

Proiect: aplicatie E-Event

Proiect creat de Dobârceanu Mihai și Dincă Flavian

Info 2 Grupa 2

Materie: Programare orientata spre obiecte (POO)

IDE folosit: NetBeans

Limbaj: JAVA

Definitie problema:

Un antreprenor intenționează să transmită publicului informații cu privire la diverse evenimente culturale prin intermediul unei sistem software. Acesta accesează sistemul pe partea de server cu o parolă de administrator pe care o poate schimba. Ca administrator al acestui sistem, el va introduce prin intermediul formularelor evenimentele cu detaliile aferente acestora cât și categoriile din care fac parte.

Fiecare utilizator va avea acces la sistem prin intermediul unui cont unic cu nume de utilizator bazat pe adresa de email și parolă pe care îl poate crea la pornirea aplicației. Acest cont va fi folosit pentru a se identifica în sistem la începutul fiecărei sesiuni de lucru în parte.

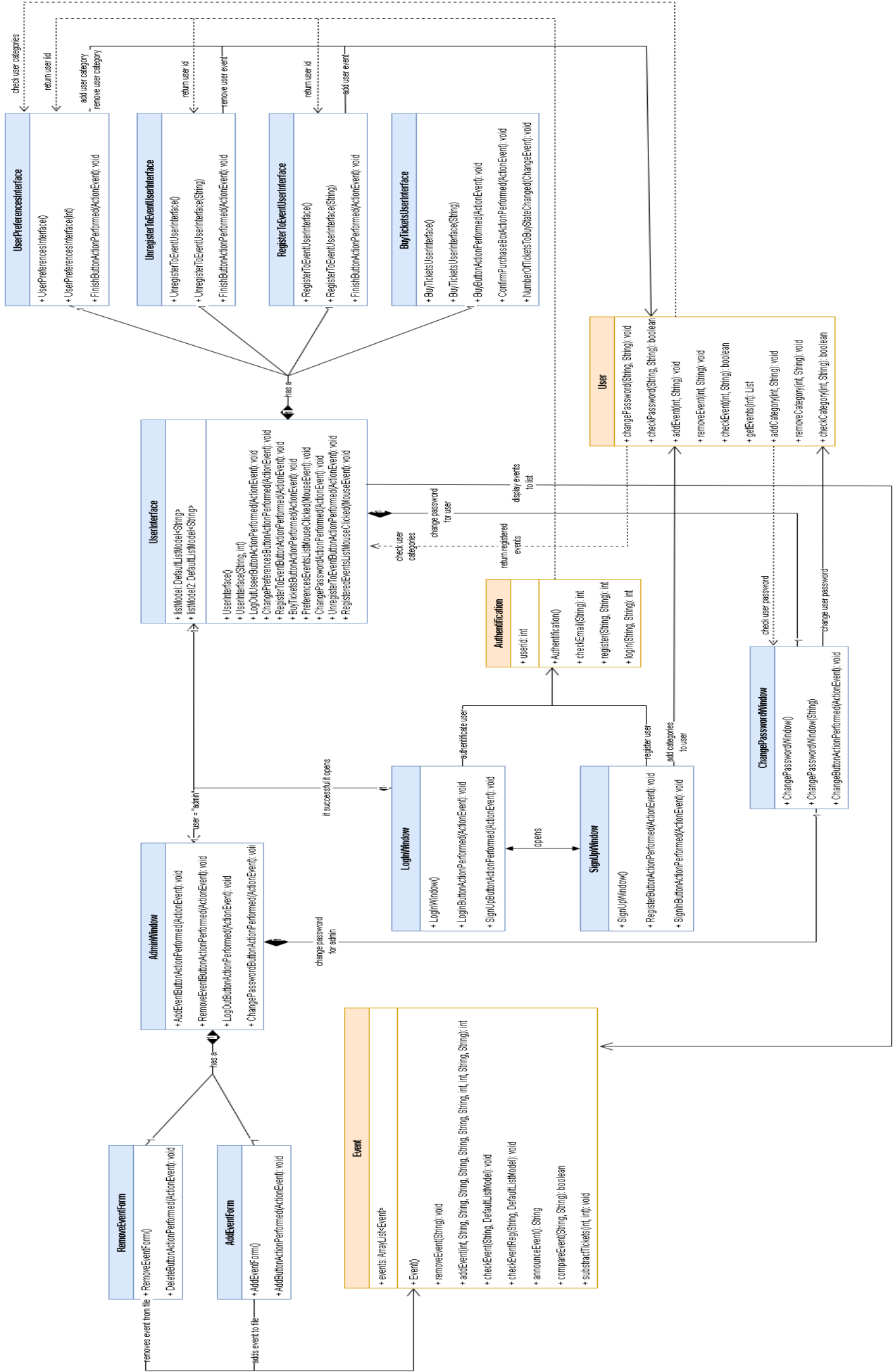
Prin intermediul acestui cont un utilizator poate opta pentru notificări personalizate pe anumite categorii generale sau specifice de evenimente odată ce acestea devin disponibile. Mai precis, utilizatorul se înregistrează la categoriile culturale de evenimente (muzică, film, expoziții de artă, etc) sau poate să specifice evenimentele la care vrea să participe și din acest motiv, aplicatia afiseaza data la care va avea loc și alte informații de interes (loc, preț bilet, etc). De exemplu, dacă utilizatorul este interesat să meargă la festivalul Untold, se va înregistra în aplicație care va afisa când va avea loc și când sunt puse în vânzare bilete la acest festival. Ulterior, utilizatorul își poate rezerva un loc primind în schimb un bilet listat la imprimanta de catre aplicatie.

Scopul principal este acela de a promova eficient evenimentele culturale gestionate de antreprenor printr-o experiență croită pe nevoile personale ale fiecărui utilizator. Obiectivele ce duc la îndeplinirea acestui scop sunt următoarele:

- Autentificarea utilizatorilor prin cont unic
- Memorarea preferințelor fiecărui utilizator
- Gestionarea evenimentelor și categoriilor de care aparțin
- Punerea la dispoziție de bilete pentru evenimentele disponibile

Cerintele proiectului:

- 1.Sa se creeze diagrama UML de clase a aplicatiei E- Event
- 2.Sa se implementeze in Java aplicatia E- Event.
- 3.Aplicatia trebuie sa indeplineasca toate functiunile specificate in descrierea problemei.
- 4.Aplicatia memoreaza pe suport extern (fisiere) toate datele necesare functionarii sale astfel incat existenta datelor nu depinde de oprirea temporara (accidentala sau nu) a calculatoarelor pe care ruleaza.
4. Pentru realizarea interfetelor grafice ale aplicatiei va fi folosita tehnologia Swing.
5. Pentru realizarea diagramei UML de clase si a interfetelor grafice ale aplicatiei nu vor fi folosite programe software care faciliteaza acest lucru.
6. Pentru gestiunea evenimentelor generate de utilizatorii aplicatiei vor fi folosite clase interne.
7. Pentru gestiunea evenimentelor, etc. vor fi folosite colectii dinamice de obiecte.



