import java.io.\*;  
import java.util.\*;  
public class Engine {  
 private ArrayList<Team> teams = new ArrayList<>();  
 private ArrayList<Discipline> disciplines = new ArrayList<>();  
 private ArrayList<Subscription> subscriptions = new ArrayList<>();  
 private ArrayList<Member> members = new ArrayList<>();  
 private ArrayList<Record> records = new ArrayList<>();  
  
 public ArrayList<Discipline> getDisciplines() {  
 return disciplines;  
 }  
 public ArrayList<Subscription> getSubscriptions(){  
 return subscriptions;  
 }  
 public ArrayList<Team> getTeams() {  
 return teams;  
 }  
 public ArrayList<Record> getRecords(){  
 return records;  
 }  
 //Bertram  
 public ArrayList<Record> getPersonalRecordsFromMember(CompetitiveMember cm, boolean onlyBest) {  
 //get all records sorted  
 ArrayList<Record> sortedRecords = selectionSortRecords(getRecords());  
 //get disciplines for member  
 ArrayList<Discipline> activeDisciplines = cm.getDisciplines();  
 //create arraylist for members records  
 ArrayList<Record> pRecords = new ArrayList<>();  
 for (Discipline discipline : activeDisciplines) {  
 for (Record r : sortedRecords) {  
 //if the record is not from a competiton, in the right discipline and belong to the member  
 if (!r.isFromCompetition() && discipline == r.getDiscipline() && cm == r.getMember()) {  
 pRecords.add(r);  
 if (onlyBest) {  
 break;  
 }  
 }  
 }  
 }  
 return pRecords;  
 }  
   
 //Bertram  
 public ArrayList<Record> getPersonalRecordsFromAll() {  
 //get all records sorted  
 ArrayList<Record> sortedRecords = selectionSortRecords(getRecords());  
 //get all disciplines so we can order records by discipline  
 ArrayList<Discipline> disciplines = getDisciplines();  
 //create arraylist for personal records  
 ArrayList<Record> pRecords = new ArrayList<>();  
 for (Discipline discipline : disciplines) {  
 for (Record r : sortedRecords) {  
 //if the record is not from a competiton and is in the current discipline  
 if (!r.isFromCompetition() && discipline == r.getDiscipline()) {  
 pRecords.add(r);  
 }  
 }  
 }  
 return pRecords;  
 }  
   
 //Bertram  
 public Record getLastCompetitiveRecord() {  
 Collections.reverse(records);  
 for (Record r : records) {  
 if (r.isFromCompetition()) {  
 Collections.reverse(records);  
 return r;  
 }  
 }  
 return null;  
 }  
   
 //Bertram  
 public Record getRecordById(int id) {  
 for (Record r : records) {  
 if (r.getId() == id)  
 return r;  
 }  
 return null;  
 }  
   
 //Bertram  
 public ArrayList<Record> getTop5RecordsByDiscplineAndTeam(Discipline discipline, Team team) {  
 //create a list for members in the team that was passed as a parameter  
 ArrayList<CompetitiveMember> teamMembers = getTeamMembers(team);  
 //create a list for top5 records  
 ArrayList<Record> top5 = new ArrayList<>();  
 //get all records and sort them  
 ArrayList<Record> records = selectionSortRecords(getRecords());  
 //loop all records  
 for (Record r : records) {  
 //if record is in the right discipline  
 if (r.getDiscipline() == discipline) {  
 boolean memberIsInTeam = false;  
 boolean memberExistsInTop5 = false;  
 //check if record belongs to anyone on the team  
 for (Member tm : teamMembers) {  
 if (r.getMember() == tm) {  
 memberIsInTeam = true;  
 break;  
 }  
 }  
 if (memberIsInTeam) {  
 //check if the recordholder already is in top5  
 for (Record top5R : top5) {  
 if (top5R.getMember() == r.getMember()) {  
 memberExistsInTop5 = true;  
 break;  
 }  
 }  
 }  
 //finally if the recorderholder is on the team and not already added, add the record  
 if (memberIsInTeam && !memberExistsInTop5) {  
 top5.add(r);  
 }  
 }  
 //if we already have found 5, plz stop.  
 if (top5.size() == 5) {  
 break;  
 }  
 }  
 return top5;  
 }  
   
 //Bertram  
 public ArrayList<Record> getCompetitionRecords() {  
 ArrayList<Record> cRecords = new ArrayList<>();  
 for (Record r : records) {  
 if (r.isFromCompetition()) {  
 cRecords.add(r);  
 }  
 }  
 return cRecords;  
 }  
 public ArrayList<Member> getMembers() {  
 return members;  
 }  
   
