Proiect: aplicatie E-Event

Proiect creat de Dobîrceanu 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 înfăptuirea 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.

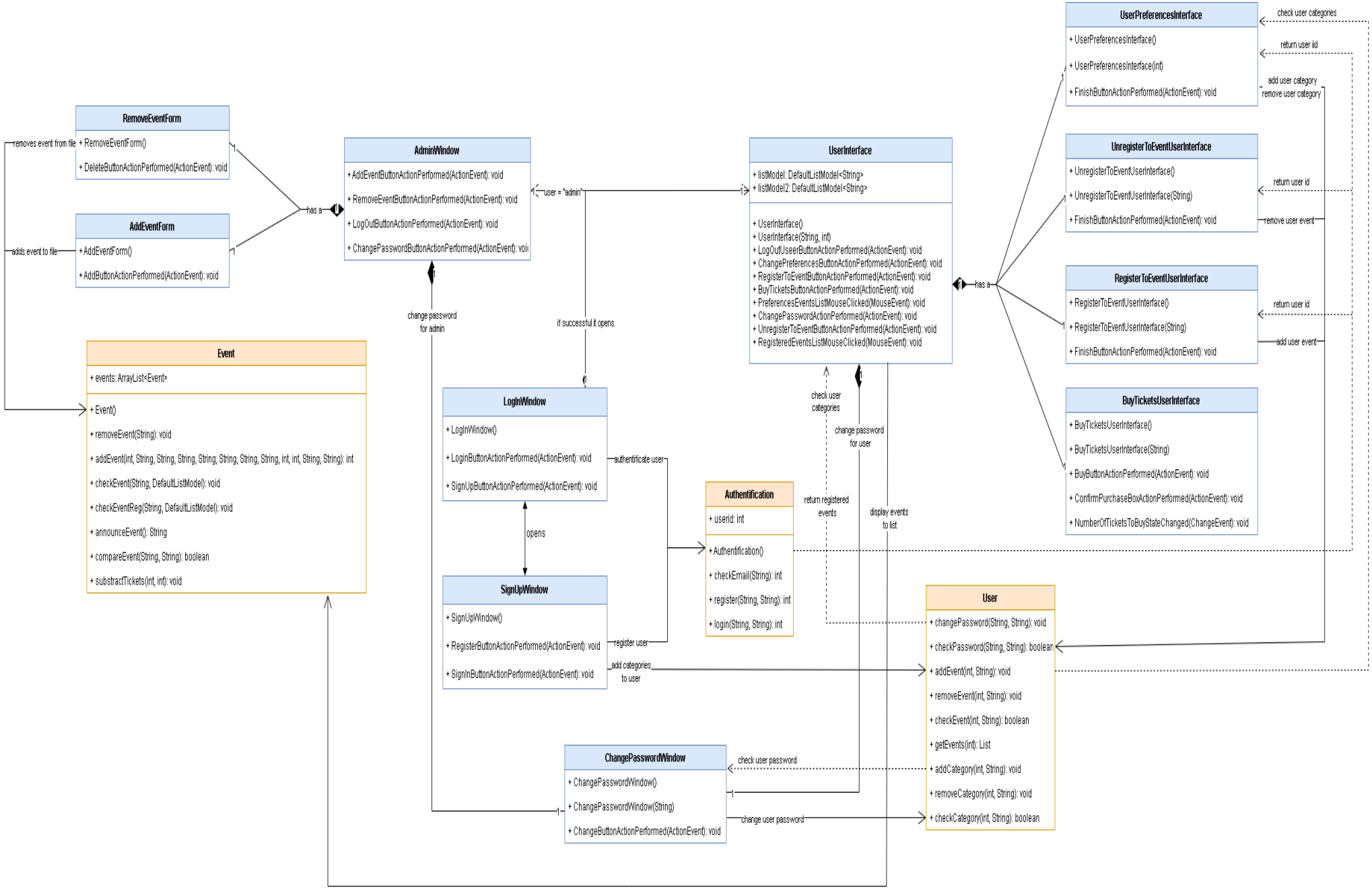
4.Aplicatia memoreaza pe suport extern (fisiere) toate datele necesare functionarii sale astfel incat existenta datelor nu depinde de oprirea temporara (accidentala sau nu) a calculatoarelor pe care ruleaza.

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.


RemoveEventForm
+ RemoveEventForm()
+ DeleteButtonActionPerformed(ActionEvent): void
removes event from file
AddEventForm
+ AddEventForm()
+ AddButtonActionPerformed(ActionEvent): void
adds event to file
has a
AdminWindow
+ AddEventButtonActionPerformed(ActionEvent): void
+ RemoveEventButtonActionPerformed(ActionEvent): void
+ LogOutButtonActionPerformed(ActionEvent): void
+ ChangePasswordButtonActionPerformed(ActionEvent): voi
change password for admin
user = "admin"
if successful it opens
UserInterface
+ listModel: DefaultListModel<String>
+ listModel2: DefaultListModel<String>
+ UserInterface()
+ UserInterface(String, int)
+ LogOutUseerButtonActionPerformed(ActionEvent): void
+ ChangePreferencesButtonActionPerformed(ActionEvent): void
+ RegisterToEventButtonActionPerformed(ActionEvent): void
+ BuyTicketsButtonActionPerformed(ActionEvent): void
+ PreferencesEventsListMouseClicked(MouseEvent): void
+ ChangePasswordActionPerformed(ActionEvent): void
+ UnregisterToEventButtonActionPerformed(ActionEvent): void
+ RegisteredEventsListMouseClicked(MouseEvent): void
has a
UserPreferencesInterface
+ UserPreferencesInterface()
+ UserPreferencesInterface(int)
+ FinishButtonActionPerformed(ActionEvent): void
check user categories
return user iid
add user category
remove user category
UnregisterToEventUserInterface
+ UnregisterToEventUserInterface()
+ UnregisterToEventUserInterface(String)
+ FinishButtonActionPerformed(ActionEvent): void
return user id
remove user event
RegisterToEventUserInterface
+ RegisterToEventUserInterface()
+ RegisterToEventUserInterface(String)
+ FinishButtonActionPerformed(ActionEvent): void
return user id
add user event
BuyTicketsUserInterface
+ BuyTicketsUserInterface()
+ BuyTicketsUserInterface(String)
+ BuyButtonActionPerformed(ActionEvent): void
+ ConfirmPurchaseBoxActionPerformed(ActionEvent): void
+ NumberOfTicketsToBuyStateChanged(ChangeEvent): void
Event
+ events: ArrayList<Event>
+ Event()
+ removeEvent(String): void
+ addEvent(int, String, String, String, String, String, String, String, int, int, String, String): int
+ checkEvent(String, DefaultListModel): void
+ checkEventReg(String, DefaultListModel): void
+ announceEvent(): String
+ compareEvent(String, String): boolean
+ substractTickets(int, int): void
LogInWindow
+ LogInWindow()
+ LoginButtonActionPerformed(ActionEvent): void
+ SignUpButtonActionPerformed(ActionEvent): void
authentificate user
opens
SignUpWindow
+ SignUpWindow()
+ RegisterButtonActionPerformed(ActionEvent): void
+ SignInButtonActionPerformed(ActionEvent): void
register user
add categories to user
Authentification
+ userid: int
+ Authentification()
+ checkEmail(String): int
+ register(String, String): int
+ login(String, String): int
return registered events
display events to list
check user categories
change password for user
User
+ changePassword(String, String): void
+ checkPassword(String, String): boolean
+ addEvent(int, String): void
+ removeEvent(int, String): void
+ checkEvent(int, String): boolean
+ getEvents(int): List
+ addCategory(int, String): void
+ removeCategory(int, String): void
+ checkCategory(int, String): boolean
ChangePasswordWindow
+ ChangePasswordWindow()
+ ChangePasswordWindow(String)
+ ChangeButtonActionPerformed(ActionEvent): void
check user password
change user password

