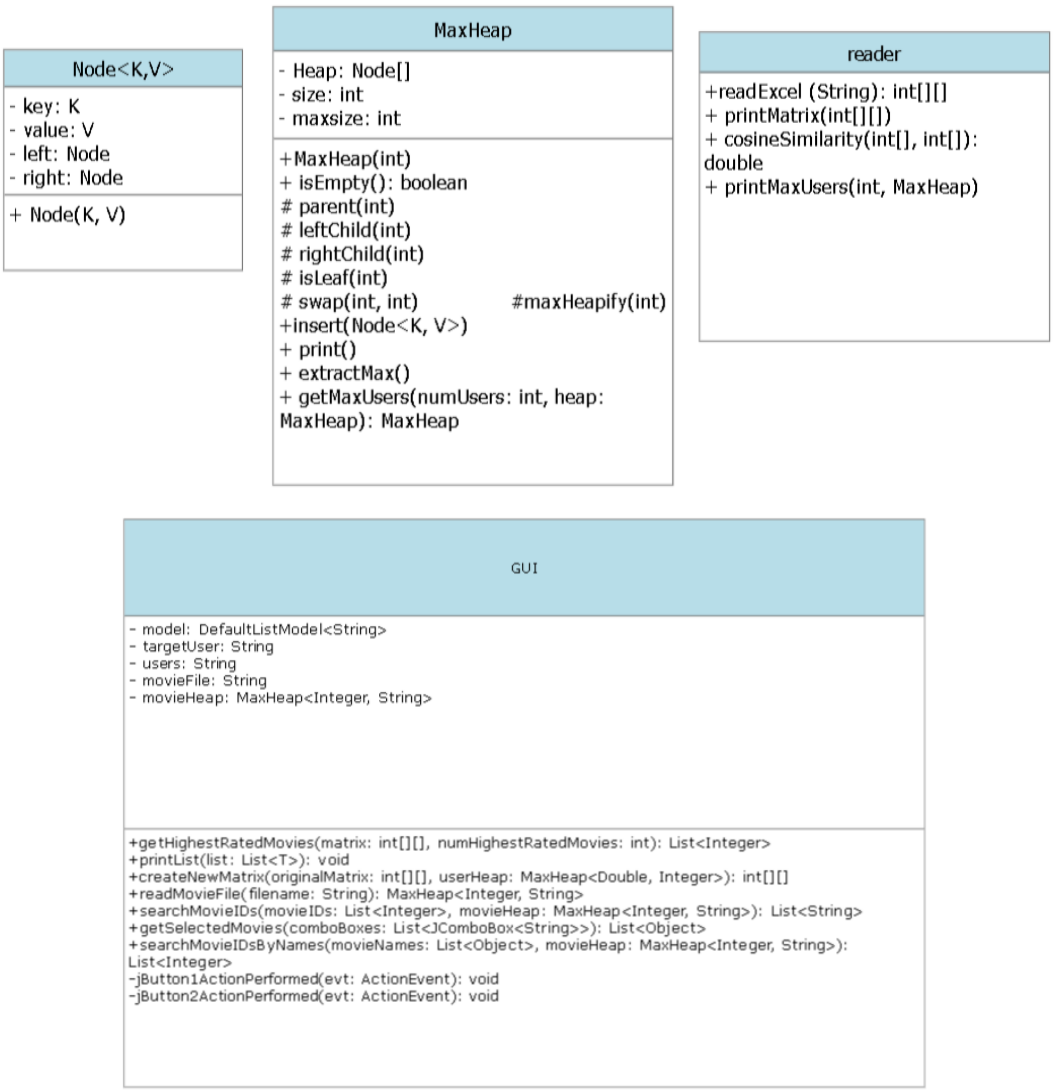
BLM19202E Data Structures Programming Assignment #3
Movie Recommendation System using Heap-Based Collaborative Filtering

HAMZA ALAHMEDİ– 2021221358 / RAMİ EYÜPOĞLU–1921221355

1- Design:



Pseudo-code and algorthim steps:

a. First Window :

1- Read the target_user file and main_data file and create a matrix for each file: Use the readExcel method in the Reader class to read files and obtain a matrix representation of the data.
2- Calculate the cosine similarity: Iterate over each user in the main data matrix. For each user, calculate the cosine similarity with the target user by using the cosineSimilarity method in the Reader class. Store the user ID and cosine similarity value as a pair.
3- Add user IDs with cosine similarity to a multi-value MaxHeap: Create a new instance of the MaxHeap class with the appropriate data types. Add the user IDs and cosine similarity values as nodes to the heap.

4- Get the x most similar users: Use the extractMax method in the MaxHeap class to extract the x highest cosine similarity users from the heap. Create a new instance of the MaxHeap class specifically for the similar users and add the extracted nodes to this new heap.
5- Read the main data file and create a new matrix: Iterate over each user ID in the similar users heap. For each user ID, retrieve the entire row of data from the main data matrix corresponding to that user. Create a new matrix with these rows.
6- Get the K highest rated movies IDs: Determine the K highest rated movies by implementing the appropriate logic. Create a list to store the movie IDs.
7- Read the movies file and create a MaxHeap with movie ID and name: Use the readExcel method in the Reader class to read the movies file and obtain a matrix representation of the data. Create a new instance of the MaxHeap class to store the movie IDs and names.
8- Get the movie names from the MaxHeap: Iterate over the movie IDs in the list of K highest rated movies. Search for each movie ID in the movie heap and retrieve the corresponding movie name. Add the movie names to a JList or any suitable data structure.

