**Playlist - Requirement 6**

**Requirement 6:**

In this requirement, predict the state of mind of the user. The state of mind depends on the song type the user listened recently. Given the list of songs, he heard recently predict the mind state of the user.   
  
A user is happy if he has heard many songs of type Celebration.   
A user is depressed if he has heard many songs of type Emotional.   
A user is energetic if he has heard many songs of type Motivational.   
  
a) Create a **Song**Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_name | string |
| \_artist | string |
| \_songType | string |
| \_dateDownloaded | DateTime |
| \_rating | double |
| \_numberOfDownloads | int |

Mark all the attributes as private.  
Include appropriate properties.  
Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Song(string \_name, string \_artist, string \_songType,DateTime \_dateDownloaded, double \_rating, int \_numberOfDownloads)**   
  
b) Create the following static methods in **Song** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static Dictionary<string,int> CalculateTypeCount(List<Song> list) | This method accepts a list of Songs as arguments and create a Dictionary with the songType as key and number of songs for the particular songType as value and return the Dictionary. |
| static string PredictState(Dictionary<string,int> perTypeCount) | This method accepts the song type Dictionary as argument and returns the state of mind as given below, if song type is Emotional -> return "depressed" if song type is Celebration -> return "happy" if song type is Motivational -> return "energetic" |

The input format of Song details is separated by comma in the below order,  
(\_name, \_artist, \_songType, \_dateDownloaded, \_rating, \_numberOfDownloads)   
  
**Note:** The statement "**The user is feeling [state of mind]"**is displayed in the main method. Where **[state of mind]** represents the mind state of the user, which is returned by predictState method.   
  
**Sample INPUT & OUTPUT 1:**  
  
Enter the number of songs  
**4  
Here comes the sun,The Beatles,Motivational,20-01-2018,4.5,150000  
Don't Give Up,Peter Gabriel,Motivational,15-12-2017,4.7,10000  
Let It Be,The Beatles,Motivational,11-01-2010,4.8,500000  
Mad World,Gary Jules,Emotional,12-09-2017,4,10000**  
The user is feeling energetic  
  
**Sample INPUT & OUTPUT 2:**  
  
Enter the number of songs  
**5  
Mad World,Gary Jules,Emotional,12-09-2017,4.2,15000  
Someone Like You,Adele,Emotional,15-12-2017,4.7,10000  
Hurt,Johnny Cash,Emotional,11-01-2010,4.8,500000  
The Way,Fastball,Emotional,12-09-2017,4,10000  
Here comes the sun,The Beatles,Motivational,20-01-2018,4.5,150000**  
The user is feeling depressed

**Customer Table Booking - Requirement 2**

**Requirement 2:**

One of the main features of any application is searching. In this requirement, you need to search customers based on \_name, \_birthdate, and \_rating.  
  
a) Create a **Customer**Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_id | long |
| \_name | string |
| \_mobileNumber | string |
| \_birthdate | DateTime |
| \_averageSpendAmount | double |
| \_totalAmount | double |
| \_dateEnrolled | DateTime |
| \_rating | double |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Customer(long \_id, string \_name, string \_mobileNumber, DateTime \_birthdate, double \_averageSpendAmount, double \_totalAmount, DateTime \_dateEnrolled, double \_rating)**  
  
b) Create the following static methods in the **CustomerBO** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static Customer CreateCustomer(string line) | This method accepts a string as argument. Comma-separated Customer detail is passed to this method. Split the value then create a customer object and return the customer object. |
| static List<Customer> FindCustomer(List<Customer> customerList,string name) | This method accepts customer list and a customer name as arguments. Find the list of customers with given name and return the list. If no customers found with the given name return null, then print "**No customers found with the given name**". |
| static List<Customer> FindCustomer(List<Customer> customerList,DateTime birth) | This method accepts customer list and birth date as arguments. Find the list of customers with the given birth date and return the list. If no customers found with the given birth date return null, then print "**No customers found with the given birth date**". |
| static List<Customer> FindCustomer(List<Customer> customerList,double rating) | This method accepts customer list and rating as arguments. Find the list of customers with the given rating value and return the list. If no customers found with the given rating return null, then print "**No customers found with the given rating**". |

The input format consists of customer details separated by comma in the below order,  
(\_id, \_name, \_mobileNumber, \_birthdate, \_averageSpendAmount, \_totalAmount, \_dateEnrolled, \_rating)  
  
