|  |  |
| --- | --- |
| **Student Name** | **Azhar Ali** |
| **Roll Number** | **21SW087** |
| **Section #** | **II** |
| **PBL #** | **2** |

**Class Solution:**

*package* PBL2;  
  
*import* java.io.File;  
*import* java.io.FileNotFoundException;  
*import* java.util.ArrayList;  
*import* java.util.Scanner;  
  
*public class* Solution {  
 ArrayList<CrashCase>[] month=*new* ArrayList[12];*//ArrayList array that stores ArrayList of each month* MyQueue monthWiseQueue=*new* MyQueue();*//Queue storing data month wise* MyQueue clearWeatherCrashCaseQueue=*new* MyQueue();  
 MyQueue snowWeatherCrashCaseQueue=*new* MyQueue();  
 MyQueue rainWeatherCrashCaseQueue=*new* MyQueue();  
 MyQueue monday=*new* MyQueue();*//queue having only monday cases* MyQueue tuesday=*new* MyQueue();*//queue having only tuesday cases* MyQueue wednesday=*new* MyQueue();*//queue having wednesday cases* MyQueue thursday=*new* MyQueue();*//queue having thursday cases* MyQueue friday=*new* MyQueue();*//queue having friday cases* MyQueue saturday=*new* MyQueue();*//queue having saturday cases* MyQueue sunday=*new* MyQueue();*//queue having sunday cases  
  
// I have stored the keys as January being root, even no months on left and odd no months on right in both the trees* BinaryTree hit\_and\_run\_tree;*//tree having only no of hit\_and\_run cases with month* BinaryTree not\_hit\_and\_run\_tree;*//tree having only no of not\_hit\_and\_run cases with month  
 //filling month ArrayList array  
 public void* setMonth(){  
 File file=*new* File("ped\_crashes.csv");  
 *try* {  
 Scanner scan=*new* Scanner(file);  
 scan.nextLine();  
 *while* (scan.hasNextLine()){  
 String[] parts=scan.nextLine().split(",");  
 CrashCase c=*new* CrashCase(parts[0],parts[1],parts[2],parts[3],parts[4],parts[5], parts[6],parts[7],parts[8],parts[9],parts[10],parts[11],parts[12],parts[13],parts[14]);  
 *switch* (parts[1]){  
 *case* "January"->{  
 *if*(month[0]==*null*)month[0]=*new* ArrayList();  
 month[0].add(c);  
 }  
 *case* "February"->{  
 *if*(month[1]==*null*)month[1]=*new* ArrayList();  
 month[1].add(c);  
 }  
 *case* "March"->{  
 *if*(month[2]==*null*)month[2]=*new* ArrayList();  
 month[2].add(c);  
 }  
 *case* "April"->{  
 *if*(month[3]==*null*)month[3]=*new* ArrayList();  
 month[3].add(c);  
 }  
 *case* "May"->{  
 *if*(month[4]==*null*)month[4]=*new* ArrayList();  
 month[4].add(c);  
 }  
 *case* "June"->{  
 *if*(month[5]==*null*)month[5]=*new* ArrayList();  
 month[5].add(c);  
 }  
 *case* "July"->{  
 *if*(month[6]==*null*)month[6]=*new* ArrayList();  
 month[6].add(c);  
 }  
 *case* "August"->{  
 *if*(month[7]==*null*)month[7]=*new* ArrayList();  
 month[7].add(c);  
 }  
 *case* "September"->{  
 *if*(month[8]==*null*)month[8]=*new* ArrayList();  
 month[8].add(c);  
 }  
 *case* "October"->{  
 *if*(month[9]==*null*)month[9]=*new* ArrayList();  
 month[9].add(c);  
 }  
 *case* "November"->{  
 *if*(month[10]==*null*)month[10]=*new* ArrayList();  
 month[10].add(c);  
 }  
 *case* "December"->{  
 *if*(month[11]==*null*)month[11]=*new* ArrayList();  
 month[11].add(c);  
 }  
  
