# COMPUTER ENGINEERING DEPARTMENT INTRODUCTION TO OBJECT ORIENTED PROGRAMMING PROJECT

2023-2024 Spring Semester
Ege University


05220000285 Volkan SUNGAR

# PRODUCT RATING FORECASTING

## Project Description

This project is a Customer Rating Forecasting System designed to forecast unknown values for a product according to a list of other customers. The system uses a Linked List structure to store the customer data sorted and a 2-dimensional array to store the product ratings. This system can guess the unknown rating of a product from a customer by comparing the customer's rating similarity to other customers.

## Project Mission

By forecasting the product ratings, it is intended to improve Business Strategy and Planning, Customer Satisfaction, Competitive Advantage, Financial Performance, Operation Efficiency and Consumer Insights.

Efficient inventory management, informed product development, targeted marketing strategies; personalised recommendations; staying ahead of marketing trends, strategic positioning; precise revenue forecasting, informed investment decisions; effective resource allocation can be given examples of the results of a successful forecasting. Overall, accurate rating forecasts drive growth, enhance customer satisfaction, and ensure long-term business success.

# Project Capabilities

1. Read data from a text file.
2. Store data in corresponding list structures.
3. Accept customer entry from the user.
4. Calculate average ratings for each product and also according to demographics.
5. Display each list structure.

# Technologies Used

Java programming language was used for this project.

- **Linked List Structure:** Used to store customer information.
- **BufferedReader:** Used to read files and get keyboard entry from the user.
- **HashMap:** Used to store consumer rating similarity as key: data pairs.
- **java.util.arrays:** Used the deepToString() method to display the 2-dimensional array.

# Explanation of classes and methods

## *CustomerList* Class

CustomerList class is a Linked List. It has a **head** field that stores the first Node. Its nodes have two fields; int primitive typed customerNumber field and CustomerData Object typed data field. customerNumber represents the customer ID number for entries into the linked list from both the keyboard and the text file. CustomerData is an object constructed with the information gathered from the file and the keyboard such as name and surname.

```
4   public class CustomerList {
5       public class Node {
6           public CustomerData data;
7           public int customerNumber;
8           public Node link;
9           //node no argument constructor
10          public Node() {
11              data = null;
12              customerNumber = 0;
13              link = null;
14          }
15
16          //node constructor
17          public Node(CustomerData data, int customerNumber, Node link) {
18              this.data = data;
19              this.customerNumber = customerNumber;
20              this.link = link;
21          }
22      }
```

## *ListIterator* Inner Class

ListIterator is an inner class of *CustomerList.* Its main function is to iterate through the linked list (can be done both ways; forward or backwards). It has two Node Object typed fields named **previous** and **position**. *position* stores the current node that the iterator checks and *previous* stores the one that is before.

```
24      //iterator class
25      public class ListIterator {
26          private Node position;
27          private Node previous;
28          //iterator no argument constructor
29          public ListIterator() {
30              position = head;
31              previous = null;
32          }
```

ListIterator has 5 methods except its constructor:

- **void restart()** : sets the 'position' to *head* and 'previous' to null.
- **Node next()** : sets the 'previous' to 'position' and 'position' to position.link. Thus moves one node forward.
- **void previous()** : restarts from the start and iterates until 'position' is 'previous'.
- **boolean hasNext()** : checks if there is a forward node.
- **void addHere()** : adds a new node depending on the location pointed by the iterator.

```
55          public void addHere(CustomerData newData, int customerNumber) {
56              if (position == null && previous != null)
57                      previous.link = new Node(newData, customerNumber, null);
58                  //start of the list
59                  else if (position == null || previous == null) {
60                  CustomerList.this.addToStart(newData, customerNumber);
61              }
62                  //nodes are consecutive
63                  else {
64                      Node temp = new Node(newData, customerNumber, position);
65                      previous.link = temp;
66                      previous = temp;
67              }
68          }
```

## *CustomerList Methods*

- **iterator()** : constructs and returns a ListIterator object.
- **addToStart()** : constructs and sets head node by receiving CustomerData object and int customerNumber.
- **size()** : returns the linked list size by iteration.
- **outputList()** : prints the list by iteration.

- **checkDoctor() & checkTurkey()** : checks if the node is a doctor and from Turkey respectively. Receives customerNumber as parameter.
- **equals()** : receives an Object typed list as parameter. Checks if they are the same class, has the same size and the same elements in order.