Cod:

StartApp.java

```java
package event;

public class StartApp {

    public static void main(String[] args) {
        new LogInWindow().setVisible(true);

    }
}
```

Authentification.java

```java
package event;

import java.awt.Component;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import javax.swing.JOptionPane;

public class Authentification {
    static int userid;

    public Authentification() {
    }

    public static int register(String email, String password) throws IOException {
        boolean checkEmail = false;
        boolean checkPassword = false;
        boolean flag = false;

        try {
            File userfile = new File("users.txt");
            Scanner myReader = new Scanner(userfile);

            while(myReader.hasNextLine()) {
                String data = myReader.nextLine();
                String[] parts = data.split("/");
                if (parts[0].equals(email)) {
                    flag = true;
                    break;
                }
            }
        } catch (FileNotFoundException var13) {
            System.out.println("User file not found.");
            var13.printStackTrace();
        }

        if (flag) {
            JOptionPane.showMessageDialog((Component)null, "The entered e-mail already exists");
        } else if (email.contains("@") && email.contains(".com")) {
            if (email.contains("/")) {
                JOptionPane.showMessageDialog((Component)null, "E-mail must not contain /.");
            } else {
                checkEmail = true;
            }
        } else {
            JOptionPane.showMessageDialog((Component)null, "The entered e-mail is invalid");
```

Authentification.java

Source | History

```
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");
```

Authentification.java

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     }
```

User.java

Source | History

```java
package event;

import java.io.*;
import java.nio.file.*;
import java.util.*;

public class User {
    static ArrayList<Event> userevents = new ArrayList<>();

    public static void changePassword(String username, String newPassword) {
        try {
            // Read the file line by line
            BufferedReader reader = new BufferedReader(new FileReader("users.txt"));
            String line;
            StringBuilder sb = new StringBuilder();
            while ((line = reader.readLine()) != null) {
                // Split the line into parts
                String[] parts = line.split("/");
                if (parts[0].equals(username)) {
                    // Update the password
                    parts[1] = newPassword;
                    line = parts[0] + "/" + parts[1] + "/" + parts[2] + "/";
                }
                sb.append(line).append("\n");
            }
            reader.close();

            // Write the updated file
            BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
            writer.write(sb.toString());
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static boolean checkPassword(String username, String enteredPassword) {
        try {
            // Read the file line by line
            BufferedReader reader = new BufferedReader(new FileReader("users.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                // Split the line into parts
                String[] parts = line.split("/");
                if (parts[0].equals(username)) {
                    // Check if the entered password matches the stored password
                    if (parts[1].equals(enteredPassword)) {
                        reader.close();
                        return true;
```

User.java

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);
                        newline.append(event + "-");
77                  }
78                  System.out.println(newline);
79                  lines.set(userid, newline.toString());
80
                    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
```

User.java

Source History

```
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             BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
121             for (String l : lines) {
122                 writer.write(l);
123                 writer.newLine();
124             }
125             writer.close();
126         } catch (IOException e) {
127             e.printStackTrace();
128         }
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--)
```

User.java

Source History

```
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
```

User.java

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             // Seperate all lines
203             String[] parts = line.split("/");
                int categorynr = 0;
205
206             // Seperate 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 interfene with events
243             newparts.reverse();
244             newparts.replace(0, 1, "");
```

User.java

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             // Seperate all lines
271             String[] parts = line.split("/");
272             int categorynr = 0;
273
274             // Seperate 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];
```

User.java

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 interfene with events
311             newparts.reverse();
312             newparts.replace(0, 1, "");
313             newparts.reverse();
314
315             lines.set(userid, newparts.toString());
316
317             BufferedWriter writer = new BufferedWriter(new FileWriter("users.txt"));
318             for (String l : lines) {
319                 writer.write(l);
320                 writer.newLine();
321             }
322             writer.close();
323         } catch (IOException e) {
324             e.printStackTrace();
325         }
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("/");
```

User.java

Source History

```
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

```java
package event;

import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Scanner;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class Event {

    static ArrayList<Event> events = new ArrayList<>();

    public Event(){
    }

    public static void addEvent(int id, String name, String category, String startDate, String endDate,
            String country, String city, String location, int priceTicket, int nrTickets, String duration) throws FileNotFoundException, IOException{

        boolean flag = false;
        boolean checkEvent = false;

        int index = 1;
        try
        {
            File userfile = new File("events.txt");
            Scanner myReader = new Scanner(userfile);
            while (myReader.hasNextLine()) {

                String data = myReader.nextLine();
                index++;
                String[] parts = data.split(" / ");
                if (parts[1].equals(name)) {

                    flag = true;
                    break;
                }
            }
        } catch (FileNotFoundException e) {
            System.out.println("User file not found.");
            e.printStackTrace();
        }

        if(flag){
            JOptionPane.showMessageDialog(null, "The entered event already exists");
```