 //SÃ¸ren  
 public Member getMember(String cpr){  
 for (Member m : members) {  
 if (cpr.equals(m.getCpr())) {  
 return m;  
 }  
 }  
 return null;  
 }  
   
 //Bertram  
 public ArrayList<CompetitiveMember> getTeamMembers(Team team) {  
 ArrayList<CompetitiveMember> teamMembers = new ArrayList<>();  
 ArrayList<Member> members = getMembers();  
 for (Member m : members) {  
 //find all competitive members  
 if (m.getIsCompetitive()) {  
 CompetitiveMember cm = (CompetitiveMember)m;  
 //if the member belong to the team, add them to our list   
 if (cm.getTeam() == team) {  
 teamMembers.add(cm);  
 }   
 }  
 }  
 return teamMembers;  
 }  
  
 public void addRecord(Record r){  
 records.add(r);  
 }  
 public void addMember(Member m) {  
 members.add(m);  
 }  
   
 //SÃ¸ren, Martin, Bertram  
 public void deleteMember(Member member) {  
 int i = 0;  
 int index = 0;  
 for (Member m : members) {  
 if (m.getCpr().equals(member.getCpr())) {  
 index = i;  
 }  
 i++;  
 }  
 members.remove(index);  
 }  
   
 //Bertram  
 public void deleteRecord(int id) {  
 int i = 0;  
 int index = 0;  
 for (Record r : records) {  
 if (r.getId() == id) {  
 index = i;  
 }  
 i++;  
 }  
 records.remove(index);  
 }  
   
 //Bertram  
 public void deleteRecordsByMember(Member m) {  
 ArrayList<Record> recordsToRemove = new ArrayList<>();  
 for (Record r : records) {  
 if (r.getMember() == (CompetitiveMember)m) {  
 recordsToRemove.add(r);  
 }  
 }  
 for (Record r : recordsToRemove) {  
 deleteRecord(r.getId());  
 }  
 }  
   
 //Bertram, SÃ¸ren, Martin  
 public void saveMembers() throws IOException{  
 FileWriter fw = new FileWriter("data/members.txt");  
 int endIndex = members.size()-1;  
 for (int i = 0; i < endIndex; i++) {  
 fw.write(members.get(i).toString() + "\n");  
 }  
 if (members.get(endIndex) != null) {  
 fw.write(members.get(endIndex).toString());  
 }  
 fw.close();  
 }  
   
 //Bertram  
 public void saveRecords() throws IOException{  
 FileWriter fw = new FileWriter("data/records.txt");  
 int endIndex = records.size()-1;  
 for (int i = 0; i < endIndex; i++) {  
 fw.write(records.get(i).toString() + "\n");  
 }  
 if (records.get(endIndex) != null) {  
 fw.write(records.get(endIndex).toString());  
 }  
 fw.close();  
 }  
   
 //Bertram  
 public void loadData() throws Exception {  
 //Most Danish Windows OS have their locale set to da;Danish and thus expecting decimal types from scanners to contain ',' instead of '.'  
 //Hardcoding the locale as us;English will remove that annoying feature.  
 Locale.setDefault(new Locale("en", "US"));  
   
   
 //fill all the lists with data stored in their respective text files, one by one  
 //the order is important, because some objects are depending on other objects to have been initialized  
 Scanner teamScanner = new Scanner(new File("data/teams.txt"));  
 while (teamScanner.hasNextLine()){  
 String line = teamScanner.nextLine();  
 String[] str = line.split(":");  
 Team t = new Team(Integer.parseInt(str[0]), str[1], str[2]);  
 teams.add(t);  
 }  
   
 Scanner disciplineScanner = new Scanner(new File("data/disciplines.txt"));  
 while (disciplineScanner.hasNextLine()){  
 String line = disciplineScanner.nextLine();  
 String[] str = line.split(":");  
 Discipline d = new Discipline(Integer.parseInt(str[0]), str[1]);  
 disciplines.add(d);  
 }  
   
 Scanner subscriptionReader = new Scanner(new File("data/subscriptions.txt"));  
 while (subscriptionReader.hasNextLine()){  
 int id = subscriptionReader.nextInt();  
 String type = subscriptionReader.next();  
 double price = subscriptionReader.nextDouble();  
 Subscription s = new Subscription(id, type, price);  
 subscriptions.add(s);  
 }  
   