b. Second Window :
1- Obtain the newUserId from a text field.
2- Retrieve the selected movie names from combo boxes.
3- Collect the movie ratings from text fields.
4- Search the movie heap for the movie IDs corresponding to the selected movie names.
5- Create a new vector to hold the newUserId and the movie ratings, with each rating stored at the index corresponding to its movie ID.
6- Calculate the cosine similarity between the new user and all other users in the system.
7- Add the user IDs along with their cosine similarity values to a multi-value MaxHeap data structure.
8- Retrieve the x most similar users by extracting the highest cosine similarity users from the MaxHeap.
9- Read the main data file and create a new matrix containing the data of the similar users.
10- Determine the K highest rated movie IDs by implementing the appropriate logic.
11- Read the movies file and create a MaxHeap data structure to store movie IDs and names.
12- Retrieve the movie names from the MaxHeap by searching for each movie ID in the list of highest rated movies.
13- Populate a JList or suitable data structure with the recommended movie names.

implementation details:

```
public class Node<K extends Comparable<K>, V> {

    K key; // the cossinSimilraty
    V value; // the User id
    Node<K, V> left, right;

    public Node(K key, V value) {
        this.key = key;
        this.value = value;
        left = right = null;
    }
}
```

The Node class(Generic type) has the following attributes:
key: Represents the cosine similarity, value: Represents the user ID.
left: Represents the left child node, right: Represents the right child node.
The class provides a constructor that takes a key and a value as parameters and initializes the attributes accordingly. The left and right child nodes are initially set to null.

Reader class :

```java
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Reader {

    public static int[][] readExcel(String filePath) {
        int[][] matrix = null;
        try {
            File file = new File(filePath);
            FileInputStream fis = new FileInputStream(file);
            Scanner scanner = new Scanner(fis);

            List<String[]> lines = new ArrayList<>();

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] values = line.split(",");
                lines.add(values);
            }

            int numRows = lines.size();
            int numCols = lines.get(0).length;

            matrix = new int[numRows][numCols];

            for (int i = 0; i < numRows; i++) {
                String[] values = lines.get(i);
                for (int j = 0; j < numCols; j++) {
                    matrix[i][j] = Integer.parseInt(values[j]);
                }
            }

            scanner.close();
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return matrix;
    }
```

readExcel(String filePath)*:* This method takes a file path as input and reads an Excel file. It reads the contents of the file and stores them in a 2D integer array (matrix) by splitting each line at commas. It returns the resulting matrix.

```java
    public static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int element : row) {
                System.out.print(element + "\t");  // Use "\t" for tab-separated columns
            }
            System.out.println();  // Move to the next line after printing each row
        }
    }
    public static void printMaxUsers(int numUsers, MaxHeap heap) {
        for (int i = 0; i < numUsers; i++) {
            Node<Double, Integer> maxNode = heap.extractMax();
            if (maxNode == null) {
                break;  // If there are no more nodes in the heap, exit the loop
            }
            System.out.println("User ID: " + maxNode.value + ", Value: " + maxNode.key);
        }
    }
```

printMatrix(int[][] matrix): This method takes a 2D integer array (matrix) as input and prints its contents. It iterates over each row and column of the matrix and prints each element, separated by a tab (\t). After printing each row, it moves to the next line.

printMaxUsers(int numUsers, MaxHeap heap): This method takes the number of users to print (numUsers) and a MaxHeap object (heap) as input. It iterates numUsers times and extracts the maximum node from the heap. If there are no more nodes in the heap, the loop is exited. It then prints the user ID and value of the extracted node.

```java
    // Method to calculate the cosine similarity
    public static double cosineSimilarity(int[] vectorA, int[] vectorB) {
        double dotProduct = 0.0;
        double normA = 0.0;
        double normB = 0.0;
        for (int i = 0; i < vectorA.length; i++) {
            dotProduct += vectorA[i] * vectorB[i];
            normA += Math.pow(vectorA[i], 2);
            normB += Math.pow(vectorB[i], 2);
        }
        return dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
    }
```

cosineSimilarity(int[] vectorA, int[] vectorB): This method takes two integer arrays (vectorA and vectorB) as input and calculates the cosine similarity between them. It calculates the dot product, as well as the norms of the vectors, and then divides the dot product by the product of the vector norms. The resulting cosine similarity value is returned.

MaxHeap class :

```java
 *
 * @author Hamza
 */
public class MaxHeap<K extends Comparable<K>, V> {
    Node<K, V>[] Heap;
    int size;
    int maxsize;

    public MaxHeap(int capacity) {
        Heap = new Node[capacity];
        size = 0;
        maxsize = capacity;
    }


    public boolean isEmpty() {
    return size == 0;
}

    // Calculates the index of the parent node of the given position
    private int parent(int pos) {
        return (pos - 1) / 2;
    }

    // Calculates the index of the left child node of the given position
    private int leftChild(int pos) {
        return (2 * pos) + 1;
    }

    // Calculates the index of the right child node of the given position
    private int rightChild(int pos) {
        return (2 * pos) + 2;
    }

    // Checks if the given position is a leaf node
    private boolean isLeaf(int pos) {
        return pos > (size / 2) && pos <= size;
    }
```