Cod:

```
StartApp.java x
Source History
1 package event;
2
3 public class StartApp {
4
5     public static void main(String[] args) {
6         new LogInWindow().setVisible(true);
7     }
8 }
9
10
```

```
Authentication.java x
Source History
1 package event;
2
3 import java.awt.Component;
4 import java.io.BufferedWriter;
5 import java.io.File;
6 import java.io.FileNotFoundException;
7 import java.io.FileWriter;
8 import java.io.IOException;
9 import java.util.Scanner;
10 import javax.swing.JOptionPane;
11
12 public class Authentication {
13     static int userid;
14
15     public Authentication() {
16     }
17
18     public static int register(String email, String password) throws IOException {
19         boolean checkEmail = false;
20         boolean checkPassword = false;
21         boolean flag = false;
22
23         try {
24             File userfile = new File("users.txt");
25             Scanner myReader = new Scanner(userfile);
26
27             while(myReader.hasNextLine()) {
28                 String data = myReader.nextLine();
29                 String[] parts = data.split("/");
30                 if (parts[0].equals(email)) {
31                     flag = true;
32                     break;
33                 }
34             }
35         } catch (FileNotFoundException var13) {
36             System.out.println("User file not found.");
37             var13.printStackTrace();
38         }
39
40         if (flag) {
41             JOptionPane.showMessageDialog((Component)null, "The entered e-mail already exists");
42         } else if (email.contains("@") && email.contains(".com")) {
43             if (email.contains("/")) {
44                 JOptionPane.showMessageDialog((Component)null, "E-mail must not contain /.");
45             } else {
46                 checkEmail = true;
47             }
48         } else {
49             JOptionPane.showMessageDialog((Component)null, "The entered e-mail is invalid");
50         }
51     }
52 }
```

```
50     }
51
52     if (password.contains("/")) {
53         JOptionPane.showMessageDialog((Component) null, "Password must not contain /."
54     } else {
55         checkPassword = true;
56     }
57
58     String userformat = email + "/" + password + "/0-0-0-0-0/";
59     int index = -1;
60     if (checkEmail && checkPassword) {
61         FileWriter fw = new FileWriter("users.txt", true);
62         BufferedWriter bw = new BufferedWriter(fw);
63
64         try {
65             bw.write(userformat + "0-");
66         } catch (Throwable var12) {
67             try {
68                 bw.close();
69             } catch (Throwable var11) {
70                 var12.addSuppressed(var11);
71             }
72
73             throw var12;
74         }
75
76         bw.close();
77         File userfile = new File("users.txt");
78         Scanner myReader = new Scanner(userfile);
79
80         while(myReader.hasNextLine()) {
81             ++index;
82             myReader.nextLine();
83         }
84     }
85
86     return index;
87 }
88
89 public static int login(String email, String password) {
90     int index = -1;
91     boolean flag = false;
92
93     File userfile;
94     Scanner myReader;
95     String data;
96     String[] parts;
97     try {
98         userfile = new File("users.txt");
```

```
Source History
99     myReader = new Scanner(userfile);
100
101     while(myReader.hasNextLine()) {
102         ++index;
103         data = myReader.nextLine();
104         parts = data.split("/");
105         if (parts[0].equals(email)) {
106             flag = true;
107             break;
108         }
109     }
110 } catch (FileNotFoundException var9) {
111     System.out.println("User file not found.");
112     var9.printStackTrace();
113 }
114
115 if (flag) {
116     try {
117         userfile = new File("users.txt");
118         myReader = new Scanner(userfile);
119
120         for(int i = 0; i < index; ++i) {
121             myReader.nextLine();
122         }
123
124         data = myReader.nextLine();
125         parts = data.split("/");
126         if (password.equals(parts[1])) {
127             return index;
128         } else {
129             JOptionPane.showMessageDialog((Component)null, "Incorrect password, please try again");
130             return -1;
131         }
132     } catch (FileNotFoundException var8) {
133         System.out.println("User file not found.");
134         var8.printStackTrace();
135         return index;
136     }
137 } else {
138     JOptionPane.showMessageDialog((Component)null, "No user found using that e-mail");
139     return -1;
140 }
141 }
142 }
```

```
1 package event;
2
3 import java.io.*;
4 import java.nio.file.*;
5 import java.util.*;
6
7 public class User {
8     static ArrayList<Event> userevents = new ArrayList<>();
9
10    public static void changePassword(String username, String newPassword) {
11        try {
12            // Read the file line by line
13             BufferedReader reader = new BufferedReader(new FileReader("users.txt"));
14            String line;
15            StringBuilder sb = new StringBuilder();
16            while ((line = reader.readLine()) != null) {
17                // Split the line into parts
18                String[] parts = line.split("/");
19                if (parts[0].equals(username)) {
20                    // Update the password
21                    parts[1] = newPassword;
22                    line = parts[0] + "/" + parts[1] + "/" + parts[2] + "/";
23                }
24                sb.append(line).append("\n");
25            }
26            reader.close();
27
28            // Write the updated file
29            BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
30            writer.write(sb.toString());
31            writer.close();
32        } catch (IOException e) {
33            e.printStackTrace();
34        }
35    }
36
37    public static boolean checkPassword(String username, String enteredPassword) {
38        try {
39            // Read the file line by line
40             BufferedReader reader = new BufferedReader(new FileReader("users.txt"));
41            String line;
42            while ((line = reader.readLine()) != null) {
43                // Split the line into parts
44                String[] parts = line.split("/");
45                if (parts[0].equals(username)) {
46                    // Check if the entered password matches the stored password
47                    if (parts[1].equals(enteredPassword)) {
48                        reader.close();
49                        return true;
50                    }
51                }
52            }
53            reader.close();
54            return false;
55        } catch (IOException e) {
56            e.printStackTrace();
57        }
58    }
59 }
```

```
Source History
50         } else {
51             reader.close();
52             return false;
53         }
54     }
55 }
56 reader.close();
57 } catch (IOException e) {
58     e.printStackTrace();
59 }
60 return false;
61 }
62
63 public static void addEvent(int userid, String event) throws IOException{
64
65     try {
66         // Read the file and store each line in a list
67         List<String> lines = Files.readAllLines(Paths.get("users.txt"));
68
69         // Get the line to be updated
70         String line = lines.get(userid);
71
72         // Adding event at the end of the line
73         StringBuilder newline = new StringBuilder(line);
74         if(!User.checkEvent(userid, event)) {
75             System.out.println("adding event " + event);
76             newline.append(event + "_");
77         }
78         System.out.println(newline);
79         lines.set(userid, newline.toString());
80
81         BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
82         for (String l : lines) {
83             writer.write(l);
84             writer.newLine();
85         }
86         writer.close();
87     } catch (IOException e) {
88         e.printStackTrace();
89     }
90 }
91
92
93
94
95 public static void removeEvent(int userid, String event) throws IOException{
96
97     try {
98         // Read the file and store each line in a list
```



```
99      List<String> lines = Files.readAllLines(Paths.get("users.txt"));
100
101      // Get the line to be updated
102      String line = lines.get(userid);
103
104      // Making a new line without event
105      StringBuilder newline = new StringBuilder(line);
106      if(User.checkEvent(userid, event)) {
107          System.out.println("removing event " + event);
108          newline.reverse();
109          int index = newline.indexOf("-" + event);
110          if (index != -1)
111          {
112              newline.replace(index, index+2, "");
113          }
114          newline.reverse();
115      }
116
117      System.out.println(newline);
118      lines.set(userid, newline.toString());
119
120      
121      BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
122      for (String l : lines) {
123          writer.write(l);
124          writer.newLine();
125      }
126      writer.close();
127      } catch (IOException e) {
128          e.printStackTrace();
129      }
130  }
131
132  public static boolean checkEvent(int userid, String event) throws IOException{
133      try
134      {
135          File userfile = new File("users.txt");
136          Scanner myReader = new Scanner(userfile);
137
138          for(int i = 0; i < userid; i++) {
139              myReader.nextLine();
140          }
141          String data = myReader.nextLine();
142          String[] parts = data.split("/");
143
144          String[] events = parts[3].split("-");
145
146          int check = 0;
147          for(int i = events.length-1; i >= 1; i--)
```

```
148     {
149         // Checks if user is registered to event
150         if(events[i].equals(event))
151         {
152             check = 1;
153         }
154     }
155     if (check == 1) return true;
156     else return false;
157
158 } catch (FileNotFoundException e) {
159     System.out.println("User file not found.");
160     e.printStackTrace();
161 }
162 return false;
163 }
164
165 public static List getEvents(int userid)throws IOException{
166
167     List<String> regEvents = new ArrayList<String>();
168     try
169     {
170         File userfile = new File("users.txt");
171         Scanner myReader = new Scanner(userfile);
172
173         for(int i = 0; i < userid; i++) {
174             myReader.nextLine();
175         }
176         String data = myReader.nextLine();
177         String[] parts = data.split("/");
178
179         String[] events = parts[3].split("-");
180
181         for(int i = events.length-1; i >= 1; i--)
182         {
183             regEvents.add(events[i]);
184         }
185
186     } catch (FileNotFoundException e) {
187         System.out.println("User file not found.");
188         e.printStackTrace();
189     }
190     return regEvents;
191 }
192
193 public static void addCategory(int userid, String category)throws IOException{
194
195     try {
196         // Read the file and store each line in a list
```

```
Source History
196 // Read the file and store each line in a list
197 List<String> lines = Files.readAllLines(Paths.get("users.txt"));
198
199 // Get the line to be updated
200 String line = lines.get(userid);
201
202 // Separate all lines
203 String[] parts = line.split("/");
204 int categorynr = 0;
205
206 // Separate the categories
207 String[] categories = parts[2].split("-");
208
209 // Find what boolean represents the category
210 switch(category)
211 {
212     case "Other": categorynr = 4; break;
213     case "Festival": categorynr = 3; break;
214     case "Movie": categorynr = 2; break;
215     case "Art": categorynr = 1; break;
216     case "Concert":
217
218     default: categorynr = 0; break;
219 }
220
221 categories[categorynr] = "1";
222
223 // Make the new line
224 String newcategories = categories[0] + "-" + categories[1] + "-" + categories[2] + "-" + categories[3] + "-" + categories[4];
225 StringBuilder newparts = new StringBuilder();
226
227 int count = 0;
228 for (String s : parts)
229 {
230     if (count == 2)
231     {
232         newparts.append(newcategories);
233         newparts.append("/");
234     }
235     else
236     {
237         newparts.append(s);
238         newparts.append("/");
239     }
240     count++;
241 }
242 // Remove last "/" to not interfere with events
243 newparts.reverse();
244 newparts.replace(0, 1, "");
```

```
User.java x
Source History
244     newparts.replace(0, 1, "");
245     newparts.reverse();
246
247     lines.set(userid, newparts.toString());
248
249     BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
250     for (String l : lines) {
251         writer.write(l);
252         writer.newLine();
253     }
254     writer.close();
255 } catch (IOException e) {
256     e.printStackTrace();
257 }
258
259 }
260
261 public static void removeCategory(int userid, String category) throws IOException {
262
263     try {
264         // Read the file and store each line in a list
265         List<String> lines = Files.readAllLines(Paths.get("users.txt"));
266
267         // Get the line to be updated
268         String line = lines.get(userid);
269
270         // Separate all lines
271         String[] parts = line.split("/");
272         int categorynr = 0;
273
274         // Separate the categories
275         String[] categories = parts[2].split("-");
276
277         // Find what boolean represents the category
278         switch(category)
279         {
280             case "Other": categorynr = 4; break;
281             case "Festival": categorynr = 3; break;
282             case "Movie": categorynr = 2; break;
283             case "Art": categorynr = 1; break;
284             case "Concert":
285
286                 default: categorynr = 0; break;
287         }
288
289         categories[categorynr] = "0";
290
291         // Make the new line
292         String newcategories = categories[0] + "-" + categories[1] + "-" + categories[2] + "-" + categories[3] + "-" + categories[4];
```

```
Source History
292 String newcategories = categories[0] + "-" + categories[1] + "-" + categories[2] + "-" + categories[3] + "-" + categories[4];
293 StringBuilder newparts = new StringBuilder();
294
295 int count = 0;
296 for (String s : parts)
297 {
298     if (count == 2)
299     {
300         newparts.append(newcategories);
301         newparts.append("/");
302     }
303     else
304     {
305         newparts.append(s);
306         newparts.append("/");
307     }
308     count++;
309 }
310 // Remove last "/" to not interfere with events
311 newparts.reverse();
312 newparts.replace(0, 1, "");
313 newparts.reverse();
314
315 lines.set(userid, newparts.toString());
316
317
318 BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
319 for (String l : lines) {
320     writer.write(l);
321     writer.newLine();
322 }
323 writer.close();
324 } catch (IOException e) {
325     e.printStackTrace();
326 }
327
328
329 public static boolean checkCategory(int userid, String category) throws IOException{
330
331     try
332     {
333         File userfile = new File("users.txt");
334         Scanner myReader = new Scanner(userfile);
335
336         for(int i = 0; i < userid; i++) {
337             myReader.nextLine();
338         }
339         String data = myReader.nextLine();
340         String[] parts = data.split("/");
```

```
338     }
339     String data = myReader.nextLine();
340     String[] parts = data.split("/");
341
342     int categorynr = 0;
343
344     switch(category)
345     {
346         case "Other": categorynr = 4; break;
347         case "Festival": categorynr = 3; break;
348         case "Movie": categorynr = 2; break;
349         case "Art": categorynr = 1; break;
350         case "Concert":
351
352         default: categorynr = 0; break;
353     }
354
355     String[] categories = parts[2].split("-");
356
357     int check = Integer.parseInt(categories[categorynr]);
358     if (check == 1)
359     {
360         return true;
361     }
362
363 } catch (FileNotFoundException e) {
364     System.out.println("User file not found.");
365     e.printStackTrace();
366 }
367 return false;
368 }
369 }
370 }
```

```
Event.java x
Source History
1 package event;
2
3 import java.io.*;
4 import java.text.ParseException;
5 import java.text.SimpleDateFormat;
6 import java.util.ArrayList;
7 import java.util.Calendar;
8 import java.util.Date;
9 import java.util.List;
10 import java.util.Scanner;
11 import javax.swing.DefaultListModel;
12 import javax.swing.JOptionPane;
13
14 public class Event {
15
16     static ArrayList<Event> events = new ArrayList<>();
17
18     public Event() {
19     }
20
21     public static void addEvent(int id, String name, String category, String startDate, String endDate,
22         String country, String city, String location, int priceTicket, int nrTickets, String duration) throws FileNotFoundException, IOException{
23
24         boolean flag = false;
25         boolean checkEvent = false;
26
27         int index = 1;
28         try
29         {
30             File userfile = new File("events.txt");
31             Scanner myReader = new Scanner(userfile);
32             while (myReader.hasNextLine()) {
33
34                 String data = myReader.nextLine();
35                 index++;
36                 String[] parts = data.split(" / ");
37                 if (parts[1].equals(name)) {
38
39                     flag = true;
40                     break;
41                 }
42             }
43         } catch (FileNotFoundException e) {
44             System.out.println("User file not found.");
45             e.printStackTrace();
46         }
47
48         if(flag){
49             JOptionPane.showMessageDialog(null, "The entered event already exists");
50
51         }
52     }
53
54     String eventFormat = index + " / " + name + " / " + category + " / " + startDate + " / " + endDate + " / " + country + " / " + city + " / " + location + " / " + priceTicket + " / " + nrTickets + " / " + duration;
55
56     if(checkEvent == true){
57         FileWriter fw = new FileWriter("events.txt", true);
58         try (BufferedWriter bw = new BufferedWriter(fw)) {
59             bw.write(eventFormat);
60             bw.newLine();
61         }
62     }
63
64     File userfile = new File("events.txt");
65     Scanner myReader = new Scanner(userfile);
66
67     while(myReader.hasNextLine()) {
68         index++;
69         myReader.nextLine();
70     }
71 }
72
73
74
75
76 public static void removeEvent(String value) throws FileNotFoundException, IOException, ParseException{
77     File file = new File("events.txt");
78     List<String> lines = new ArrayList<>();
79
80     try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
81         String line;
82
83         while ((line = reader.readLine()) != null) {
84             String[] parts = line.split(" / ");
85             SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
86             Date today = Calendar.getInstance().getTime();
87             Date dateToGet = format.parse(parts[3]);
88             long diffInMilliseconds = dateToGet.getTime() - today.getTime();
89             long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
90             String days = Long.toString(diffInDays);
91             String print = parts[1] + " - " + parts[2] + " | Tickets available: " + parts[9] + " | " + parts[3] + " - " + days + " days until";
92             if (!print.equals(value)) {
93                 lines.add(line);
94             }
95         }
96     } catch (IOException e) {
97         e.printStackTrace();
98     }
99 }
```

```
Event.java x
Source History
50
51 }
52 else {
53     checkEvent = true;
54     JOptionPane.showMessageDialog(null, "Event added successfully");
55 }
56
57 String eventFormat = index + " / " + name + " / " + category + " / " + startDate + " / " + endDate + " / " + country + " / " + city + " / " + location + " / " + priceTicket + " / " + nrTickets + " / " + duration;
58
59 if(checkEvent == true){
60     FileWriter fw = new FileWriter("events.txt", true);
61     try (BufferedWriter bw = new BufferedWriter(fw)) {
62         bw.write(eventFormat);
63         bw.newLine();
64     }
65 }
66
67 File userfile = new File("events.txt");
68 Scanner myReader = new Scanner(userfile);
69
70 while(myReader.hasNextLine()) {
71     index++;
72     myReader.nextLine();
73 }
74
75
76
77 public static void removeEvent(String value) throws FileNotFoundException, IOException, ParseException{
78     File file = new File("events.txt");
79     List<String> lines = new ArrayList<>();
80
81     try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
82         String line;
83
84         while ((line = reader.readLine()) != null) {
85             String[] parts = line.split(" / ");
86             SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
87             Date today = Calendar.getInstance().getTime();
88             Date dateToGet = format.parse(parts[3]);
89             long diffInMilliseconds = dateToGet.getTime() - today.getTime();
90             long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
91             String days = Long.toString(diffInDays);
92             String print = parts[1] + " - " + parts[2] + " | Tickets available: " + parts[9] + " | " + parts[3] + " - " + days + " days until";
93             if (!print.equals(value)) {
94                 lines.add(line);
95             }
96         }
97     } catch (IOException e) {
98         e.printStackTrace();
99     }
100 }
```

```
Event.java x
Source History
99
100
101     try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
102         for (String line : lines) {
103             writer.write(line);
104             writer.newLine();
105         }
106     } catch (IOException e) {
107         e.printStackTrace();
108     }
109
110 public static void checkEvent(String category, DefaultListModel l) throws ParseException{
111     List<String> lines = new ArrayList<>();
112     try {
113         BufferedReader br = new BufferedReader(new FileReader("events.txt"));
114         String line;
115         while ((line = br.readLine()) != null) {
116             String[] parts = line.split(" / ");
117
118             SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
119             Date today = Calendar.getInstance().getTime();
120             Date dateToGet = format.parse(parts[3]);
121             long diffInMilliseconds = dateToGet.getTime() - today.getTime();
122             long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
123             String days = Long.toString(diffInDays);
124
125             if(Integer.parseInt(days) > 0){
126                 lines.add(line);
127             }
128             if (parts[2].trim().equals(category)) {
129                 if(parts[9].equals("0")){
130                     parts[9] = "Sold out";
131                 }
132
133                 String print = parts[1] + " - " + parts[2] + " | Tickets available: " + parts[9] + " | " + parts[3] + " - " + days + " days until";
134                 l.addElement(print);
135             }
136         }
137         br.close();
138     } catch (IOException e) {
139         e.printStackTrace();
140     }
141     try (BufferedWriter writer = new BufferedWriter(new FileWriter("events.txt"))) {
142         for (String line : lines) {
143             writer.write(line);
144             writer.newLine();
145         }
146     } catch (IOException e) {
147         e.printStackTrace();
148     }
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
```

```
Event.java x
Source History
148
149
150
151
152 public static String announceEvent() throws ParseException{
153     try {
154         BufferedReader br = new BufferedReader(new FileReader("events.txt"));
155         String line;
156         while ((line = br.readLine()) != null) {
157             String[] parts = line.split(" / ");
158
159             SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
160             Date today = Calendar.getInstance().getTime();
161             Date dateToGet = format.parse(parts[3]);
162             long diffInMilliseconds = dateToGet.getTime() - today.getTime();
163             long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
164             String days = Long.toString(diffInDays);
165
166             if(days.equals("10") || days.equals("5") || days.equals("1")){
167                 return days + " more day/s until " + parts[1] + " begins!";
168             }
169
170         }
171         br.close();
172     } catch (IOException e) {
173         e.printStackTrace();
174     }
175     return "There are no events events happening soon";
176
177
178 public static void checkEventReg(String id, DefaultListModel l) throws ParseException {
179     try {
180         BufferedReader br = new BufferedReader(new FileReader("events.txt"));
181
182         String line;
183         while((line = br.readLine()) != null) {
184             String[] parts = line.split(" / ");
185             SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
186             Date today = Calendar.getInstance().getTime();
187             Date dateToGet = format.parse(parts[3]);
188             long diffInMilliseconds = dateToGet.getTime() - today.getTime();
189             long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
190             String days = Long.toString(diffInDays);
191             if (parts[0].trim().equals(id)) {
192                 String print = parts[1] + " - " + parts[2] + " | Tickets available: " + parts[9] + " | " + parts[3] + " - " + days + " days until";
193                 l.addElement(print);
194             }
195         }
196     }
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