Event.java

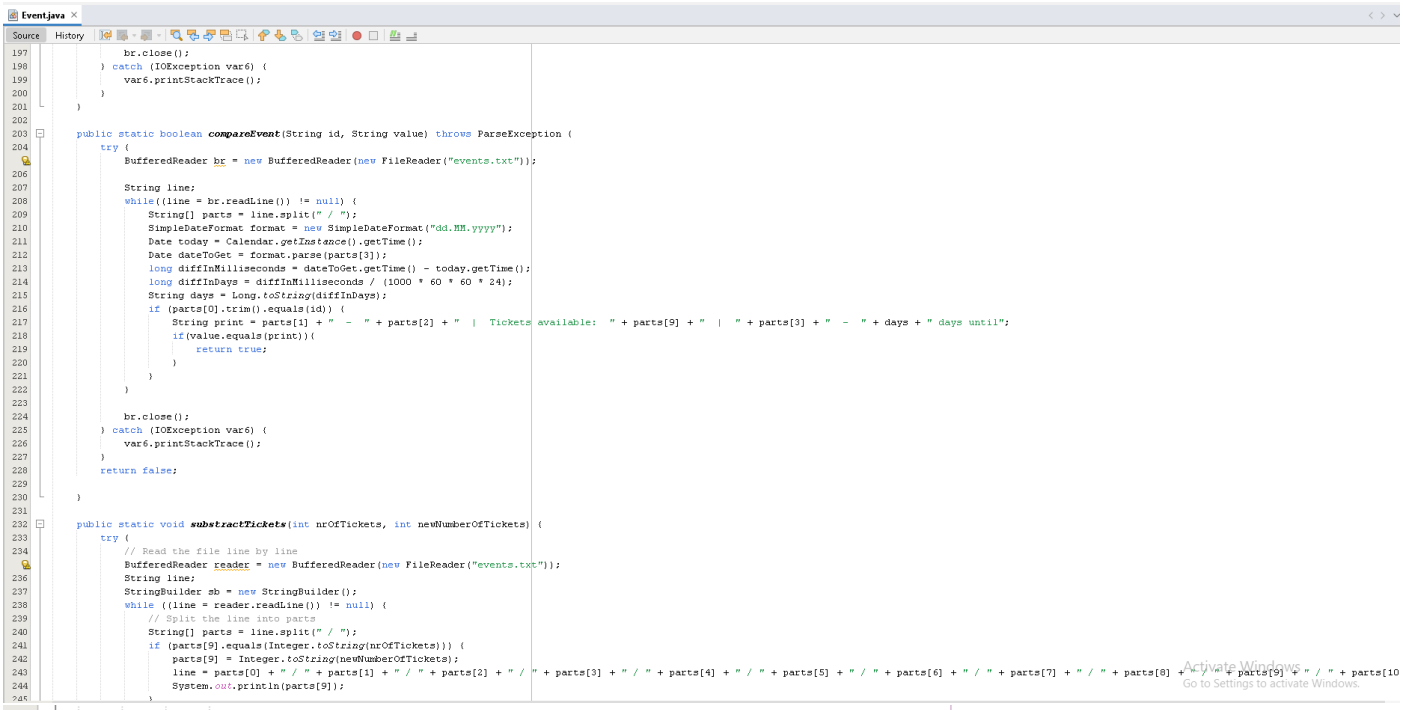
```java
        }
        else {
            checkEvent = true;
            JOptionPane.showMessageDialog(null, "Event added successfully");
        }

        String eventFormat = index + " / " + name + " / " + category + " / " + startDate + " / " + endDate + " / " + country + " / " + city + " / " + location + " / " + priceTicket + " / " + nrTickets + " / " + duration;

        if(checkEvent == true){
            FileWriter fw = new FileWriter("events.txt", true);
            try (BufferedWriter bw = new BufferedWriter(fw)) {
                bw.write(eventFormat);
                bw.newLine();
            }

            File userfile = new File("events.txt");
            Scanner myReader = new Scanner(userfile);


            while(myReader.hasNextLine()) {
                index++;
                myReader.nextLine();
            }
        }
    }

    public static void removeEvent(String value) throws FileNotFoundException, IOException, ParseException{
        File file = new File("events.txt");
        List<String> lines = new ArrayList<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
            String line;

            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(" / ");
                SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
                Date today = Calendar.getInstance().getTime();
                Date dateToGet = format.parse(parts[3]);
                long diffInMilliseconds = dateToGet.getTime() - today.getTime();
                long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
                String days = Long.toString(diffInDays);
                String print = parts[1] + "  -  " + parts[2] + "  |  Tickets available:  " + parts[9] + "  |  " + parts[3] + "  -  " + days + " days until";
                if (!print.equals(value)) {
                    lines.add(line);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Event.java

```
99
100         try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
101             for (String line : lines) {
102                 writer.write(line);
103                 writer.newLine();
104             }
105         } catch (IOException e) {
106             e.printStackTrace();
107         }
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();
```

Event.java

```
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
```



```
            br.close();
        } catch (IOException var6) {
            var6.printStackTrace();
        }
    }

    public static boolean compareEvent(String id, String value) throws ParseException {
        try {
            BufferedReader br = new BufferedReader(new FileReader("events.txt"));

            String line;
            while((line = br.readLine()) != null) {
                String[] parts = line.split(" / ");
                SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
                Date today = Calendar.getInstance().getTime();
                Date dateToGet = format.parse(parts[3]);
                long diffInMilliseconds = dateToGet.getTime() - today.getTime();
                long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
                String days = Long.toString(diffInDays);
                if (parts[0].trim().equals(id)) {
                    String print = parts[1] + "  -  " + parts[2] + "  |  Tickets available:  " + parts[9] + "  |  " + parts[3] + "  -  " + days + " days until";
                    if(value.equals(print)){
                        return true;
                    }
                }
            }

            br.close();
        } catch (IOException var6) {
            var6.printStackTrace();
        }
        return false;

    }

    public static void substractTickets(int nrOfTickets, int newNumberOfTickets) {
        try {
            // Read the file line by line
            BufferedReader reader = new BufferedReader(new FileReader("events.txt"));
            String line;
            StringBuilder sb = new StringBuilder();
            while ((line = reader.readLine()) != null) {
                // Split the line into parts
                String[] parts = line.split(" / ");
                if (parts[9].equals(Integer.toString(nrOfTickets))) {
                    parts[9] = Integer.toString(newNumberOfTickets);
                    line = parts[0] + " / " + parts[1] + " / " + parts[2] + " / " + parts[3] + " / " + parts[4] + " / " + parts[5] + " / " + parts[6] + " / " + parts[7] + " / " + parts[8] + " / " + parts[9] + " / " + parts[10
                    System.out.println(parts[9]);
                }
```



