

**Mehran University of Engineering &** T**echnology, Jamshoro, Sindh**

Problem-Based Assignment (21SW-III)

Of

Data Structure and Algorithms

|  |  |
| --- | --- |
| Name | Abdul Lahad |
| Roll No | 21SW007 |
| Section | III or 3rd |

Problem

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).

CODE

Class Main :

import java.util.LinkedList;

import java.util.Scanner;

public class Main {

    // collection along with magnitude.

    // (Collection of each year means 52 collections)

    LinkedList<Collections> col = new LinkedList<Collections>(); // step 1

    // queue storing biggest (with highest magnitude)

    // quake of each year with magnitude and

    // country, starting from 1965 to 2016.

    Queue q = new Queue(); // step 2

    // 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).

    LinkedList<LinkedStack> scol = new LinkedList<LinkedStack>(); // step 3

    // linked list which saves the one most recent

    // earthquake with magnitude and country name

    // from each country

    Linked\_list ll; // step 4

    Scanner sc = new Scanner(System.in);

    list date, time, country, city, stat, mag = new list(), lat, lon;

    Database d = new Database();

    Location lo = new Location();

    public static void main(String[] args) {

        Main m = new Main();

        m.step\_1(); // making yearly collection of earthquake

        m.step\_2(); // Make a queue storing biggest quake of each year

        m.step\_3(); // Make a stack, one for each country

        m.step\_4(); // Make a linked list which saves the one most

                    // recent earthquake from each country

        int num = 0;

        do {

            num = m.me();

            if (num == 2) {

                int n;

                do {

                    n = m.problems();

                    switch (n) {

                        case 1: {

                            m.problem\_1();

                            break;

                        }

                        case 2: {

                            m.problem\_2();

                            break;

                        }

                        case 3: {

                            m.problem\_3();

                            break;

                        }

                        case 4: {

                            m.problem\_4();

                            break;

                        }

                        case 0: {

                            break;

                        }

                        default:

                            System.out.println("\ninvalid input ;");

                    }

                } while (n != 0);

            } else if (num == 1) {

                int n;

                do {

                    n = m.steps();

                    switch (n) {

                        case 1: {

                            m.s1();

                            break;

                        }

                        case 2: {

                            m.s2();

                            break;

                        }

                        case 3: {

                            m.s3();

                            break;

                        }

                        case 4: {

                            m.s4();

                            break;

                        }

                        case 0: {

                            break;

                        }

                        default:

                            System.out.println("\ninvalid input ;");

                    }

                } while (n != 0);

            }

        } while (num != 0);

    }

    public void s1() {

        int pos = -1;

        while (pos != 0) {

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

            pos = sc.nextInt();

            pos -= 1965;

            Collections c = col.get(pos);

            System.out.println(c);

            System.out.print("press 0 to exit step\_1 or 1 to continue : ");

            pos = sc.nextInt();

        }

    }

    public void s2() {

        System.out.println(q);

    }

    public void s3() {

        int pos = -1;

        while (pos != 0) {

            System.out.print("Enter number between 0 to " + (scol.size() - 1) + " : ");

            pos = sc.nextInt();

            LinkedStack s = scol.get(pos);

            System.out.println("Country : " + s.Country\_name);

            System.out.println(s);

            System.out.print("press 0 to exit step\_3 or 1 to continue : ");

            pos = sc.nextInt();

        }

    }

    public void s4() {

        System.out.println(ll);

    }

    public int problems() {

        System.out.println("\n");

        System.out.println(

                "1 > problem 1 (average number of earthquakes per year for each country and which country is most vulnerable to earthquakes)");

        System.out.println("2 > problem 2 (biggest earthquakes from 2005 to 2015 )");

        System.out.println("3 > problem 3 (recent 5 earthquakes from each country)");

        System.out.println("4 > problem 4 (recent above 6 magnitude earthquakes)");

        System.out.println("0 > Go back");

        System.out.print("\nEnter your choice : ");

        int num = sc.nextInt();

        System.out.println("\n");

        return num;

    }

    public int me() {

        System.out.println("\n");

        System.out.println("1 > data structurs");

        System.out.println("2 > problems");