```

EventJava x
Source History
197         br.close();
198     } catch (IOException var6) {
199         var6.printStackTrace();
200     }
201 }
202
203 public static boolean compareEvent(String id, String value) throws ParseException {
204     try {
205         BufferedReader br = new BufferedReader(new FileReader("events.txt"));
206
207         String line;
208         while((line = br.readLine()) != null) {
209             String[] parts = line.split(" / ");
210             SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
211             Date today = Calendar.getInstance().getTime();
212             Date dateToGet = format.parse(parts[3]);
213             long diffInMilliseconds = dateToGet.getTime() - today.getTime();
214             long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
215             String days = Long.toString(diffInDays);
216             if (parts[0].trim().equals(id)) {
217                 String print = parts[1] + " - " + parts[2] + " | Tickets available: " + parts[9] + " | " + parts[3] + " - " + days + " days until";
218                 if (value.equals(print)) {
219                     return true;
220                 }
221             }
222         }
223
224         br.close();
225     } catch (IOException var6) {
226         var6.printStackTrace();
227     }
228     return false;
229 }
230
231
232 public static void subtractTickets(int nrOfTickets, int newNumberOfTickets) {
233     try {
234         // Read the file line by line
235         BufferedReader reader = new BufferedReader(new FileReader("events.txt"));
236         String line;
237         StringBuilder sb = new StringBuilder();
238         while ((line = reader.readLine()) != null) {
239             // Split the line into parts
240             String[] parts = line.split(" / ");
241             if (parts[9].equals(Integer.toString(nrOfTickets))) {
242                 parts[9] = Integer.toString(newNumberOfTickets);
243                 line = parts[0] + " / " + parts[1] + " / " + parts[2] + " / " + parts[3] + " / " + parts[4] + " / " + parts[5] + " / " + parts[6] + " / " + parts[7] + " / " + parts[8] + " / " + parts[9] + " / " + parts[10];
244                 System.out.println(parts[9]);
245             }
246             sb.append(line).append("\n");
247         }
248         reader.close();
249
250         // Write the updated file
251         BufferedWriter writer = new BufferedWriter(new FileWriter("events.txt"));
252         writer.write(sb.toString());
253         writer.close();
254     } catch (IOException e) {
255         e.printStackTrace();
256     }
257 }
258
259 }

```

