**File handling introduction**

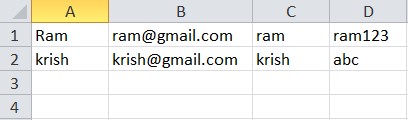
File handling is an important technique that you need to accustom to it. File reading and writing are types of handling. Let's practice file reading for now. There is a Class called FileReader that will help us with file reading. You'll be provided with a file that contains the data in CSV format. Using FileReader, read the file and parse the data contained in it to below specified format.  
  
Provided "**input.csv**" which have User details. Read all the user information stored in CSV format and create a user object by parsing the line. Add all the user objects to the ArrayList. At last, display the user list.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class called **User** with following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| name | String |
| email | String |
| username | String |
| password | String |

Include getters and setters.  
Create a default constructor and parameterized constructor.  
Format for the parameterized constructor is **User(String name, String email, String username, String password)**  
  
Create **UserBO** class with following methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| public List<User> readFromFile(BufferedReader br) | This method accepts the BufferedReader object as input and parses the data in the file to User objects and adds them to a list. Finally, it returns the list of User objects. |
| public void display(List<User> list) | This method accepts a list of User objects and displays the user details by iterating the list. Use **"%-15s %-20s %-15s %s\n"**to print the details. |

Create a driver class called **Main**. If the List of Users is empty print "**The list is empty**" in the main method. Else display the user detail by calling the display method.  
  
**Note :** Use BufferedReader br=new BufferedReader(new FileReader("input.csv")) for file reading.  
  
**Input format:**  
Read the input from the **"input.csv"** file which contains the user details.  
  
**Output format:**  
Use **"%-15s %-20s %-15s %s\n"**to print statements for the heading of the details in the Main method.  
  
**Sample Input: (input.csv)**



**Sample Output :**  
  
Name            Email                Username        Password  
Ram             ram@gmail.com        ram             ram123  
krish           krish@gmail.com     krish           abc

Input,csv

Ram,ram@gmail.com,ram,ram123

krish,krish@gmail.com,krish,abc

user.java

public class User {

private String name;

private String email;

private String username;

private String password;

public User() {

}

public User(String name, String email, String username, String password) {

super();

this.name = name;

this.email = email;

this.username = username;

this.password = password;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

Main.java

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.List;

public class Main {

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

BufferedReader br = new BufferedReader(new FileReader("input.csv"));

UserBO userBO = new UserBO();

List<User> userList = userBO.readFromFile(br);

if (userList.isEmpty()) {

System.out.println("The list is empty");

} else {

String name="Name", email="Email", username="Username", password="Password" ;

System.out.printf("%-15s %-20s %-15s %s\n", name, email, username, password);

userBO.display(userList);

}

}

}

userBo.java’

import java.io.BufferedReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class UserBO {

public List<User> readFromFile(BufferedReader br) throws IOException {

List<User> userList = new ArrayList<User>();

User user;

try {

String line;

while ((line = br.readLine()) != null) {

String[] tokens = line.split(",");

user = new User();

user.setName(tokens[0]);

user.setEmail(tokens[1]);

user.setUsername(tokens[2]);

user.setPassword(tokens[3]);

userList.add(user);

}

br.close();

} catch (IOException ex) {

ex.printStackTrace();

}

return userList;

}

public void display(List<User> list) {

for (User user : list) {

System.out.printf("%-15s %-20s %-15s %s\n", user.getName(), user.getEmail(), user.getUsername(), user.getPassword());

}

}

}Top of Form

Bottom of Form

**File Writing**

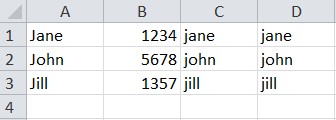
The file we write can be of several formats. But for now, we are just going to write a CSV text file, in which all the fields are separated by comma delimiter. Use FileWriter and BufferedWriter to write the data to a file.  
  
As a first thing, we are gonna create a file that contains the record of all the users registered. So write a program that can write all the user details from the console into a file "**output.csv**".  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class **User** with the following attributes,

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| name | String |
| mobileNumber | String |
| username | String |
| password | String |

Create a class **UserBO** with the following methods,

|  |  |
| --- | --- |
| **Method** | **Description** |
| public static void writeFile(ArrayList<User> userList, BufferedWriter bw) | This method gets a list of the user as argument and writes all the user details in the list into a file |

Create a driver class **Main** and use the main method to get the details from the user.  
  
