```java
/**
 * stub class to run the application.
 *
 * @author dfof
 *
 */
public class Main
{

  /**
   * @param args
   */
  public static void main(String[] args)
  {
    Application app = new Application();
    app.run();
  }

}

```

```
1  import java.util.Date;
2
3  /**
4   * Class representing an Intervention
5   */
6  class Intervention
7  {
8    public Date date;
9    public int duration;
10   public String description;
11   public Person technician;
12   public boolean resolved;
13
14   /**
15   * Contructor
16   */
17   Intervention()
18   {
19     /**
20      * Intervention date
21      */
22     this.date = null;
23     /**
24      * Intervention duration in minutes
25      */
26     this.duration = 0;
27     /**
28      * Intervention description
29      */
30     this.description = "";
31     /**
32      * Technician that did the intervention
33      */
34     this.technician = null;
35     /**
36      * Did the intervention resolved the ticket
37      */
38     this.resolved = false;
39   }
40
41   Intervention(int id, Date date, int duration, String description,
   Person technician, boolean resolved)
42   {
43     this();
44
45     this.date = date;
46     this.duration = duration;
47     this.description = description;
48     this.technician = technician;
49     this.resolved = resolved;
50   }
51
52   /**
53    * toString Overide
54    */
55   public String toString()
```

```
56      {
57        return "date: " + this.date +
58          "| duration: " + this.duration +
59          "| description: " + this.description +
60          "| technician: " + this.technician.id +
61          "| resolved: " + this.resolved;
62      }
63    }
64
```

```java
import java.io.Serializable;

/**
 * Class Representing a person.
 */
class Person implements Serializable
{
  /**
   * required for serialization
   */
  private static final long serialVersionUID = -2286451351076678805L;

  /**
   * Fake enum for PersonTypes
   */
  public final class PersonTypes {
     public static final int ALL = 0;
     public static final int USER = 1;
     public static final int WORKER = 2;
  }

  /**
   * Id of the Object
   */
  public int id;
  /**
   * Name of the person (must be unique)
   */
  public String name;
  /**
   * Person type of a PersonTypes value
   */
  public int type;

  Person()
  {
    this.id = 0;
    this.name = "";
  }

  Person(int type, String name)
  {
    this();
    this.type = type;
    this.name = name;
  }

  public String toString()
  {
    return "id: " + this.id +
      "| type: " +this.type +
      "| name: " + this.name;
  }
}
```

```java
import java.io.Serializable;

/**
 * Class Representing a Station.
 */
class Station implements Serializable
{
  /**
   * Required for serialization
   */
  private static final long serialVersionUID = 220727677960360309L;

  /**
   * Station ID
   */
  public int id;
  /**
   * Station's Name (must be unique)
   */
  public String name;
  /**
   * Operating System
   */
  public String os;
  /**
   * CPU
   */
  public String cpu;
  /**
   * Hard Drive
   */
  public float hdd;
  /**
   * RAM
   */
  public float ram;

  /**
  * Contructor
  */
  Station()
  {
    this.id = 0;
    this.name = "";
    this.os = "";
    this.cpu = "";
    this.hdd = 0;
    this.ram = 0;
  }

  Station(String name, String os ,String cpu, int hdd, int ram)
  {
    this();

    this.name = name;
    this.os = os;
```

```
57        this.cpu = cpu;
58        this.hdd = hdd;
59        this.ram = ram;
60      }
61
62      /**
63       * toString Overide
64       */
65      public String toString()
66      {
67        return "id: " + this.id +
68          "| name: " + this.name +
69          "| os: " + this.os +
70          "| cpu: " + this.cpu +
71          "| hdd: " + this.hdd +
72          "| ram: " + this.ram;
73      }
74    }
75
```

```java
import java.util.Date;

/**
 * Class representing a SystemTicket (senha de sistema).