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

        System.out.print("\nEnter your choice : ");

        int num = sc.nextInt();

        System.out.println("\n");

        return num;

    }

    public int steps() {

        System.out.println("\n");

        System.out.println("1 > step 1 [yearly earthquake collection]");

        System.out.println("2 > step 2 [queue storing biggest (with highest magnitude) quake of each year]");

        System.out.println("3 > step 3 [stack , one for each country (the most recent event on top)]");

        System.out.println(

                "4 > step 4 [linked list, which saves the one most recent earthquake with magnitude and country name from each country]");

        System.out.println("0 > Go back");

        System.out.print("\nEnter your choice : ");

        int num = sc.nextInt();

        System.out.println("\n");

        return num;

    }

    public void step\_1() {

        // ------making collections yearly--------

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

            col.add(new Collections(i)); // initilizing objects yearly

        }

        // fatching data from database

        date = d.column\_return(0, "Writesheet");

        time = d.column\_return(1, "Writesheet");

        country = d.column\_return(2, "Writesheet");

        city = d.column\_return(4, "Writesheet");

        mag = d.column\_return(5, "Writesheet");

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

            Collections c = col.get(i); // get objects from collection

            c.yearly\_insert(date, time, country, city, mag);

            col.set(i, c); // setting object again in collection

        }

        // ------ending of collections yearly--------

    }

    public void step\_2() {

        q.highest\_magnitudQueue(col);

    }

    public void step\_3() {

        country = d.column\_return(2, "Writesheet");

        for (node temp = country.head; temp != null; temp = temp.next) {

            String s = String.valueOf(temp.data);

            boolean boool = true;

            if (scol.size() > 0) {

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

                    LinkedStack l = scol.get(i);

                    if (l.Country\_name.equals(s)) {

                        boool = false;

                        break;

                    }

                }

            }

            if (boool) {

                LinkedStack st = new LinkedStack(s);

                scol.addFirst(st);

            }

        }

        System.out.println("completed country insertion into collection");

        // collection of country is ready

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

            LinkedStack ls = scol.get(i);

            // populating the stack

            ls.country\_stack(col);

            // seting stack again to collection

            scol.set(i, ls);

        }

        scol.remove(scol.size() - 1);

    }

    public void step\_4() {

        LinkedStack d = scol.getFirst();

        Object[] oo = d.peek();

        ll = new Linked\_list(d.Country\_name, oo[0], oo[1]);

        Linked\_list t = ll;

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

            d = scol.get(i);

            oo = d.peek();

            t.next = new Linked\_list(d.Country\_name, oo[0], oo[1]);

            t = t.next;

        }

    }

    public void problem\_1() {

        double highest\_avr = 0.0;

        int j = 0, t = 0;

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

            LinkedStack lStack = scol.get(i);

            double avr = lStack.size();

            avr /= 52;

            System.out.println("Country : " + lStack.Country\_name + " \t| TOTAL NO. of magnitudes :" + lStack.size()

                    + " \t| AVERAGE NO. of magnitude : " + avr);

            t += lStack.size();

            if (highest\_avr < avr) {

                highest\_avr = avr;

                j = i;

            }

        }

        System.out.println("TOTAL NUMBER OF MAGNITUD OF ALL COUNTRYS : " + t);

        System.out.println("\nMost vulnerable country to earthquakes : ");

        System.out.print(scol.get(j).Country\_name + "\t| TOTAL NO. of magnitudes :" + scol.get(j).size()

                + "\t| HIGHEST AVERAGE Magnitude : " + highest\_avr);

    }

    public void problem\_2() {

        q.high\_Quake\_2005\_to\_2015();

    }

    public void problem\_3() {

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

            LinkedStack lStack = scol.get(i);

            lStack.most\_5\_recent();

        }

        System.out.println("\nThe total number of country : " + scol.size());

    }

    public void problem\_4() {

        Linked\_list tempList = ll;

        while (tempList != null) {

            if ((Double) tempList.magnitude > 6) {

                System.out.print("| COUNTRY : " + tempList.country\_name);

                System.out.print(" | DATE : " + tempList.date);

                System.out.println(" | MAGNITUDE : " + tempList.magnitude + " |");