```
Source Design History
package event;

import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 *
 * @author dobir
 */
public class LogInWindow extends javax.swing.JFrame {

    /**
     * Creates new form AuthenticationWindow
     */
    public LogInWindow() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void EmailFieldActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void PasswordFieldActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void LoginButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if(EmailField.getText().isEmpty() || PasswordField.getText().isEmpty()){
            JOptionPane.showMessageDialog(null, "Please fill all the forms!");
        }
        else{
            String email = EmailField.getText();
            String password = PasswordField.getText();

            int user = Authentication.login(email, password);
            if(email.equals("admin") && password.equals("admin")){
                dispose();
                new AdminWindow().setVisible(true);
            }
        }
    }
}
```

```

195     }
196     else if (user >= 0) {
197         Authentication.userid = user;
198         dispose();
199
200         try {
201             new UserInterface(email, user).setVisible(true);
202         } catch (IOException | ParseException ex) {
203             Logger.getLogger(LoginWindow.class.getName()).log(Level.SEVERE, null, ex);
204         }
205     }
206 }
207
208 private void SignUpButtonActionPerformed(java.awt.event.ActionEvent evt) {
209     dispose();
210     new SignUpWindow().setVisible(true);
211 }
212
213 /**
214  * @param args the command line arguments
215  */
216 public static void main(String args[]) {
217     /* Set the Nimbus look and feel */
218     Look and feel setting code (optional)
219     //</editor-fold>
220
221     /* Create and display the form */
222     java.awt.EventQueue.invokeLater(new Runnable() {
223         public void run() {
224             new LoginWindow().setVisible(true);
225         }
226     });
227 }
228
229 // Variables declaration - do not modify
230 private javax.swing.JTextField EmailField;
231 private javax.swing.JLabel LoginLabel;
232 private javax.swing.JButton LoginButton;
233 private javax.swing.JPasswordField PasswordField;
234 private javax.swing.JButton SignUpButton;
235 private javax.swing.Box.Filler filler1;
236 private javax.swing.JLabel jLabel1;
237 private javax.swing.JSeparator jSeparator2;
238 private javax.swing.JSeparator jSeparator3;
239 // End of variables declaration
240
241 }
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262

```

```
1 package event;
2
3 import java.io.IOException;
4 import java.util.logging.Level;
5 import java.util.logging.Logger;
6 import javax.swing.JOptionPane;
7
8 /**
9  *
10  * @author dobir
11  */
12 public class SignUpWindow extends javax.swing.JFrame {
13
14     /**
15      * Creates new form SignUpWindow
16      */
17     public SignUpWindow() {
18         initComponents();
19     }
20
21     /**
22      * This method is called from within the constructor to initialize the form.
23      * WARNING: Do NOT modify this code. The content of this method is always
24      * regenerated by the Form Editor.
25      */
26     @SuppressWarnings("unchecked")
27     Generated Code
28
29     private void PasswordFieldActionPerformed(java.awt.event.ActionEvent evt) {
30         // TODO add your handling code here:
31     }
32
33     private void EmailFieldActionPerformed(java.awt.event.ActionEvent evt) {
34         // TODO add your handling code here:
35     }
36
37     private void RegisterButtonActionPerformed(java.awt.event.ActionEvent evt) {
38
39         if(EmailField.getText().isEmpty() || PasswordField.getText().isEmpty()) {
40             JOptionPane.showMessageDialog(null, "Please fill all the forms!");
41         } else {
42             String email = EmailField.getText();
43             String password = PasswordField.getText();
44             int user = -1;
45
46             if(MovieCheckBox.isSelected() || ArtCheckBox.isSelected() || ConcertCheckBox.isSelected() || FestivalCheckBox.isSelected() || OtherCheckBox.isSelected()) {
47                 try {
48                     user = Authentication.register(email, password);
49                 } catch (IOException ex) {
50
51                 }
52             }
53         }
54     }
55 }
```

```
Source Design History
261         Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
262     }
263     if (user >= 0) {
264
265         Authentication.userid = user;
266
267         if (MovieCheckBox.isSelected()) {
268             try {
269                 User.addCategory(user, "Movie");
270             } catch (IOException ex) {
271                 Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
272             }
273         }
274         if (ArtCheckBox.isSelected()) {
275             try {
276                 User.addCategory(user, "Art");
277             } catch (IOException ex) {
278                 Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
279             }
280         }
281         if (ConcertCheckBox.isSelected()) {
282             try {
283                 User.addCategory(user, "Concert");
284             } catch (IOException ex) {
285                 Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
286             }
287         }
288         if (FestivalCheckBox.isSelected()) {
289             try {
290                 User.addCategory(user, "Festival");
291             } catch (IOException ex) {
292                 Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
293             }
294         }
295
296         if (OtherCheckBox.isSelected()) {
297             try {
298                 User.addCategory(user, "Other");
299             } catch (IOException ex) {
300                 Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
301             }
302         }
303
304         JOptionPane.showMessageDialog(null, "Registration successful!");
305         dispose();
306         new LogInWindow().setVisible(true);
307     }
308 }
309 } else {
```

SignUpWindow.java x

SourceDesignHistory

310JOptionPane.showMessageDialog(null, "Check at least one box");

311}

312}

313}

314}

315private void SignInButtonActionPerformed(java.awt.event.ActionEvent evt) {

316dispose();

317new LogInWindow().setVisible(true);

318}

319}

320private void ArtCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {

321// TODO add your handling code here:

322}

323}

324private void ConcertCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {

325// TODO add your handling code here:

326}

327}

328/**

329 * @param args the command line arguments

330 */

331public static void main(String args[]) {

332/* Set the Nimbus look and feel */

333Look and feel setting code (optional)

334}

335/* Create and display the form */

336java.awt.EventQueue.invokeLater(new Runnable() {

337public void run() {

338new SignUpWindow().setVisible(true);

339}

340});

341}

342}

343// Variables declaration - do not modify

344private javax.swing.JCheckBox ArtCheckBox;

345private javax.swing.JLabel CategoriesLabel;

346private javax.swing.JCheckBox ConcertCheckBox;

347private javax.swing.JTextField EmailField;

348private javax.swing.JCheckBox FestivalCheckBox;

349private javax.swing.JCheckBox MovieCheckBox;

350private javax.swing.JCheckBox OtherCheckBox;

351private javax.swing.JPasswordField PasswordField;

352private javax.swing.JButton RegisterButton;

353private javax.swing.JLabel SelectEventLabel;

354private javax.swing.JButton SignInButton;

355private javax.swing.JLabel SignUpLabel;

356private javax.swing.Box.Filler filler1;

357private javax.swing.JLabel jLabel1;

358private javax.swing.JSeparator jSeparator2;

```
Source Design History
package event;

import javax.swing.JOptionPane;

/**
 *
 * @author dobir
 */
public class ChangePasswordWindow extends javax.swing.JFrame {
    String user;

    /**
     * Creates new form ChangePasswordWindow
     */
    public ChangePasswordWindow() {
        initComponents();
    }

    public ChangePasswordWindow(String username) {
        initComponents();
        user = username;
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

147
private void ChangeButtonActionPerformed(java.awt.event.ActionEvent evt) {
149     if(OldPasswordField.getText().isEmpty() || NewPasswordField.getText().isEmpty() || ConfirmPasswordField.getText().isEmpty()){
150         JOptionPane.showMessageDialog(null, "All text fields must be filled.");
151     }else{
152         if(User.checkPassword(user, OldPasswordField.getText()){
153             if(User.checkPassword(user, NewPasswordField.getText()){
154                 JOptionPane.showMessageDialog(null, "New password can't be the same as the old password.");
155             }else{
156                 if(NewPasswordField.getText().equals(ConfirmPasswordField.getText()){
157                     User.changePassword(user, NewPasswordField.getText());
158                     dispose();
159                     new LogInWindow().setVisible(true);
160                 }else{
161                     JOptionPane.showMessageDialog(null, "Please make sure your passwords match.");
162                 }
163             }
164         }
165     }
166 }

167 private void OldPasswordFieldActionPerformed(java.awt.event.ActionEvent evt) {
```

```
169     private void OldPasswordFieldActionPerformed(java.awt.event.ActionEvent evt) {  
170     }  
171  
172     /**  
173     * @param args the command line arguments  
174     */  
175     public static void main(String args[]) {  
176         /* Set the Nimbus look and feel */  
177         Look and feel setting code (optional)  
178  
179         /* Create and display the form */  
180         java.awt.EventQueue.invokeLater(new Runnable() {  
181             public void run() {  
182                 new ChangePasswordWindow().setVisible(true);  
183             }  
184         });  
185     }  
186  
187     // Variables declaration - do not modify  
188     private javax.swing.JButton ChangeButton;  
189     private javax.swing.JLabel ConfirmNewPasswordLabel;  
190     private javax.swing.JPasswordField ConfirmPasswordField;  
191     private javax.swing.JPasswordField NewPasswordField;  
192     private javax.swing.JLabel NewPasswordLabel;  
193     private javax.swing.JPasswordField OldPasswordField;  
194     private javax.swing.JLabel jLabel1;  
195     private javax.swing.JLabel jLabel2;  
196     private javax.swing.JSeparator jSeparator1;  
197     private javax.swing.JSeparator jSeparator2;  
198     // End of variables declaration  
199 }  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220
```