Node<K extends Comparable<K>, V>: This is a nested class representing a node in the heap. It has a key of type K (which must be comparable) and a value of type V.
MaxHeap(int capacity): This is the constructor of the MaxHeap class. It initializes the heap with a given capacity. It creates an array Heap of Node objects with the specified capacity, sets the initial size to 0, and stores the maximum capacity.
isEmpty(): This method checks if the heap is empty by comparing the size with 0. It returns true if the heap is empty, and false otherwise.
parent(int pos): This private method calculates the index of the parent node given a position pos in the heap.

leftChild(int pos): This private method calculates the index of the left child node given a position pos in the heap.
rightChild(int pos): This private method calculates the index of the right child node given a position pos in the heap.
isLeaf(int pos): This private method checks if the node at the given position pos is a leaf node. It returns true if the node is a leaf, and false otherwise.

```
    // Swaps the nodes at the given positions in the Heap array
    private void swap(int fpos, int spos) {
        Node<K, V> tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
        Heap[spos] = tmp;
    }

    // Maintains the max heap property starting from the given position
    // Maintains the max heap property starting from the given position
private void maxHeapify(int pos) {
    int largest = pos;
    int left = leftChild(pos);
    int right = rightChild(pos);

    // Compare the left child with the current position
    if (left < size && Heap[left].key.compareTo(Heap[largest].key) > 0) {
        largest = left;
    }

    // Compare the right child with the current largest
    if (right < size && Heap[right].key.compareTo(Heap[largest].key) > 0) {
        largest = right;
    }

    // If the largest element is not the current position, swap them
    if (largest != pos) {
        swap(pos, largest);
        maxHeapify(largest);
    }
}
```

swap(int fpos, int spos): This private method swaps the nodes at the given positions fpos and spos in the Heap array.
maxHeapify(int pos): This private method maintains the max heap property starting from the given position pos. It compares the node at pos with its children, and if necessary, swaps them to maintain the max heap property. It recursively calls itself on the child nodes.

```
// Inserts a new node into the max heap
public void insert(Node<K, V> node) {
    if (size >= maxsize) {
        throw new IllegalStateException("Heap is full. Cannot insert more elements.");
    }

    Heap[size] = node;
    int current = size;
    size++;

    // Bubble-up the inserted node until it reaches the appropriate position
    while (current > 0 && Heap[current].key.compareTo(Heap[parent(current)].key) > 0) {
        swap(current, parent(current));
        current = parent(current);
    }

    // Perform maxHeapify on the root and its children to ensure the maximum three nodes property
    maxHeapify(0);
    if (leftChild(0) < size) {
        maxHeapify(leftChild(0));
    }
    if (rightChild(0) < size) {
        maxHeapify(rightChild(0));
    }
}
```

insert(Node<K, V> node): This method inserts a new node into the max heap. It checks if the heap is already full and throws an exception if so. It adds the node to the Heap array at the current size position and increments the size. Then, it bubbles up the inserted node by comparing it with its parent and swapping them if necessary. Finally, it performs ***maxHeapify*** on the root and its children to ensure the maximum heap property is maintained.

```
104
105     private void PrintHelper(int index) {
106         if (index >= size) {
107             return;
108         }
109
110         // Traverse the left subtree
111         PrintHelper(leftChild(index));
112
113         // Traverse the right subtree
114         PrintHelper(rightChild(index));
115
116         // Print the current node
117         System.out.println(" key: " + Heap[index].key + " value: " + Heap[index].value + ", ");
118     }
119
120     public void Print() {
121         PrintHelper(0);
122     }
```

PrintHelper(int index): This private recursive method is called by Print() to traverse the heap in a post-order traversal. It prints the nodes by recursively traversing the left and right subtrees and then printing the current node.

Print(): This method prints the nodes in the heap in a post-order traversal. It calls the PrintHelper method to traverse the left subtree, then the right subtree, and finally prints the current node.

```
// Extracts the maximum node from the max heap
public Node<K, V> extractMax() {
    Node<K, V> popped = Heap[0];
    Heap[0] = Heap[--size];
    maxHeapify(0);
    return popped;
}
```

extractMax(): This method extracts the maximum node from the max heap. It removes the root node (which contains the maximum value) and replaces it with the last node in the heap. It then calls maxHeapify(0) to restore the max heap property and returns the extracted node.

```
132     // Retrieve the maximum users from the heap and return a new MaxHeap object
133     public static MaxHeap<Double, Integer> getMaxUsers(int numUsers, MaxHeap heap) {
134         MaxHeap<Double, Integer> maxUserheap = new MaxHeap<>(numUsers);
135         for (int i = 0; i < numUsers; i++) {
136             Node<Double, Integer> maxNode = heap.extractMax();
137             if (maxNode == null) {
138                 break;  // If there are no more nodes in the heap, exit the loop
139             }
140             maxUserheap.insert(maxNode);
141         }
142         return maxUserheap;
143     }
144
145 }
```

getMaxUsers(int numUsers, MaxHeap heap): This static method retrieves the maximum users from the heap and returns a new MaxHeap object. It extracts the maximum node numUsers times from

the given heap and inserts them into a new MaxHeap called maxUserHeap. It returns maxUserHeap containing the maximum numUsers nodes.