**Input format:**  
The first line of input consists of an integer that corresponds to the number of users.  
The next n line of input consists of user details in  the CSV format (name, mobileNumber, username, password)  
Refer to sample Input for other further details.  
  
**Output format:**  
Write the user details in the output.csv file.  
Refer to sample Output for other further details.  
  
**Sample Input:**  
**[All Texts in bold corresponds to the input and rest are output]**  
  
Enter the number of users:  
**3**  
Enter the details of user :1  
**Jane,1234,jane,jane**  
Enter the details of user :2  
**John,5678,john,john**  
Enter the details of user :3  
**Jill,1357,jill,jill**

**Sample Output: (output.csv)  
  
**

Top of Form

Bottom of Form

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.util.ArrayList;

import java.util.Scanner;

public class Main {

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

ArrayList<User> userList = new ArrayList<>();

Scanner s = new Scanner(System.in);

System.out.println("Enter No of User");

int no = s.nextInt();

s.nextLine();

User user = null;

for (int i = 0; i < no; i++) {

System.out.println("Enter the details of user:" + (i + 1));

String input = s.nextLine();

String[] elements = input.split(",");

user = new User();

user.setName(elements[0]);

user.setMobileNumber(elements[1]);

user.setUsername(elements[2]);

user.setPassword(elements[3]);

userList.add(user);

user = null;

}

FileWriter fw = new FileWriter("output.csv");

BufferedWriter bw = new BufferedWriter(fw);

UserBO.writeFile(userList, bw);

}

}

import java.io.BufferedWriter;

import java.util.ArrayList;

public class UserBO {

public static void writeFile(ArrayList<User>userList, BufferedWriter bw) throws Exception {

for(User u:userList)

{

bw.write(u.getName()+",");

bw.write(u.getMobileNumber()+",");

bw.write(u.getUsername()+",");

bw.write(u.getPassword());

bw.write("\n");

}

bw.flush();

bw.close();

}

}

public class User {

String name;

String mobileNumber;

String username;

String password;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getMobileNumber() {

return mobileNumber;

}

public void setMobileNumber(String mobileNumber) {

this.mobileNumber = mobileNumber;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

**Hall details**

In the last problem, we read a file and got its contents. Now we are gonna write some contents into our desired file. Writing details in a file is so important as the file can be easily transferred to whenever we want. So in our application for the exhibition, we have many many halls. So a file with all the hall details should be fine for future reference. So get the hall details in CSV format in the console and store them as the list of objects. Then write this list into the file "output.txt".  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

Create a class **Hall**with the following private attributes,

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| name | String |
| contact | String |
| costPerDay | Double |
| owner | String |

Create default constructor and a parameterized constructor with arguments in order Hall(String name, String contact, Double costPerDay, String owner).  
Include appropriate getters and setters.  
  
Create the following methods in the **Hall** class,

|  |  |
| --- | --- |
| **Method** | **Description** |
| static void writeHallDetails(List<Hall> halls) | In this method, get the list of hall details as parameters. write the hall details to hall.csv file as comma-separated in the given order, (name,contact,costPerDay,owner) |

Read a list of halls and write the hall details to the **hall.csv** file.  
  
Create a driver class **Main** to test the above classes.  
  
**Input Format:**  
The next line of the input corresponds to the total number of halls 'n'.  
The next 'n' line of input contains hall details (name, contact,costperday, and owner separated by comma[,]).  
  
**Output Format:**  
The output is **hall.csv** file with hall details.  
The hall.csv file consists of hall details separated by commas[,] in order (name,contact,costPerDay, owner).  
Refer to sample output for formatting specifications.  
  