 //members and the subclass, competitive members, is stored in the same file  
 //by splitting each text-line with a delimiters we can have more 'tokens' for our competitive members  
 Scanner memberReader = new Scanner(new File("data/members.txt"));  
 while (memberReader.hasNextLine()) {  
 String line = memberReader.nextLine();  
 //split the line into a string-array foreach '#'  
 String[] comp = line.split("#");  
 //the first string in the array is the info for the super class, split that once more with another delimiter so we can use the parameters to create member objects.  
 String[] str = comp[0].split(":");  
   
 //if member is not competitive  
 if (!Boolean.parseBoolean(str[6])) {  
 Member m = new Member(str[0], str[1], str[2], str[3], str[4], Boolean.parseBoolean(str[5]), Boolean.parseBoolean(str[6]), Double.parseDouble(str[7]), subscriptions);  
 members.add(m);  
 //if member is competitive  
 }   
 else {  
 //a competitive member has one or more disciplines, we store those disciplines in the text files as a number  
 //each digit in that number represents a discipline id, so we split the digits and collect each discipline from the list and add them to a new list for the member  
 ArrayList<Discipline> activeDisciplines = new ArrayList<>();  
 for (int i = 0; i < comp[1].length(); i++) {  
 activeDisciplines.add(disciplines.get(Character.getNumericValue(comp[1].charAt(i))));  
 }  
 CompetitiveMember m = new CompetitiveMember(str[0], str[1], str[2], str[3], str[4], Boolean.parseBoolean(str[5]), Boolean.parseBoolean(str[6]), Double.parseDouble(str[7]), subscriptions, teams, activeDisciplines);  
 members.add(m);  
 }  
 }  
 //records and the subclass competitive records is also stored in the same file  
 //using the same trick as before we split the text-lines with delimiters  
 Scanner recordScanner = new Scanner(new File("data/records.txt"));  
 while(recordScanner.hasNextLine()) {  
 String line = recordScanner.nextLine();  
 String[] comp = line.split("#");  
 String[] str = comp[0].split(":");  
 //a record holds a member object, pointing to the member who set the record  
 //in the textfile this connection is stored as the cpr of that member  
 //search the memberlist for the cpr string and get the correct member object  
 CompetitiveMember recordHolder = null;  
 for (Member m : members) {  
 if (str[1].equals(m.getCpr())) {  
 recordHolder = (CompetitiveMember)m;  
 break;  
 }  
 }  
 //a record also has a discipline  
 //using the stored id of the discipline, get that discipline object  
 Discipline discipline = null;  
 for (Discipline d : disciplines) {  
 if (Integer.parseInt(str[2]) == d.getId()) {  
 discipline = d;  
 break;  
 }  
 }  
 //if the record is not from a competition  
 if (!Boolean.parseBoolean(str[5])) {  
 Record r = new Record(Integer.parseInt(str[0]), recordHolder, discipline, Double.parseDouble(str[3]), str[4], Boolean.parseBoolean(str[5]));  
 records.add(r);  
 //if it is, use the subclass and load the extra parameters for the constructor  
 }   
 else {  
 CompetitionRecord r = new CompetitionRecord(Integer.parseInt(str[0]), recordHolder, discipline, Double.parseDouble(str[3]), str[4], Boolean.parseBoolean(str[5]), comp[1], Integer.parseInt(comp[2]));  
 records.add(r);  
 }  
 }  
 }  
   
 //Bertram  
 public int generateNextId(String objectToString) {  
 String[] str = objectToString.split(":");  
 return (Integer.parseInt(str[0])+1);  
 }  
   
 public CompetitiveMember castCompetitiveMember(Member member) {  
 if (member.getIsCompetitive()) {  
 return (CompetitiveMember)member;   
 }  
 return null;  
 }  
   
 //Bertram  
 public ArrayList<Record> selectionSortRecords(ArrayList<Record> records) {  
 //convert the ArrayList to an array of same size, because its syntax is easier to handle for this swapping/sorting algorithm.  
 Record[] arr = records.toArray(new Record[records.size()]);  
 //loop the length of the array -1  
 for (int i = 0; i < arr.length-1; i++) {  
 int index = i;  
 //make another loop, this time through all the elements after [i], compare all record times and save the index of the fastest  
 for (int j = i+1; j < arr.length; j++) {  
 if (arr[j].getTime() < arr[index].getTime()) {  
 index = j;  
 }  
 }  
 //swap [i]'s position in the array with the fastest record  
 Record tempRecord = arr[index];  
 arr[index] = arr[i];  
 arr[i] = tempRecord;  
 }  
 //when the sorting is finished, convert the array back into an ArrayList and return it.  
 ArrayList<Record> sortedRecords = new ArrayList<>(Arrays.asList(arr));  
 return sortedRecords;  
 }   
}