```
UserInterface.java
Source Design History
1 package event;
2
3 import java.io.File;
4 import java.io.IOException;
5 import java.text.ParseException;
6 import java.util.List;
7 import java.util.Scanner;
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10 import javax.swing.DefaultListModel;
11 import javax.swing.JOptionPane;
12
13 public class UserInterface extends javax.swing.JFrame {
14     DefaultListModel<String> listModel = new DefaultListModel();
15     DefaultListModel<String> listModel2 = new DefaultListModel();
16
17     public UserInterface() {
18         initComponents();
19     }
20
21     public UserInterface(String username, int userid) throws IOException, ParseException {
22         initComponents();
23
24         RegisterToEventButton.setEnabled(false);
25         UnregisterToEventButton.setEnabled(false);
26         BuyTicketsButton.setEnabled(false);
27
28         UsernameLabel.setText(username);
29         PreferencesEventsList.setModel(listModel);
30         RegisteredEventsList.setModel(listModel2);
31
32         if(User.checkCategory(userid, "Concert")){
33             Event.checkEvent("Concert", listModel);
34         }
35         if(User.checkCategory(userid, "Art")){
36             Event.checkEvent("Art", listModel);
37         }
38         if(User.checkCategory(userid, "Movie")){
39             Event.checkEvent("Movie", listModel);
40         }
41         if(User.checkCategory(userid, "Festival")){
42             Event.checkEvent("Festival", listModel);
43         }
44         if(User.checkCategory(userid, "Other")){
45             Event.checkEvent("Other", listModel);
46         }
47
48         File eventfile = new File("events.txt");
49         Scanner myReader = new Scanner(eventfile);
```

```
Source Design History
50
51     List<String> regEvents = User.getEvents(userid);
52
53     for(int i = 0; i < regEvents.size(); ++i) {
54         Event.checkEventReg((String)regEvents.get(i), this.listModel2);
55     }
56
57     Announce.setText(Event.announceEvent());
58
59 }
60
61 @SuppressWarnings("unchecked")
62 Generated Code
305
306 private void LogOutUserButtonActionPerformed(java.awt.event.ActionEvent evt) {
307     dispose();
308     new LogInWindow().setVisible(true);
309 }
310
311 private void ChangePreferencesButtonActionPerformed(java.awt.event.ActionEvent evt) {
312     dispose();
313     new UserPreferencesInterface().setVisible(true);
314 }
315
316 private void RegisterToEventButtonActionPerformed(java.awt.event.ActionEvent evt) {
317
318     try {
319         File eventfile = new File("events.txt");
320         Scanner myReader = new Scanner(eventfile);
321
322         List<String> regEvents = User.getEvents(Authentication.userid);
323         boolean check = false;
324         for(int i = 0; i < regEvents.size(); ++i) {
325             if(Event.compareEvent((String)regEvents.get(i), PreferencesEventsList.getSelectedValue())){
326                 check = false;
327                 break;
328             }else{
329                 check = true;
330             }
331         }
332     }
333
334     if(check){
335         new RegisterToEventUserInterface(PreferencesEventsList.getSelectedValue()).setVisible(true);
336         dispose();
337     }else{
338         JOptionPane.showMessageDialog(null, "You are already registered to this event.");
339     }
340 } catch (IOException | ParseException ex) {
```

```
Source Design History
Logger.getLogger(UserInterface.class.getName()).log(Level.SEVERE, null, ex);
}
}

private void BuyTicketsButtonActionPerformed(java.awt.event.ActionEvent evt) {
    if(!PreferencesEventsList.isSelectionEmpty())
        new BuyTicketsUserInterface(PreferencesEventsList.getSelectedValue()).setVisible(true);
    if(!RegisteredEventsList.isSelectionEmpty())
        new BuyTicketsUserInterface(RegisteredEventsList.getSelectedValue()).setVisible(true);
}

private void PreferencesEventsListMouseClicked(java.awt.event.MouseEvent evt) {
    if(PreferencesEventsList.getSelectedIndex() > -1){
        RegisteredEventsList.clearSelection();
        RegisterToEventButton.setEnabled(true);
        BuyTicketsButton.setEnabled(true);
        UnregisterToEventButton.setEnabled(false);
    }
}

private void ChangePasswordActionPerformed(java.awt.event.ActionEvent evt) {
    dispose();
    new ChangePasswordWindow(UsernameLabel.getText()).setVisible(true);
}

private void UnregisterToEventButtonActionPerformed(java.awt.event.ActionEvent evt) {
    dispose();
    new UnregisterToEventUserInterface(RegisteredEventsList.getSelectedValue()).setVisible(true);
}

private void RegisteredEventsListMouseClicked(java.awt.event.MouseEvent evt) {
    if(RegisteredEventsList.getSelectedIndex() > -1){
        PreferencesEventsList.clearSelection();
        UnregisterToEventButton.setEnabled(true);
        BuyTicketsButton.setEnabled(true);
        RegisterToEventButton.setEnabled(false);
    }
}

private void AnnounceActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
```

```
387 public static void main(String args[]) {
388     /* Set the Nimbus look and feel */
389     Look and feel setting code (optional)
410
411     /* Create and display the form */
412     java.awt.EventQueue.invokeLater(new Runnable() {
413         public void run() {
414             new UserInterface().setVisible(true);
415         }
416     });
417 }
418
419 // Variables declaration - do not modify
420 private javax.swing.JTextField Announce;
421 private javax.swing.JButton BuyTicketsButton;
422 private javax.swing.JButton ChangePassword;
423 private javax.swing.JButton ChangePreferencesButton;
424 private javax.swing.JButton LogOutUserButton;
425 private javax.swing.JLabel MenuLabel;
426 private javax.swing.JLabel ParticipatingLabel;
427 private javax.swing.JLabel PreferencesButton;
428 private javax.swing.JList<String> PreferencesEventsList;
429 private javax.swing.JButton RegisterToEventButton;
430 public javax.swing.JList<String> RegisteredEventsList;
431 private javax.swing.JButton UnregisterToEventButton;
432 public static javax.swing.JLabel UsernameLabel;
433 private javax.swing.JLabel WelcomeLabel;
434 private javax.swing.JScrollPane jScrollPane1;
435 private javax.swing.JScrollPane jScrollPane2;
436 private javax.swing.JSeparator jSeparator1;
437 // End of variables declaration
438 }
439
```

```
1 package event;
2
3 import java.io.IOException;
4 import java.text.ParseException;
5 import java.util.logging.Level;
6 import java.util.logging.Logger;
7 import javax.swing.JOptionPane;
8
9 /**
10  *
11  * @author dobir
12  */
13 public class UserPreferencesInterface extends javax.swing.JFrame {
14
15     /**
16      * Creates new form UserPreferencesInterface
17      */
18     public UserPreferencesInterface() {
19         initComponents();
20
21         try {
22             if (User.checkCategory(Authentication.userid, "Art")) {
23                 ArtCheckBox.setSelected(true);
24             }
25             if (User.checkCategory(Authentication.userid, "Concert")) {
26                 ConcertCheckBox.setSelected(true);
27             }
28             if (User.checkCategory(Authentication.userid, "Movie")) {
29                 MovieCheckBox.setSelected(true);
30             }
31             if (User.checkCategory(Authentication.userid, "Festival")) {
32                 FestivalCheckBox.setSelected(true);
33             }
34             if (User.checkCategory(Authentication.userid, "Other")) {
35                 OtherCheckBox.setSelected(true);
36             }
37         } catch (IOException ex) {
38             Logger.getLogger(UserPreferencesInterface.class.getName()).log(Level.SEVERE, null, ex);
39         }
40     }
41
42     public UserPreferencesInterface(int userid) {
43         initComponents();
44     }
45
46     /**
47      * This method is called from within the constructor to initialize the form.
48      * WARNING: Do NOT modify this code. The content of this method is always
49      * regenerated by the Form Editor.
```

```
UserPreferencesInterface.java x
Source Design History
50 +/
51 @SuppressWarnings("unchecked")
52 Generated Code
193
195 private void FinishButtonActionPerformed(java.awt.event.ActionEvent evt) {
196     try {
197         if (MovieCheckBox.isSelected() || ArtCheckBox.isSelected() || ConcertCheckBox.isSelected() || FestivalCheckBox.isSelected() || OtherCheckBox.isSelected()) {
198             if (ArtCheckBox.isSelected()) {
199                 if (!User.checkCategory(Authentication.userid, "Art")) {
200                     User.addCategory(Authentication.userid, "Art");
201                 }
202             } else {
203                 User.removeCategory(Authentication.userid, "Art");
204             }
205             if (ConcertCheckBox.isSelected()) {
206                 if (!User.checkCategory(Authentication.userid, "Music")) {
207                     User.addCategory(Authentication.userid, "Music");
208                 }
209             } else {
210                 User.removeCategory(Authentication.userid, "Music");
211             }
212             if (MovieCheckBox.isSelected()) {
213                 if (!User.checkCategory(Authentication.userid, "Movie")) {
214                     User.addCategory(Authentication.userid, "Movie");
215                 }
216             } else {
217                 User.removeCategory(Authentication.userid, "Movie");
218             }
219             if (FestivalCheckBox.isSelected()) {
220                 if (!User.checkCategory(Authentication.userid, "Festival")) {
221                     User.addCategory(Authentication.userid, "Festival");
222                 }
223             } else {
224                 User.removeCategory(Authentication.userid, "Festival");
225             }
226             if (OtherCheckBox.isSelected()) {
227                 if (!User.checkCategory(Authentication.userid, "Other")) {
228                     User.addCategory(Authentication.userid, "Other");
229                 }
230             } else {
231                 User.removeCategory(Authentication.userid, "Other");
232             }
233             dispose();
234             new UserInterface(UserInterface.UsernameLabel.getText(), Authentication.userid).setVisible(true);
235         } else {
236             JOptionPane.showMessageDialog(null, "Check at least one box");
237         }
238     } catch (IOException ex) {
```

```

Source Design History
239         Logger.getLogger(UserPreferencesInterface.class.getName()).log(Level.SEVERE, null, ex);
240     } catch (ParseException ex) {
241         Logger.getLogger(UserPreferencesInterface.class.getName()).log(Level.SEVERE, null, ex);
242     }
243
244
245     }
246
247     private void FestivalCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
248
249     }
250
251     private void MovieCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
252
253     }
254
255     private void ArtCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
256
257     }
258
259     private void ConcertCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
260
261     }
262
263     private void OtherCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
264         // TODO add your handling code here:
265     }
266
267     /**
268     * @param args the command line arguments
269     */
270     public static void main(String args[]) {
271         /* Set the Nimbus look and feel */
272         Look and feel setting code (optional)
273
274         /* Create and display the form */
275         java.awt.EventQueue.invokeLater(new Runnable() {
276             public void run() {
277                 new UserPreferencesInterface().setVisible(true);
278             }
279         });
280     }
281
282     // Variables declaration - do not modify
283     public static javax.swing.JCheckBox ArtCheckBox;
284     private javax.swing.JLabel CategoriesLabel;
285     public static javax.swing.JCheckBox ConcertCheckBox;
286     public static javax.swing.JCheckBox FestivalCheckBox;
287     private javax.swing.JButton FinishButton;

```

Source Design History