**[All text in bold corresponds to input and rest corresponds to output]  
Sample Input:**  
Enter the number of halls:  
**3  
Party hall,9876543210,4000.0,Jarviz  
Disco hall,9876543201,5000.0,Starc  
Dining hall,9873216540,3000.0,Chris**  
  
**Sample Output:**  
hall.csv  
Party hall,9876543210,4000.0,Jarviz  
Disco hall,9876543201,5000.0,Starc  
Dining hall,9873216540,3000.0,Chris

import java.io.FileWriter;

import java.io.IOException;

import java.util.List;

public class Hall {

private String name;

private String contact;

private double costPerDay;

private String owner;

public Hall(String name, String contact, double costPerDay, String owner) {

super();

this.name = name;

this.contact = contact;

this.costPerDay = costPerDay;

this.owner = owner;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getContact() {

return contact;

}

public void setContact(String contact) {

this.contact = contact;

}

public double getCostPerDay() {

return costPerDay;

}

public void setCostPerDay(double costPerDay) {

this.costPerDay = costPerDay;

}

public String getOwner() {

return owner;

}

public void setOwner(String owner) {

this.owner = owner;

}

/\*\*

\* n this method, get the list of hall details as parameters. write the hall

\* details to hall.csv file as comma-separated in the given order,

\* (name,contact,costPerDay,owner)

\*

\* @param halls

\* @throws IOException

\*/

public static void writeHallDetails(List<Hall> halls) throws IOException {

FileWriter fi = new FileWriter("hall.csv", false);

for (Hall s : halls) {

fi.write(s.getName() + "," + s.getContact() + "," + s.getCostPerDay() + "," + s.getOwner());

fi.write("\n");

}

fi.close();

}

}

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Main {

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

List<Hall> hallsList = new ArrayList<Hall>();

Scanner s = new Scanner(System.in);

System.out.println("Enter the number of halls:");

int no = s.nextInt();

s.nextLine();

Hall hall = null;

for (int i = 0; i < no; i++) {

String input = s.nextLine();

String[] elements = input.split(",");

hall = new Hall(elements[0], elements[1], Double.parseDouble(elements[2]), elements[3]);

hallsList.add(hall);

hall = null;

}

Hall.writeHallDetails(hallsList);

}

}

**Fixed Length format**

There are many types of files available, each has its own advantages and disadvantages of using. In this problem, we are gonna use Fixed Length Format files. In these files, each field has a fixed length. If a variable of the field is shorter than its size, it is filled with space so that all variables of a particular field is of the same size. By this, we can get all the variables of the field using **substring()** and **trim()** methods.  
  
When the file is given as input, read each line, split it into a number of fields using substring() as per the size of each field then use trim() to get only the variable and removing the unwanted spaces. Note that the trim() method only removes spaces in the starting and ending of the word and not in between.  
  
Now in our application, it is normal to have all the records stored in a file. When we want to enter the content of the file into the application, we cannot enter each detail in the console one by one. So we must develop a program that can parse the details in the file into required objects.  
  
Now we have got a situation where all the event details we want are in a file. So we want to read all the details in it and store them as objects.   
  
Write a program to do this and also to display the events organized by specific persons.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class **Event** with the following private attributes,

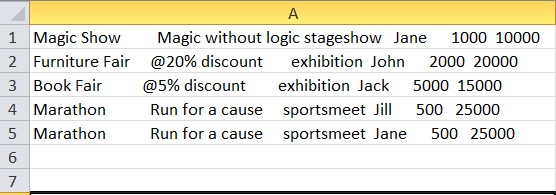
|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| name | String |
| detail | String |
| type | String |
| organiser | String |
| attendeesCont | Integer |
| projectedExpense | Double |

Include appropriate getter/setter, default constructor and parameterized constructor.  
  
Override **toString()** and print the details in the tabular form.  
  
Create a class **EventBO** with the following **public static** methods,

|  |  |
| --- | --- |
| **Method** | **Description** |
| ArrayList<Event> readFile(BufferedReader br) | this method reads the lines from the file and stores it as ArrayList of Event Object It returns the ArrayList |
| ArrayList<Event> eventsByPerson(ArrayList<Event> eventList, String organiser) | this method takes the list of events and organizer name It returns a list of events that are organized by that organizer. |

Create a driver class **Main** and use the main method to get the file and details from the user.  
  