```
                    }
                    sb.append(line).append("\n");
                }
                reader.close();

                // Write the updated file
                BufferedWriter writer = new BufferedWriter(new FileWriter("events.txt"));
                writer.write(sb.toString());
                writer.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
```

LogInWindow.java

Source | Design | History

```java
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 AuthentificationWindow
     */
    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 = Authentification.login(email, password);
            if(email.equals("admin") && password.equals("admin")){
                dispose();
                new AdminWindow().setVisible(true);
```

LogInWindow.java

Source | Design | History

```
195                 }
196                 else if (user>=0) {
197                     Authentification.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 
209     private void SignUpButtonActionPerformed(java.awt.event.ActionEvent evt) {
210         dispose();
211         new SignUpWindow().setVisible(true);
212     }
213 
214     /**
215      * @param args the command line arguments
216      */
217     public static void main(String args[]) {
218         /* Set the Nimbus look and feel */
219         Look and feel setting code (optional)
240         //</editor-fold>
241 
242         /* Create and display the form */
243         java.awt.EventQueue.invokeLater(new Runnable() {
244             public void run() {
245                 new LogInWindow().setVisible(true);
246             }
247         });
248     }
249 
250     // Variables declaration - do not modify
251     private javax.swing.JTextField EmailField;
252     private javax.swing.JLabel LogInLabel;
253     private javax.swing.JButton LoginButton;
254     private javax.swing.JPasswordField PasswordField;
255     private javax.swing.JButton SignUpButton;
256     private javax.swing.Box.Filler filler1;
257     private javax.swing.JLabel jLabel1;
258     private javax.swing.JSeparator jSeparator2;
259     private javax.swing.JSeparator jSeparator3;
260     // End of variables declaration
261 }
262 
```

SignUpWindow.java

Source | Design | History

```
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
239
241     private void PasswordFieldActionPerformed(java.awt.event.ActionEvent evt) {
242         // TODO add your handling code here:
243     }
245     private void EmailFieldActionPerformed(java.awt.event.ActionEvent evt) {
246         // TODO add your handling code here:
247     }
249     private void RegisterButtonActionPerformed(java.awt.event.ActionEvent evt) {
250
251         if(EmailField.getText().isEmpty() || PasswordField.getText().isEmpty()){
252             JOptionPane.showMessageDialog(null, "Please fill all the forms!");
253         }else{
254             String email = EmailField.getText();
255             String password = PasswordField.getText();
256             int user = -1;
257
258             if(MovieCheckBox.isSelected() || ArtCheckBox.isSelected() || ConcertCheckBox.isSelected() || FestivalCheckBox.isSelected() || OtherCheckBox.isSelected()){
259                 try {
260                     user = Authentification.register(email, password);
                    } catch (IOException ex) {
```

SignUpWindow.java

Source | Design | History

```
261                     Logger.getLogger(SignUpWindow.class.getName()).log(Level.SEVERE, null, ex);
262                 }
263                 if (user>=0) {
264 
265                     Authentification.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

Source | Design | History

```
310                 JOptionPane.showMessageDialog(null, "Check at least one box");
311             }
312         }
313     }
314
315     private void SignInButtonActionPerformed(java.awt.event.ActionEvent evt) {
316         dispose();
317         new LogInWindow().setVisible(true);
318     }
319
320     private void ArtCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {
321         // TODO add your handling code here:
322     }
323
324     private 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      */
331     public static void main(String args[]) {
332         /* Set the Nimbus look and feel */
333         Look and feel setting code (optional)
354
355         /* Create and display the form */
356         java.awt.EventQueue.invokeLater(new Runnable() {
357             public void run() {
358                 new SignUpWindow().setVisible(true);
359             }
360         });
361     }
362
363     // Variables declaration - do not modify
364     private javax.swing.JCheckBox ArtCheckBox;
365     private javax.swing.JLabel CategoriesLabel;
366     private javax.swing.JCheckBox ConcertCheckBox;
367     private javax.swing.JTextField EmailField;
368     private javax.swing.JCheckBox FestivalCheckBox;
369     private javax.swing.JCheckBox MovieCheckBox;
370     private javax.swing.JCheckBox OtherCheckBox;
371     private javax.swing.JPasswordField PasswordField;
372     private javax.swing.JButton RegisterButton;
373     private javax.swing.JLabel SelectEventLabel;
374     private javax.swing.JButton SignInButton;
375     private javax.swing.JLabel SignUpLabel;
376     private javax.swing.Box.Filler filler1;
377     private javax.swing.JLabel jLabel1;
378     private javax.swing.JSeparator jSeparator2;
```

ChangePasswordWindow.java

Source | Design | History

```
1   package event;
2
3   import javax.swing.JOptionPane;
4
5   /**
6    *
7    * @author dobir
8    */
9   public class ChangePasswordWindow extends javax.swing.JFrame {
10      String user;
11      /**
12       * Creates new form ChangePasswordWindow
13       */
14      public ChangePasswordWindow() {
15          initComponents();
16      }
17      public ChangePasswordWindow(String username) {
18          initComponents();
19          user = username;
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
147
148     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
168     private void OldPasswordFieldActionPerformed(java.awt.event.ActionEvent evt) {
```

ChangePasswordWindow.java

Source | Design | History

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

169
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)
198
199         /* Create and display the form */
            java.awt.EventQueue.invokeLater(new Runnable() {
                public void run() {
202                 new ChangePasswordWindow().setVisible(true);
203             }
204         });
205     }
206
207     // Variables declaration - do not modify
208     private javax.swing.JButton ChangeButton;
209     private javax.swing.JLabel ConfirmNewPasswordLabel;
210     private javax.swing.JPasswordField ConfirmPasswordField;
211     private javax.swing.JPasswordField NewPasswordField;
212     private javax.swing.JLabel NewPasswordLabel;
213     private javax.swing.JPasswordField OldPasswordField;
214     private javax.swing.JLabel jLabel1;
215     private javax.swing.JLabel jLabel2;
216     private javax.swing.JSeparator jSeparator1;
217     private javax.swing.JSeparator jSeparator2;
218     // End of variables declaration
219 }
220
```

UserInterface.java

Source | Design | History