 */
class SystemTicket extends Ticket
{
  /**
   * Station to which the ticket was required for
   */
  public Station station;
  /**
   * Description of the issue
   */
  public String description;


  public SystemTicket()
  {
    super();
    this.type = "system";
    this.station = null;
    this.description = "";
  }

  SystemTicket(Person author, Date date,
      Station station, String description)
  {
    // calls parent constructor
    super( author, date);

    this.station = station;
    this.description = description;
  }

  public String toString()
  {
    return super.toString() +
    "| station: " + this.station.id +
    "| description: " + this.description;
  }
}
```

```java
import java.util.Date;
import java.util.ArrayList;

/**
 * Class representing a Ticket (senha)
 */
class Ticket
{
  /**
   * Static final string defining the type of ticket
   */
  public String type;
  /**
   * Ticket id
   */
  public int id;
  /**
   * Date of the ticket
   */
  public Date date;
  /**
   * Person that put up the ticket
   */
  public Person author;
  /**
   * ArrayList of interventions
   */
  public ArrayList<Intervention> interventions;

  /**
   * Constructor
   */
  public Ticket()
  {
    // incializes the ArrayList
    this.interventions = new ArrayList<Intervention>();

    this.id = 0;
    this.date = null;
    this.author = null;
    this.type = "";
  }

  public Ticket(Person author, Date date)
  {

    this();

    this.author = author;
    this.date = date;
  }

  /**
   * Overloading of toString
   */
  public String toString()
```

```
57      {
58        return "id: " + this.id +
59        "| type: " + this.type +
60        "| author: " + this.author.id +
61        "| date: " + this.date;
62      }
63
64      /**
65       * Check if the Ticket is resolved.
66       *
67       * @return true if it's resolved. false otherwise
68       */
69      public boolean isResolved()
70      {
71        // if no interventions exit right away with 0
72        if (this.interventions.isEmpty()) return false;
73
74        //TODO: is this properly indexed?
75        if (this.interventions.get(this.interventions.size()-1).resolved
    == true)
76        {
77          return true;
78        }
79        return false;
80      }
81
82      /**
83       * Returns the number of times the ticket been marked as resolved
84       *
85       * @return number of times ticket been marked resolved
86       */
87      public int timesResolved()
88      {
89        int count = 0;
90
91        // if no interventions exit right away with 0
92        if (this.interventions.isEmpty()) return 0;
93
94        for (int i = 0; i < this.interventions.size(); i++)
95        {
96          if (this.interventions.get(i).resolved == true)
97          {
98            count++;
99          }
100       }
101
102       return count;
103     }
104
105     /**
106      * Returns the time a ticket took to become resolved.
107      *
108      * @return time to resolved in minutes
109      */
110     public long resolveTime()
111     {
```

```
112       if (!this.isResolved()) return 0;
113
114       int time = 0;
115       for (int i = 0; i < this.interventions.size(); i++)
116       {
117         time += this.interventions.get(i).duration;
118       }
119
120       return time;
121     }
122   }
123
```

```java
import java.util.Date;

/**
 * Class representing a TrainningTicket (pedido de formação)
 */
class TrainningTicket extends Ticket
{
  /**
   * Topic of the training required
   */
  public String topic;

  /**
   *  Contructor
   */
  TrainningTicket()
  {
    super();
    this.type = "trainning";
    this.topic = "";
  }

  TrainningTicket(Person author, Date date, String topic)
  {
    super(author, date);


    this.topic = topic;
  }

  public String toString()
  {
    return super.toString() +
    "| topic: " + this.topic;
  }
}

```

```java
/**
 * Application class with CL interface
 *
 * @author dfof
 * @version 1.0
 */
class Application
{
  /**
   * Fake enum for MainMenu options
   */
  public final class MainMenuChoises {
     public static final int EXIT = 0;
     public static final int LIST_TICKETS = 1;
     public static final int EDIT_TICKETS = 2;
     public static final int NEW_TICKET = 3;
     public static final int OTHERS = 4;
     public static final int STATISTICS = 5;
  }

  /**
   * Fake enum for OtherMenu
   */
  public final class ManageOthersMenuChoises {
     public static final int EXIT = 0;
     public static final int USERS = 1;
     public static final int TECHS = 2;
     public static final int POSTS = 3;
  }

  // list variables
  public PersonList users;
  public PersonList techs;
  public StationList posts;
  public TicketList tickets;

  /**
   * Contructor
   */
  Application()
  {
    // inicalizes the lists
    users = new PersonList("Pessoas.dat", Person.PersonTypes.USER);
    techs = new PersonList("Pessoas.dat", Person.PersonTypes.WORKER);
    posts = new StationList("Postos_trab.dat");

    tickets = new TicketList("Pedidos.txt", "Intervencoes.txt", users,
techs, posts);

    // loads files
    posts.loadFile();
    users.loadFile();
    techs.loadFile();
    tickets.loadFile();
  }

```

```
56      /**
57       * exit's the application
58       */
59      public void exit()
60      {
61        // saves posts
62        posts.saveFile();