When the “customer” object is printed, it should display the following format  
Print format:  
**Console.WriteLine("{0} {1,10} {2,15} {3,15} {4,15} {5,15} {6,15} {7,15}", "Id", "Name", "Mobile Number", "Date of Birth", "Average spent amount", "Total amount", "Date Enrolled", "Rating");**  
  
**Sample INPUT & OUTPUT 1:**  
  
Enter the number of customers:  
**3  
1,John,9876543210,12-12-1990,5000,25000,12-12-2012,3  
2,James,9876543201,12-12-1991,6000,35000,12-12-2013,4  
3,John,9567843201,14-09-1987,6000,35000,12-12-2013,4**  
Enter the search type:  
1.By name  
2.By birth date  
3.By rating  
**1**  
Enter the name of the customer to be searched:  
**John**  
Id    Name            Mobile Number   Date of Birth   Average spent amount Total amount    Date Enrolled   Rating  
1     John            9876543210      12-12-1990      5000.0               25000.0         12-12-2012      3.0  
3     John            9567843201      14-09-1987      6000.0               35000.0         12-12-2013      4.0  
  
**Sample INPUT & OUTPUT 2:**  
  
Enter the number of customers:  
**3  
1,John,9876543210,12-12-1990,5000,25000,12-12-2012,3  
2,James,9876543201,12-12-1991,6000,35000,12-12-2013,4  
3,Parker,9567843201,14-09-1987,6000,35000,12-12-2013,4**  
Enter the search type:  
1.By name  
2.By birth date  
3.By rating  
**3**  
Enter the rating of the customer to be searched:  
**4**  
Id    Name            Mobile Number   Date of Birth   Average spent amount Total amount    Date Enrolled   Rating  
2     James           9876543201      12-12-1991      6000.0               35000.0         12-12-2013      4.0  
3     Parker          9567843201      14-09-1987      6000.0               35000.0         12-12-2013      4.0  
  
**Sample INPUT & OUTPUT 3:**  
  
Enter the number of customers:  
**3  
1,John,9876543210,12-12-1990,5000,25000,12-12-2012,3  
2,James,9876543201,12-12-1990,6000,35000,12-12-2013,4  
3,Parker,9567843201,14-09-1987,6000,35000,12-12-2013,4**  
Enter the search type:  
1.By name  
2.By birth date  
3.By rating  
**2**  
Enter the birth date of the customer to be searched:  
**12-12-1990**  
Id    Name            Mobile Number   Date of Birth   Average spent amount Total amount    Date Enrolled   Rating  
1     John            9876543210      12-12-1990      5000.0               25000.0         12-12-2012      3.0  
2     James           9876543201      12-12-1990      6000.0               35000.0         12-12-2013      4.0  
  
**Sample INPUT & OUTPUT 4:**  
  
Enter the number of customers:  
**3  
1,John,9876543210,12-12-1990,5000,25000,12-12-2012,3  
2,James,9876543201,12-12-1991,6000,35000,12-12-2013,4  
3,John,9567843201,14-09-1987,6000,35000,12-12-2013,4**  
Enter the search type:  
1.By name  
2.By birth date  
3.By rating  
**1**  
Enter the name of the customer to be searched:  
**Starc**  
No customers found with the given name  
  
**Sample INPUT & OUTPUT 5:**  
  
Enter the number of customers:  
**3  
1,John,9876543210,12-12-1990,5000,25000,12-12-2012,3  
2,James,9876543201,12-12-1991,6000,35000,12-12-2013,4  
3,Parker,9567843201,14-09-1987,6000,35000,12-12-2013,4**  
Enter the search type:  
1.By name  
2.By birth date  
3.By rating  
**3**  
Enter the rating of the customer to be searched:  
**3.5**  
No customers found with the given rating  
  
**Sample INPUT & OUTPUT 6:**  
  
Enter the number of customers:  
**3  
1,John,9876543210,12-12-1990,5000,25000,12-12-2012,3  
2,James,9876543201,12-12-1990,6000,35000,12-12-2013,4  
3,Parker,9567843201,14-09-1987,6000,35000,12-12-2013,4**  
Enter the search type:  
1.By name  
2.By birth date  
3.By rating  
**2**  
Enter the birth date of the customer to be searched:  
**10-10-1998**  
No customers found with the given birth date

**Customer Table Booking - Requirement 3**

**Requirement 3:**  
   In this requirement, you need to validate the name and age of a customer. Also, you need to find if the customer is lucky or not.  
  
i)  The name should contain only alphabets and not numbers or special characters.  
  
ii)  The age is considered valid only if it is greater than equal to 18.(Take the current date as "01-01-2018")  
  
iii) A customer is said to be lucky if the repetitive sum of his mobile number is equal to 1.  
  