**Input format:**  
The first line of the input is the name of the organiser.  
Read all the details of that organiser from the input file.  
The input file is "**input.csv**". The sizes of fields in the fixed-length file are,

|  |  |
| --- | --- |
| **Field** | **Size** |
| name | 0-19 |
| details | 19-39 |
| type | 39-51 |
| organiser | 51-61 |
| attendeesCount | 61-67 |
| projectedExpense | 67-74 |

**Output format:**  
Use "**%-15s%-20s%-15s%-15s%-15s**" to display details in tabular format.  
Print "**The given person has no upcoming events**" if the person is not in the file.  
Refer to sample output for other further details and format of the output.  
  
**Sample Input file: (input.csv)  
  
**  
  
**[All Texts in bold corresponds to the input and rest are output]  
Sample Input and Output 1:**  
  
Enter the name of the person whose events to be shown:  
**Jane**  
Name           Detail              Type           Attendees CountProjected Expense  
Magic Show     Magicwithoutlogicstageshow      1000           10000.0          
Marathon       Run for a cause    sportsmeet     500            25000.0          
Do you want to continue?(y/n)  
**y**  
Enter the name of the person whose events to be shown:  
**Jack**  
Name           Detail              Type           Attendees CountProjected Expense  
Book Fair      @5% discount        exhibition     5000           15000.0          
Do you want to continue?(y/n)  
**y**  
Enter the name of the person whose events to be shown:  
**Jim**  
The given person has no upcoming events  
Do you want to continue?(y/n)  
**n**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

public class Main {

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

//write your code here

Scanner sc = new Scanner(System.in);

String ch = null;

BufferedReader br = new BufferedReader(new FileReader("input.csv"));

ArrayList<Event> e = EventBO.readFile(br);

do {

System.out.println("Enter the name of the person whose events to be shown:");

String s = sc.nextLine();

ArrayList<Event> e1 = EventBO.eventsByPerson(e, s);

if(e1.isEmpty()) {

System.out.println("The given person has no upcoming events");

}else {

String name ="Name",detail="Detail",type="Type",attendees="Attendees Count",cpe="Projected Expense";

System.out.printf("%-15s%-20s%-15s%-15s%-15s\n" ,name,detail,type,attendees,cpe);

for(int i=0;i<e1.size();i++) {

System.out.print(e1.get(i).toString());

}

}

System.out.println("Do you want to continue?(y/n)");

ch = sc.nextLine();

}while(ch.equals("y"));

}

}

public class Event {

//write your code here

String name;

String detail;

String type;

String organiser;

int attendeesCont;

double projectedExpense;

public Event(String name, String detail, String type, String organiser, int attendeesCont,

double projectedExpense) {

super();

this.name = name;

this.detail = detail;

this.type = type;

this.organiser = organiser;

this.attendeesCont = attendeesCont;

this.projectedExpense = projectedExpense;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDetail() {

return detail;

}

public void setDetail(String detail) {

this.detail = detail;

}

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public String getOrganiser() {

return organiser;

}

public void setOrganiser(String organiser) {

this.organiser = organiser;

}

public int getAttendeesCont() {

return attendeesCont;

}

public void setAttendeesCont(int attendeesCont) {

this.attendeesCont = attendeesCont;

}

public double getProjectedExpense() {

return projectedExpense;

}

public void setProjectedExpense(double projectedExpense) {

this.projectedExpense = projectedExpense;

}

@Override

public String toString() {

return String.format("%-15s%-20s%-15s%-15s%-15s\n",this.getName(),this.getDetail(),this.type,this.attendeesCont,this.projectedExpense);

}

}

import java.io.BufferedReader;

import java.util.ArrayList;

public class EventBO {

public static ArrayList<Event> readFile(BufferedReader br) throws Exception {

ArrayList<Event> e = new ArrayList<Event>();

Event event;

String line;

while ((line = br.readLine()) != null) {

String name = line.substring(0, 19).trim();

String detail = line.substring(19, 39).trim();

String type = line.substring(39, 51).trim();

String organiser = line.substring(51, 61).trim();

String attendeesCont = line.substring(61, 67).trim();

String projectedExpense = line.substring(67, 74).trim();

event = new Event(name, detail, type, organiser, Integer.parseInt(attendeesCont), Double.parseDouble(projectedExpense));

e.add(event);

}

br.close();

return e;

//write your code here

}

public static ArrayList<Event> eventsByPerson(ArrayList<Event> eventList,String organiser) {