63
64        // saves users and techs in one file
65        PersonList pl = new PersonList("Pessoas.dat", Person.PersonTypes.
    ALL);
66
67        for (int i = 0; i < users.size(); i++) pl.add(users.get(i));
68        for (int i = 0; i < techs.size(); i++) pl.add(techs.get(i));
69        pl.saveFile();
70
71        tickets.saveFile();
72
73        System.exit(0);
74      }
75
76      /**
77       * Runs the App by showing the main menu handleling
78       * user choises.
79       */
80      public void run()
81      {
82        for (;/*show menu*/;)
83        {
84          showMainMenu();
85          int opt = User.readInt();
86
87          switch (opt)
88          {
89          case MainMenuChoises.LIST_TICKETS:
90            tickets.clPrintAll();
91            break;
92
93          case MainMenuChoises.EDIT_TICKETS:
94            tickets.clEdit();
95            break;
96
97          case MainMenuChoises.NEW_TICKET:
98            tickets.clNew();
99
100
101           boolean done;
102           do
103           {
104             done = true;
105             System.out.print("Adicionar intervenção? (s/n): ");
106             String r = User.readString();
107
108             if (!(r.equals("s") || r.equals("n")))
109             {
110               System.out.println("Escolha invalida.");
```

```
111               done = false;
112             }
113
114             if (r.equals("s"))
115             {
116               tickets.clNewIntervention(tickets.size()-1);
117             }
118           }
119           while (!done);
120           break;
121
122         case MainMenuChoises.OTHERS:
123           manageOthers();
124           break;
125
126         case MainMenuChoises.STATISTICS:
127           showStatistics();
128           break;
129
130         case MainMenuChoises.EXIT:
131           this.exit();
132           return;
133
134         default:
135           System.out.println("opção invalida");
136         }
137       }
138     }
139
140     /**
141      * shows the manageOthers menu and handles user
142      * choises.
143      */
144     public void manageOthers()
145     {
146       @SuppressWarnings("unchecked")
147       CustomList list = null;
148
149       for (;;)
150       {
151         showManageOthersMenu();
152         int op = User.readInt();
153
154         switch(op)
155         {
156         case ManageOthersMenuChoises.USERS:
157           list = users;
158           break;
159         case ManageOthersMenuChoises.TECHS:
160           list = techs;
161           break;
162         case ManageOthersMenuChoises.POSTS:
163           list = posts;
164           break;
165         case ManageOthersMenuChoises.EXIT:
166           return;
```

```
167
168         default:
169           System.out.println("Opção invalida");
170         }
171
172         String eop;
173         do
174         {
175           System.out.println("## Listagem ##");
176           list.clPrintAll();
177           showEditOptions();
178           eop = User.readString();
179
180           // makes sure there's something there
181           if (eop.length() != 1) eop = "i";
182
183           switch(eop.charAt(0))
184           {
185           case 'n':
186             list.clNew();
187             break;
188           case 'e':
189             list.clEdit();
190             break;
191           case 'a':
192             list.clDelete();
193             break;
194
195           case 'v':
196             break;
197
198           default:
199             System.out.println("Opcao invalida.");
200           }
201
202         }
203         while (eop.charAt(0) != 'v');
204       }
205     }
206
207     /**
208      * Shows some statistics as specified in requirements
209      */
210     public void showStatistics()
211     {
212       System.out.println("## Estatisticas ##");
213
214       System.out.println("Numero de pedidos no sistema: " + tickets.size
      ());
215       System.out.println("Numero de pedidos resolvidos: " + tickets.
      resolvedCount());
216
217       System.out.println("Numero de intervenções medio: " + tickets.
      averageInterventions(false));
218       System.out.println("Numero de intervenções medio em pedidos
      resolvidos: " + tickets.averageInterventions(true));
```