Ex:- Mobile number 9635285233 is **lucky.**  
          step1- 9+6+3+5+2+8+5+2+3+3 - 46  
         step 2 -  4+6 - 10  
         step 3 - 1+0 - 1  
  
a)Create a Class **Program** with the following static methods:

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static bool ValidateAge(DateTime birth) | This method accepts DateTime as argument and returns a boolean. Calculate the age and return **True**ifthe age of the customer is greater than or equal to 18, else return **False** |
| static bool ValidateName(string name) | This method accepts a String as argument and returns a boolean.Validate the name of the customer and return **True** if the name is valid, else return **False** |
| static bool ValidateLuckyCustomer(string mobile) | This method accepts a String as argument and returns a boolean.Check whether the mobile number of the customer is lucky and return **True**, else return **False** |

Print the following statements in the Main method.  
Print "Age is valid" if the age is greater than or equal to18, else print "Age is invalid".  
Print "Name is valid" if the name contains only alphabets (a-z) or (A-Z), else print "Name is invalid".  
Print "Lucky Customer" if the sum of the digits of the mobile number is 1, else print "Unlucky Customer".  
  
**Sample Input and Output 1:**  
  
1.Validate Age  
2.Validate Name  
3.Lucky Customer  
Enter your choice:  
**1**  
Enter birthdate:  
**25-12-1989**  
Age is valid  
  
**Sample Input and Output 2:**  
  
1.Validate Age  
2.Validate Name  
3.Lucky Customer  
Enter your choice:  
**1**  
Enter birthdate:  
**29-02-2000**  
Age is invalid  
  
**Sample Input and Output 3:**  
  
1.Validate Age  
2.Validate Name  
3.Lucky Customer  
Enter your choice:  
**2**  
Enter name:  
**Jane**  
Name is valid  
  
**Sample Input and Output 4:**  
  
1.Validate Age  
2.Validate Name  
3.Lucky Customer  
Enter your choice:  
**2**  
Enter name:  
**Jane Doe**  
Name is invalid  
  
**Sample Input and Output 5:**  
  
1.Validate Age  
2.Validate Name  
3.Lucky Customer  
Enter your choice:  
**3**  
Enter mobile number:  
**9597074311**  
Lucky Customer  
  
**Sample Input and Output 6:**  
  
1.Validate Age  
2.Validate Name  
3.Lucky Customer  
Enter your choice:  
**3**  
Enter mobile number:  
**9876543210**  
Unlucky Customer

**Customer Table Booking - Requirement 4**

**Requirement 4:**

In this requirement, build a feature in which user can book tables. If the table is not available suggest alternate tables in ascending order of table number.  
  
a) Create a **Booking**Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_id | long |
| \_customerName | string |
| \_table | Table |
| \_membersPresent | int |
| \_billamount | double |
| \_bookingTime | DateTime |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Booking(long \_id, string \_customerName, Table \_table, int \_membersPresent, double \_billamount, DateTime \_bookingTime)**  
  
b) Create a Table Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_id | long |
| \_number | string |
| \_capacity | string |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Table(long \_id, int \_number, int \_capacity)**  
  
c) The Table class should implement the IComparer interface, then implement **Compare** method, This Compare method sorts the table list based on table id.  
  
d) Create the following static methods in the **Table** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static Table CreateTable(string line) | This method will get a table detail split the value, then create a table object and return the table object. |

e) Create the following static methods in the **Booking** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static void CreateBooking(List<Table> tableList,List<Booking> bookingList,string line) | This method will get the table list, booking list, and a booking detail from the user and check the table is already booked or not, If the table is not booked create a Booking object and add it to the list and display "**Table successfully booked**", if it is already booked display the available table details. If no tables are available display "**Sorry, No tables available**". [The booking for the table is checked only on the basis of table number, not based on other factors] |

The input format of Table details are separated by comma in the below order,  
(\_id, \_number, \_capacity)  
  
The input format of Booking details is separated by comma in the below order,  
(\_id, \_customerName, \_tableNumber, \_membersPresent, \_billamount, \_bookingTime)  
  
When the “table” object is printed, it should display the following format  
Print format:  
**Console.WriteLine("{0} {1,15} {2,15}","ID","Number","Capacity");**  
  
The above print format is used to print the table details, this will be placed in CreateBooking method.  
  
