double* magnitude;  
  
 *public* EarthQuake(String country,String city,*double* magnitude,String date){  
 *this*.country=country;  
 *this*.city=city;  
 *this*.magnitude=magnitude;  
 *this*.date=date;  
 }  
 *public* EarthQuake(*double* magnitude,String date,String country){  
 *this*.country=country;  
 *this*.date=date;  
 *this*.magnitude=magnitude;  
  
 }  
 *public* EarthQuake(String type, *double* magnitude,String date){  
 *this*.date=date;  
 *this*.type=type;  
 *this*.magnitude=magnitude;  
 }  
 *public* EarthQuake(){}  
  
 *public* String getCity() {  
 *return* city;  
 }  
  
 *public* String getCountry() {  
 *return* country;  
 }  
  
 *public double* getMagnitude() {  
 *return* magnitude;  
 }  
  
 *public* String getType() {  
 *return* type;  
 }  
  
 *public* String getDate() {  
 *return* date;  
 }  
  
 @Override  
 *public* String toString(){  
*// return "Type: "+type+"\tMagnitude: "+magnitude+"\tDate: "+date;  
 return* "Date: "+date+" , Magnitude: "+magnitude+" , Country: "+country;  
 }

}

**Algorithm**

The "MyLinkedList" class has several methods, including:

"insert(EarthQuake data)": This method inserts a new node with the given EarthQuake object at the beginning of the linked list. If the list is empty, it sets the new node as the head.

"getHead()": This method returns the head node of the linked list.

"traverse()": This method traverses the linked list and prints the data of each node.

"delete()": This method removes and returns the head node of the linked list. If the list is empty, it throws an IllegalStateException.

"copy()": This method creates a new instance of "MyLinkedList", copies all nodes from the original list to the new list, and returns the new list.

"reverse()": This method reverses the order of the nodes in the linked list by creating an array to store the data of each node, and then re-inserting the nodes in reverse order.

"getMax()": This method traverses the linked list to find the node with the highest magnitude of EarthQuake object, and returns the data of that node.

"getSize()": This method returns the size of the linked list.

*public class* MyLinkedList {  
 *private static class* Node{  
 EarthQuake data;  
 Node next;  
 *public* Node(EarthQuake data){  
 *this*.data=data;  
 }  
 }  
 *private int* size;  
 *private* Node head;  
 *public void* insert(EarthQuake data){  
 *if*(head==*null*){  
 head=*new* Node(data);  
 size++;  
 *return*;  
 }  
 Node n=*new* Node(data);  
 n.next=head;  
 head=n;  
 size++;  
 }  
 *public* Node getHead(){*return* head;}  
 *public void* traverse(){  
 *for*(Node n=head;n!=*null*;n=n.next) System.***out***.println(n.data); }  
 *public* EarthQuake delete(){  
 *if*(head==*null*)*throw new* IllegalStateException("List is exception");  
 Node n=head;  
 head=head.next;  
 size--;  
 *return* n.data;  
  
 }  
 *public* MyLinkedList copy(){  
 MyLinkedList list=*new* MyLinkedList();  
 *for*(Node n=head;n!=*null*;n=n.next)list.insert(n.data);  
 *return* list;  
 }  
 *public void* reverse(){  
 *//creating array to store data of linked list* EarthQuake[] quakes =*new* EarthQuake[*this*.getSize()];  
 *int* i=0;  
 *for*(Node n=head;n!=*null*;n=n.next)quakes[i++]=n.data;  
 *//now inserting values into linked list  
 for*(*int* k=quakes.length-1;k>=0;k--)*this*.insert(quakes[k]);  
 }  
 *public* EarthQuake getMax(){  
 Node max=head;  
 *for*(Node n=head.next;n!=*null*;n=n.next)*if*(max.data.getMagnitude()<n.data.getMagnitude())max=n;  
 *return* max.data;  
 }  
  
 *public int* getSize() {  
 *return* size;  
  