```
219       System.out.println("Tempo medio de resolução: " + tickets.
      averageResolveTime() + " minutos");
220
221       System.out.println("");
222     }
223
224     /**
225      * Prints the main menu
226      */
227     public void showMainMenu()
228     {
229       System.out.println("## Menu Principal ##");
230       System.out.println("1) Listar Pedidos");
231       System.out.println("2) Ver/Editar Pedidos");
232       System.out.println("3) Novo Pedido");
233       System.out.println("---------------");
234       System.out.println("4) Gerir outros dados");
235       System.out.println("5) Estatisticas");
236       System.out.println("---------------");
237       System.out.println("0) sair");
238     }
239
240     /**
241      * Prints the ManageOthers menu
242      */
243     public void showManageOthersMenu()
244     {
245       System.out.println("## Gerir outros dados ##");
246       System.out.println("1) Utilizadores");
247       System.out.println("2) Tecnicos");
248       System.out.println("3) Postos de Trabalho");
249       System.out.println("---------------");
250       System.out.println("0) sair");
251     }
252
253     /**
254      * Prints edit options
255      */
256     public void showEditOptions()
257     {
258       System.out.print("Opcoes: ");
259       System.out.print("(e)ditar, ");
260       System.out.print("(n)ovo, ");
261       System.out.print("(a)pagar, ");
262       System.out.println("(v)oltar ao menu.");
263     }
264   }
```

```java
import java.io.File;
import java.util.ArrayList;

/**
* Custom List for holding database like Objects.
*
* @author dfof
* @version 1.0
*/
class CustomList<E> extends ArrayList<E>
{
  /**
   * required to extend ArrayList
   */
  private static final long serialVersionUID = -2287290890185696 65L;

  /**
   * File used by loadFile, saveFile and clearFile
   */
  public String filename;

  /**
   * last ID used
   */
  protected int lastid;

  /**
  * Contructor. Inicializes protected lastId.
  */
  CustomList()
  {
    this.lastid = 0;
  }

  CustomList(String filename)
  {
    this();
    this.filename = filename;
  }

  /**
  * get's the last id used on.
  *
  * @return id returns the last id used in the list.
  */
  public int getLastid()
  {
    return this.lastid;
  }

  /**
   * Returns an object element given it's id.
   *
   * @param id of the element
   * @return object if found. null otherwise
   */
```

```
57      public E getById(int id)
58      {
59        // Implemented by extendor
60        return null;
61      }
62
63      /**
64       * Returns the position of an element given it's id.
65       *
66       * @param id of the element
67       * @return position of the element. negative value if not found.
68       */
69      public int findId(int id)
70      {
71        // Implemented by extendor
72        return -1;
73      }
74
75      /**
76       * Add's an element to the list. By adding the element the lastId
    will be
77       * incremented and used to overide the .id member of the given object.
78       *
79       * @param Object Object to be added, note that the oject must have a
    public
80       *        id property or the add will fail.
81       *
82       * @return added returns true if the
83       */
84      public boolean add(E o)
85      {
86        lastid++;
87        return super.add(o);
88      }
89
90      /**
91       * Delete's an element of the list by id
92       *
93       * @param id of the element
94       * @return true if sucessfull. false on error
95       */
96      public boolean delete(int id)
97      {
98        int pos;
99        if (0 < (pos = this.findId(id)))
100       {
101         remove(pos);
102         return true;
103       }
104
105       return false;
106     }
107
108     /**
109      * Overide of toString
110      */
```

```
111     public String toString()
112     {
113       String output = "";
114 
115       for (int i = 0; i < this.size(); i++)
116       {
117         output += this.get(i).toString();
118       }
119 
120       return output;
121     }
122 
123     /**
124     * Load's a list from a file. The function uses the filename
    property as
125     * the target from were to load the file. it will then attempt to read
126     * object's from it and add them to the list.
127     *
128     * @return readCount number of elements read from file, in case of
    failure
129     *         0 will be returned and the apropriate exeption will be
    trown.
130     */
131     public int loadFile()
132     {
133       // Implemented by extendor
134       return 0;
135     }
136 
137     /**
138     * Save's the list to a file. Will use the file indicated in public
139     * property filename to save.
140     *
141     * @return writeCount number of elements writen to the file, in case
    of
142     *         failure 0 will be returned and the apropriate exeption
    will be
143     *         trown.
144     */
145     public int saveFile()
146     {
147       // Implemented by extendor
148       return 0;
149     }
150 
151     /**
152      * Clears the file of it's content. It will open the file and
    overwrite it
153      * with empty content.