**Sample INPUT & OUTPUT 1:**  
  
Enter the number of tables:  
**2  
101,10,10  
102,20,15**  
Enter the booking details:  
**10,John,10,8,1200,12-01-2018**  
Table successfully booked  
Do you want to continue(yes/no)  
**yes**  
Enter the booking details:  
**20,Peter,2,9,1300,12-01-2018**  
Sorry the table is not available  
The available tables are:  
ID    Number     Capacity  
102   20         15  
Do you want to continue(yes/no)  
**yes**  
Enter the booking details:  
**20,Peter,10,8,1300,12-01-2018**  
Sorry the table is not available  
The available tables are:  
ID    Number     Capacity  
102   20         15  
Do you want to continue(yes/no)  
**yes**  
Enter the booking details:  
**20,Peter,20,8,1300,12-01-2018**  
Table successfully booked  
Do you want to continue(yes/no)  
**no**

**Customer Table Booking - Requirement 5**

**Requirement 6:**  
In this requirement, find the lucky table of a given seater type. A lucky table is one which generated maximum revenue of a particular seater type.

a) Create a Booking Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_id | long |
| \_customerName | string |
| \_tableNumber | int |
| \_membersPresent | int |
| \_billamount | double |
| \_bookingTime | DateTime |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Booking(long \_id, string \_customerName, int \_tableNumber, int \_membersPresent, double \_billamount, DateTime \_bookingTime)**  
  
b) Create the follwing static methods in **Booking** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static void SortedDictionary<DateTime,double> CalculateDayBilling(List<Booking> bookings) | This method will get a list of booking objects as parameter.  Create a sortedDictionary which contains the booking date as key(DateTime) and total amount for the Booking(double) as value.  In this method iterate all the booking and put the booking date of the booking as key in the sortedDictionary and total amount of the booking as the value for the sortedDictionary and return that sortedDictionary . |

The input format of Booking details is separated by comma in the below order,  
(\_id, \_customerName, \_tableNumber, \_membersPresent, \_billamount, \_bookingTime)

Read the number of bookings and booking details and create a sortedDictionary using  booking date as key and total amount for the booking date as value and find the booking date with the maximum total amount

Create a class named as **Program**, which contains Main method. All the input and output operations are done this Main method.  
And also in Main method used to access the above classes to done this requirement.

Print the sortedDictionary details in the below format.  
Print format:  
**Console.WriteLine("{0} {1,15}","Date","Amount");**  
  
**Sample INPUT & OUTPUT:**  
  
Enter the number of booking details:  
**5  
10,John,10,8,1200,12-01-2018  
20,Peter,20,8,1300,13-01-2018  
30,Starc,30,4,1200,13-01-2018  
40,Mark,10,4,900,14-01-2018  
50,Jack,30,2,500,14-01-2018**  
Date          Amount  
12-01-2018            1200  
13-01-2018            2500  
14-01-2018            1400

Problem Description

**Customer Table Booking - Requirement 6**

**Requirement 6:**

In this requirement, find the lucky table of a given seater type. A lucky table is one which generated maximum revenue of a particular seater type.  
  
a) Create a **Booking**Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_id | long |
| \_customerName | string |
| \_table | Table |
| \_membersPresent | int |
| \_billamount | double |
| \_bookingTime | DateTime |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Booking(long \_id, string \_customerName, Table \_table, int \_membersPresent, double \_billamount, DateTime \_bookingTime)**  
  
b) Create a **Table**Class with the following private attributes:

|  |  |
| --- | --- |
| **Member Field Name** | **Type** |
| \_id | long |
| \_number | string |
| \_capacity | string |

Mark all the attributes as private, Create / Generate appropriate Getters & Setters, Add a default constructor and a parameterized constructor to take in all attributes in the given order: **Table(long \_id, int \_number, int \_capacity)**  
  
b) Create the following static methods in **Booking** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static Dictionary<int,double> CalculateTableBilling(List<Booking> bookings,int type) | This method will get a list of booking objects and seating type as arguments. If seating type=1 -> Check the values of 2 seater tables. If seating type=2 -> Check the values of 4 seater tables. If seating type=3 -> Check the values of 6 seater tables. Create a dictianary with table number as key(Integer) and total amount for the table(double) as value In this method iterate the booking list and get all the bookings with the given seating type. Put the table number of the booking as key in the dictionary and total amount of the booking as the value for the dictionary and return the dictionary. |
| static int CalculateHighest(Dictionary<int,double> perTableBilling) | This method accepts a dictionary as the argument and returns the table number with the highest cost(Integer). |

c) Create the following static methods in the **Table** class,

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| static Table CreateTable(string line) | This method will get a table detail split the value, then create a table object and return the table object. |

Create a class named as **Program**, which contains Main method. All the input and output operations are done this Main method.  
And also in Main method used to access the above classes to done this requirement.  
  