 }  
 }  
 }*catch* (FileNotFoundException e){  
 System.***out***.println(e.getMessage());  
 }  
 }  
 *//inserting data into wise queue month wise with January on first in and first out  
 public void* setMonthWiseQueue(){  
 *for*(ArrayList<CrashCase> list:month)  
 *for* (CrashCase CrashCase : list) monthWiseQueue.add(CrashCase);  
 }  
 *public void* setSubQueues(){  
 MyQueue queue=monthWiseQueue.copy();  
 *int* size= queue.size();  
 *for*(*int* i=0;i<size;i++){  
 CrashCase c=queue.remove();  
 *switch* (c.getWeather\_condition()) {  
 *case* "Clear" -> clearWeatherCrashCaseQueue.add(c);  
 *case* "Rain" -> rainWeatherCrashCaseQueue.add(c);  
 *case* "Snow" -> snowWeatherCrashCaseQueue.add(c);  
 }  
 }  
 }  
 *public void* setWeekQueues(){  
 MyQueue queue=monthWiseQueue.copy();  
 *int* size= queue.size();  
 *for*(*int* i=0;i<size;i++){  
 CrashCase c=queue.remove();  
 *switch* (c.getDay().toLowerCase()){  
 *case* "monday"->monday.add(c);  
 *case* "tuesday"->tuesday.add(c);  
 *case* "wednesday"->wednesday.add(c);  
 *case* "thursday"->thursday.add(c);  
 *case* "friday"->friday.add(c);  
 *case* "saturday"->saturday.add(c);  
 *case* "sunday"->sunday.add(c);  
 }  
 }  
 }  
 *public void* fillTree(){  
 MyQueue queue=monthWiseQueue.copy();  
 *int* size= queue.size();  
 String month="January";  
 *int* hit\_run=0,not\_hit\_run=0;  
 *for*(*int* i=0;i<size;i++){  
 CrashCase c=queue.remove();  
 *if*(i==size-1){  
 *if*(c.getHit\_run().equals("Hit-and-run"))hit\_run++;  
 *else if* (c.getHit\_run().equals("Not hit-and-run"))not\_hit\_run++;  
 hit\_and\_run\_tree.add("left",hit\_run+","+month);  
 not\_hit\_and\_run\_tree.add("left",not\_hit\_run+", "+month);  
 *return*;  
 }  
 *if* (!month.equals(c.getMonth())) {  
 *boolean* b = month.equals("February") || month.equals("April") || month.equals("June") || month.equals("August") || month.equals("October") ;  
 *if* (hit\_and\_run\_tree == *null*) hit\_and\_run\_tree = *new* BinaryTree(hit\_run+", "+month);  
 *else* {  
 *if* (b) hit\_and\_run\_tree.add("left", hit\_run+", "+month);  
 *else* hit\_and\_run\_tree.add("right", hit\_run+", "+month);  
 }  
 *if* (not\_hit\_and\_run\_tree == *null*) not\_hit\_and\_run\_tree = *new* BinaryTree(not\_hit\_run+", "+month);  
 *else* {  
 *if* (b) not\_hit\_and\_run\_tree.add("left", not\_hit\_run+", "+month);  
 *else* not\_hit\_and\_run\_tree.add("right", not\_hit\_run+", "+month);  
 }  
 month = c.getMonth();  
 hit\_run = 0;  
 not\_hit\_run = 0;  
 }  
 *if*(c.getHit\_run().equals("Hit-and-run"))hit\_run++;  
 *else if* (c.getHit\_run().equals("Not hit-and-run"))not\_hit\_run++;  
  