            }

            tempList = tempList.next;

        }

    }

}

Class Collections :

class detail\_collection {

    Object country;

    Object city;

    Object magnitude;

    Object date;

    Object time;

    Object stat;

    detail\_collection next;

    public detail\_collection(Object co, Object ci, Object m, Object d, Object t) {

        country = co;

        city = ci;

        magnitude = m;

        date = d;

        time = t;

    }

    public detail\_collection(Object m, Object d, Object t) {

        magnitude = m;

        date = d;

        time = t;

    }

    public detail\_collection(Object co, Object ci, Object m, Object d, Object t, detail\_collection n) {

        country = co;

        city = ci;

        magnitude = m;

        date = d;

        time = t;

        next = n;

    }

    public detail\_collection filter(){

        detail\_collection co=this;

        int counter=1,num=0;

        if(this.country.equals("NOT")){

            System.out.println("working head");

            co=co.next;

        }

        detail\_collection temp=co,n=co.next;

        while(n!=null){

            counter++;

            System.out.print("row : "+counter);

            String s=String.valueOf(n.country);

            if(s.equals("NOT")){

                    temp.next=n.next;

                    num++;

                    System.out.println("      change : "+num+" ; "+s);

                }

        }

        return co;

    }

}

public class Collections {

    int year;

    detail\_collection head;

    Collections(int year) {

        this.year = year;

    }

    void yearly\_insert(list date\_List, list time\_list, list country\_List, list city\_List, list magnitude\_List) {

        node t, t1, t2, t3, t4;

        t = date\_List.head.next;

        t1 = country\_List.head.next;

        t2 = city\_List.head.next;

        t3 = magnitude\_List.head.next;

        t4 = time\_list.head.next;

        while (t != null) {

            String y = String.valueOf(t.data);

            y = y.substring(6);

            int a = Integer.parseInt(y);

            if (a == year) {

                break;

            }

            t = t.next;

            t1 = t1.next;

            t2 = t2.next;

            t3 = t3.next;

            t4 = t4.next;

        }

        this.head = new detail\_collection(t1.data, t2.data, t3.data, t.data, t4.data);

        detail\_collection temp1 = this.head;

        t = t.next;

        t1 = t1.next;

        t2 = t2.next;

        t3 = t3.next;

        t4 = t4.next;

        while (t != null) {

            String y = String.valueOf(t.data);

            y = y.substring(6);

            int a = Integer.parseInt(y);

            if (a != year) {

                break;

            }

            temp1.next = new detail\_collection(t1.data, t2.data, t3.data, t.data, t4.data);

            temp1 = temp1.next;

            t = t.next;

            t1 = t1.next;

            t2 = t2.next;

            t3 = t3.next;

            t4 = t4.next;

        }

    }

    public String toString() {

        StringBuilder b = new StringBuilder("[");

        String S;

        detail\_collection X = head;

        b.append("{" + " DATE " + " ,   " + " TIME " +"     ,   "+ "COUNTRY " + "  , " + " CITY  " +"   ,   " + " MAGNITUDE "+ "}\n");

        while (X != null) {

            b.append("{" + X.date + " , " + X.time +" , "+ X.country + " , " + X.city +" , " + X.magnitude+ "}");

            if (X.next != null)

                b.append("\n");

            X = X.next;

        }

        b.append("]");

        S = b.toString();

        return S;

    }

}

Class Linked\_list :

import java.util.LinkedList;

public class Linked\_list {

     Object country\_name;

     Object magnitude;

     Object date;

     Linked\_list next;

     public Linked\_list(Object c, Object m, Object d) {

          country\_name = c;

          magnitude = m;

          date = d;

     }

     public Linked\_list(Object c, Object m, Object d, Linked\_list n) {

          country\_name = c;

          magnitude = m;

          date = d;

          next = n;

     }

     public void store\_recently\_quake(LinkedList<LinkedStack> l) {

          Linked\_list temp = this;

               LinkedStack ls = l.get(0);

               Object[] obj = ls.peek();

               temp=new Linked\_list(ls.Country\_name,obj[0], obj[1]);

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

                ls = l.get(i);

                obj = ls.peek();

                temp.next=new Linked\_list(ls.Country\_name,obj[0], obj[1]);

                temp=temp.next;