```java
package event;

import java.io.File;
import java.io.IOException;
import java.text.ParseException;
import java.util.List;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.DefaultListModel;
import javax.swing.JOptionPane;

public class UserInterface extends javax.swing.JFrame {
    DefaultListModel<String> listModel = new DefaultListModel();
    DefaultListModel<String> listModel2 = new DefaultListModel();

    public UserInterface() {
        initComponents();
    }

    public UserInterface(String username, int userid) throws IOException, ParseException {
        initComponents();

        RegisterToEventButton.setEnabled(false);
        UnregisterToEventButton.setEnabled(false);
        BuyTicketsButton.setEnabled(false);

        UsernameLabel.setText(username);
        PreferencesEventsList.setModel(listModel);
        RegisteredEventsList.setModel(listModel2);

        if(User.checkCategory(userid, "Concert")){
            Event.checkEvent("Concert", listModel);
        }
        if(User.checkCategory(userid, "Art")){
            Event.checkEvent("Art", listModel);
        }
        if(User.checkCategory(userid, "Movie")){
            Event.checkEvent("Movie", listModel);
        }
        if(User.checkCategory(userid, "Festival")){
            Event.checkEvent("Festival", listModel);
        }
        if(User.checkCategory(userid, "Other")){
            Event.checkEvent("Other", listModel);
        }

        File eventfile = new File("events.txt");
        Scanner myReader = new Scanner(eventfile);
```

UserInterface.java

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 LogOutUseerButtonActionPerformed(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(Authentification.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) {
```

UserInterface.java

Source | Design | History

```
341             Logger.getLogger(UserInterface.class.getName()).log(Level.SEVERE, null, ex);
342         }
343     }
344
345     private void BuyTicketsButtonActionPerformed(java.awt.event.ActionEvent evt) {
346         if(!PreferencesEventsList.isSelectionEmpty())
347             new BuyTicketsUserInterface(PreferencesEventsList.getSelectedValue()).setVisible(true);
348         if(!RegisteredEventsList.isSelectionEmpty())
349             new BuyTicketsUserInterface(RegisteredEventsList.getSelectedValue()).setVisible(true);
350     }
351
352     private void PreferencesEventsListMouseClicked(java.awt.event.MouseEvent evt) {
353        if(PreferencesEventsList.getSelectedIndex() > -1){
354             RegisteredEventsList.clearSelection();
355             RegisterToEventButton.setEnabled(true);
356             BuyTicketsButton.setEnabled(true);
357             UnregisterToEventButton.setEnabled(false);
358         }
359     }
360
361     private void ChangePasswordActionPerformed(java.awt.event.ActionEvent evt) {
362         dispose();
363         new ChangePasswordWindow(UsernameLabel.getText()).setVisible(true);
364     }
365
366     private void UnregisterToEventButtonActionPerformed(java.awt.event.ActionEvent evt) {
367         dispose();
368         new UnregisterToEventUserInterface(RegisteredEventsList.getSelectedValue()).setVisible(true);
369     }
370
371     private void RegisteredEventsListMouseClicked(java.awt.event.MouseEvent evt) {
372         if(RegisteredEventsList.getSelectedIndex() > -1){
373             PreferencesEventsList.clearSelection();
374             UnregisterToEventButton.setEnabled(true);
375             BuyTicketsButton.setEnabled(true);
376             RegisterToEventButton.setEnabled(false);
377         }
378     }
379
380     private void AnnounceActionPerformed(java.awt.event.ActionEvent evt) {
381         // TODO add your handling code here:
382     }
383
384     /**
385      * @param args the command line arguments
386      */
387     public static void main(String args[]) {
388         /* Set the Nimbus look and feel */
389         Look and feel setting code (optional)
```

UserInterface.java

Source Design History

```
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 */
            java.awt.EventQueue.invokeLater(new Runnable() {
                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 LogOutUseerButton;
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
```

UserPreferencesInterface.java

Source | Design | History

```java
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 UserPreferencesInterface extends javax.swing.JFrame {

    /**
     * Creates new form UserPreferencesInterface
     */
    public UserPreferencesInterface() {
        initComponents();

        try {
            if(User.checkCategory(Authentification.userid, "Art")){
                ArtCheckBox.setSelected(true);
            }
            if(User.checkCategory(Authentification.userid, "Concert")){
                ConcertCheckBox.setSelected(true);
            }
            if(User.checkCategory(Authentification.userid, "Movie")){
                MovieCheckBox.setSelected(true);
            }
            if(User.checkCategory(Authentification.userid, "Festival")){
                FestivalCheckBox.setSelected(true);
            }
            if(User.checkCategory(Authentification.userid, "Other")){
                OtherCheckBox.setSelected(true);
            }
        } catch (IOException ex) {
            Logger.getLogger(UserPreferencesInterface.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public UserPreferencesInterface(int userid) {
        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.
```

UserPreferencesInterface.java

Source | Design | History

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

UserPreferencesInterface.java

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)
293
294         /* Create and display the form */
295         java.awt.EventQueue.invokeLater(new Runnable() {
296             public void run() {
297                 new UserPreferencesInterface().setVisible(true);
298             }
299         });
300     }
301
302     // Variables declaration - do not modify
303     public static javax.swing.JCheckBox ArtCheckBox;
304     private javax.swing.JLabel CategoriesLabel;
305     public static javax.swing.JCheckBox ConcertCheckBox;
306     public static javax.swing.JCheckBox FestivalCheckBox;
307     private javax.swing.JButton FinishButton;
```

UserPreferencesInterface.java

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)
293
294             /* Create and display the form */
                java.awt.EventQueue.invokeLater(new Runnable() {
                    public void run() {
297                     new UserPreferencesInterface().setVisible(true);
298                 }
299             });
300         }
301
302         // Variables declaration - do not modify
303         public static javax.swing.JCheckBox ArtCheckBox;
304         private javax.swing.JLabel CategoriesLabel;
305         public static javax.swing.JCheckBox ConcertCheckBox;
306         public static javax.swing.JCheckBox FestivalCheckBox;
307         private javax.swing.JButton FinishButton;
308         public static javax.swing.JCheckBox MovieCheckBox;
309         public static javax.swing.JCheckBox OtherCheckBox;
310         private javax.swing.JLabel PreferencesLabel;
311         private javax.swing.JLabel YourPreferencesLabel;
312         private javax.swing.JSeparator jSeparator1;
313         private javax.swing.JSeparator jSeparator2;
314         // End of variables declaration
315     }
316
```