GUI class :

```
42              // Column index to read from CSV
43              int columnIndex = 0;
44
45              try {
46                  BufferedReader br = new BufferedReader(new FileReader(targetUser));
47                  String line;
48                  int lineNumber = 0;
49                  while ((line = br.readLine()) != null) {
50                      lineNumber++;
51                      String[] columns = line.split(",");
52                      if (columns.length > 0) {
53                          String columnValue = columns[0]; // Get the value of the first column
54                          jComboTarget.addItem(columnValue);
55                      } else {
56                          System.err.println("Invalid column index on line " + lineNumber);
57                      }
58                  }
59                  br.close();
60              } catch (IOException e) {
61                  System.out.println(e);
62              }
63              jComboTarget.setSelectedIndex(1);
64          }
```

this code reads a CSV file, extracts values from the first column, and populates a JComboBox with those values, while also handling error cases where the column index is invalid or an IO exception occurs during file reading.

```
25      public static List<Integer> getHighestRatedMovies(int[][] matrix, int numHighestRatedMovies) {
26          List<Integer> highestRatedMovieIndices = new ArrayList<>();
27
28          for (int user = 0; user < matrix.length; user++) {
29              int[] ratings = matrix[user];
30              List<Integer> userHighestRatedMovies = new ArrayList<>();
31
32              for (int i = 0; i < numHighestRatedMovies; i++) {
33                  int maxRatingIndex = -1;
34                  int maxRating = 0;
35
36                  for (int movie = 1; movie < ratings.length; movie++) {
37                      int rating = ratings[movie];
38
39                      if (rating > maxRating && !userHighestRatedMovies.contains(movie)) {
40                          maxRating = rating;
41                          maxRatingIndex = movie;
42                      }
43                  }
44
45                  if (maxRatingIndex != -1) {
46                      userHighestRatedMovies.add(maxRatingIndex);
47                  }
48              }
49
50              highestRatedMovieIndices.addAll(userHighestRatedMovies);
51          }
52
53          return highestRatedMovieIndices;
54      }
```

getHighestRatedMovies : The given method takes a two-dimensional array (matrix) representing movie ratings by users, and an integer (numHighestRatedMovies) specifying the number of highest-rated movies to be returned.
The code finds the indices of the highest-rated movies across all users based on a given movie rating matrix. It returns a list of these indices, limited to a specified number of highest-rated movies per user.

```java
public static int[][] createNewMatrix(int[][] originalMatrix, MaxHeap<Double, Integer> userHeap) {
    int numRows = userHeap.size;
    int numCols = originalMatrix[0].length;
    int[][] newMatrix = new int[numRows][numCols];

    int rowIndex = 0;
    while (!userHeap.isEmpty()) {
        int userId = userHeap.extractMax().value;
        for (int col = 0; col < numCols; col++) {
            newMatrix[rowIndex][col] = originalMatrix[userId][col];
        }
        rowIndex++;
    }

    return newMatrix;
}
```

creatNewMatrix :This method creates a new matrix based on an original matrix and a max heap of user IDs. It extracts user IDs from the max heap in descending order of their priority (max value) and assigns the corresponding row from the original matrix to the new matrix. The new matrix will have a number of rows equal to the size of the max heap and the same number of columns as the original matrix. The resulting new matrix represents a rearrangement of rows from the original matrix based on the priority of user IDs in the max heap.

```java
public static MaxHeap<Integer, String> readMovieFile(String filename) {
    MaxHeap<Integer, String> movieHeap = new MaxHeap<>(10000);

    try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split(",", 3);
            if (parts.length == 3) {
                int movieId = Integer.parseInt(parts[0].trim());
                String movieName = parts[1].trim();
                Node<Integer, String> node = new Node<>(movieId, movieName);
                movieHeap.insert(node);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return movieHeap;
}
```

readMovieFile(String filename) : This method reads a movie file specified by the filename parameter and creates a max heap (MaxHeap<Integer, String>) based on the data in the file.
The method uses a BufferedReader to read the file line by line. Each line is split into three parts using a comma as the delimiter. If a line contains three parts, indicating a valid movie entry, the method extracts the movie ID (parsed as an integer) and the movie name from the parts array.
Then, a Node<Integer, String> object is created using the movie ID and movie name, and this node is inserted into the movieHeap max heap. The max heap is initialized with a capacity of 10000.
After processing all the lines in the file, the method returns the populated movieHeap max heap containing the movie IDs and names read from the file.