ArrayList<Event> e = new ArrayList<Event>();

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

if(eventList.get(i).getOrganiser().equals(organiser)) {

e.add(eventList.get(i));

}

}

return e;

//write your code here

}

}

**ItemType details in a file**

Let's get more practice in reading data from fixed-length format files. You are provided with ItemType details in "**input.txt**". Write a program that reads the file and displays the contents of the file in the required format. The length and position of attributes are as follows.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Length** | **Position** |
| name | 15 | 1 |
| deposit | 7 | 16 |
| costPerDay | 5 | 22 |

**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

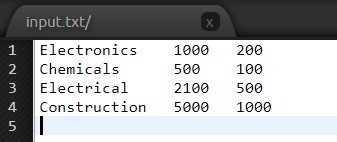
Create an **ItemType**class with following private attributes

|  |  |
| --- | --- |
| **Attribute** | **Datatype** |
| name | String |
| deposit | Double |
| costPerDay | Double |

Include appropriate default constructor, getters and setters for the attributes of the Item class.  
Include a parameterized Constructor for the ItemType class.

Create an **ItemTypeBO**class with the following methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| public List<ItemType> readFromFile(BufferedReader br) | This method accepts the BufferedReader object as input and parses the data in the file to ItemType objects and adds them to a list. Finally, it returns the list of ItemType objects. |
| public List<ItemType> depositList(List<ItemType> list) | This method accepts a list of ItemType objects and returns the list of ItemType objects with deposit amount greater than 2000 |
| public void display(List<ItemType> list) | This method accepts a list of ItemType objects and displays the ItemType details by iterating the list. Refer to the sample Output for the formatting specifications. |

Create a driver class called **Main**. If the list of ItemType objects that have a deposit amount greater than 2000, then print "**No item type has deposit more than 2000**" in the main method. Else print the details of the ItemType.  
  
**Input format:**  
Read the details from the input file "**input.txt**"  
  
**Output format:**  
Use "**%-15s %-15s %s\n**" for displaying ItemType details in tabular form in the console  
Print one digit after the decimal for the double datatype.  
Print the statement for the heading of the details is present in the main method.  
Refer to the sample output for the formatting specifications.  
  
**Sample Input:**  
  
  
  
**Sample Output :**  
  
Item type       Deposit         Cost per day  
Electrical      2100.0          500.0  
Construction    5000.0          1000.0

 import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.List;

public class Main {

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

BufferedReader br = new BufferedReader(new FileReader("input.txt"));

ItemTypeBO it = new ItemTypeBO();

List<ItemType> itList = it.readFromFile(br);

List<ItemType> itList1 = it.depositList(itList);

if(itList1.isEmpty()) {

System.out.println("No item type has deposit more than 2000");

}else {

String type ="Item type",dep="Deposit",cost="Cost per day";

System.out.printf("%-15s %-15s %s\n" ,type,dep,cost);

it.display(itList1);

}

}

}

Top of Form

Bottom of Form

public class ItemType {

//Your code here

String name;

double deposit;

double costPerDay;

public ItemType() {

super();

}

public ItemType(String name, double deposit, double costPerDay) {

super();

this.name = name;

this.deposit = deposit;

this.costPerDay = costPerDay;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getDeposit() {

return deposit;

}

public void setDeposit(double deposit) {

this.deposit = deposit;

}

public double getCostPerDay() {

return costPerDay;

}

public void setCostPerDay(double costPerDay) {

this.costPerDay = costPerDay;

}

}

Electronics 1000 200

Chemicals 500 100

Electrical 2100 500

Construction 5000 1000

import java.io.BufferedReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class ItemTypeBO {

//Your code here

public List<ItemType> readFromFile(BufferedReader br){

List<ItemType> itList = new ArrayList<ItemType>();

ItemType it;

try {

String line;

while ((line = br.readLine()) != null) {

String sp[] = line.split("\\s+");

it = new ItemType(sp[0], Double.parseDouble(sp[1]), Double.parseDouble(sp[2]));

itList.add(it);

}

br.close();

} catch (IOException ex) {

ex.printStackTrace();

}

return itList;

}

public List<ItemType> depositList(List<ItemType> list){

List<ItemType> itList = new ArrayList<ItemType>();