 }  
 }  
 *public* String deadliestDay(MyQueue day){  
 *//max1: maximum cases for current date and max2 for maximum no of cases on any date  
 int* max1=1,max2=1;  
 *//duplicate queue so that remove method can be called* MyQueue queue=day.copy();  
 *//size of queue  
 int* size= queue.size();  
 *//CrashCase object for getting starting date* CrashCase c=queue.remove();  
 *//current date* String date=c.getDate()+"/"+c.getMonth()+"/"+c.getYear();  
 *//initializing tempDate* String tempDate=date;  
 String date1="";  
 *for*(*int* i=1;i<size;i++){  
 *//getting one by one object* CrashCase c1=queue.remove();  
 *//getting date of every object* date1=c1.getDate()+"/"+c1.getMonth()+"/"+c1.getYear();  
 *//checking weather date is same  
 if*(date1.equals(date))max1++;  
 *//if date has been changed  
 else*{  
 *//if cases on previous date are greater than the previous maximum cases  
 if*(max1>max2)tempDate=date;  
 *//getting the max no of cases* max2=Math.*max*(max1,max2);  
 max1=1;  
 *//updating the current date* date=date1;  
 }  
 }  
 *if*(max1>max2)tempDate=date;  
 *return* Math.*max*(max1,max2)+", were the maximum number of cases happened on "+tempDate+" on "+c.getDay();  
 }  
 *public void* question1( ){  
 *int* max1,max2,maxFriday;  
 String fridayDate;  
 String[] s=deadliestDay(monday).split(",");  
 String str=s[1];  
 max1=Integer.*parseInt*(s[0]);  
 String[] s1=deadliestDay(tuesday).split(",");  
 max2=Integer.*parseInt*(s1[0]);  
 *if*(max2>max1)str=s1[1];  
 max1=Math.*max*(max1,max2);  
 s1=deadliestDay(wednesday).split(",");  
 max2=Integer.*parseInt*(s1[0]);  
 *if*(max2>max1)str=s1[1];  
 max1=Math.*max*(max1,max2);  
 s1=deadliestDay(thursday).split(",");  
 max2=Integer.*parseInt*(s1[0]);  
 *if*(max2>max1)str=s1[1];  
 max1=Math.*max*(max2,max1);  
 s1=deadliestDay(friday).split(",");  
 max2=Integer.*parseInt*(s1[0]);  
 maxFriday=max2;  
 fridayDate=s1[1];  
 *if*(max2>max1)str=s1[1];  
 max1=Math.*max*(max1,max2);  
 s1=deadliestDay(saturday).split(",");  
 max2=Integer.*parseInt*(s1[0]);  
 *if*(max2>max1)str=s1[1];  
 max1=Math.*max*(max1,max2);  
 s1=deadliestDay(sunday).split(",");  
 max2=Integer.*parseInt*(s1[0]);  
 *if*(max2>max1)str=s1[1];  
 max1=Math.*max*(max1,max2);  
 System.***out***.println(max1+": "+str);  
 System.***out***.println(maxFriday+","+fridayDate+". It was the deadliest Friday among all the Fridays");  
  
 }  
 *public void* question2(){  
 System.***out***.println("Rainy day Crashes of every month");  
 MyQueue queue=rainWeatherCrashCaseQueue.copy();  
 *int* size=queue.size();  
 *int* count=0;  
 String month=queue.remove().getMonth();  
 *for*(*int* i=1;i<size;i++){  
 String month1=queue.remove().getMonth();  
 *if*(month1.equals(month))count++;  
 *else* {  
 System.***out***.println(month+" CrashCase "+count);  
 count=0;  
 month=month1;  
 }  
 }  
 System.***out***.println(month+" CrashCase "+count);  
 }  
 *public void* question3(){  
 System.***out***.println("1. Which month had the lowest \"hit and run\" CrashCase?\t\t"+hit\_and\_run\_tree.getSmallest());  
 System.***out***.println("2. Which month had the highest \"hit and run\" CrashCase?\t\t"+hit\_and\_run\_tree.getLargest());  
 System.***out***.println("3. Which month had the lowest \"not hit and run\" CrashCase?\t\t"+not\_hit\_and\_run\_tree.getSmallest());  
 System.***out***.println("4. Which month had the highest \"not hit and run\" CrashCase?\t\t"+not\_hit\_and\_run\_tree.getLargest());  
 }  
 *public void* postulate1(){  
 *int* mondaysCrashCase=0,tuesdaysCrashCase=0,saturdaysCrashCase=0,sundayCrashCase=0;  
 MyQueue[] queues={monday,tuesday,saturday,sunday};  
 *for*(MyQueue q:queues){  
 *int* size=q.size();  
 *for*(*int* i=0;i<size;i++){  
 String lightCondition=q.remove().getLight\_condition();  
 *if*(lightCondition.equals("Dark lighted")){  
 *if* (monday.equals(q)) mondaysCrashCase++;  
 *else if* (tuesday.equals(q)) tuesdaysCrashCase++;  
 *else if* (saturday.equals(q)) saturdaysCrashCase++;  
 *else if* (sunday.equals(q)) sundayCrashCase++;  
 }  
 }  
 }  
 System.***out***.println("CrashCase when light condition was dark lighted");  
 System.***out***.println("Monday CrashCase "+mondaysCrashCase);  
 System.***out***.println("Tuesday CrashCase "+tuesdaysCrashCase);  
 System.***out***.println("Saturday CrashCase "+saturdaysCrashCase);  
 System.***out***.println("Sunday CrashCase "+sundayCrashCase);  
 }  
  