154      *
155      * @return 0 on sucess other on failure
156      */
157     public int clearFile()
158     {
159       if (this.filename == null) return -1;
160       try
```

```
161       {
162         File file = new File(this.filename);
163 
164         if (file.exists())
165         {
166           file.delete();
167         }
168       }
169       catch (Exception e)
170       {
171         e.printStackTrace();
172         return 1;
173       }
174       return 0;
175     }
176 
177     /**
178      * Outputs to stdout a formated list of content.
179      */
180     public void clPrintAll()
181     {
182       // Implemented by extendor
183     }
184 
185     /**
186      * Prompts on the command line for an id of an element
187      * and them prints it in stdout.
188      */
189     public void clPrint()
190     {
191       System.out.print("id: ");
192       int id = User.readInt();
193 
194       if (0 < this.findId(id))
195       {
196         this.clPrint(id);
197       }
198       else
199       {
200         System.out.println("Id não existe.");
201       }
202     }
203 
204     /**
205      * Outputs to stdout a formated element by id.
206      *
207      * @param id of element in list
208      */
209     public void clPrint(int id)
210     {
211       // Implemented by extendor
212     }
213 
214     /**
215      * Prompts in the command line for a new element
216      * on the list and adds it.
```

```
217       *
218       * @return true if sucessfull. false on error
219       */
220      public boolean clNew()
221      {
222        // Implemented by extendor
223        return false;
224      }
225
226      /**
227       * Prompts in the command line for an element
228       * and requests updated fields for it.
229       *
230       * @return true if sucessfull. false on error
231       */
232      public boolean clEdit()
233      {
234        System.out.print("id: ");
235        int id = User.readInt();
236
237        return this.clEdit(id);
238      }
239
240      /**
241       * Prompts in the command line for updated fields
242       * of the element with the given id.
243       *
244       * @param id of the element
245       * @return true if sucessfull. false on error
246       */
247      public boolean clEdit(int id)
248      {
249        // Implemented by extendor
250        return false;
251      }
252
253      /**
254       * Prompts in the command line for an element
255       * and asks for comfirmation before deleting it.
256       *
257       * @return true if sucessfull. false on error
258       */
259      public boolean clDelete()
260      {
261        System.out.print("id: ");
262        int id = User.readInt();
263
264        return this.clDelete(id);
265      }
266
267      /**
268       * Prompts in the command line for comfirmation
269       * and deletes the element.
270       *
271       * @param id of the element
272       * @return true if sucessfull. false on error
```

```
273       */
274      public boolean clDelete(int id)
275      {
276        int pos;
277        if (0 > (pos = this.findId(id)))
278        {
279          System.out.println("id não existe.");
280          return false;
281        }
282
283        this.clPrint(id);
284        System.out.println("Tem a certesa que quer apagar (s/n)?");
285        char op = User.readString().charAt(0);
286
287        if (op == 's')
288        {
289          this.remove(pos);
290          return true;
291        }
292        return false;
293      }
294    }
295
```

```
1  import java.io.EOFException;
2  import java.io.FileInputStream;
3  import java.io.FileNotFoundException;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.ObjectInputStream;
7  import java.io.ObjectOutputStream;
8
9  /**
10  * Custom List for holding database like Objects.