RegisterToEventUserInterface.java

Source | Design | History

```java
package event;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;

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

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

    public RegisterToEventUserInterface(String value) throws IOException {
        initComponents();



        try (BufferedReader br = new BufferedReader(new FileReader("events.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                String eventName = value.substring(0, value.indexOf(" - "));
                String[] parts = line.split(" / ");
                if (parts[1].trim().equals(eventName.trim())) {
                    SelectedEventForm.setText(parts[1]);
                    CountryField.setText(parts[5]);
                    CityField.setText(parts[6]);
                    AddressField.setText(parts[7]);
                    DateOfEventField.setText(parts[3]);
                    TicketsAvailableField.setText(parts[8]);
                    PriceOfTicketsField.setText(parts[9]);
                }
            }
        }

    }

    /**
     * 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
```

RegisterToEventUserInterface.java

Source | Design | History

```
50          * regenerated by the Form Editor.
51          */
52         @SuppressWarnings("unchecked")
53         Generated Code
260
262
263         }
264
266             // TODO add your handling code here:
267         }
268
270             // TODO add your handling code here:
271         }
272
274             // TODO add your handling code here:
275         }
276
278             // TODO add your handling code here:
279             try
280             {
281                 String eventid = "0";
282
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(Authentification.userid, eventid);
293                 dispose();
294                 new UserInterface(UserInterface.UsernameLabel.getText(), Authentification.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
```

RegisterToEventUserInterface.java

Source | Design | History

```
50          * regenerated by the Form Editor.
51          */
52         @SuppressWarnings("unchecked")
53         Generated Code
260
262
263         }
264
266             // TODO add your handling code here:
267         }
268
270             // TODO add your handling code here:
271         }
272
274             // TODO add your handling code here:
275         }
276
278             // TODO add your handling code here:
279             try
280             {
281                 String eventid = "0";
282
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(Authentification.userid, eventid);
293                 dispose();
294                 new UserInterface(UserInterface.UsernameLabel.getText(), Authentification.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
```

RegisterToEventUserInterface.java

Source | Design | History

```
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)
334
335             /* Create and display the form */
                java.awt.EventQueue.invokeLater(new Runnable() {
                    public void run() {
338                     new RegisterToEventUserInterface().setVisible(true);
339                 }
340             });
341         }
342
343         // Variables declaration - do not modify
344         private javax.swing.JTextField AddressField;
345         private javax.swing.JLabel AddressLabel;
346         private javax.swing.JTextField CityField;
347         private javax.swing.JLabel CityLabel;
348         private javax.swing.JTextField CountryField;
349         private javax.swing.JLabel CountryLabel;
350         private javax.swing.JTextField DateOfEventField;
351         private javax.swing.JLabel DateOfTheEventLabel;
352         private javax.swing.JLabel DateOfTheEventLabel1;
353         private javax.swing.JButton FinishButton;
354         private javax.swing.JTextField PriceOfTicketsField;
355         private javax.swing.JLabel PriceOfTicketsLabel;
356         private javax.swing.JLabel RegisterToAnEventLabel;
357         private javax.swing.JTextField SelectedEventForm;
358         private javax.swing.JLabel SelectedEventLabel;
359         private javax.swing.JTextField TicketsAvailableField;
360         private javax.swing.JSeparator jSeparator1;
361         private javax.swing.JSeparator jSeparator2;
362         // End of variables declaration
363     }
364
```

UnregisterToEventUserInterface.java

Source | Design | History

```java
package event;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.text.ParseException;
import java.util.logging.Level;
import java.util.logging.Logger;

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

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

    public UnregisterToEventUserInterface(String value) {
        initComponents();

        String eventName = value.substring(0, value.indexOf(" - "));

        try {
            BufferedReader br = new BufferedReader(new FileReader("events.txt"));
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(" / ");
                if (parts[1].trim().equals(eventName.trim())) {
                    SelectedEventForm.setText(parts[1]);
                    CountryField.setText(parts[5]);
                    CityField.setText(parts[6]);
                    AddressField.setText(parts[7]);
                    DateOfEventField.setText(parts[3]);
                    TicketsAvailableField.setText(parts[8]);
                    PriceOfTicketsField.setText(parts[9]);
                }
            }
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

UnregisterToEventUserInterface.java

Source | Design | History

```
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(Authentification.userid, eventid);
296             dispose();
297             new UserInterface(UserInterface.UsernameLabel.getText(), Authentification.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     }
```

UnregisterToEventUserInterface.java

Source | Design | History

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

BuyTicketsUserInterface.java

Source | Design | History

```java
package event;

import java.awt.print.PrinterException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

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

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

    public BuyTicketsUserInterface(String value) {
        initComponents();

        BuyButton.setEnabled(false);

        String eventName = value.substring(0, value.indexOf(" | "));

        try {
            BufferedReader br = new BufferedReader(new FileReader("events.txt"));
            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(" / ");
                if (parts[1].trim().equals(eventName.trim())) {
                    SelectedEventBuyField.setText(parts[1]);
                    CountryBuyField.setText(parts[5]);
                    CityBuyField.setText(parts[6]);
                    AddressBuyField.setText(parts[7]);
                    DateOfEventBuyField.setText(parts[3]);
                    TicketsAvailableBuyField.setText(parts[9]);
                    PriceOfTicketsBuyField.setText(parts[8]);
                }
            }
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
```

BuyTicketsUserInterface.java

Source | Design | History

```
        }

        NumberOfTicketsToBuy.setModel(new javax.swing.SpinnerNumberModel(1, 1, Integer.parseInt(TicketsAvailableBuyField.getText()), 1));
        TotalBuyField.setText((Integer) NumberOfTicketsToBuy.getValue() * Integer.parseInt(PriceOfTicketsBuyField.getText()) + " Lei");

        area.setText("                    ----------------------------------------------------------------\n\n");

        area.setText(area.getText() + "                                                    Receipt\n");
        area.setText(area.getText() + "\n                    ----------------------------------------------------------------\n\n\n");