 *public static void* main(String[] args) {  
 Solution s=*new* Solution();  
 s.setMonth();  
 s.setMonthWiseQueue();  
 s.setSubQueues();  
 s.setWeekQueues();  
 s.fillTree();  
 *while* (*true*){  
 System.***out***.println("\n\n\n");  
 System.***out***.println("1.See which day was the deadliest among all the days from Monday to Sunday and Which month's Friday was deadliest?");  
 System.***out***.println("2.See no of Rainy Crashes in every month");  
 System.***out***.println("3.Lowest and highest hit\_and\_run and not\_hit\_and\_run");  
 System.***out***.println("4.Comparing Monday and Tuesday crashes with Saturday and Sunday (Postulate1)");  
 System.***out***.println("5.Exit");  
 Scanner scan=*new* Scanner(System.***in***);  
 *switch* (scan.next()){  
 *case* "1"->s.question1();  
 *case* "2"->s.question2();  
 *case* "3"->s.question3();  
 *case* "4"->s.postulate1();  
 *case* "5"->System.*exit*(0);  
 }  
 }  
 }  
}

**Class CrashCases:**

*package* PBL2;  
  
*public class* CrashCase {  
 *//Attributes  
 private final* String year,month,date,time,day,city,intersection,hit\_run,light\_condition,weather\_condition,speed,injury\_type,partyType,age,gender;  
  
 *//Argumentative constructor  
 public* CrashCase(String year, String month, String date, String time, String day, String city, String intersection, String hit\_run, String light\_condition, String weather\_condition, String speed, String injury\_type, String partyType, String age, String gender) {  
 *this*.year = year;  
 *this*.month = month;  
 *this*.date = date;  
 *this*.time = time;  
 *this*.day = day;  
 *this*.city = city;  
 *this*.intersection = intersection;  
 *this*.hit\_run = hit\_run;  
 *this*.light\_condition = light\_condition;  
 *this*.weather\_condition = weather\_condition;  
 *this*.speed = speed;  
 *this*.injury\_type = injury\_type;  
 *this*.partyType = partyType;  
 *this*.age = age;  
 *this*.gender = gender;  
 }  
  