          }

     }

     public String toString() {

          StringBuffer b = new StringBuffer("[");

          for (Linked\_list temp = this; temp != null; temp = temp.next) {

              b.append("{" + temp.country\_name + " , " + temp.date + " , " + temp.magnitude + "}");

              if (temp.next != null) {

                  b.append("\n");

              }

          }

          b.append("]");

          String s = b.toString();

          return s;

      }

}

Class LinkedStack :

import java.util.LinkedList;

interface Stack {

    Object peek();

    Object pop();

    void push(Object obj1, Object obj2, Object obj3);

    int size();

}

public class LinkedStack implements Stack {

    public String Country\_name;

    private Node node;

    private int size;

    public LinkedStack(){

    }

    public LinkedStack(String c){

        Country\_name=c;

    }

    private class Node {

        Object magnitude;

        Object date;

        Object time;

        Node next;

        Node(Object mag, Object d, Object t, Node n) {

            magnitude = mag;

            date = d;

            time = t;

            next = n;

        }

    }

    @Override

    public Object[] peek() {

        // TODO Auto-generated method stub

        Object[] obj = { node.magnitude , node.date , node.time };

        return obj;

    }

    @Override

    public Object[] pop() {

        // TODO Auto-generated method stub

        Object[] o = { node.magnitude, node.date, node.time };

        node = node.next;

        --size;

        return o;

    }

    @Override

    public void push(Object magnitude, Object date, Object time) {

        // TODO Auto-generated method stub

        node = new Node(magnitude, date, time, node);

        size++;

    }

    @Override

    public int size() {

        // TODO Auto-generated method stub

        return size;

    }

    boolean isEmpty() {

        return (size == 0);

    }

    public String toString() {

        StringBuffer b = new StringBuffer("[");

        for (Node temp = node; temp != null; temp = temp.next) {

            b.append("{" + temp.magnitude + " , " + temp.date + " , " + temp.time + "}");

            if (temp.next != null) {

                b.append("\n");

            }

        }

        b.append("]");

        String s = b.toString();

        return s;

    }

    public void country\_stack(LinkedList<Collections> l) {

        // System.out.println("outer");

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

            // System.out.println(i+"loop in");

            Collections cl = l.get(i);

            detail\_collection temp = cl.head;

            while (temp != null) {

                // System.out.println(i+"while in ");

                String s = String.valueOf(temp.country);

                if (this.Country\_name.equals(s)) {

                    this.push(temp.magnitude, temp.date, temp.time);

                }

                temp=temp.next;

            }

            // System.out.println(i+"while out");

        }

        // System.out.println("loop out");

    }

    public void most\_5\_recent(){

        Node n=node;

        System.out.println("COUNTRY : " + Country\_name);

        for(int i=0 ; i<5;i++){

            System.out.print("| DATE "+n.date);

            System.out.println("| MAGNITUDE "+n.magnitude+" |");

            if(n.next==null)

              break;

             n=n.next;

        }

    }

}

Class Queue :

import java.util.LinkedList;

interface Queue\_bluePrint {

    public void add(Object a, Object b, Object c);

    public Object[] first();

    public Object[] remove();

    public int size();

}

public class Queue implements Queue\_bluePrint {

    private Node head = new Node(null, null, null);

    private int size;

    private class Node {

        Object year;

        Object country;

        Object magnitude;

        Node next = this, prev = this;

        public Node(Object obj, Object mag, Object y) {

            country = obj;

            magnitude = mag;

            year = y;

        }

        public Node(Object obj, Object mag, Object y, Node n, Node p) {

            country = obj;

            magnitude = mag;

            year = y;

            next = n;

            prev = p;

        }

    }

    public int Size() {

        return size;

    }

    public boolean isEpmty() {

        return (size == 0);

    }

    @Override

    public void add(Object country, Object magnitud, Object year) {

        // TODO Auto-generated method stub

        head.prev.next = new Node(country, magnitud, year, head, head.prev);

        head.prev = head.prev.next;

        ++size;

    }

    @Override

    public Object[] first() {

        // TODO Auto-generated method stub

        if (isEpmty()) {

            System.out.println("No elements ");

            return null;