## *CustomerData* Class

CustomerData is a class to store customer information. It has 5 String typed fields; name, surname, country, city and occupation. Its methods are basic constructors and setter and getters.

```
1    public class CustomerData {
2        private String name;
3        private String surname;
4        private String country;
5        private String city;
6        private String occupation;
```

## *Main* Class

The 'Main' class has 8 static fields to be used inside every scope throughout the program. **ratings** field represent the 2-dimensional array. **list** field is the CustomerList typed linked list structure. **products** is allocated for the first line in the text file, thus stores the product names and the count. **lineCount** is for counting the lines in the text file, this was implemented to calculate and enter into customer data and ratings separately. **names** is the product names.

```
public class Main {
    final static int MAX_CUSTOMER = 200;
    static String[] products = new String[0];
    static String names = "";
    static int productCount = 0;
    static String[][] ratings = new String[0][];
    static CustomerList list = new CustomerList();
    static int lineCount = 0;
    static int customerCount;
```

### Static Main Methods

There are two static methods inside the Main class; **addOrdered** and **calculateAverage**.

## *addOrdered()*

```
170     public static boolean addOrdered(CustomerList list, CustomerData data, int customerNumber) {
171         CustomerList.ListIterator iterator = list.iterator();
172 
173         while(iterator.hasNext()) {
174             int compareResult = (iterator.next().customerNumber-customerNumber);
175             if (compareResult==0){
176                 iterator.addHere(data, customerNumber);
177                 return true;
178             }else if(compareResult > 0) {
179                 iterator.previous();
180                 iterator.addHere(data, customerNumber);
181                 return true;
182             }
183         }
184         iterator.addHere(data, customerNumber);
185         return true;
186     }
```

addOrdered() receives a CustomerList typed linked list, CustomerData typed object data, and primitive typed int customerNumber.

Its main function is to iterate through the list and move pointer or add the customer respectively to their customer numbers, ordered.

compareResult parameter is used to store the data of the difference between the pointed customer's number and the passed customer's number. If the pointed number is bigger than the past number, it will iterate to the previous index and place the past customer there.

## *calculateAverage()*

```
190         for (int i = 0; i < productCount; i++) {
191             double totalRating = 0;
192             double turkeyTotal = 0;
193             double doctorTotal = 0;
194             double notTurkeyTotal = 0;
195             int totalCustomer = 0;
196             int turkeyCustomer = 0;
197             int doctorCustomer = 0;
198             int notTurkeyCustomer = 0;
199             for (int j = 0; j < MAX_CUSTOMER; j++) {
200                 //iterate in ratings to get the value for each product and add them as totalRating
201                 if (ratings[j][i + 1] != null) {
202                     totalRating += Double.parseDouble(ratings[j][i + 1]);
203                     if (list.checkDoctor(Integer.parseInt(ratings[j][0]))) {
204                         doctorTotal += Double.parseDouble(ratings[j][i + 1]);
205                         doctorCustomer++;
206                     }
207                     if (list.checkTurkey(Integer.parseInt(ratings[j][0]))) {
208                         turkeyTotal += Double.parseDouble(ratings[j][i + 1]);
209                         turkeyCustomer++;
210                     } else {
211                         notTurkeyTotal += Double.parseDouble(ratings[j][i + 1]);
212                         notTurkeyCustomer++;
213                     }
214                     totalCustomer++;
215 
216                 }
217             }
```

This method receives a String that can be either “turkey”, “not turkey”, or “doctor” Depending on the inputted string, method will calculate and print the average ratings for each product by adding every rating for each product then later dividing them to corresponding customer counts.

## *Main* Method

The Main method first prints out the menu to the console then uses switch case to evaluate the choice of the user.

```
41              case "1":
42                  //read the file and create data structures
43                  File file = new File("Firma.txt");
44                  BufferedReader br = new BufferedReader(new FileReader(file));
45                  names = br.readLine();
46                  products = names.split(",");
47                  productCount = Integer.parseInt(products[0]);
48                  ratings = new String[MAX_CUSTOMER][productCount + 1];
49
50                  String st;
51                  lineCount = 0;
52                  while ((st = br.readLine()) != null) {
53                      String[] customerInfo = st.split(",");
54                      if (lineCount % 2 == 0) {
55                          ratings[lineCount / 2][0] = customerInfo[0];
56                          String name = customerInfo[1];
57                          String surname = customerInfo[2];
58                          String country = customerInfo[3];
59                          String city = customerInfo[4];
60                          String occupation = customerInfo[5];
61                          //add customers to linkedlist
62                          CustomerData customer = new CustomerData(name, surname, country, city, occupation);
63                          addOrdered(list, customer, Integer.parseInt(customerInfo[0]));
64                      } else {
65                          //add customers to 2d array
66                          System.arraycopy(customerInfo, 0, ratings[lineCount / 2], 1, productCount);
67                      }
68
69                      lineCount++;
70                  }
71                  customerCount = lineCount / 2;
```

For case 1, it first creates a File object then reads the first line with BufferedReader and stored as ‘names’. ‘products’ is basically the array version of ‘names’. productCount is the first element of the ‘names’ array as the project paper suggested for the text file format.