 *//Getters  
 public* String getYear() {  
 *return* year;  
 }  
  
 *public* String getMonth() {  
 *return* month;  
 }  
  
 *public* String getDate() {  
 *return* date;  
 }  
  
 *public* String getTime() {  
 *return* time;  
 }  
  
 *public* String getDay() {  
 *return* day;  
 }  
  
 *public* String getCity() {  
 *return* city;  
 }  
  
 *public* String getIntersection() {  
 *return* intersection;  
 }  
  
 *public* String getHit\_run() {  
 *return* hit\_run;  
 }  
  
 *public* String getLight\_condition() {  
 *return* light\_condition;  
 }  
  
 *public* String getWeather\_condition() {  
 *return* weather\_condition;  
 }  
  
 *public* String getSpeed() {  
 *return* speed;  
 }  
  
 *public* String getInjury\_type() {  
 *return* injury\_type;  
 }  
  
 *public* String getPartyType() {  
 *return* partyType;  
 }  
  
 *public* String getAge() {  
 *return* age;  
 }  
  
 *public* String getGender() {  
 *return* gender;  
 }  
  
 *//toString method to print the information of the object* @Override  
 *public* String toString() {  
 *return* "Date: "+date+"\tMonth: "+month+"\tYear: "+year+"\tDay: "+day+"\tTime: "+time+"\tCity: "+city+"\tCrashCase Intersection: "+  
 intersection+"\tCrashCase Hit and Run: "+hit\_run+"\tSpeed: "+speed+"\tWorst Injury in CrashCase: "+injury\_type+  
 "\tParty type: "+partyType+"\tPerson Age: "+age+"\tGender: "+gender;  
  
 }  
}

**Class MyStack:**

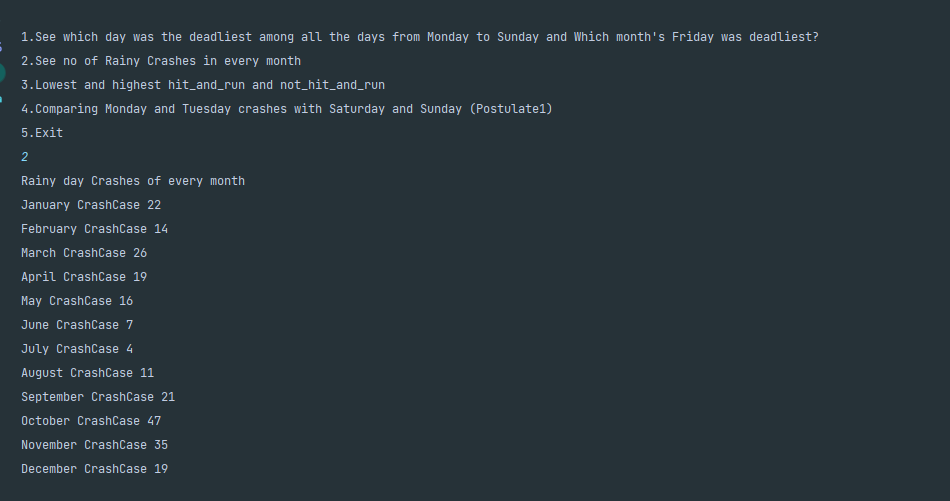
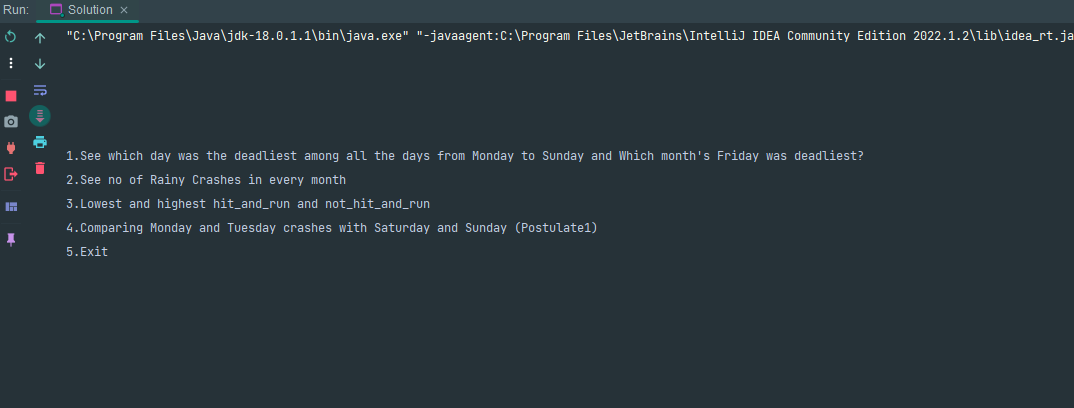
*package* PBL2;  
  