        }

        Object[] obj = { head.next.country, head.next.magnitude, head.next.year };

        return obj;

    }

    @Override

    public Object[] remove() {

        // TODO Auto-generated method stub

        Object[] obj = { head.next.country, head.next.magnitude, head.next.year };

        head.next = head.next.next;

        head.next.prev = head;

        return obj;

    }

    @Override

    public int size() {

        // TODO Auto-generated method stub

        return size;

    }

    public String toString(){

        StringBuffer b = new StringBuffer("");

        for (Node temp = head.next; temp != head; temp = temp.next) {

            b.append("[ Country : " + temp.country + " | Magnitude : " + temp.magnitude + " | Year : " + temp.year + " ] ");

            if (temp.next != head)

                b.append("\n");

        }

        String s = b.toString();

        return s;

    }

    public void highest\_magnitudQueue(LinkedList<Collections> l) {

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

            Collections cl = l.get(i);

            detail\_collection high\_node, temp = cl.head;

            high\_node=cl.head;

            double high\_mag = 0.0;

            while (temp != null) {

                double mag = (Double) temp.magnitude;

                if (high\_mag < mag) {

                    high\_mag = mag;

                    high\_node = temp;

                }

                temp = temp.next;

            }

            this.add(high\_node.country, high\_node.magnitude, cl.year);

        }

    }

    public void high\_Quake\_2005\_to\_2015() {

        for (Node temp = this.head.next; temp != this.head; temp = temp.next) {

            int num = Integer.parseInt(String.valueOf(temp.year));

            if (num >= 2005 && num <= 2015) {

                System.out.print("Year : " + temp.year);

                System.out.print("| Country : " + temp.country);

                System.out.println("| highest Magnitude of Earthquake : " + temp.magnitude);

            }

        }

    }

}

Class List :

class node {

    Object data;

    node next;

    node(Object data)

    {

        this.data = data;

        this.next = null;

    }

}

class list {

    node head;

    private node tail;

    private int length = 0;

    list() { this.head = null; }

    Object first(){

        return head.data;

    }

    void add(Object data)

    {

        node temp = new node(data);

        if (this.head == null) {

        tail = head = temp;

        }

        else {

            tail.next=temp;

            tail=tail.next;

        }

        length++;

    }

    void add(int position, Object data)

    {

        if (position > length + 1) {

            System.out.println(

                "Position Unavailable in LinkedList");

            return;

        }

        if (position == 1) {

            node temp = head;

            head = new node(data);

            head.next = temp;

            return;

        }

        node temp = head;

        node prev = new node(null);

        while (position - 1 > 0) {

            prev = temp;

            temp = temp.next;

            // decreasing position counter

            position--;

        }

        // previous node now points to new value

        prev.next = new node(data);

        // new value now points to former current node

        prev.next.next = temp;

    }

    void remove(Object key)

    {

        node prev = new node(null);

        prev.next = head;

        node next = head.next;

        node temp = head;

        boolean exists = false;

        if (head.data.equals(key)) {

            head = head.next;

            exists = true;

        }

        while (temp.next != null) {

            if (String.valueOf(temp.data).equals(

                    String.valueOf(key))) {

                prev.next = next;

                exists = true;

                break;

            }

            prev = temp;

            temp = temp.next;

            next = temp.next;

        }

        if (exists == false

            && String.valueOf(temp.data).equals(

                String.valueOf(key))) {

            prev.next = null;

            exists = true;

        }

        if (exists) {

            length--;

        }

        else {

            System.out.println(

                "Given Value is not present in linked list");

        }

    }

    void clear()

    {

        head = null;

        length = 0;

    }

    boolean empty()

    {

        if (head == null) {

            return true;

        }

        return false;

    }

    int length() { return this.length; }

    public String toString()

    {

        StringBuilder b=new StringBuilder("[");

        String S;

        node X = head;

        while (X != null) {

            b.append(String.valueOf(X.data));

             if(X.next!= null)

                b.append(" -> ");

            X = X.next;

        }

        b.append("]");

        S=b.toString();

        return S ;