 }  
  
}

**Algorithm**

The MyLinkedQueue class provides the following methods:

add(EarthQuake data): adds a new EarthQuake object to the end of the queue. This is done by creating a new Node object and inserting it before the head node.

remove(): removes and returns the first EarthQuake object in the queue. This is done by setting the next reference of the head node to the second node in the list, and returning the EarthQuake object stored in the first node.

first(): returns the first EarthQuake object in the queue, without removing it. This is done by returning the EarthQuake object stored in the second node in the list.

size(): returns the number of elements in the queue.

print(): prints the elements in the queue, starting from the first element.

copy(): creates a copy of the queue and returns it as a new MyLinkedQueue object.

isEmpty(): returns true if the queue is empty, false otherwise.

The InvalidOpenTypeException is thrown if a method is called on an empty queue

*import* javax.management.openmbean.InvalidOpenTypeException;  
  
*public class* MyLinkedQueue {  
 *private static class* Node{  
 EarthQuake data;  
 *private* Node next=*this*;  
 *private* Node pre=*this*;  
 *public* Node(EarthQuake data){*this*.data=data;}  
 *public* Node(EarthQuake data, Node pre, Node next){  
 *this*.data=data;  
 *this*.pre=pre;  
 *this*.next=next;  
 }  
 }  
 *private int* size=0;  
 *private final* Node head=*new* Node(*null*);  
 *public void* add(EarthQuake data) {  
 head.pre.next=*new* Node(data,head.pre,head);  
 head.pre=head.pre.next;  
 size++;  
 }  
  
 *public* EarthQuake remove() {  
 *if*(isEmpty())*throw new* InvalidOpenTypeException("Queue is empty!");  
 EarthQuake temp=head.next.data;  
 head.next=head.next.next;  
 size--;  
 *return* temp;  
 }  
 *public* EarthQuake first() {  
 *if*(isEmpty())*throw new* InvalidOpenTypeException("Queue is empty!");  
 *return* head.next.data;  
 }  
  
 *public int* size() {  
 *return* size;  
 }  
 *public void* print(){  
 *for*(Node n=head.next;n!=head;n=n.next) System.***out***.println(n.data);  
 }  
 *public* MyLinkedQueue copy(){  
 MyLinkedQueue queue=*new* MyLinkedQueue();  
 *for*(Node n=head.next;n!=head;n=n.next)queue.add(n.data);  
 *return* queue;  
 }  
 *public boolean* isEmpty(){  
 *return* size==0;  
 }  
}

**Algorithm**

MyLinkedStack is an implementation of a stack using a singly linked list. It has basic operations such as push(), pop(), peek(), getSize(), search(), copy(), display(), and isEmpty(). The copy() method creates a new stack that contains the same elements as the original stack.

*import* java.util.EmptyStackException;  
  
*public class* MyLinkedStack{  
 *private final* String stackName;  
 *public* MyLinkedStack(String stackName){  
 *this*.stackName=stackName;  
 }  
  
 *public* String getStackName() {  
 *return* stackName;  
 }  
  
 *private static class* Node{  
 EarthQuake data;  
 Node next;  
 Node(EarthQuake data){  
 *this*.data=data;  
 }  
 }  
 *private* Node head=*null*;  
 *private int* size=0;  
 *public boolean* isEmpty(){  
 *return* head==*null*;  
 }  
 *public int* getSize(){  
 *return* size;  
 }  
 *public int* search(EarthQuake target){  
 *int* count=1;  
 *for*(Node n=head;n!=*null*;n=n.next){  
 *if*(n.data.equals(target))*return* count;  
 count++;  
 }  
 *return* -1;  
 }  
 *public void* push(EarthQuake t){  
 Node node= *new* Node(t);  
 node.next=head;  
 head=node;  
 size++;  
 }  
 *public* EarthQuake peek(){  
 *if*(isEmpty())*throw new* EmptyStackException();  
 *return* head.data;}  
 *public* EarthQuake pop(){  
 EarthQuake t=head.data;  
 head=head.next;  
 --size;  
 *return* t;  
 }  
 *public* MyLinkedStack copy(){  
 MyLinkedStack stack=*new* MyLinkedStack(*this*.getStackName());  
 EarthQuake[] a=*new* EarthQuake[*this*.size];  
 *int* count=0;  
 *for*(Node n=head;n!=*null*;n=n.next)a[count++]=n.data;  
 *for*(*int* i=a.length-1;i>=0;i--)stack.push(a[i]);  
 *return* stack;  
 }  
 *public void* display(){  
 *for*(Node n=head;n!=*null*;n=n.next) System.***out***.println(n.data);  
 }  
}