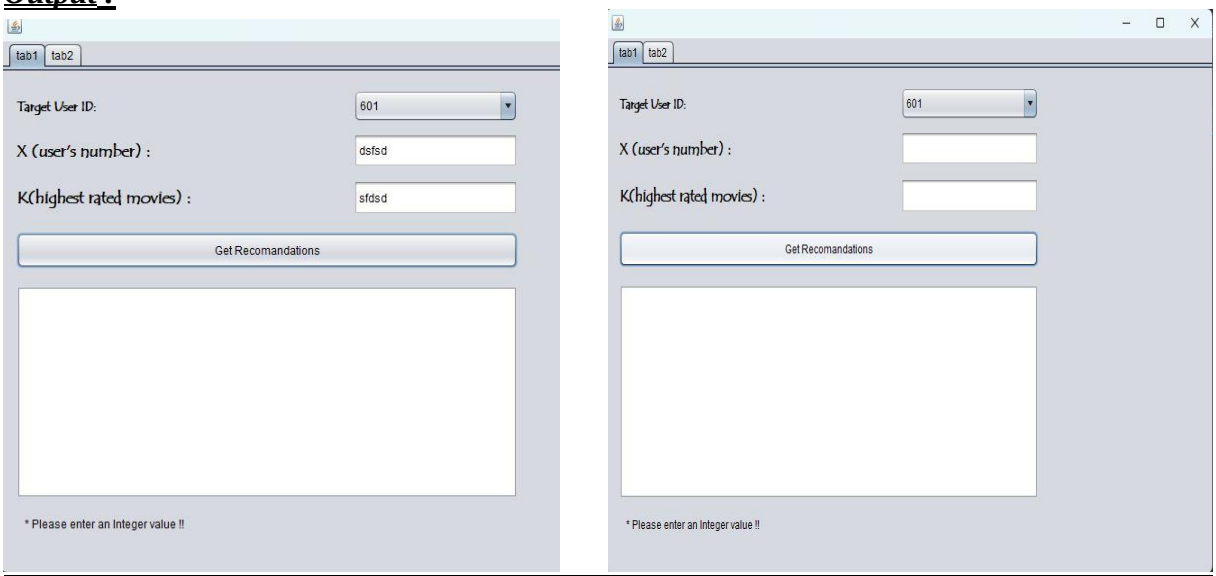
```java
public static List<String> searchMovieIDs(List<Integer> movieIDs, MaxHeap<Integer, String> movieHeap) {
    List<String> movieNames = new ArrayList<>();
    List<Node<Integer, String>> extractedNodes = new ArrayList<>();

    // Extract all nodes from the movie heap
    while (!movieHeap.isEmpty()) {
        extractedNodes.add(movieHeap.extractMax());
    }

    // Search for movie IDs in the extracted nodes
    for (int movieID : movieIDs) {
        boolean found = false;
        for (Node<Integer, String> node : extractedNodes) {
            if (node.key.equals(movieID)) {
                String movieName = node.value;
                movieNames.add(movieName);
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("Movie ID " + movieID + " not found in the movie heap.");
            movieNames.add("Movie Not Found");
        }
    }

    return movieNames;
}
```

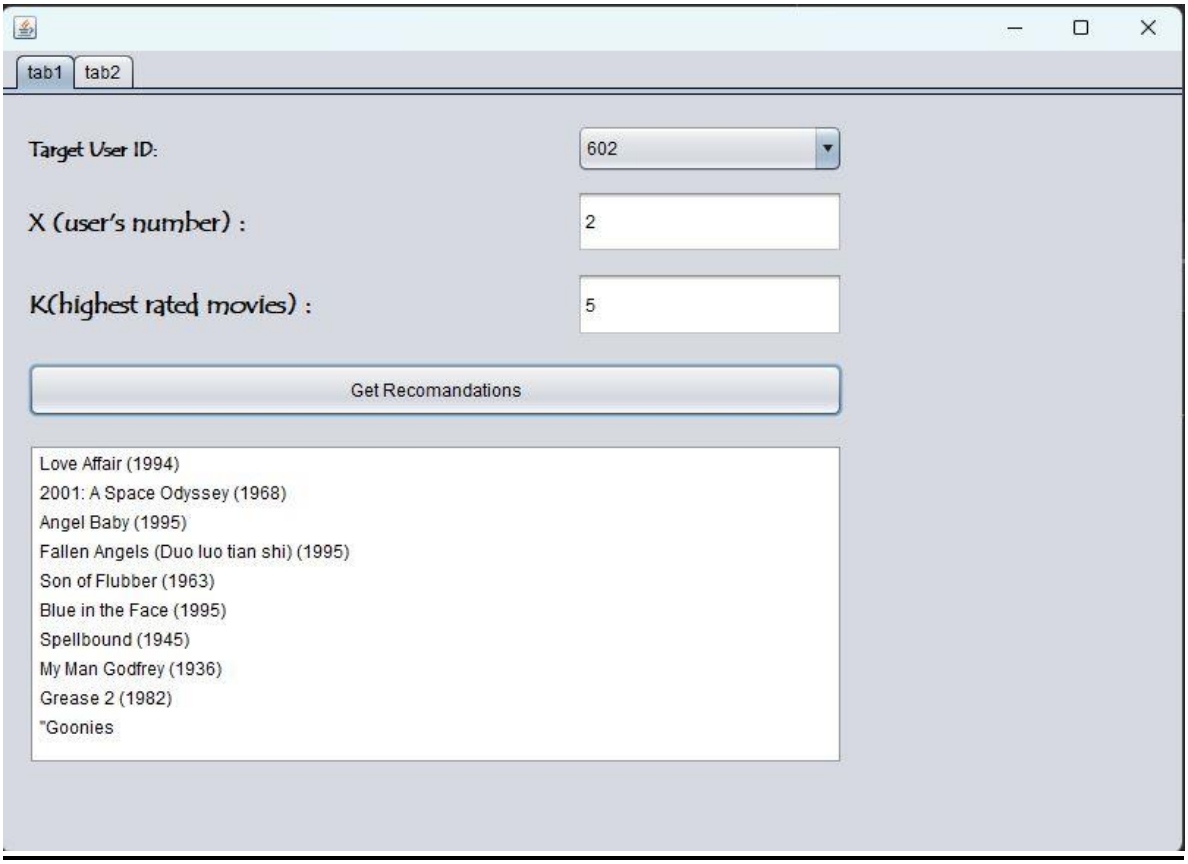
This method searches for movie names corresponding to a given list of movie IDs within a movieHeap max heap. it initializes an empty list movieNames to store the found movie names and an empty list extractedNodes to store the extracted nodes from the movieHeap.
Next, it extracts all nodes from the movieHeap using the extractMax() method and adds them to the extractedNodes list .Then, for each movie ID in the movieIDs list, it searches for a matching movie ID within the extractedNodes. If a match is found, it retrieves the corresponding movie name from the node and adds it to the movieNames list. If no match is found, it prints a message indicating that the movie ID was not found and adds the string "Movie Not Found" to the movieNames list.
Finally, it returns the movieNames list, which contains the names of the movies corresponding to the provided movie IDs, or "Movie Not Found" for IDs that were not found in the movieHeap

```java
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // Control if X and K are positive Integer values.
    String XKregex = "\\d+";
    String xValue = Users.getText();
    String kValue = Movie.getText();

    if (!xValue.matches(XKregex) || !kValue.matches(XKregex)) {
        jLabel1.setText("* Please enter an Integer value !!");
    } else {
        jLabel1.setText("");

        // Read the target user matrix from the file
        int[][] target_user_matrix = Reader.readExcel(TargetUser);

        // Read the main data matrix from the file
        int[][] main_data_matrix = Reader.readExcel(Users);

        // Create a new max heap
        MaxHeap<Double, Integer> maxHeap = new MaxHeap<>(10000);

        for (int user_id = 0; user_id < main_data_matrix.length; user_id++) {
            double similarity = Reader.cosineSimilarity(target_user_matrix[TargetUser.getSelectedIndex()], main_data_matrix[user_id]);
            Node<Double, Integer> node = new Node<>(similarity, user_id);
            maxHeap.insert(node);
        }
        int numberOfUsers = Integer.parseInt(xValue);
        int numberOfMovies = Integer.parseInt(kValue);
        // Create a new max heap for the most similar users
        MaxHeap<Double, Integer> maxUserHeap = MaxHeap.getMaxUsers(numberOfUsers, maxHeap);
        //maxUserHeap.Print();
        int maxUserMatrix[][] = createNewMatrix(main_data_matrix, maxUserHeap);
        //Reader.printMatrix(maxUserMatrix);
        List<Integer> movieIds = getHighestRatedMovies(maxUserMatrix, numberOfMovies);
        printList(movieIds);
        MaxHeap<Integer, String> movieHeap = readMovieFile(movieFile);
        //movieHeap.Print();
        List<String> movieNames = searchMovieIDs(movieIds, movieHeap);
        printList(movieNames);
        model.clear();
        for (String movie : movieNames) {
            model.addElement(movie);
        }
        jList1.setModel(model);
```