    }

    list concteList(list end){

        node temp=this.head;

        while(temp.next!=null){

        temp=temp.next;

        }

        temp.next=end.head;

        return this;

    }

}

Class Database : (excessing the data from excel file)

import java.util.Iterator;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.Workbook;

import org.apache.poi.xssf.usermodel.XSSFRow;

import org.apache.poi.xssf.usermodel.XSSFSheet;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class Database {

    public list column\_return(int column\_no) {

        list l1 = new list();

        Workbook wb = null; // initialize Workbook null

        try {

            FileInputStream fis = new FileInputStream("database.xlsx");

            wb = new XSSFWorkbook(fis);

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        } catch (IOException e1) {

            e1.printStackTrace();

        }

        Sheet sheet = wb.getSheetAt(0); // getting the XSSFSheet object at given index

        Iterator<Row> itr = sheet.iterator(); // iterating over excel file

        while (itr.hasNext()) {

            Row row = itr.next();

            Iterator<Cell> cellIterator = row.cellIterator(); // iterating over each column

            Cell cell = row.getCell(column\_no);

            switch (cell.getCellType()) {

                case Cell.CELL\_TYPE\_STRING: // field that represents string cell type

                    l1.add(cell.getStringCellValue());

                    break;

                case Cell.CELL\_TYPE\_NUMERIC: // field that represents number cell type

                    l1.add(cell.getNumericCellValue());

                    break;

                default:

            }

        }

        return l1;

    }

    public list column\_return(int column\_no,String Filename) {

        list l1 = new list();

        Workbook wb = null; // initialize Workbook null

        try {

            FileInputStream fis = new FileInputStream(Filename+".xlsx");

            wb = new XSSFWorkbook(fis);

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        } catch (IOException e1) {

            e1.printStackTrace();

        }

        Sheet sheet = wb.getSheetAt(0); // getting the XSSFSheet object at given index

        Iterator<Row> itr = sheet.iterator(); // iterating over excel file

        while (itr.hasNext()) {

            Row row = itr.next();

            Iterator<Cell> cellIterator = row.cellIterator(); // iterating over each column

            Cell cell = row.getCell(column\_no);

            switch (cell.getCellType()) {

                case Cell.CELL\_TYPE\_STRING: // field that represents string cell type

                    l1.add(cell.getStringCellValue());

                    break;

                case Cell.CELL\_TYPE\_NUMERIC: // field that represents number cell type

                    l1.add(cell.getNumericCellValue());

                    break;

                default:

            }

        }

        return l1;

    }

         public void insert\_excel(list l1,list l2,list l3) throws Exception

         {

                node temp1=l1.head, temp2=l2.head , temp3=l3.head;

                 //Create blank workbook

                 XSSFWorkbook workbook = new XSSFWorkbook();

                 //Create a blank sheet

                 XSSFSheet spreadsheet = workbook.createSheet( " Employee Info ");

                 //Create row object

                 XSSFRow row;

                 int rowid = 0;

                 Object o="";

                 while(temp3!=null){

                    int col=0;

                    String s;

                    row = spreadsheet.createRow(rowid++);

                        s= String.valueOf(temp1.data);

                       Cell cell = row.createCell(col);

                       if(s.equals("") || !isLetter(s.charAt(0))){

                        s="NOT";

                         }

                       cell.setCellValue(s);

                       col++;

                      s=String.valueOf(temp2.data);

                       Cell cell1 = row.createCell(col);

                       if(s.equals("") || !isLetter(s.charAt(0))){

                        s="NOT";

                         }

                       cell1.setCellValue(s);

                       col++;

                       s= String.valueOf(temp3.data);

                       Cell cell2 = row.createCell(col);

                       if(s.equals("") || !isLetter(s.charAt(0))){

                            s="NOT";

                             }

                        cell2.setCellValue(s);

                       temp1=temp1.next;

                       temp2=temp2.next;

                       temp3=temp3.next;

                    }

                 //Write the workbook in file system

                 FileOutputStream out = new FileOutputStream(

                    new File("Writesheet.xlsx"));

                 workbook.write(out);

                 out.close();

                 System.out.println("Writesheet.xlsx written successfully");

              }

              boolean isLetter(char ch){

                return (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z');