```
264 // TODO add your handling code here:
265 }
266
267 /**
268  * @param args the command line arguments
269  */
270 public static void main(String args[]) {
271     /* Set the Nimbus look and feel */
272     Look and feel setting code (optional)
273
274     /* Create and display the form */
275     java.awt.EventQueue.invokeLater(new Runnable() {
276         public void run() {
277             new UserPreferencesInterface().setVisible(true);
278         }
279     });
280 }
281
282 // Variables declaration - do not modify
283 public static javax.swing.JCheckBox ArtCheckBox;
284 private javax.swing.JLabel CategoriesLabel;
285 public static javax.swing.JCheckBox ConcertCheckBox;
286 public static javax.swing.JCheckBox FestivalCheckBox;
287 private javax.swing.JButton FinishButton;
288 public static javax.swing.JCheckBox MovieCheckBox;
289 public static javax.swing.JCheckBox OtherCheckBox;
290 private javax.swing.JLabel PreferencesLabel;
291 private javax.swing.JLabel YourPreferencesLabel;
292 private javax.swing.JSeparator jSeparator1;
293 private javax.swing.JSeparator jSeparator2;
294 // End of variables declaration
295 }
296
297 }
```



```
Source Design History
1 package event;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.text.ParseException;
7 import java.util.logging.Level;
8 import java.util.logging.Logger;
9
10 /**
11  *
12  * @author dobir
13  */
14 public class RegisterToEventUserInterface extends javax.swing.JFrame {
15
16     /**
17      * Creates new form RegisterToEventUserInterface
18      */
19     public RegisterToEventUserInterface() {
20         initComponents();
21     }
22
23     public RegisterToEventUserInterface(String value) throws IOException {
24         initComponents();
25
26
27
28         try (BufferedReader br = new BufferedReader(new FileReader("events.txt"))) {
29             String line;
30             while ((line = br.readLine()) != null) {
31                 String eventName = value.substring(0, value.indexOf(" - "));
32                 String[] parts = line.split(" / ");
33                 if (parts[1].trim().equals(eventName.trim())) {
34                     SelectedEventForm.setText(parts[1]);
35                     CountryField.setText(parts[5]);
36                     CityField.setText(parts[6]);
37                     AddressField.setText(parts[7]);
38                     DateOfEventField.setText(parts[3]);
39                     TicketsAvailableField.setText(parts[8]);
40                     PriceOfTicketsField.setText(parts[9]);
41                 }
42             }
43         }
44
45     }
46
47     /**
48      * This method is called from within the constructor to initialize the form.
49      * WARNING: Do NOT modify this code. The content of this method is always
```

```

50      * regenerated by the Form Editor.
51      */
52      @SuppressWarnings("unchecked")
53      Generated Code
260
261      private void SelectedEventFormActionPerformed(java.awt.event.ActionEvent evt) {
262
263      }
264
265      private void CountryFieldActionPerformed(java.awt.event.ActionEvent evt) {
266          // TODO add your handling code here:
267      }
268
269      private void CityFieldActionPerformed(java.awt.event.ActionEvent evt) {
270          // TODO add your handling code here:
271      }
272
273      private void AddressFieldActionPerformed(java.awt.event.ActionEvent evt) {
274          // TODO add your handling code here:
275      }
276
277      private void FinishButtonActionPerformed(java.awt.event.ActionEvent evt) {
278          // TODO add your handling code here:
279          try
280          {
281              String eventid = "0";
282
283              BufferedReader br = new BufferedReader(new FileReader("events.txt"));
284              String line;
285              while ((line = br.readLine()) != null) {
286                  String[] parts = line.split(" / ");
287                  if (parts[1].trim().equals(SelectedEventForm.getText())) {
288                      eventid = parts[0];
289                  }
290              }
291              br.close();
292              User.addEvent(Authentication.userid, eventid);
293              dispose();
294              new UserInterface(UserInterface.UsernameLabel.getText(), Authentication.userid).setVisible(true);
295          }
296          catch (IOException e)
297          {
298              e.printStackTrace();
299          } catch (ParseException ex) {
300              Logger.getLogger(RegisterToEventUserInterface.class.getName()).log(Level.SEVERE, null, ex);
301          }
302      }
303
304      private void PriceOfTicketsFieldActionPerformed(java.awt.event.ActionEvent evt) {

```

```
Source Design History
50      * regenerated by the Form Editor.
51      */
52      @SuppressWarnings("unchecked")
53      Generated Code
260
261      private void SelectedEventFormActionPerformed(java.awt.event.ActionEvent evt) {
262
263      }
264
265      private void CountryFieldActionPerformed(java.awt.event.ActionEvent evt) {
266          // TODO add your handling code here:
267      }
268
269      private void CityFieldActionPerformed(java.awt.event.ActionEvent evt) {
270          // TODO add your handling code here:
271      }
272
273      private void AddressFieldActionPerformed(java.awt.event.ActionEvent evt) {
274          // TODO add your handling code here:
275      }
276
277      private void FinishButtonActionPerformed(java.awt.event.ActionEvent evt) {
278          // TODO add your handling code here:
279          try
280          {
281              String eventid = "0";
282
283              BufferedReader br = new BufferedReader(new FileReader("events.txt"));
284              String line;
285              while ((line = br.readLine()) != null) {
286                  String[] parts = line.split(" / ");
287                  if (parts[1].trim().equals(SelectedEventForm.getText())) {
288                      eventid = parts[0];
289                  }
290              }
291              br.close();
292              User.addEvent(Authentication.userid, eventid);
293              dispose();
294              new UserInterface(UserInterface.UsernameLabel.getText(), Authentication.userid).setVisible(true);
295          }
296          catch (IOException e)
297          {
298              e.printStackTrace();
299          } catch (ParseException ex) {
300              Logger.getLogger(RegisterToEventUserInterface.class.getName()).log(Level.SEVERE, null, ex);
301          }
302      }
303
304      private void PriceOfTicketsFieldActionPerformed(java.awt.event.ActionEvent evt) {
```

```

305     // TODO add your handling code here:
306 }
307
308 /**
309  * @param args the command line arguments
310  */
311 public static void main(String args[]) {
312     /* Set the Nimbus look and feel */
313     Look and feel setting code (optional)
314
315     /* Create and display the form */
316     java.awt.EventQueue.invokeLater(new Runnable() {
317         public void run() {
318             new RegisterToEventUserInterface().setVisible(true);
319         }
320     });
321 }
322
323 // Variables declaration - do not modify
324 private javax.swing.JTextField AddressField;
325 private javax.swing.JLabel AddressLabel;
326 private javax.swing.JTextField CityField;
327 private javax.swing.JLabel CityLabel;
328 private javax.swing.JTextField CountryField;
329 private javax.swing.JLabel CountryLabel;
330 private javax.swing.JTextField DateOfEventField;
331 private javax.swing.JLabel DateOfTheEventLabel;
332 private javax.swing.JLabel DateOfTheEventLabel1;
333 private javax.swing.JButton FinishButton;
334 private javax.swing.JTextField PriceOfTicketsField;
335 private javax.swing.JLabel PriceOfTicketsLabel;
336 private javax.swing.JLabel RegisterToAnEventLabel;
337 private javax.swing.JTextField SelectedEventForm;
338 private javax.swing.JLabel SelectedEventLabel;
339 private javax.swing.JTextField TicketsAvailableField;
340 private javax.swing.JSeparator jSeparator1;
341 private javax.swing.JSeparator jSeparator2;
342 // End of variables declaration
343 }
344
345
346

```

```
Source Design History
1 package event;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.text.ParseException;
7 import java.util.logging.Level;
8 import java.util.logging.Logger;
9
10 /**
11  *
12  * @author dobir
13  */
14 public class UnregisterToEventUserInterface extends javax.swing.JFrame {
15
16     /**
17      * Creates new form RegisterToEventUserInterface
18      */
19     public UnregisterToEventUserInterface() {
20         initComponents();
21     }
22
23     public UnregisterToEventUserInterface(String value) {
24         initComponents();
25
26         String eventName = value.substring(0, value.indexOf(" - "));
27
28         try {
29             BufferedReader br = new BufferedReader(new FileReader("events.txt"));
30             String line;
31             while ((line = br.readLine()) != null) {
32                 String[] parts = line.split(" / ");
33                 if (parts[1].trim().equals(eventName.trim())) {
34                     SelectedEventForm.setText(parts[1]);
35                     CountryField.setText(parts[5]);
36                     CityField.setText(parts[6]);
37                     AddressField.setText(parts[7]);
38                     DateOfEventField.setText(parts[3]);
39                     TicketsAvailableField.setText(parts[8]);
40                     PriceOfTicketsField.setText(parts[9]);
41                 }
42             }
43             br.close();
44         } catch (IOException e) {
45             e.printStackTrace();
46         }
47     }
48
49 }
```

```

50  /**
51  * This method is called from within the constructor to initialize the form.
52  * WARNING: Do NOT modify this code. The content of this method is always
53  * regenerated by the Form Editor.
54  */
55  @SuppressWarnings("unchecked")
56  Generated Code
263
264  private void SelectedEventFormActionPerformed(java.awt.event.ActionEvent evt) {
265
266  }
267
268  private void CountryFieldActionPerformed(java.awt.event.ActionEvent evt) {
269      // TODO add your handling code here:
270  }
271
272  private void CityFieldActionPerformed(java.awt.event.ActionEvent evt) {
273      // TODO add your handling code here:
274  }
275
276  private void AddressFieldActionPerformed(java.awt.event.ActionEvent evt) {
277      // TODO add your handling code here:
278  }
279
280  private void FinishButtonActionPerformed(java.awt.event.ActionEvent evt) {
281      // TODO add your handling code here:
282      try
283      {
284          String eventid = "0";
285
286          BufferedReader br = new BufferedReader(new FileReader("events.txt"));
287          String line;
288          while ((line = br.readLine()) != null) {
289              String[] parts = line.split(" / ");
290              if (parts[1].trim().equals(SelectedEventForm.getText())) {
291                  eventid = parts[0];
292              }
293          }
294          br.close();
295          User.removeEvent(Authentication.userid, eventid);
296          dispose();
297          new UserInterface(UserInterface.UsernameLabel.getText(), Authentication.userid).setVisible(true);
298      }
299      catch (IOException e)
300      {
301          e.printStackTrace();
302      } catch (ParseException ex) {
303          Logger.getLogger(UnregisterToEventUserInterface.class.getName()).log(Level.SEVERE, null, ex);
304      }
305  }

```