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

if(list.get(i).getDeposit()>2000) {

itList.add(list.get(i));

}

}

return itList;

}

public void display(List<ItemType> list) {

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

System.out.printf("%-15s %-15s %s\n" ,list.get(i).getName(),list.get(i).getDeposit(),list.get(i).getCostPerDay());

}

}

}

**Serialize Customer**

During IO Journey, we started off by character level read-write and then byte level read-write. Java also has yet another unique option of persisting data / objects. An Object can be directly saved into a file and can be later reloaded. This process is called Serialization and the restoration is called De-serialization. The extension of the file is ".ser".  
An Object is considered ready for serialization if its class implements Serializable Interface.  
  
Serializable Interface is called Marker Interface. A Marker Interface is one which has no methods and member variables, but just indicates / marks that the given class is implementing the Interface.  
Use ObjectOutputStream class for converting an object to serialized file.  
  
Quickly, write a code to serialize a customer object, get the customer details from the console and store the customer object into a file called **“Serialize.ser”**. Class must **implement Serializable.**

**Input and Output Format:**

Refer sample input and output for formatting specifications.

**[All text in bold corresponds to input and the rest corresponds to output]**

Enter the Customer name

**Amarnath**

Enter the Customer number

**12**

Enter the Customer address

**Coimbatore**

**Read and Write**

In our application, we are gonna read the contents of a file and write it into another file with some conditions. There can be many events for the same organization or owner. But we want only one event for every owner available. So we are gonna read the event details from the file **"input.txt"** and then write the event details into "**output.txt**" but after removal of duplicate entries.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

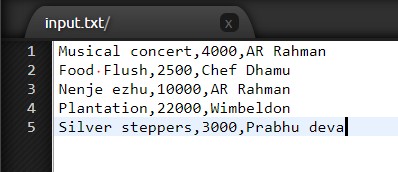
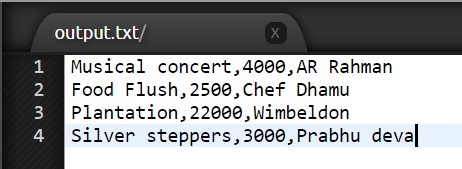
Create a class **Event**with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| eventName | String |
| attendeesCount | Integer |
| ownerName | String |

Include appropriate getters and setters  
Create default constructor and a parameterized constructor with arguments in order**Event(String eventName, Integer attendeesCount, String ownerName).**  
  
Create a class **EventBO**to access the above class. Include the following public methods.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public List<Event> readFromFile(BufferedReader br) | This method accepts the BufferedReader object as input and parses the data in the file to Event objects and then adds them to a list. Finally, it returns the list of Event objects. |
| void writeFile(List<Event> eventList,FileWriter fr) | This method takes event list and file writer as the arguments and writes into "**output.txt**" with the removal of duplicate event details (i.e) an event having the same owner name. |

Create a driver class named **Main** which reads data from console and to test the above class.

**Input:**  
Read the contents[event details] from the file **"input.txt".**  
  
**Output:**  
Write the contents[event details] with the removal of duplicate events into the **"output.txt".  
  
Sample Input: (input.txt)**  
  
  
  
  
**Sample Output: (output.txt)  
  
**

**Item Count**

We have got ourselves into a situation here. We have the item list in a fixed-length format file. But for manipulation and record maintenance we want it as objects. So we need a program that can read the file and create the item with the itemtype that we already have. We want the list in the file in a consolidated form that gives only the itemtype and count of items with that itemtype. So write a program to do the requirement and hard code the itemtype list.  
  
Use a treemap so that the order of the itemtype remains the same.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**  
  
Create a class **Item** with the following attributes,

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| number | String |
| itemType | ItemType |
| vendor | String |

Create a class **ItemType** with the following attributes,

|  |  |
| --- | --- |
| **Attribute** | **Data type** |
| name | String |
| deposit | Double |
| costPerDay | Double |

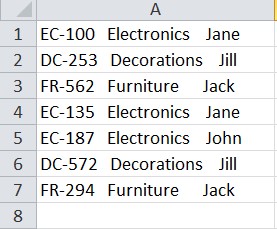
Include appropriate getter/setter, default constructor and parameterized constructor. Prefill ItemType with some data.  
  