11  *
12  * @author dfof
13  * @version 1.0
14  */
15 class PersonList extends CustomList<Person>
16 {
17   /**
18    * required for serialization
19    */
20   private static final long serialVersionUID = -3502716191976118845L;
21
22   /**
23    * Type of person we're storing
24    */
25   public int type;
26
27   PersonList()
28   {
29     super();
30     this.type = Person.PersonTypes.ALL;
31   }
32
33   PersonList(String filename, int type)
34   {
35     super(filename);
36     this.type = type;
37   }
38
39   public boolean add(Person p)
40   {
41     if (this.type == Person.PersonTypes.ALL || p.type == this.type)
42     {
43       p.id = this.lastid;
44       return super.add(p);
45     }
46     return false;
47   }
48
49   public Person getById(int id)
50   {
51     for (int i = 0; i < this.size(); i++)
52     {
53       if (id == this.get(i).id)
54       {
55         return this.get(i);
56       }
```

```
57        }
58
59        return null;
60      }
61
62      public int findId(int id)
63      {
64        for (int i = 0; i < this.size(); i++)
65        {
66          if (id == this.get(i).id)
67          {
68            return i;
69          }
70        }
71
72        return -1;
73      }
74
75      public int loadFile()
76      {
77        try
78        {
79          FileInputStream fis = new FileInputStream(this.filename);
80          ObjectInputStream ois = new ObjectInputStream(fis);
81
82          for (;;)
83          {
84            Person p = (Person)ois.readObject();
85
86            if (p.type == Person.PersonTypes.ALL || p.type == this.type)
87            {
88              if (p.id > this.lastid)
89              {
90                this.lastid = p.id;
91              }
92              this.add(p);
93            }
94          }
95
96        }
97        catch (FileNotFoundException e)
98        {
99          return 0;
100       }
101       catch (EOFException e)
102       {
103         return this.size();
104       }
105       catch (ClassNotFoundException e)
106       {
107         return this.size();
108       }
109       catch (IOException e)
110       {
111         e.printStackTrace();
112       }
```

```
113       return this.size();
114     }
115
116     public int saveFile()
117     {
118       // if empty nothing to save
119       if (this.isEmpty()) return 0;
120       int writeCount = 0;
121
122       try
123       {
124         // opens in append mode
125         FileOutputStream fos = new FileOutputStream(this.filename);
126         ObjectOutputStream oos = new ObjectOutputStream(fos);
127
128         for (int i = 0; i < this.size(); i++)
129         {
130           oos.writeObject(this.get(i));
131           writeCount++;
132         }
133
134         oos.close();
135         fos.close();
136       }
137       catch (IOException e)
138       {
139         e.printStackTrace();
140       }
141       return writeCount;
142     }
143
144     public void clPrintAll()
145     {
146       String formating = "%-3s|%-30s\n";
147       Person p;
148
149       // prints header
150       if (this.isEmpty())
151       {
152         System.out.print("Nao a ");
153
154         System.out.println(
155           (this.type == Person.PersonTypes.USER)?
156               "Utilizador" :
157               "Tecnico"
158         );
159       }
160       else
161       {
162         System.out.printf(formating,
163             "id",
164             "nome"
165           );
166       }
167
168       for (int i = 0; i < this.size(); i++)
```

```
169       {
170         p = this.get(i);
171 
172         System.out.printf(formating,
173             p.id,
174             p.name
175           );
176       }
177       System.out.println("");
178     }
179 
180     public void clPrint(int id)
181     {
182       int pos;
183       if (0 <= (pos = this.findId(id)))
184       {
185         Person p = this.get(pos);
186 
187         System.out.println("nome: " + p.name);
188       }
189     }
190 
191     public boolean clNew()
192     {
193       Person p = new Person();
194       p.type = this.type;
195 
196       boolean done;
197       do
198       {
199         done = true;
200         System.out.print("nome: ");
201         p.name = User.readString();
202 
203         for (int i = 0; i < this.size(); i++)
204         {
205           if (this.get(i).name.equals(p.name))
206           {
207             System.out.println("nome já existe, escolher outro.");
208             done = false;
209           }
210         }
211       }
212       while (!done);
213 
214       this.add(p);
215       return true;
216     }
217 
218     public boolean clEdit(int id)
219     {
220       int pos;
221       if (0 > (pos = this.findId(id)))
222       {
223         System.out.println("Id não existe.");
224         return false;
```

```
225         }
226
227         Person p = this.get(pos);
228
229         System.out.print("nome ["+ p.name +"]: ");
230         p.name = User.readString();
231         return true;
232       }
233     }
```

```
1  import java.io.*;
2
3  /**
4   * Custom List for holding database like Objects.