**Algorithm**

This is a Java program that reads earthquake data from a CSV file, organizes the data into different data structures, and performs various operations on the data. The program is divided into several methods that perform specific tasks such as fetching data from a file, inserting data into collections, and setting various attributes of the earthquake data;

The program uses several data structures such as arrays, linked lists, stacks, and queues to organize and process the earthquake data. The program reads earthquake data from a CSV file that contains information about the location, magnitude, and date of earthquakes that occurred in different parts of the world. The program then organizes this data into different collections such as arrays and linked lists based on various attributes such as the year in which the earthquake occurred.

The program also performs various operations on the earthquake data such as finding the maximum magnitude earthquake in each year, adding the maximum magnitude earthquake from each year to a queue, and creating a list of recent earthquakes sorted by country. These operations involve iterating over the different collections of earthquake data and performing various computations such as comparing dates and magnitudes of earthquakes.

*import* java.io.\*;  
*import* java.util.Scanner;  
  
  
*public class* Main {  
 *private final double*[] magnitude=*new double*[1104];  
  *private final* String[] city=*new* String[1104];  
 *private final int*[] year=*new int*[1104];  
  *private final* String[] date=*new* String[1104];  
 *private final* String[] earthQuackType=*new* String[1104];  
  *private final* String[] totalDifferentCountries=*new* String[52];  
 */   
 private final* MyLinkedList[] yearWiseLL=*new* MyLinkedList[52];  
  *private final* MyLinkedQueue earthQuackQueue=*new* MyLinkedQueue();  
  *private final* MyLinkedList recentCountryWiseQuacksList=*new* MyLinkedList();  
  *private final* MyLinkedStack[] countryStack=*new* MyLinkedStack[52];  
  *public* Main(){  
 fetchingDataFromCSVFile();  
 insertCountry\_City();  
 insertIntoCollections();  
 insertIntoQueue();  
 setTotalDifferentCountries();  
  *for*(*int* i=0;i<countryStack.length;i++)countryStack[i]=*new* MyLinkedStack(totalDifferentCountries[i]);  
 *for* (*int* i=0;i< country.length;i++)  
 *for* (MyLinkedStack myLinkedStack : countryStack)  
 *if* (country[i].equals(myLinkedStack.getStackName()))  
 myLinkedStack.push(*new* EarthQuake(earthQuackType[i], magnitude[i], date[i]));  
 setRecentCountryWiseQuacksList();  
 }  
 *public void* fetchingDataFromCSVFile(){  
 File file=*new* File("Valid countries and cities.csv");  
 *try* {  
 *int* i=0;  
Scanner scan=*new* Scanner(file);  
 scan.nextLine();  
 *while* (scan.hasNextLine()){  
 String[] s=scan.nextLine().split(",");  
  *char*[] s1=s[2].toCharArray();  
 StringBuilder sdb=*new* StringBuilder();  
 *for*(*int* j=s1.length-4;j<s1.length;j++)sdb.append(s1[j]);  
year[i]= Integer.*parseInt*(sdb+"");  
 date[i]=s[2];  
 magnitude[i]=Double.*parseDouble*(s[10]);  
 earthQuackType[i]=s[6];  
 i++;  
 }  
 }*catch* (FileNotFoundException e){  
 System.***out***.println(e.getMessage());  
 }  
 }  
 *public void* insertCountry\_City(){  
 File f=*new* File("County.txt");  
 File f1=*new* File("Countries\_Cities.txt");  
 *try* {  
  