        Date obj = new Date();
        String date = obj.toString();

        area.setText(area.getText() + "                    Event: " + SelectedEventBuyField.getText() + "\n\n");
        area.setText(area.getText() + "                    Country and city: " + CountryBuyField.getText() + " " + CityBuyField.getText() + "\n\n");
        area.setText(area.getText() + "                    Location: " + AddressBuyField.getText() + "\n\n");
        area.setText(area.getText() + "                    Tickets:                                                " + PriceOfTicketsBuyField.getText() + "  x" + NumberOfTicketsToBuy.getValue() + "");

        area.setText(area.getText() + "\n\n\n                    ----------------------------------------------------------------\n\n");
        area.setText(area.getText() + "                    Total:                                                  " + TotalBuyField.getText());
        area.setText(area.getText() + "\n\n                    ----------------------------------------------------------------");
        area.setText(area.getText() + "\n\n\n                " + date + "            Signature: " + UserInterface.UsernameLabel.getText());
    }

    /**
     * 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 BuyButtonActionPerformed(java.awt.event.ActionEvent evt) {
        BuyButton.setEnabled(false);
        int value = Integer.parseInt(TicketsAvailableBuyField.getText()) - (Integer) NumberOfTicketsToBuy.getValue();
        Event.substractTickets(Integer.parseInt(TicketsAvailableBuyField.getText()), value);
        try {
            area.print();
        } catch (PrinterException ex) {
            Logger.getLogger(BuyTicketsUserInterface.class.getName()).log(Level.SEVERE, null, ex);
        }
        dispose();
        JOptionPane.showMessageDialog(null, "Purchase complete!");

    }

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


BuyTicketsUserInterface.java

Source | Design | History

```
    }

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

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

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

    private void ConfirmPurchaseBoxActionPerformed(java.awt.event.ActionEvent evt) {
        if(ConfirmPurchaseBox.isSelected()){
            BuyButton.setEnabled(true);
        }else{
            BuyButton.setEnabled(false);
        }
    }

    private void TotalBuyFieldActionPerformed(java.awt.event.ActionEvent evt) {

    }

    private void NumberOfTicketsToBuyStateChanged(javax.swing.event.ChangeEvent evt) {

        Integer myInt = (Integer) NumberOfTicketsToBuy.getValue() * Integer.parseInt(PriceOfTicketsBuyField.getText());
        String spinner = myInt.toString();
        TotalBuyField.setText(spinner + " Lei");
        area.setText(null);
        area.setText("                    ----------------------------------------------------------------\n\n");

        area.setText(area.getText() + "                                                    Receipt\n");
        area.setText(area.getText() + "\n                    ----------------------------------------------------------------\n\n\n");

        Date obj = new Date();
        String date = obj.toString();

        area.setText(area.getText() + "                    Event: " + SelectedEventBuyField.getText() + "\n\n");
        area.setText(area.getText() + "                    Country and city: " + CountryBuyField.getText() + " " + CityBuyField.getText() + "\n\n");
        area.setText(area.getText() + "                    Location: " + AddressBuyField.getText() + "\n\n");
        area.setText(area.getText() + "                    Tickets:                                                " + PriceOfTicketsBuyField.getText() + "  x" + NumberOfTicketsToBuy.getValue() + "");

        area.setText(area.getText() + "\n\n\n                    ----------------------------------------------------------------\n\n");
        area.setText(area.getText() + "                    Total:                                                  " + TotalBuyField.getText());
        area.setText(area.getText() + "\n\n                    ----------------------------------------------------------------");
        area.setText(area.getText() + "\n\n\n                " + date + "            Signature: " + UserInterface.UsernameLabel.getText());
```

BuyTicketsUserInterface.java

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)
466
467         /* Create and display the form */
468         java.awt.EventQueue.invokeLater(new Runnable() {
469             public void run() {
470                 new BuyTicketsUserInterface().setVisible(true);
471             }
472         });
473     }
474
475     // Variables declaration - do not modify
476     private javax.swing.JTextField AddressBuyField;
477     private javax.swing.JLabel AddressBuyLabel;
478     private javax.swing.JButton BuyButton;
479     private javax.swing.JLabel BuyTicketsLabel;
480     private javax.swing.JTextField CityBuyField;
481     private javax.swing.JLabel CityBuyLabel;
482     private javax.swing.JCheckBox ConfirmPurchaseBox;
483     private javax.swing.JTextField CountryBuyField;
484     private javax.swing.JLabel CountryBuyLabel;
485     private javax.swing.JTextField DateOfEventBuyField;
486     private javax.swing.JLabel DateOfTheEventBuyLabel;
487     private javax.swing.JLabel NoOfTicketsBuyLabel;
488     private javax.swing.JSpinner NumberOfTicketsToBuy;
489     private javax.swing.JTextField PriceOfTicketsBuyField;
490     private javax.swing.JLabel PriceOfTicketsBuyLabel;
491     private javax.swing.JTextField SelectedEventBuyField;
492     private javax.swing.JLabel SelectedEventBuyLabel;
493     private javax.swing.JTextField TicketsAvailableBuyField;
494     private javax.swing.JLabel TicketsAvailableBuyLabel;
495     private javax.swing.JTextField TotalBuyField;
496     private javax.swing.JLabel TotalBuyLabel;
497     private javax.swing.JTextArea area;
498     private javax.swing.JScrollPane jScrollPane1;
499     private javax.swing.JSeparator jSeparator1;
500     private javax.swing.JSeparator jSeparator2;
501     private javax.swing.JSeparator jSeparator3;
502     // End of variables declaration
```

AdminWindow.java

Source | Design | History

```
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
142     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

```
package event;

import java.io.IOException;
import java.text.*;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class AddEventForm extends javax.swing.JFrame {
    SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
    /**
     * Creates new form AddEventForm
     */
    public AddEventForm() {
        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 NameOfTheNewEventActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

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

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

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

    private void AddButtonActionPerformed(java.awt.event.ActionEvent evt) {
        if(NameOfTheNewEvent.getText().isEmpty() || CountryNewEvent.getText().isEmpty() || CityNewEvent.getText().isEmpty() || AddressField.getText().isEmpty() ||
                StartEventCalendar.getDate() == null || EndEventDate.getDate() == null || PriceTicketField.getText().isEmpty() || NrTicketsField.getText().isEmpty()){
            JOptionPane.showMessageDialog(null, "Please fill all the forms!");
        }else{
```