*import* java.util.EmptyStackException;  
*//Stack class  
public class* MyStack{  
 *//Node class  
 private static class* Node{  
 CrashCase data;*//for storing object of CrashCase* Node next;  
 *public* Node(CrashCase data){*this*.data=data;}  
 }  
 *private int* size;*//size of stack  
 private* Node top;*//top of stack  
 private* String name;*//name of stack  
 public* MyStack(String name){*this*.name=name;}  
 *public boolean* isEmpty(){  
 *return* size==0;  
 }  
 *public void* push(CrashCase data){  
 Node n=*new* Node(data);  
 n.next=top;  
 top=n;  
 size++;  
 }  
 *public* CrashCase pop(){  
 *if*(isEmpty())*throw new* EmptyStackException();  
 Node n=top;  
 top=top.next;  
 size--;  
 *return* n.data;  
 }  
 *public* CrashCase peek(){  
 *if*(isEmpty())*throw new* IllegalArgumentException();  
 *return* top.data;  
 }  
  
 *public int* getSize() {  
 *return* size;  
 }  
 *public* String getName() {  
 *return* name;  
 }  
}

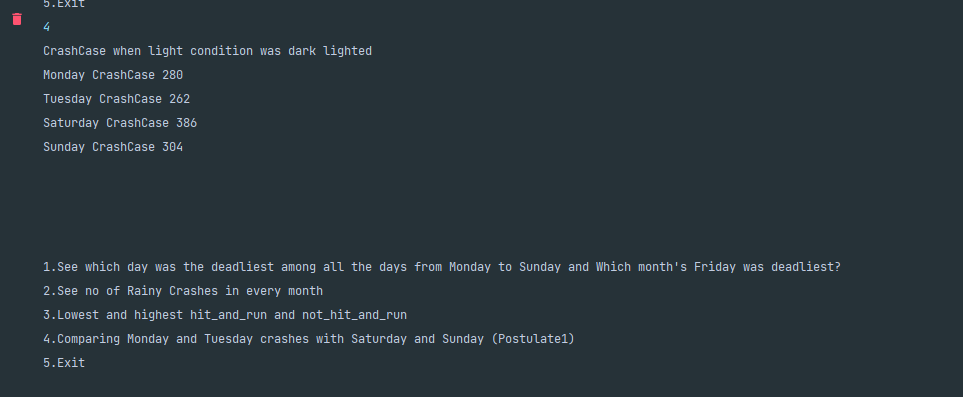
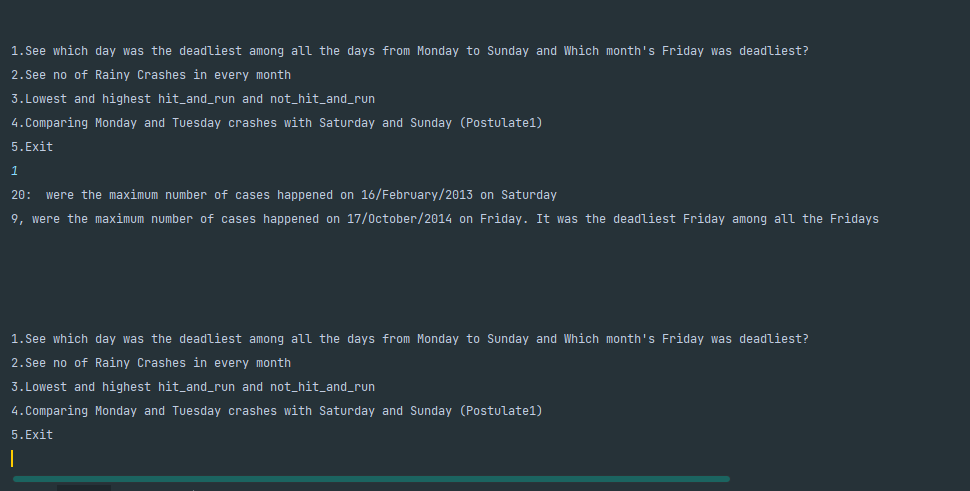
**Class MyQueue:**