The input format of Table details is separated by comma in the below order,  
(\_id, \_number, \_capacity)  
  
The input format of Booking details is separated by comma in the below order,  
(\_id, \_customerName, \_tableNumber, \_membersPresent, \_billamount, \_bookingTime)  
  
Read the number of tables, table details, number of bookings , booking details and seating type and create a dictioary using table number as key and total amount for the table as value for the given seating type and find the table number with maximum total amount for the given seating type.  
  
**Sample INPUT & OUTPUT 1:**  
  
Enter the number of tables:  
**5  
101,10,2  
102,20,4  
103,30,6  
104,40,2  
105,50,6**  
Enter the number of booking details:  
**18  
10,John,10,8,1200,12-01-2018  
20,Peter,30,8,1300,12-01-2018  
30,Mark,20,8,2600,12-01-2018  
40,Ricky,50,8,1200,12-01-2018  
50,Sam,10,8,1300,12-01-2018  
60,James,20,8,2600,12-01-2018  
70,Rose,40,8,1200,12-01-2018  
80,Jack,50,8,1300,12-01-2018  
90,Parker,40,8,2600,12-01-2018  
100,Tony,10,8,1200,12-01-2018  
110,Lee,30,8,1300,12-01-2018  
120,Starc,50,8,2600,12-01-2018  
130,Amanda,10,8,1200,12-01-2018  
140,Chris,30,8,1300,12-01-2018  
150,Morsh,20,8,2600,12-01-2018  
160,Clinton,20,8,1200,12-01-2018  
170,Tom,40,8,1300,12-01-2018  
180,Alice,30,8,2600,12-01-2018**  
Enter the type:  
1. Two seater table  
2. Four seater table  
3. Six seater table  
**1**  
The lucky table is table 40  
  
**Sample INPUT & OUTPUT 2:**  
  
Enter the number of tables:  
**5  
101,10,2  
102,20,4  
103,30,6  
104,40,2  
105,50,6**  
Enter the number of booking details:  
**18  
10,John,10,8,1200,12-01-2018  
20,Peter,30,8,1300,12-01-2018  
30,Mark,20,8,2600,12-01-2018  
40,Ricky,50,8,1200,12-01-2018  
50,Sam,10,8,1300,12-01-2018  
60,James,20,8,2600,12-01-2018  
70,Rose,40,8,1200,12-01-2018  
80,Jack,50,8,1300,12-01-2018  
90,Parker,40,8,2600,12-01-2018  
100,Tony,10,8,1200,12-01-2018  
110,Lee,30,8,1300,12-01-2018  
120,Starc,50,8,2600,12-01-2018  
130,Amanda,10,8,1200,12-01-2018  
140,Chris,30,8,1300,12-01-2018  
150,Morsh,20,8,2600,12-01-2018  
160,Clinton,20,8,1200,12-01-2018  
170,Tom,40,8,1300,12-01-2018  
180,Alice,30,8,2600,12-01-2018**  
Enter the type:  
1. Two seater table  
2. Four seater table  
3. Six seater table  
**2**  
The lucky table is table 20  
  
**Sample INPUT & OUTPUT 3:**  
  
Enter the number of tables:  
**5  
101,10,2  
102,20,4  
103,30,6  
104,40,2  
105,50,6**  
Enter the number of booking details:  
**18  
10,John,10,8,1200,12-01-2018  
20,Peter,30,8,1300,12-01-2018  
30,Mark,20,8,2600,12-01-2018  
40,Ricky,50,8,1200,12-01-2018  
50,Sam,10,8,1300,12-01-2018  
60,James,20,8,2600,12-01-2018  
70,Rose,40,8,1200,12-01-2018  
80,Jack,50,8,1300,12-01-2018  
90,Parker,40,8,2600,12-01-2018  
100,Tony,10,8,1200,12-01-2018  
110,Lee,30,8,1300,12-01-2018  
120,Starc,50,8,2600,12-01-2018  
130,Amanda,10,8,1200,12-01-2018  
140,Chris,30,8,1300,12-01-2018  
150,Morsh,20,8,2600,12-01-2018  
160,Clinton,20,8,1200,12-01-2018  
170,Tom,40,8,1300,12-01-2018  
180,Alice,30,8,2600,12-01-2018**  
Enter the type:  
1. Two seater table  
2. Four seater table  
3. Six seater table  
**3**  
The lucky table is table 30