5   *
6   * @author dfof
7   * @version 1.0
8   */
9  class StationList extends CustomList<Station>
10 {
11   /**
12    * Required for serialization
13    */
14   private static final long serialVersionUID = 1350364189904060519L;
15
16
17   public StationList(String filename)
18   {
19     super(filename);
20   }
21
22   public boolean add(Station s)
23   {
24     s.id = this.lastid;
25     return super.add(s);
26   }
27
28   public Station getById(int id)
29   {
30     for (int i = 0; i < this.size(); i++)
31     {
32       if (id == this.get(i).id)
33       {
34         return this.get(i);
35       }
36     }
37
38     return null;
39   }
40
41   public int findId(int id)
42   {
43     for (int i = 0; i < this.size(); i++)
44     {
45       if (id == this.get(i).id)
46       {
47         return i;
48       }
49     }
50
51     return -1;
52   }
53
54   public int loadFile()
55   {
56     try
```

```
57        {
58          FileInputStream fis = new FileInputStream(this.filename);
59          ObjectInputStream ois = new ObjectInputStream(fis);
60
61          for (;;)
62          {
63            Station s = (Station)ois.readObject();
64
65
66            if (s.id > this.lastid)
67            {
68              this.lastid = s.id;
69            }
70
71            this.add(s);
72          }
73
74        }
75        catch (FileNotFoundException e)
76        {
77          return 0;
78        }
79        catch (EOFException e)
80        {
81          return this.size();
82        }
83        catch (ClassNotFoundException e)
84        {
85          return this.size();
86        }
87        catch (IOException e)
88        {
89          e.printStackTrace();
90        }
91        return this.size();
92      }
93
94      public int saveFile()
95      {
96        // if empty nothing to save
97        if (this.isEmpty()) return 0;
98        int writeCount = 0;
99
100       try
101       {
102         FileOutputStream fos = new FileOutputStream(this.filename);
103         ObjectOutputStream oos = new ObjectOutputStream(fos);
104
105         for (int i = 0; i < this.size(); i++)
106         {
107           oos.writeObject(this.get(i));
108           writeCount++;
109         }
110
111         oos.close();
112         fos.close();
```

```
113       }
114       catch (IOException e)
115       {
116         e.printStackTrace();
117       }
118       return writeCount;
119     }
120
121     public void clPrintAll()
122     {
123       String formating = "%-3s|%-20s|%-30s|%-15s|%-6s|%-5s\n";
124       Station s;
125
126       // prints header
127       if (this.isEmpty())
128       {
129         System.out.println("Nao a postos de trabalho");
130       }
131       else
132       {
133         System.out.printf(formating,
134
135             "id",
136             "name",
137             "os",
138             "cpu",
139             "ram",
140             "hdd"
141           );
142       }
143
144       for (int i = 0; i < this.size(); i++)
145       {
146         s = this.get(i);
147
148         System.out.printf(formating,
149             s.id,
150             s.name,
151             s.os,
152             s.cpu,
153             s.ram,
154             s.hdd
155           );
156       }
157       System.out.println("");
158     }
159
160     public void clPrint(int id)
161     {
162       int pos;
163       if (0 <= (pos = this.findId(id)))
164       {
165         Station s = this.get(pos);
166
167         System.out.println("nome: " + s.name);
168         System.out.println("os: " + s.os);
```

```
169         System.out.println("cpu: " + s.cpu);
170         System.out.println("ram: " + s.ram);
171         System.out.println("hdd: " + s.hdd);
172       }
173     }
174
175     public boolean clNew()
176     {
177       Station s = new Station();
178
179       boolean done;
180       do
181       {
182         done = true;
183         System.out.print("nome: ");
184         s.name = User.readString();
185
186         for (int i = 0; i < this.size(); i++)
187         {
188           if (this.get(i).name.equals(s.name))
189           {
190             System.out.println("nome já existe, escolher outro.");
191             done = false;
192           }
193         }
194       }
195       while (!done);
196
197       System.out.print("os: ");
198       s.os = User.readString();
199       System.out.print("cpu: ");
200       s.cpu = User.readString();
201       System.out.print("hdd (gb): ");
202       s.hdd = User.readFloat();
203       System.out.print("ram (gb): ");
204       s.ram = User.readFloat();
205
206       this.add(s);
207       return true;
208     }
209
210     public boolean clEdit(int id)
211     {
212       int pos;
213       if (0 > (pos = this.findId(id)))
214       {
215         System.out.println("Id não existe.");
216         return false;
217       }
218
219       Station s = this.get(pos);
220
221       System.out.print("nome ["+ s.name +"]: ");
222       s.name = User.readString();
223       System.out.print("os ["+ s.os +"]: ");
224       s.os = User.readString();
```

```
225         System.out.print("cpu ["+ s.cpu +"]: ");
226         s.cpu = User.readString();
227         System.out.print("hdd ["+ s.hdd +"](gb): ");
228         s.hdd = User.readFloat();
229         System.out.print("ram ["+ s.ram +"[(gb): ");
230         s.ram = User.readFloat();
231         return true;
232       }
233     }
234
```

```
1  import java.io.*;
2  import java.text.ParseException;
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5  import java.util.Calendar;
6  import java.util.HashMap;
7  import java.util.Map;
8  import java.util.StringTokenizer;
9
10
11 /**
12  * Custom List based on CustomList for holding Ticket Objects.