*package* PBL2;  
*import* javax.management.openmbean.InvalidOpenTypeException;  
  
*public class* MyQueue {  
 *private static class* Node{  
 CrashCase data;  
 Node next=*this*;  
 Node pre=*this*;  
 *public* Node(CrashCase data){*this*.data=data;}  
 *public* Node(CrashCase 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(CrashCase data) {  
 head.pre.next=*new* Node(data,head.pre,head);  
 head.pre=head.pre.next;  
 size++;  
 }  
 *public* CrashCase remove() {  
 *if*(isEmpty())*throw new* InvalidOpenTypeException("Queue is empty!");  
 CrashCase temp=head.next.data;  
 head.next=head.next.next;  
 size--;  
 *return* temp;  
 }  
 *public* CrashCase first() {  
 *if*(isEmpty())*throw new* InvalidOpenTypeException("Queue is empty!");  
 *return* head.next.data;  
 }  
 *public int* size() {  
 *return* size;  
 }  
 *public* MyQueue copy(){  
 MyQueue queue=*new* MyQueue();  
 *for*(Node n=head.next;n!=head;n=n.next)queue.add(n.data);  
 *return* queue;  
 }*//for making copy of queue  
 public boolean* isEmpty(){  
 *return* size==0;  
 }  
}

**Class BinaryTree:**

*package* PBL2;  
*public class* BinaryTree {  
 Object root;  
 BinaryTree left,right;  
 *public* BinaryTree(Object root){*this*.root=root;}  
 *public* BinaryTree(Object root, BinaryTree left, BinaryTree right){  
 *this*.root=root;  
 *this*.left=left;  
 *this*.right=right;  
 }  
  
 *public* Object getRoot() {  
 *return* root;  
 }  
  
 *public* BinaryTree getLeft() {  
 *return* left;  
 }  
  
 *public* BinaryTree getRight() {  
  
 *return* right;  
 }  
 *//for getting the smallest value from the tree  
 public* String getSmallest(){  
 BinaryTree temp=*this*;*//assigning the actual tree to temp so that can be traversed  
 int* leftSmall,rightSmall,temporary;  
 String[] parts=String.*valueOf*(temp).split(",");*//because it has two parts cases before comma and month after comma* leftSmall=Integer.*parseInt*(parts[0]);  
 rightSmall=Integer.*parseInt*(parts[0]);  
 String monthLeft=parts[1],monthRight=parts[1];  
 *while* (temp.left!=*null*){  
 parts=String.*valueOf*(temp.left.root).split(",");  
 temporary=Integer.*parseInt*(parts[0]);  
 *if*(temporary<leftSmall){  
 leftSmall=temporary;  
 monthLeft=parts[1];  
 }  
 temp=temp.left;  
 }  
 temp=*this*;  
 *while* (temp.right!=*null*){  
 parts=String.*valueOf*(temp.right.root).split(",");  
 temporary=Integer.*parseInt*(parts[0]);  
 *if*(temporary<rightSmall){  
 rightSmall=temporary;  
 monthRight=parts[1];  
 }  
 temp=temp.right;  
 }  
 *if*(leftSmall<rightSmall)*return* leftSmall+", crashes in "+monthLeft;  
 *return* rightSmall+", crashes in "+monthRight;  
 }  
 *public* String getLargest(){  
 BinaryTree temp=*this*;  
 *int* leftLarge,rightLarge,temporary;  
 String[] parts=String.*valueOf*(temp).split(",");  
 leftLarge=Integer.*parseInt*(parts[0]);  
 rightLarge=Integer.*parseInt*(parts[0]);  
 String monthLeft=parts[1],monthRight=parts[1];  
 *while* (temp.left!=*null*){  
 parts=String.*valueOf*(temp.left.root).split(",");  
 temporary=Integer.*parseInt*(parts[0]);  
 *if*(temporary>leftLarge){  
 leftLarge=temporary;  
 monthLeft=parts[1];  
 }  
 temp=temp.left;  
 }  
 temp=*this*;  
 *while* (temp.right!=*null*){  
 parts=String.*valueOf*(temp.right.root).split(",");  
 temporary=Integer.*parseInt*(parts[0]);  
 *if*(temporary>rightLarge){  
 rightLarge=temporary;  
 monthRight=parts[1];  
 }  
 temp=temp.right;  
 }  
 *if*(leftLarge>rightLarge)*return* leftLarge+", crashes in "+monthLeft;  
 *return* rightLarge+", crashes in "+monthRight;  
 }  
  