It checks if the values of User and Movie text fields are positive integer values using regular expressions. If any of them is not a positive integer, it sets the text of jLabel1 to indicate that the user should enter an integer value.
If both User and Movie values are positive integers the method performs various operations including reading data from files, calculating similarity, creating and manipulating max heaps, retrieving movie information, and updating the GUI with the results based on user input.

Output :



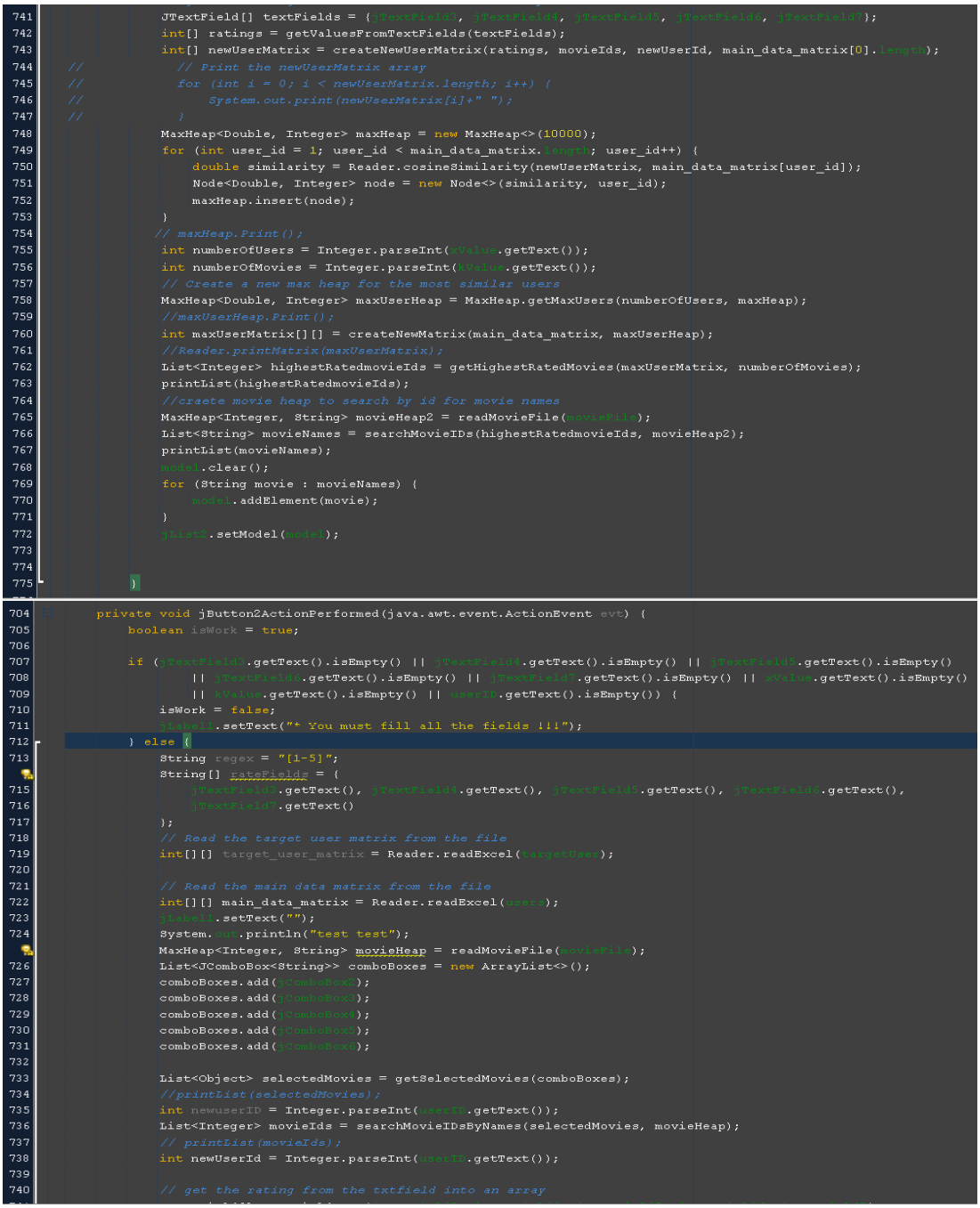


Second panel :

```
        try {
            BufferedReader br = new BufferedReader(new FileReader(movieFile));
            String line;
            Random random = new Random();

            // Skip the header line if present
            br.readLine();

            // Add random movies to each jComboBox
            for (int i = 2; i <= 6; i++) {
                javax.swing.JComboBox<String> comboBox = null;
                // add 10 movies to the choosen combobox .
                switch (i) {
                    case 2:
                        comboBox = jComboBox2;
                        break;
                    case 3:
                        comboBox = jComboBox3;
                        break;
                    case 4:
                        comboBox = jComboBox4;
                        break;
                    case 5:
                        comboBox = jComboBox5;
                        break;
                    case 6:
                        comboBox = jComboBox6;
                        break;
                }

                if (comboBox != null) {
                    int numMovies = 0;
                    while (numMovies < 10 && (line = br.readLine()) != null) {
                        String[] columns = line.split(",");
                        if (columns.length > 1) {
                            // Get the value of the second column (movie's name)
                            String movieName = columns[1];
                            comboBox.addItem(movieName);
                            numMovies++;
                        }
                    }
                }
            }

            br.close();
        } catch (IOException e) {
            System.out.println(e);
        }

```

The code reads a movie file, skips the header line, and randomly selects movies to populate several JComboBox components. Each JComboBox is associated with a specific number (2 to 6), and the code adds 10 movies from the file to each corresponding JComboBox. The goal is to populate the JComboBox components with a selection of movies for user interaction or selection purposes.

```
    public static List<Integer> searchMovieIDsByNames(List<Object> movieNames, MaxHeap<Integer, String> movieHeap) {
        List<Integer> movieID = new ArrayList<>();
        List<Node<Integer, String>> extractedNodes = new ArrayList<>();

        // Extract all nodes from the movie heap
        while (!movieHeap.isEmpty()) {
            extractedNodes.add(movieHeap.extractMax());
        }

        // Search for movie IDs in the extracted nodes
        for (Object movieName : movieNames) {
            boolean found = false;
            for (Node<Integer, String> node : extractedNodes) {
                if (node.value.equals(movieName)) {
                    int movieid = node.key;
                    movieID.add(movieid);
                    found = true;
                    break;
                }
            }

            if (!found) {
                System.out.println("Movie name " + movieName + " not found in the movie heap.");
                movieNames.add("Movie Not Found");
            }
        }

        return movieID;
    }

```



```