Create a class with **ItemBO** with the following **public static** methods,

|  |  |
| --- | --- |
| **Method** | **Description** |
| ArrayList<Item> readFile(BufferedReader br,ArrayList<ItemType> typeList) | This method takes the list of itemtype as argument, then reads the file and create a list of item objects and returns the list |
| TreeMap<String,Integer> listItem(ArrayList<Item> itemList,ArrayList<ItemType> itemTypeList) | This method takes the list of items and itemtypes as argument and return a map which has name of the itemtype as key and the count of items which belongs to that particular itemtype as value |

Create a driver class **Main** and use the main method to get the file and details from the user.  
  
**Input format:**  
  
The input file is "**input.csv**". The sizes of fields in the fixed-length file are,

|  |  |
| --- | --- |
| **Field** | **Size** |
| name | 0-9 |
| itemType name | 9-24 |
| vendor | 24-34 |

**Output format:**  
Print the name of the ItemType and its total count.

**Sample Input: (input.csv)  
  
​​​​​​​**  
  
  
  
**Sample Output:**  
  
The item types along with count of each:  
Decorations--2  
Electronics--3  
Furniture--2

**Item Details**

File reading and writing skills are indispensable for a programmer. Your skills will increase in file handling as you progress through this module and learn about the essence of file handling. In this example, read the given file and process the information present in it. The input consists of item details, you have to display the details in tabular format.  
  
**Strictly adhere to the Object-Oriented specifications given in the problem statement. All class names, attribute names and method names should be the same as specified in the problem statement.**

Create a class Item with the following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| itemNumber | Integer |
| vendor | String |
| itemType | ItemType |

Include appropriate getters and setters  
Create default constructor and a parameterized constructor with arguments in order**Item(Integer itemNumber, String vendor,ItemType itemType).**

Create a class ItemType with following private attributes

|  |  |
| --- | --- |
| **Attributes** | **Datatype** |
| itemTypeName | String |
| cost | Double |

Include appropriate getters and setters  
Create a default constructor and a parameterized constructor with arguments in order**ItemType(String itemTypeName, Double cost).**

Create a class ItemBO to access the above class. Include the following public methods.

|  |  |
| --- | --- |
| **Method** | **Description** |
| List<Item> readFile(BufferedReader br) | This method takes BufferedReader that contains a CSV file **"input.csv"** which consists of (Item Number,Vendor,Item Type Name,Cost). It store item details it in a list and returns it. |

Create a driver class named **Main**to test the above class.

**Input Format:**  
It reads details from "**input.csv**"  
Refer to sample input for formatting specifications.  
  
**Output Format:**  
The output consists of item details in the format of **System.out.format("%-20s %-20s %-20s %s\n", "Item Number","Vendor Name","Item Type","Cost").**  
Refer to sample output for formatting specifications.  
  
**Sample Input : (input.csv)**  
input.csv  
1,Martin,Leather,2000  
15,Louis,Mud,1300.50  
23,Jaques kallis,Electronics,100  
38,McCullum,Wood,3000.4  
  
**Sample Output1:**  
Item Number          Vendor Name          Item Type          Cost  
1                              Martin                     Leather             2000.0  
15                            Louis                       Mud                  1300.5  
23                            Jaques kallis           Electronics        100.0  
38                            McCullum                Wood                3000.4

**IO - Simple File Write**

Write a Java program to record the airport details into the file. Get the airport details name, cityName and countryCode from the user and write that information in a comma-separated format in a file (CSV).

Below is the format of the output file.

<name>,<cityName>,<countryCode>

**eg: Cochin International Airport,Cochin,IN**

Create a main class **"Main.java"**

**Input and Output Format:**

Get the airport details name, cityName and country code from the user.  
Print the airport details in the output file **airport.csv**

**Sample Input :**  
**[All text in bold corresponds to input and the rest corresponds to output]**

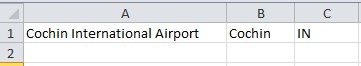
Enter the name of the airport

**Cochin International Airport**

Enter the city name

**Cochin**

Enter the country code

**IN**  
  
**Sample Output:  
  
**

Top of Form

Bottom of Form