```

Source Design History
306 private void PriceOfTicketsFieldActionPerformed(java.awt.event.ActionEvent evt) {
307     // TODO add your handling code here:
308 }
309
310
311 /**
312  * @param args the command line arguments
313  */
314 public static void main(String args[]) {
315     /* Set the Nimbus look and feel */
316     Look and feel setting code (optional)
317
318     /* Create and display the form */
319     java.awt.EventQueue.invokeLater(new Runnable() {
320         public void run() {
321             new RegisterToEventUserInterface().setVisible(true);
322         }
323     });
324 }
325
326 // Variables declaration - do not modify
327 private javax.swing.JTextField AddressField;
328 private javax.swing.JLabel AddressLabel;
329 private javax.swing.JTextField CityField;
330 private javax.swing.JLabel CityLabel;
331 private javax.swing.JTextField CountryField;
332 private javax.swing.JLabel CountryLabel;
333 private javax.swing.JTextField DateOfEventField;
334 private javax.swing.JLabel DateOfTheEventLabel;
335 private javax.swing.JLabel DateOfTheEventLabel1;
336 private javax.swing.JButton FinishButton;
337 private javax.swing.JTextField PriceOfTicketsField;
338 private javax.swing.JLabel PriceOfTicketsLabel;
339 private javax.swing.JLabel RegisterToAnEventLabel;
340 private javax.swing.JTextField SelectedEventForm;
341 private javax.swing.JLabel SelectedEventLabel;
342 private javax.swing.JTextField TicketsAvailableField;
343 private javax.swing.JSeparator jSeparator1;
344 private javax.swing.JSeparator jSeparator2;
345 // End of variables declaration
346
347 }
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367

```

```
1 package event;
2
3 import java.awt.print.PrinterException;
4 import java.io.BufferedReader;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.util.Date;
8 import java.util.logging.Level;
9 import java.util.logging.Logger;
10 import javax.swing.JOptionPane;
11
12 /**
13  *
14  * @author dobir
15  */
16 public class BuyTicketsUserInterface extends javax.swing.JFrame {
17
18     /**
19      * Creates new form BuyTicketsUserInterface
20      */
21     public BuyTicketsUserInterface() {
22         initComponents();
23     }
24
25     public BuyTicketsUserInterface(String value) {
26         initComponents();
27
28         BuyButton.setEnabled(false);
29
30         String eventName = value.substring(0, value.indexOf(" | "));
31
32         try {
33             BufferedReader br = new BufferedReader(new FileReader("events.txt"));
34             String line;
35             while ((line = br.readLine()) != null) {
36                 String[] parts = line.split(" / ");
37                 if (parts[1].trim().equals(eventName.trim())) {
38                     SelectedEventBuyField.setText(parts[1]);
39                     CountryBuyField.setText(parts[5]);
40                     CityBuyField.setText(parts[6]);
41                     AddressBuyField.setText(parts[7]);
42                     DateOfEventBuyField.setText(parts[3]);
43                     TicketsAvailableBuyField.setText(parts[9]);
44                     PriceOfTicketsBuyField.setText(parts[8]);
45                 }
46             }
47             br.close();
48         } catch (IOException e) {
49             e.printStackTrace();
50         }
51     }
52 }
```


The screenshot displays an IDE window titled "BuyTicketsUserInterface.java". The interface includes tabs for Source, Design, and History, along with standard development tool icons. The main editor area contains Java code for a ticket purchasing application. The code defines a class with two methods: `buyButtonActionPerformed` and `addressBuyFieldActionPerformed`. The `buyButtonActionPerformed` method handles the logic for purchasing tickets, including validating input fields, calculating totals, and displaying a receipt. The `addressBuyFieldActionPerformed` method is currently a placeholder with a TODO comment. The code uses Swing components like `JSpinner`, `JTextField`, and `JOptionPane` for user interaction.

```
BuyTicketsUserInterface.java
Source Design History
)
50
51
52 NumberOfTicketsToBuy.setModel(new javax.swing.SpinnerNumberModel(1, 1, Integer.parseInt(TicketsAvailableBuyField.getText()), 1));
53 TotalBuyField.setText(((Integer) NumberOfTicketsToBuy.getValue() * Integer.parseInt(PriceOfTicketsBuyField.getText()) + " Lei");
54
55 area.setText("-----\n\n");
56
57 area.setText(area.getText() + "
58                                     Receipt\n");
59 area.setText(area.getText() + "\n-----\n\n");
60
61 Date obj = new Date();
62 String date = obj.toString();
63
64 area.setText(area.getText() + "
65 Event: " + SelectedEventBuyField.getText() + "\n\n");
66 area.setText(area.getText() + "
67 Country and city: " + CountryBuyField.getText() + " " + CityBuyField.getText() + "\n\n");
68 area.setText(area.getText() + "
69 Location: " + AddressBuyField.getText() + "\n\n");
70 area.setText(area.getText() + "
71 Tickets:
72                                     " + PriceOfTicketsBuyField.getText() + " x" + NumberOfTicketsToBuy.getValue() + "");
73
74 area.setText(area.getText() + "\n\n\n
75                                     -----\n\n");
76 area.setText(area.getText() + "
77 Total:
78                                     " + TotalBuyField.getText());
79 area.setText(area.getText() + "\n\n\n
80                                     -----");
81 area.setText(area.getText() + "\n\n\n
82                                     " + date + "
83 Signature: " + UserInterface.UsernameLabel.getText());
84
85 /**
86  * This method is called from within the constructor to initialize the form.
87  * WARNING: Do NOT modify this code. The content of this method is always
88  * regenerated by the Form Editor.
89  */
90 @SuppressWarnings("unchecked")
91 // Generated Code
92
93 private void buyButtonActionPerformed(java.awt.event.ActionEvent evt) {
94     buyButton.setEnabled(false);
95     int value = Integer.parseInt(TicketsAvailableBuyField.getText()) - (Integer) NumberOfTicketsToBuy.getValue();
96     Event.subtractTickets(Integer.parseInt(TicketsAvailableBuyField.getText()), value);
97     try {
98         area.print();
99     } catch (PrinterException ex) {
100         Logger.getLogger(BuyTicketsUserInterface.class.getName()).log(Level.SEVERE, null, ex);
101     }
102     dispose();
103     JOptionPane.showMessageDialog(null, "Purchase complete!");
104 }
105
106 private void addressBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {
107     // TODO add your handling code here:
108 }
```

BuyTicketUserInterface.java

SourceDesignHistory

```
384 )
385
386 private void CountryBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {
387     // TODO add your handling code here:
388 }
389
390 private void CityBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {
391     // TODO add your handling code here:
392 }
393
394 private void SelectedEventBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {
395     // TODO add your handling code here:
396 }
397
398 private void ConfirmPurchaseBoxActionPerformed(java.awt.event.ActionEvent evt) {
399     if(ConfirmPurchaseBox.isSelected()){
400         BuyButton.setEnabled(true);
401     }else{
402         BuyButton.setEnabled(false);
403     }
404 }
405
406 private void TotalBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {
407 }
408
409 private void NumberOfTicketsToBuyStateChanged(javax.swing.event.ChangeEvent evt) {
410
411     Integer myInt = (Integer) NumberOfTicketsToBuy.getValue() * Integer.parseInt(PriceOfTicketsBuyField.getText());
412     String spinner = myInt.toString();
413     TotalBuyField.setText(spinner + " Lei");
414     area.setText(null);
415     area.setText("-----\n\n");
416
417     area.setText(area.getText() + "Receipt\n");
418     area.setText(area.getText() + "\n-----\n\n");
419
420     Date obj = new Date();
421     String date = obj.toString();
422
423     area.setText(area.getText() + "Event: " + SelectedEventBuyField.getText() + "\n\n");
424     area.setText(area.getText() + "Country and city: " + CountryBuyField.getText() + " " + CityBuyField.getText() + "\n\n");
425     area.setText(area.getText() + "Location: " + AddressBuyField.getText() + "\n\n");
426     area.setText(area.getText() + "Tickets: " + PriceOfTicketsBuyField.getText() + " x" + NumberOfTicketsToBuy.getValue() + " = " + myInt + "\n\n");
427     area.setText(area.getText() + "Total: " + TotalBuyField.getText() + "\n\n");
428     area.setText(area.getText() + "Signature: " + UserInterface.UserNameLabel.getText() + "\n\n");
429 }
```

Activate Windows
Go to Settings to activate Windows.