 Scanner scan = *new* Scanner(f);  
 Scanner scanner=*new* Scanner(f1);  
 *int* i=0;  
 *while* (scan.hasNextLine()){  
 String s=scan.nextLine();  
 String[] s1=scanner.nextLine().split(":");  
 country[i]=s;  
 city[i++]=s1[1];  
 }  
  
 } *catch* (FileNotFoundException e) {  
 *throw new* RuntimeException(e);  
 }  
  
 }  
 *public void* insertIntoCollections() {  
 *int* i=0;  
 *int* j=1;  
 *for*(*int* k=0;k<52;k++) {  
 yearWiseLL[k]=*new* MyLinkedList();  
 *while* (year[i] == year[j]) {  
 yearWiseLL[k].insert(*new* EarthQuake(country[i],city[i],magnitude[i],date[i]));  
 i++;  
 j++;  
 *if*(j==1104)*break*;  
 }  
 yearWiseLL[k].insert(*new* EarthQuake(country[i],city[i],magnitude[i],date[i]));  
 *if*(i==1104)*break*;  
 i=j;  
 j++;  
 }  
 }  
 *public void* insertIntoQueue(){  
 *for* (MyLinkedList myLinkedList : yearWiseLL) earthQuackQueue.add(myLinkedList.getMax());  
 }  
 *public void* setTotalDifferentCountries(){  
 *int* count=0;  
 label: *for*(*int* i=0;i<country.length;i++){  
 String cou=country[i];  
 *for*(*int* j=i-1;j>=0;j--)*if*(cou.equals(country[j]))*continue* label;  
 totalDifferentCountries[count++]=cou;  
 }  
 }  
 *public void* setRecentCountryWiseQuacksList( ) {  
 MyLinkedStack[] arr=countryStack;  
 *for*(*int* i=0;i<arr.length;i++){  
 *for*(*int* j=0;j<arr.length-1-i;j++){  
 String[] s=arr[j].peek().getDate().split("/");  
 *int* day1=Integer.*parseInt*(s[1]);  
 *int* month1=Integer.*parseInt*(s[0]);  
 *int* year1=Integer.*parseInt*(s[2]);  
 s=arr[j+1].peek().getDate().split("/");  
 *int* day2=Integer.*parseInt*(s[1]);  
 *int* month2=Integer.*parseInt*(s[0]);  
 *int* year2=Integer.*parseInt*(s[2]);  
 *if*(year2!=year1){  
 *if*(year1>year2){  
 MyLinkedStack temp=arr[j];  
 arr[j]=arr[j+1];  
 arr[j+1]=temp;  
 }  
 }  
 *else if*(month2!=month1){  
 *if*(month1>month2){  
 MyLinkedStack temp=arr[j];  
 arr[j]=arr[j+1];  
 arr[j+1]=temp;  
 }  
 }  
 *else* {  
 *if*(day1>day2){  
 MyLinkedStack temp=arr[j];  
 arr[j]=arr[j+1];  
 arr[j+1]=temp;  
 }  
 }  
  
 }  
  
 }  
 *for*(MyLinkedStack s:arr)  
 recentCountryWiseQuacksList.insert(*new* EarthQuake(s.peek().getMagnitude(),s.peek().getDate(),s.getStackName()));  
 }  
  
 *public double*[] getMagnitude() {  
 *return* magnitude;  
 }  
  
 *public* String[] getCountry() {  
 *return* country;  
 }  
  
 *public* String[] getCity() {  
 *return* city;  
 }  
  
 *public int*[] getYear() {  
 *return* year;  
 }  
  
 *public* String[] getDate() {  
 *return* date;  
 }  
  
 *public* String[] getEarthQuackType() {  
 *return* earthQuackType;  
 }  
  
 *public* String[] getTotalDifferentCountries() {  
 *return* totalDifferentCountries;  
 }  
  
 *public* MyLinkedList[] getYearWiseLL() {  
 *return* yearWiseLL;  
 }  
  
 *public* MyLinkedQueue getEarthQuackQueue() {  
 *return* earthQuackQueue;  
 }  
  
 *public* MyLinkedList getRecentCountryWiseQuacksList() {  
 *return* recentCountryWiseQuacksList;  
 }  
  