 *public void* add(String side, Object noOfCases){  
 BinaryTree temp=*this*;  
 *if*(side.equals("left")){  
 *while* (*true*) {  
 *if* (temp.left == *null*) {  
 temp.left = *new* BinaryTree(noOfCases);  
 *return*;  
 }  
 temp = temp.left;  
 }  
 }  
 *else if*(side.equals("right")){  
 *while* (*true*) {  
 *if* (temp.right == *null*) {  
 temp.right = *new* BinaryTree(noOfCases);  
 *return*;  
 }  
 temp = temp.right;  
 }  
 }  
 }  
 @Override  
 *public* String toString() {  
 StringBuilder s=*new* StringBuilder();  
 *if*(left!=*null*)s.append(left).append(" , ");  
 s.append(root).append("\t");  
 *if*(right!=*null*)s.append(right).append("\t");  
 *return* s+"";  
 }  
 *public int* size(){  
 *return* 1+(left!=*null*? left.size() : 0)+(right!=*null*? right.size() : 0);  
 }  
}

**Outputs:**

****

**ALGORITHM**

Step 1:  
->Make CrashCase class with proper attributes that are given in csv file  
->Now in Solution class make array of ArrayList with size 12 (no of months) with each index representing  
 every month respectively.  
->Now make a method to read file and store it into the above declared array  
->Now make a queue class and make an object of queue in the solution class and read the month array and store in it queue  
  
  
  
Step 2:  
-> Make 3 objects of queue in the solution class  
-> Now read the original queue and store the objects with checking the condition of weather and store the objects in their respective sub\_queues  
 by call the getWeather() method of each object removed from original queue  
  
  
  
Step 3:  
->Now make seven objects of queues named by the week days  
->Then make copy of month wise queue and using for loop remove one by one from copy queue and check  
 the day by calling getDay method of CrashCase and using switch case store the object in the respective sub queue.  
  
  
  
  
Step 4:  
->Now make BinaryTree class with its appropriate methods also add another method called add() with the condition that it  
 inserts the no of cases with month name associated in it. Make a condition that it inserts the even number month on the left side and odd number  
 month on the right side  
->Make two methods getLowest() and getHighest() by traversing the trees and returning the lowest of both the sides (highest for highest of both the trees)  
->Then make two references of binary tree in the solution class  
->Then make a method like fillTree and by counting the cases for each month (two int type hit\_and\_run and not\_hit\_and\_run separately) and  
 by calling the add method of binary tree and pass the argument by concatenating the cases with month name as an object with the appropriate conditions.  
  
  
Problem 1:  
->Make a method the deadliest day which takes the queue as an argument for the week\_day\_queue by traversing the week days sub queues  
 and counting the cases of the same date and save the maximum no of cases of the same date and then return the max no of cases with day as concatenating.  
->Now make another method called question1 that calls the above method for each day and by comparing the cases of each day now at the  
 end of the method print both the statements that which of the days has the most cases with date associated also friday cases with complete date.  
  
  
  
Problem 2:  
->For this traverse the array of ArrayList and count the rainy day crashes by checking the condition by calling the method getWeatherCondition() of CrashCases class  
 and print the counter variable with month name (hint use for loop to traverse the array)  
  
  
  
Problem 3:  
->call the getLowest and getHighest methods of BinaryTree class already defined above by using appropriate reference of the tree.  
  
  
  
Postulate 1:  
->Call the deadliest method by providing the queues of the monday, tuesday, saturday and sunday and then see the result and conclude on the basis of the result  
 you can see that the postulate is true.