            JTextField[] textFields = {jTextField3, jTextField4, jTextField5, jTextField6, jTextField7};
            int[] ratings = getValuesFromTextFields(textFields);
            int[] newUserMatrix = createNewUserMatrix(ratings, movieIds, newUserId, main_data_matrix[0].length);
//              // Print the newUserMatrix array
//              for (int i = 0; i < newUserMatrix.length; i++) {
//                  System.out.print(newUserMatrix[i]+" ");
//              }
            MaxHeap<Double, Integer> maxHeap = new MaxHeap<>(10000);
            for (int user_id = 1; user_id < main_data_matrix.length; user_id++) {
                double similarity = Reader.cosineSimilarity(newUserMatrix, main_data_matrix[user_id]);
                Node<Double, Integer> node = new Node<>(similarity, user_id);
                maxHeap.insert(node);
            }
           // maxHeap.Print();
            int numberOfUsers = Integer.parseInt(kValue.getText());
            int numberOfMovies = Integer.parseInt(kValue.getText());
            // Create a new max heap for the most similar users
            MaxHeap<Double, Integer> maxUserHeap = MaxHeap.getMaxUsers(numberOfUsers, maxHeap);
            //maxUserHeap.Print();
            int maxUserMatrix[][] = createNewMatrix(main_data_matrix, maxUserHeap);
            //Reader.printMatrix(maxUserMatrix);
            List<Integer> highestRatedmovieIds = getHighestRatedMovies(maxUserMatrix, numberOfMovies);
            printList(highestRatedmovieIds);
            //craete movie heap to search by id for movie names
            MaxHeap<Integer, String> movieHeap2 = readMovieFile(movieFile);
            List<String> movieNames = searchMovieIDs(highestRatedmovieIds, movieHeap2);
            printList(movieNames);
            model.clear();
            for (String movie : movieNames) {
                model.addElement(movie);
            }
            jList2.setModel(model);


        }
```

```
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        boolean isWork = true;

        if (jTextField3.getText().isEmpty() || jTextField4.getText().isEmpty() || jTextField5.getText().isEmpty()
                || jTextField6.getText().isEmpty() || jTextField7.getText().isEmpty() || kValue.getText().isEmpty()
                || kValue.getText().isEmpty() || userID.getText().isEmpty()) {
            isWork = false;
            jLabel1.setText("* You must fill all the fields !!!");
        } else {
            String regex = "[1-5]";
            String[] rateFields = {
                jTextField3.getText(), jTextField4.getText(), jTextField5.getText(), jTextField6.getText(),
                jTextField7.getText()
            };
            // Read the target user matrix from the file
            int[][] target_user_matrix = Reader.readExcel(targetUser);

            // Read the main data matrix from the file
            int[][] main_data_matrix = Reader.readExcel(users);
            jLabel1.setText("");
            System.out.println("test test");
            MaxHeap<Integer, String> movieHeap = readMovieFile(movieFile);
            List<JComboBox<String>> comboBoxes = new ArrayList<>();
            comboBoxes.add(jComboBox2);
            comboBoxes.add(jComboBox3);
            comboBoxes.add(jComboBox4);
            comboBoxes.add(jComboBox5);
            comboBoxes.add(jComboBox6);

            List<Object> selectedMovies = getSelectedMovies(comboBoxes);
            //printList(selectedMovies);
            int newuserID = Integer.parseInt(userID.getText());
            List<Integer> movieIds = searchMovieIDsByNames(selectedMovies, movieHeap);
            // printList(movieIds);
            int newUserId = Integer.parseInt(userID.getText());

            // get the rating from the txtfield into an array
```