 *public* MyLinkedStack[] getCountryStack() {  
 *return* countryStack;  
 }  
  
}

*import* java.util.Scanner;  
  
*public class* Solution {  
 *private static* Object *countryName*;  
  
 *public* Solution() {  
 *new* Main();  
 }  
 *public* String[][] question1() {  
 MyLinkedStack[] countries = *new* Main().getCountryStack();  
 String[][] quakes = *new* String[52][2];  
 *int* index = 0;  
 *double* maxAverage = 0;String country = ""; *int* countYears = 0; *int* count = 0; *int* counter = 1965; *int* a = 1; *for* (*int* i = 1965; i <= 2016; i++) {  
MyLinkedStack stack = countries[count++].copy();  
  *int* x = stack.getSize();  
 *for* (*int* y = 0; y < x; y++) {  
String[] date = stack.pop().getDate().split("/");  
  *int* year = Integer.*parseInt*(date[2]);  
 *if* (a == 2) {  
 *if* (year != counter) {  
 countYears++;  
 counter = year;  
 }  
 } *else* {  
 counter = year;  
 countYears++;  
 a++;  
 }  
 }  
 *double* average = (*double*) x / countYears;  
  *if* (average > maxAverage) {  
 maxAverage = average;  
 country = stack.getStackName();  
 }  
 quakes[index][0] = "Name: " + stack.getStackName();  
 quakes[index++][1] = "Average: " + average;  
 countYears = 0;  
 a = 1;  
 }  
 System.***out***.println("Country having maximum average of earth quack in each year is " + country + " and maximum average is: " + maxAverage);  
 *return* quakes;  
 }  
  
 *public* MyLinkedQueue question2() {  
 MyLinkedQueue queue = *new* Main().getEarthQuackQueue().copy();  
 MyLinkedQueue queue1 = *new* MyLinkedQueue();  
 *for* (*int* i = 1965; i < 2016; i++) {  
 *if* (i >= 2005) queue1.add(queue.remove());  
 *else* queue.remove();  
 }  
 *return* queue1;  
 }  
 *public* MyLinkedStack question3(String countryName) {  
 MyLinkedStack stack = *new* MyLinkedStack(countryName);  
  *for* (MyLinkedStack s : *new* Main().getCountryStack()) {  
 *if* (countryName.equals(s.getStackName())) {  
 stack = s.copy();  
 *break*;  
 }  
 }  
 MyLinkedStack stack1 = *new* MyLinkedStack(countryName);  
EarthQuake[] quakes = *new* EarthQuake[5];  
 *for* (*int* i = 0; i < 5; i++) quakes[i] = stack.pop();  
 *for* (*int* j = quakes.length - 1; j >= 0; j--) stack1.push(quakes[j]);  
 *return* stack1;  
 }  
  
 *public* MyLinkedList question4() {  
 MyLinkedList list = *new* Main().getRecentCountryWiseQuacksList().copy();  
 MyLinkedList list1 = *new* MyLinkedList();  
 *int* size = list.getSize();  
 *for* (*int* i = 0; i < size; i++) {  
 EarthQuake quake = list.delete();  
 *if* (quake.getMagnitude() >= 6) list1.insert(quake);  
 }  
 list1.reverse();  
 *return* list1;  
 }  
  
 *public static void* main(String[] args) {  
 Solution s = *new* Solution();  
  
  
*// String[][] list = s.question1();  
// while (list.length == -1) {  
// System.out.println(list);  
// }  
  
// MyLinkedQueue list2 = s.question2();  
// int k = list2.size();  
// for (int i = 0; i < k; i++) System.out.println(list2.remove());  
  
  
// MyLinkedStack stack = s.question3("Japan");  
// int z = stack.getSize();  
// for (int i = 0; i < z; i++) System.out.println(stack.pop());* MyLinkedList list = s.question4();  
 *int* j = list.getSize();  
 *for* (*int* i = 0; i < j; i++)  
 System.***out***.println(list.delete());  
}   
}