AddEventForm.java

```
            String category = null;
            int id = 0;
            if(MovieCheckBox.isSelected() || ArtCheckBox.isSelected() || ConcertCheckBox.isSelected() || FestivalCheckBox.isSelected() || OtherCheckBox.isSelected()){
                if(MovieCheckBox.isSelected())
                    category = "Movie";
                if(ArtCheckBox.isSelected())
                    category = "Art";
                if(ConcertCheckBox.isSelected())
                    category = "Concert";
                if(FestivalCheckBox.isSelected())
                    category = "Festival";
                if(OtherCheckBox.isSelected())
                    category = "Other";

                try {
                    Date startDate = format.parse(((JTextField)StartEventCalendar.getDateEditor().getUiComponent()).getText());
                    Date endDate = format.parse(((JTextField)EndEventDate.getDateEditor().getUiComponent()).getText());

                    long diffInMilliseconds = endDate.getTime() - startDate.getTime();
                    long duration = diffInMilliseconds / (1000 * 60 * 60 * 24);

                    Event.addEvent(id, NameOfTheNewEvent.getText(), category, ((JTextField)StartEventCalendar.getDateEditor().getUiComponent()).getText(), ((JTextField)EndEventDate.getDateEditor().getUiComponent()).getText(),
                        CountryNewEvent.getText(), CityNewEvent.getText(), AddressField.getText(), Integer.parseInt(PriceTicketField.getText()), Integer.parseInt(NrTicketsField.getText()), Long.toString(duration));

                } catch (ParseException | IOException ex) {
                    Logger.getLogger(AddEventForm.class.getName()).log(Level.SEVERE, null, ex);
                }

            }else{
                JOptionPane.showMessageDialog(null, "Check one box");
            }
        }
    }

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

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

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

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

AddEventForm.java

Source | Design | History

```
369     }
370
372     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)
401
402         /* Create and display the form */
405         java.awt.EventQueue.invokeLater(new Runnable() {
405             public void run() {
406                 new AddEventForm().setVisible(true);
407             }
408         });
409     }
410
410     // Variables declaration - do not modify
411     private javax.swing.JButton AddButton;
412     private javax.swing.JLabel AddEventLabel;
413     private javax.swing.JTextField AddressField;
414     private javax.swing.JCheckBox ArtCheckBox;
415     private javax.swing.JLabel CategoriesLabel;
416     private javax.swing.JTextField CityNewEvent;
417     private javax.swing.JCheckBox ConcertCheckBox;
418     private javax.swing.JTextField CountryNewEvent;
419     private javax.swing.JLabel DateEventLabel;
420     private com.toedter.calendar.JDateChooser EndEventDate;
421     private javax.swing.JLabel EndEventDateLabel;
422     private javax.swing.JCheckBox FestivalCheckBox;
423     private javax.swing.JCheckBox MovieCheckBox;
424     private javax.swing.JTextField NameOfTheNewEvent;
425     private javax.swing.JTextField NrTicketsField;
426     private javax.swing.JCheckBox OtherCheckBox;
427     private javax.swing.JLabel PriceTicketEventLabel;
428     private javax.swing.JLabel PriceTicketEventLabel2;
429     private javax.swing.JTextField PriceTicketField;
430     private com.toedter.calendar.JDateChooser StartEventCalendar;
431     private javax.swing.ButtonGroup buttonGroup1;
432     private javax.swing.JSeparator jSeparator1;
433     private javax.swing.JSeparator jSeparator2;
434     // End of variables declaration
435 }
436
```

RemoveEventForm.java

Source | Design | History

```
1   package event;
2
3   import java.io.BufferedReader;
4   import java.io.FileNotFoundException;
5   import java.io.FileReader;
6   import java.io.IOException;
7   import java.text.ParseException;
8   import java.text.SimpleDateFormat;
9   import java.util.Calendar;
10  import java.util.Date;
11  import java.util.logging.Level;
12  import java.util.logging.Logger;
13  import javax.swing.DefaultListModel;
14  import javax.swing.JOptionPane;
15
16  /**
17   *
18   * @author dobir
19   */
20  public 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
152
          private void DeleteButtonActionPerformed(java.awt.event.ActionEvent evt) {
```

RemoveEventForm.java

Source | Design | History

```
154
155         try
156         {
157             if(!EventsList.isSelectionEmpty())
158             {
159                 Event.removeEvent(EventsList.getSelectedValue());
160
161                 listModel.clear();
162
163                 // Read the contents of the text file and add them to the list
164                 BufferedReader br = new BufferedReader(new FileReader("events.txt"));
165                   String line;
166
167                 while ((line = br.readLine()) != null) {
168                     String[] parts = line.split(" / ");
169                     SimpleDateFormat format = new SimpleDateFormat("dd.MM.yyyy");
170                     Date today = Calendar.getInstance().getTime();
171                     Date dateToGet = format.parse(parts[3]);
172                     long diffInMilliseconds = dateToGet.getTime() - today.getTime();
173                     long diffInDays = diffInMilliseconds / (1000 * 60 * 60 * 24);
174                     String days = Long.toString(diffInDays);
175                     String print = parts[1] + "  -  " + parts[2] + "  |  Tickets available:  " + parts[9] + "  |  " + parts[3] + "  -  " + days + " days until";
176                     listModel.addElement(print);
177                 }
178             }
179         } catch (IOException ioe) {
180             System.out.println(ioe);
181         } catch (ParseException ex) {
182             Logger.getLogger(RemoveEventForm.class.getName()).log(Level.SEVERE, null, ex);
183         }
184         JOptionPane.showMessageDialog(null, "Event deleted successfully!");
185
186     }
187
188     private void BackButtonActionPerformed(java.awt.event.ActionEvent evt) {
189         dispose();
190     }
191
192     /**
193      * @param args the command line arguments
194      */
195     public static void main(String args[]) {
196         /* Set the Nimbus look and feel */
197         Look and feel setting code (optional)
218
219         /* Create and display the form */
220         java.awt.EventQueue.invokeLater(new Runnable() {
221             public void run() {
222                 new RemoveEventForm().setVisible(true);
223             }
224         });
225     }
226
227     // Variables declaration - do not modify
228     private javax.swing.JButton BackButton;
229     private javax.swing.JButton DeleteButton;
230     private javax.swing.JList<String> EventsList;
231     private javax.swing.JLabel RemoveEventLabel;
232     private javax.swing.JLabel SelectEventLabel;
233     private javax.swing.JScrollPane jScrollPane1;
234     private javax.swing.JSeparator jSeparator1;
235     // End of variables declaration
236 }
237
```