```
Source Design History
434 }
435
436 private void PriceOfTicketsBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {
437     // TODO add your handling code here:
438 }
439
440 /**
441  * @param args the command line arguments
442  */
443 public static void main(String args[]) {
444     /* Set the Nimbus look and feel */
445     Look and feel setting code (optional)
446
447     /* Create and display the form */
448     java.awt.EventQueue.invokeLater(new Runnable() {
449         public void run() {
450             new BuyTicketsUserInterface().setVisible(true);
451         }
452     });
453 }
454
455 // Variables declaration - do not modify
456 private javax.swing.JTextField AddressBuyField;
457 private javax.swing.JLabel AddressBuyLabel;
458 private javax.swing.JButton BuyButton;
459 private javax.swing.JLabel BuyTicketsLabel;
460 private javax.swing.JTextField CityBuyField;
461 private javax.swing.JLabel CityBuyLabel;
462 private javax.swing.JCheckBox ConfirmPurchaseBox;
463 private javax.swing.JTextField CountryBuyField;
464 private javax.swing.JLabel CountryBuyLabel;
465 private javax.swing.JTextField DateOfEventBuyField;
466 private javax.swing.JLabel DateOfTheEventBuyLabel;
467 private javax.swing.JLabel NoOfTicketsBuyLabel;
468 private javax.swing.JSpinner NumberOfTicketsToBuy;
469 private javax.swing.JTextField PriceOfTicketsBuyField;
470 private javax.swing.JLabel PriceOfTicketsBuyLabel;
471 private javax.swing.JTextField SelectedEventBuyField;
472 private javax.swing.JLabel SelectedEventBuyLabel;
473 private javax.swing.JTextField TicketsAvailableBuyField;
474 private javax.swing.JLabel TicketsAvailableBuyLabel;
475 private javax.swing.JTextField TotalBuyField;
476 private javax.swing.JLabel TotalBuyLabel;
477 private javax.swing.JTextArea area;
478 private javax.swing.JScrollPane jScrollPane1;
479 private javax.swing.JSeparator jSeparator1;
480 private javax.swing.JSeparator jSeparator2;
481 private javax.swing.JSeparator jSeparator3;
482 // End of variables declaration
```

```
1 package event;
```

```
2
```

```
3 /**
4  *
5  * @author dobir
6  */
```

```
7 public class AdminWindow extends javax.swing.JFrame {
```

```
8
```

```
9 /**
10  * Creates new form AdminWindow
11  */
```

```
12 public AdminWindow() {
13     initComponents();
14 }
```

```
15
```

```
16 /**
17  * This method is called from within the constructor to initialize the form.
18  * WARNING: Do NOT modify this code. The content of this method is always
19  * regenerated by the Form Editor.
20  */
```

```
21 @SuppressWarnings("unchecked")
```

```
22 Generated Code
```

```
141 private void AddEventButtonActionPerformed(java.awt.event.ActionEvent evt) {
143     new AddEventForm().setVisible(true);
144 }
145
```

```
146 private void RemoveEventButtonActionPerformed(java.awt.event.ActionEvent evt) {
147     new RemoveEventForm().setVisible(true);
148 }
149
```

```
150 private void LogOutButtonActionPerformed(java.awt.event.ActionEvent evt) {
151     dispose();
152     new LogInWindow().setVisible(true);
153 }
154
```

```
155 private void ChangePasswordButton1ActionPerformed(java.awt.event.ActionEvent evt) {
156     dispose();
157     new ChangePasswordWindow("admin").setVisible(true);
158 }
159
```

```
160 /**
```

```
161  * @param args the command line arguments
162  */
```

```
163 public static void main(String args[]) {
```

```
164     /* Set the Nimbus look and feel */
```

```
165     Look and feel setting code (optional)
```

```
186
187     /* Create and display the form */
```

```
188     java.awt.EventQueue.invokeLater(new Runnable() {
189         public void run() {
190             new AdminWindow().setVisible(true);
191         }
192     });
193 }
```

```
194
195 // Variables declaration - do not modify
```

```
196 private javax.swing.JButton AddEventButton;
```

```
197 private javax.swing.JLabel AdminLabel;
```

```
198 private javax.swing.JButton ChangePasswordButton1;
```

```
199 private javax.swing.JButton LogOutButton;
```

```
200 private javax.swing.JButton RemoveEventButton;
```

```
201 private javax.swing.JSeparator jSeparator1;
```

```
202 private javax.swing.JSeparator jSeparator2;
```

```
203 // End of variables declaration
```

```
204 }
```

```
205
```

```

AddEventForm.java
Source Design History
1 package event;
2
3 import java.io.IOException;
4 import java.text.*;
5 import java.util.Date;
6 import java.util.logging.Level;
7 import java.util.logging.Logger;
8 import javax.swing.JOptionPane;
9 import javax.swing.JTextField;
10
11 public class AddEventForm extends javax.swing.JFrame {
12     SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
13     /**
14      * Creates new form AddEventForm
15      */
16     public AddEventForm() {
17         initComponents();
18     }
19
20     /**
21      * This method is called from within the constructor to initialize the form.
22      * WARNING: Do NOT modify this code. The content of this method is always
23      * regenerated by the Form Editor.
24      */
25     @SuppressWarnings("unchecked")
26     // Generated Code
27
28     private void NameOfTheNewEventActionPerformed(java.awt.event.ActionEvent evt) {
29         // TODO add your handling code here:
30     }
31
32     private void CountryNewEventActionPerformed(java.awt.event.ActionEvent evt) {
33         // TODO add your handling code here:
34     }
35
36     private void CityNewEventActionPerformed(java.awt.event.ActionEvent evt) {
37         // TODO add your handling code here:
38     }
39
40     private void AddressFieldActionPerformed(java.awt.event.ActionEvent evt) {
41         // TODO add your handling code here:
42     }
43
44     private void AddButtonActionPerformed(java.awt.event.ActionEvent evt) {
45         if (NameOfTheNewEvent.getText().isEmpty() || CountryNewEvent.getText().isEmpty() || CityNewEvent.getText().isEmpty() || AddressField.getText().isEmpty() ||
46             StartEventCalendar.getDate() == null || EndEventDate.getDate() == null || PriceTicketField.getText().isEmpty() || NrTicketsField.getText().isEmpty()) {
47             JOptionPane.showMessageDialog(null, "Please fill all the forms!");
48         } else {
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
```

```

369     }
370
371     private void NrTicketsFieldActionPerformed(java.awt.event.ActionEvent evt) {
372         // TODO add your handling code here:
373     }
374
375     /**
376     * @param args the command line arguments
377     */
378     public static void main(String args[]) {
379         /* Set the Nimbus look and feel */
380         Look and feel setting code (optional)
381
382         /* Create and display the form */
383         java.awt.EventQueue.invokeLater(new Runnable() {
384             public void run() {
385                 new AddEventForm().setVisible(true);
386             }
387         });
388     }
389
390     // Variables declaration - do not modify
391     private javax.swing.JButton AddButton;
392     private javax.swing.JLabel AddEventLabel;
393     private javax.swing.JTextField AddressField;
394     private javax.swing.JCheckBox ArtCheckBox;
395     private javax.swing.JLabel CategoriesLabel;
396     private javax.swing.JTextField CityNewEvent;
397     private javax.swing.JCheckBox ConcertCheckBox;
398     private javax.swing.JTextField CountryNewEvent;
399     private javax.swing.JLabel DateEventLabel;
400     private com.toedter.calendar.JDateChooser EndEventDate;
401     private javax.swing.JLabel EndEventDateLabel;
402     private javax.swing.JCheckBox FestivalCheckBox;
403     private javax.swing.JCheckBox MovieCheckBox;
404     private javax.swing.JTextField NameOfTheNewEvent;
405     private javax.swing.JTextField NrTicketsField;
406     private javax.swing.JCheckBox OtherCheckBox;
407     private javax.swing.JLabel PriceTicketEventLabel;
408     private javax.swing.JLabel PriceTicketEventLabel2;
409     private javax.swing.JTextField PriceTicketField;
410     private com.toedter.calendar.JDateChooser StartEventCalendar;
411     private javax.swing.ButtonGroup buttonGroup1;
412     private javax.swing.JSeparator jSeparator1;
413     private javax.swing.JSeparator jSeparator2;
414     // End of variables declaration
415 }
416

```

RemoveEventForm.java x

SourceDesignHistory

1package event;

2

3import java.io.BufferedReader;

4import java.io.FileNotFoundException;

5import java.io.FileReader;

6import java.io.IOException;

7import java.text.ParseException;

8import java.text.SimpleDateFormat;

9import java.util.Calendar;

10import java.util.Date;

11import java.util.logging.Level;

12import java.util.logging.Logger;

13import javax.swing.DefaultListModel;

14import javax.swing.JOptionPane;

15

16/**

17 *

18 * @author dobir

19 */

20public class RemoveEventForm extends javax.swing.JFrame {

21 DefaultListModel<String> listModel = new DefaultListModel();

22 /**

23 * Creates new form RemoveEventForm

24 */

25 public RemoveEventForm() {

26 try {

27 initComponents();

28

29 EventsList.setModel(listModel);

30

31 Event.checkEvent("Concert", listModel);

32 Event.checkEvent("Art", listModel);

33 Event.checkEvent("Movie", listModel);

34 Event.checkEvent("Festival", listModel);

35 Event.checkEvent("Other", listModel);

36 } catch (ParseException ex) {

37 Logger.getLogger(RemoveEventForm.class.getName()).log(Level.SEVERE, null, ex);

38 }

39 }

40

41 /**

42 * This method is called from within the constructor to initialize the form.

43 * WARNING: Do NOT modify this code. The content of this method is always

44 * regenerated by the Form Editor.

45 */

46 @SuppressWarnings("unchecked")

47 Generated Code

152private void DeleteButtonActionPerformed(java.awt.event.ActionEvent evt) {

```

154
155
156
157     try
158     {
159         if(!EventsList.isSelectionEmpty())
160         {
161             Event.removeEvent(EventsList.getSelectedValue());
162
163             listModel.clear();
164
165             // Read the contents of the text file and add them to the list
166             BufferedReader br = new BufferedReader(new FileReader("events.txt"));
167             String line;
168
169             while ((line = br.readLine()) != null) {
170                 String[] parts = line.split(" / ");
171                 SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
172                 Date today = Calendar.getInstance().getTime();
173                 Date dateToGet = format.parse(parts[3]);
174                 long diffInMilliseconds = dateToGet.getTime() - today.getTime();
175                 long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
176                 String days = Long.toString(diffInDays);
177                 String print = parts[1] + " - " + parts[2] + " | Tickets available: " + parts[9] + " | " + parts[3] + " - " + days + " days until";
178                 listModel.addElement(print);
179             }
180         }
181         catch (IOException ioe) {
182             System.out.println(ioe);
183         }
184         catch (ParseException ex) {
185             Logger.getLogger(RemoveEventForm.class.getName()).log(Level.SEVERE, null, ex);
186         }
187         JOptionPane.showMessageDialog(null, "Event deleted successfully!");
188     }
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237

```

```

private void BackButtonActionPerformed(java.awt.event.ActionEvent evt) {
    dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    LookAndFeel lookAndFeel = new NimbusLookAndFeel();
    lookAndFeel.installLookAndFeel();

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new RemoveEventForm().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton BackButton;
private javax.swing.JButton DeleteButton;
private javax.swing.JList<String> EventsList;
private javax.swing.JLabel RemoveEventLabel;
private javax.swing.JLabel SelectEventLabel;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JSeparator jSeparator1;
// End of variables declaration
}

```