Output

Q:1

Country having maximum average of earth quack in each year is Haiti and maximum average is: 7.0  
  
 Process finished with exit code 0

Q:2

Date: 11/17/2005 , Magnitude: 6.8 , Country: Bolivia  
Date: 11/17/2006 , Magnitude: 6.2 , Country: Japan  
Date: 10/15/2007 , Magnitude: 6.8 , Country: New Zealand  
Date: 6/13/2008 , Magnitude: 6.9 , Country: Japan  
Date: 8/28/2009 , Magnitude: 6.3 , Country: China  
Date: 1/12/2010 , Magnitude: 7.0 , Country: Haiti  
Date: 4/7/2011 , Magnitude: 7.1 , Country: Japan  
Date: 10/28/2012 , Magnitude: 7.8 , Country: Canada  
Date: 4/16/2013 , Magnitude: 7.7 , Country: Iran  
Date: 8/18/2014 , Magnitude: 6.2 , Country: Iran  
Date: 4/25/2015 , Magnitude: 7.8 , Country: Nepal  
  
Process finished with exit code 0

Q:3

Date: 11/11/2016 , Magnitude: 6.1 , Country: *null* Date: 10/21/2016 , Magnitude: 6.2 , Country: *null* Date: 10/4/2016 , Magnitude: 5.7 , Country: *null* Date: 4/18/2016 , Magnitude: 5.5 , Country: *null* Date: 4/15/2016 , Magnitude: 5.5 , Country: *null* Process finished with exit code 0

Q:4

Date: 12/18/2016 , Magnitude: 6.2 , Country: Micronesia  
Date: 11/11/2016 , Magnitude: 6.1 , Country: Japan  
Date: 11/4/2016 , Magnitude: 6.3 , Country: Chile  
Date: 9/10/2016 , Magnitude: 6.1 , Country: Peru  
Date: 4/20/2016 , Magnitude: 6.0 , Country: Ecuador  
Date: 2/5/2016 , Magnitude: 6.4 , Country: Taiwan  
Date: 1/14/2016 , Magnitude: 6.1 , Country: Bolivia  
Date: 11/17/2015 , Magnitude: 6.5 , Country: Greece  
Date: 7/8/2014 , Magnitude: 6.2 , Country: Vanuatu  
Date: 3/25/2012 , Magnitude: 7.1 , Country: Chilean  
Date: 4/11/2010 , Magnitude: 6.3 , Country: Spain  
Date: 11/30/1999 , Magnitude: 6.6 , Country: Chili  
Date: 6/11/1986 , Magnitude: 6.3 , Country: Venezuela  
Date: 10/29/1985 , Magnitude: 6.4 , Country: Papua Niugini  
Date: 12/18/2016 , Magnitude: 6.2 , Country: Micronesia  
Date: 11/11/2016 , Magnitude: 6.1 , Country: Japan  
Date: 11/4/2016 , Magnitude: 6.3 , Country: Chile  
Date: 9/10/2016 , Magnitude: 6.1 , Country: Peru  
Date: 4/20/2016 , Magnitude: 6.0 , Country: Ecuador  
Date: 2/5/2016 , Magnitude: 6.4 , Country: Taiwan  
Date: 1/14/2016 , Magnitude: 6.1 , Country: Bolivia  
Date: 11/17/2015 , Magnitude: 6.5 , Country: Greece  
Date: 7/8/2014 , Magnitude: 6.2 , Country: Vanuatu  
Date: 3/25/2012 , Magnitude: 7.1 , Country: Chilean  
Date: 4/11/2010 , Magnitude: 6.3 , Country: Spain  
Date: 11/30/1999 , Magnitude: 6.6 , Country: Chili  
Date: 6/11/1986 , Magnitude: 6.3 , Country: Venezuela  
Date: 10/29/1985 , Magnitude: 6.4 , Country: Papua Niugini  
  
Process finished with exit code 0