In a while loop the remaining lines in the text file is read. If the line number is even, we know that we are reading the information about the customer so we pass the values to corresponding variables. Then later constructed a CustomerData object using those variables, and used addOrdered method to add the object to the linked list.

If the line number is odd, it is the rating line so we pass the text into the ratings array.

For case 2, we first get the information about a new customer in a while loop.

```
91                  double[] kbRatings = new double[productCount - 1];
92                  //hash map used to store similarity between the keyboard values and file values
93                  // as customer data as key
94                  HashMap<String, Double> similarity = new HashMap<>();
95                  for (int i = 0; i < productCount - 1; i++) {
96                      System.out.println("Enter rating for product " + products[i + 1]);
97                      String input = kb.readLine();
98                      if (input != null) {
99                      kbRatings[i] = Double.parseDouble(input);
100                     ratings[customerCount][i + 1] = input;
101                 }}
```

We also store the ratings we enter using the console in an array named kbRatings. Constructed a HashMap ‘similarity’ to store customer numbers and the similarity value between the the keyboard values and the file values. We later get ratings for each product and pass them to the ‘ratings’ array.

```
103                 //minimum difference is set productCount times 5 initially because it needs to get lower
104                 // as the program proceeds so the initial value should be the maximum value possible
105                 double minDifference = productCount * 5;
106                 double forecast = 0;
107                 for (int i = 0; i < MAX_CUSTOMER; i++) {
108                     double difference = 0;
109                     for (int j = 0; j < productCount - 1; j++) {
110                         if (ratings[i][j + 1] != null) {
111                             difference += abs(kbRatings[j] - Double.parseDouble(ratings[i][j + 1]));
112                         }
113                     }
114                     if (ratings[i][0] != null && difference < minDifference) {
115                         similarity.clear();
116                         forecast = 0;
117                         minDifference = difference;
118                         forecast += Double.parseDouble(ratings[i][ratings[i].length - 1]);
119                         similarity.put(ratings[i][0], difference);
120                     } else if (ratings[i][0] != null && difference == minDifference) {
121                         similarity.put(ratings[i][0], difference);
122                         forecast += Double.parseDouble(ratings[i][ratings[i].length - 1]);
123                     }
124                 }
125                 forecast = forecast / similarity.size();
126
127                 System.out.println("The forecast for the last product: "+forecast);
128                 ratings[customerCount][0] = String.valueOf(customerNumber);
129                 ratings[customerCount][productCount] = String.valueOf(forecast);
130
131                 CustomerData customer = new CustomerData(name, surname, country, city, occupation);
132                 addOrdered(list, customer, customerNumber);
```

In a for loop we iterate through the arrays ‘kbRating’ and ‘ratings’. We subtract and take the result’s absolute and add to the previous difference value to find the total difference. This was done for every customer for the same product, then later passed to the second customer and done the calculations again. After finding difference for each customer, the outer for loop continues to check which one is the smallest value. After finding te smallest value it is added to the ‘similarity’ HashMap and the ‘forecast’ is set to the last element of the most similar customer. To find the average the ‘forecast’ is divided by similarity.size(). Keyboard entered customer is updated according to the forecast and added to both the 2-dimensional array

and the linked list structure.

```
142                 case "3":
143                     calculateAverage("total");
144                     break;
145                 case "4":
146                     calculateAverage("turkey");
147                     break;
148                 case "5":
149                     calculateAverage("not turkey");
150                     break;
151                 case "6":
152                     calculateAverage("doctor");
153                     break;
154                 case "7":
155                     list.outputList();
156                     break;
157                 case "8":
158                     System.out.println(Arrays.deepToString(ratings));
159                     break;
```

For case 3, 4, 5, 6; we call the calculateAverage() method and pass the corresponding String values to calculate and print the average values.

For case 7, we print the linked list structure using its outputList() method.

For case 8, we use Arrays.deepToString() method from java.util.Arrays to print the 2-dimensional array.

## PROJECT DIAGRAM