              }

              public void insert\_excel(list l1,list l2,list l3,list l4,list l5,list l6,list l7,list l8) throws Exception

         {

                node temp1=l1.head, temp2=l2.head , temp3=l3.head,temp4=l4.head, temp5=l5.head , temp6=l6.head,temp7=l7.head, temp8=l8.head ;

                 //Create blank workbook

                 XSSFWorkbook workbook = new XSSFWorkbook();

                 //Create a blank sheet

                 XSSFSheet spreadsheet = workbook.createSheet( " Employee Info ");

                 //Create row object

                 XSSFRow row;

                 int rowid = 0;

                 Object o="";

                 while(temp3!=null){

                    int col=0;

                    String s;

                    row = spreadsheet.createRow(rowid++);

                        s= String.valueOf(temp1.data);

                       Cell cell = row.createCell(col);

                       cell.setCellValue(s);

                       col++;

                      s=String.valueOf(temp2.data);

                       Cell cell1 = row.createCell(col);

                       cell1.setCellValue(s);

                       col++;

                       s= String.valueOf(temp3.data);

                       Cell cell2 = row.createCell(col);

                        cell2.setCellValue(s);

                        col++;

                       s= String.valueOf(temp4.data);

                       Cell cell3 = row.createCell(col);

                        cell3.setCellValue(s);

                        col++;

                       s= String.valueOf(temp5.data);

                       Cell cell4 = row.createCell(col);

                        cell4.setCellValue(s);

                        col++;

                       s= String.valueOf(temp6.data);

                       Cell cell5 = row.createCell(col);

                        cell5.setCellValue(s);

                        col++;

                       s= String.valueOf(temp7.data);

                       Cell cell6 = row.createCell(col);

                        cell6.setCellValue(s);

                        col++;

                       s= String.valueOf(temp8.data);

                       Cell cell7 = row.createCell(col);

                        cell7.setCellValue(s);

                       temp1=temp1.next;

                       temp2=temp2.next;

                       temp3=temp3.next;

                       temp4=temp4.next;

                       temp5=temp5.next;

                       temp6=temp6.next;

                       temp7=temp7.next;

                       temp8=temp8.next;

                    }

                 //Write the workbook in file system

                 FileOutputStream out = new FileOutputStream(

                    new File("Writesheet.xlsx"));

                 workbook.write(out);

                 out.close();

                 System.out.println("Writesheet.xlsx written successfully");

              }

}

Class Location : ( finding country name from server )

import eu.bitm.NominatimReverseGeocoding.Address;

import eu.bitm.NominatimReverseGeocoding.NominatimReverseGeocodingJAPI;

public class Location {

    NominatimReverseGeocodingJAPI n1 = new NominatimReverseGeocodingJAPI(18);

    NominatimReverseGeocodingJAPI n2 = new NominatimReverseGeocodingJAPI(18);

    Address address;

    Database d = new Database();

    Object[] city\_country(list lat, list lon, int Starting, int ending) {

        list ls = new list();

        list l = new list();

        list lis = new list();

        ls.head = new node("country");

        l.head = new node("city");

        lis.head = new node("state");

        node temp = ls.head, tem = l.head, temp3 = lis.head;

        node temp1 = lat.head.next, temp2 = lon.head.next;

        double t, o;

        int counter = 0;

        if (Starting > 1)

            for (int i = 1; i < Starting; i++) {

                temp1 = temp1.next;

                temp2 = temp2.next;

            }

        while (temp1 != null) {

            t = (Double) temp1.data;

            o = (Double) temp2.data;

            address = n2.getAdress(t, o);

            temp.next = new node(address.getCountry());

            System.out.print((++counter) + address.getCountry() + " ");

            tem.next = new node(address.getCity());

            System.out.print(address.getCity() + " ");

            temp3.next = new node(address.getState());

            System.out.print(address.getState() + " ");

            temp3 = temp3.next;

            temp = temp.next;

            tem = tem.next;

            temp1 = temp1.next;

            temp2 = temp2.next;

            Starting++;

            if (Starting == ending) {

                break;

            }

        }

        Object[] obj = { ls, l, lis };

        return obj;

    }

}