The jButton2ActionPerformed method is triggered when a button is clicked. It performs several actions to provide movie recommendations based on user input. First, it checks if certain fields are empty and displays an error message if they are. Then, it reads data from Excel files to create a target user matrix and a main data matrix, which represent user preferences and movie ratings.

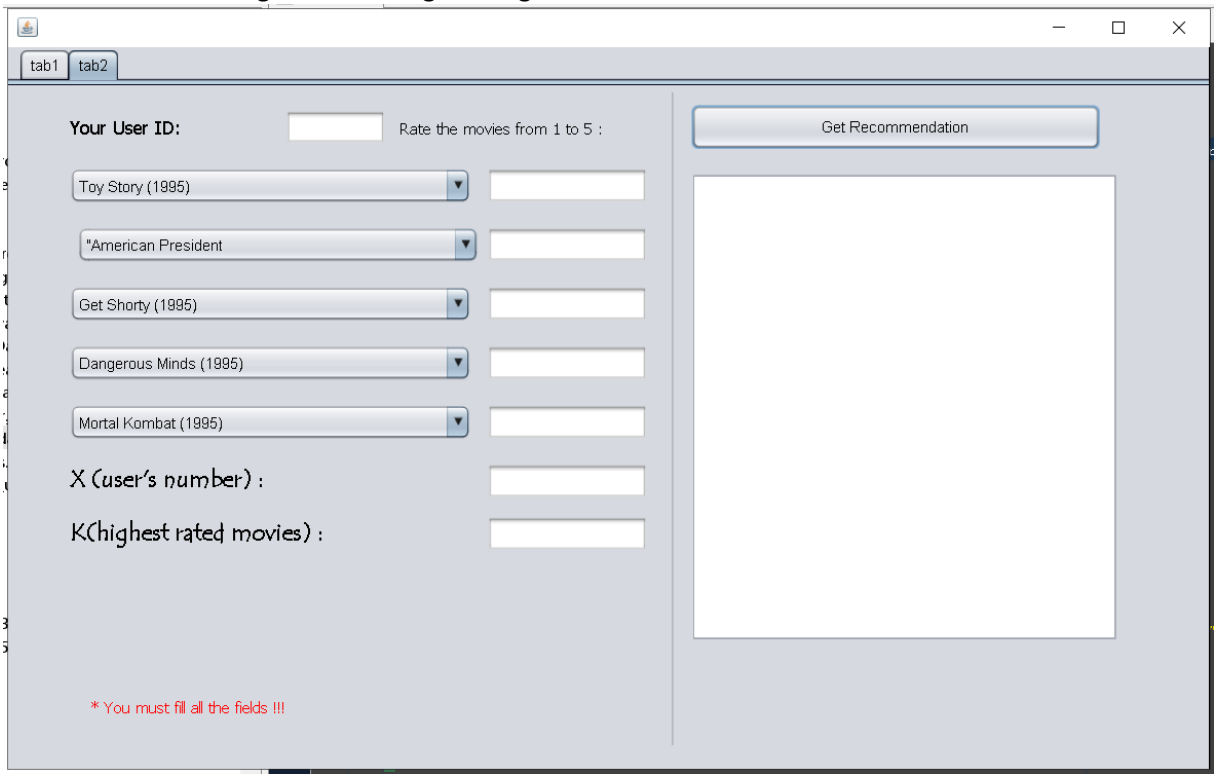
Next, it retrieves the selected movies from combo boxes and converts them into movie IDs. It also collects ratings provided by the user from text fields and creates a new user matrix using these ratings and the selected movie IDs.

To find similar users, the method calculates the similarity between the new user matrix and each user in the main data matrix using cosine similarity. It stores the similarity scores in a max heap. After that, it extracts the highest rated movies from the users with the most similar profiles. It searches for the corresponding movie names using the movie IDs.

Finally, it populates a model with the movie names and updates a list component (jList2) in the user interface with the recommended movies. This allows the user to view and select from the recommended movie list.

OutPut :

Control the field and gave a warning message .



This is the final output .