13  *
14  * @author dfof
15  * @version 1.0
16  */
17 class TicketList extends CustomList<Ticket>
18 {
19   /**
20    * Required for serialization
21    */
22   private static final long serialVersionUID = -900755006490712879L;
23   /**
24    * File were the interventions are saved
25    */
26   public String interventionFile;
27   /**
28    * "Pointer" to the list of Stations
29    */
30   private StationList posts;
31   /**
32    * "Pointer" to the list of technician
33    */
34   private PersonList techs;
35   /**
36    * "Pointer to the list of users
37    */
38   private PersonList users;
39
40   TicketList()
41   {
42     super();
43
44     this.interventionFile = "";
45     this.users = null;
46     this.techs = null;
47     this.posts = null;
48   }
49
50   TicketList(String ticketFile, String interventionFile, PersonList
   users, PersonList techs, StationList posts)
51   {
52     super(ticketFile);
53     this.interventionFile = interventionFile;
54     this.users = users;
55     this.techs = techs;
```

```
56        this.posts = posts;
57      }
58
59      public boolean add(Ticket t)
60      {
61        t.id = this.lastid;
62        return super.add(t);
63      }
64
65      public Ticket getById(int id)
66      {
67        for (int i = 0; i < this.size(); i++)
68        {
69          if (id == this.get(i).id)
70          {
71            return this.get(i);
72          }
73        }
74
75        return null;
76      }
77
78      public int findId(int id)
79      {
80        for (int i = 0; i < this.size(); i++)
81        {
82          if (id == this.get(i).id)
83          {
84            return i;
85          }
86        }
87
88        return -1;
89      }
90
91      /**
92       * Returns the number of tickets that are resolved.
93       *
94       * @return number of tickets resolved
95       */
96      public int resolvedCount()
97      {
98        int count = 0;
99
100       for (int i = 0; i < this.size(); i++)
101       {
102         if (this.get(i).isResolved())
103         {
104           count++;
105         }
106       }
107
108       return count;
109     }
110
111     /**
```

```
112     * Returns the average resolve time for the all the tickets.
113     *
114     * @return average Average resolve time of all the tickets
115     */
116     public float averageResolveTime()
117     {
118       return averageResolveTime(new Date(0), Calendar.getInstance().
      getTime());
119     }
120
121     /**
122     * Returns the average resolve time for the tickets in starting from
      the
123     * given date.
124     *
125     * @param begin Begining date
126     *
127     * @return average Average resolve time of the tickets
128     */
129     public float averageResolveTime(Date begin)
130     {
131       return averageResolveTime(begin, Calendar.getInstance().getTime());
132     }
133
134     /**
135     * Returns the average resolve time for the tickets in the given
136     * time window.
137     *
138     * @param begin Begining date of the window
139     * @param end Ending date of the window
140     *
141     * @return average Average resolve time of the tickets in the given
      window
142     */
143     public float averageResolveTime(Date begin, Date end)
144     {
145       float avg = 0;
146       int total = 0;
147
148       // in case there's are no tickets return 0 right away
149       if (this.isEmpty()) return avg;
150
151       for (int i = 0; i < this.size(); i++)
152       {
153         // not inclusive
154         if (get(i).date.after(begin) && get(i).date.before(end))
155         {
156           if (this.get(i).isResolved())
157           {
158             avg += this.get(i).resolveTime();
159             total++;
160           }
161         }
162       }
163
164       return  avg/(float)total;
```

```
165     }
166
167     /**
168      * Returns the average number of interventions for the tickets. If
    the resolved flag is true
169      * will only show resolved tickets.
170      *
171      * @param resolved true if you only want the resolved tickets
172      * @return average Average number of intervention
173      */
174     public float averageInterventions(boolean resolved)
175     {
176       return averageInterventions(resolved, new Date(0), Calendar.
    getInstance().getTime());
177     }
178
179     /**
180      * Returns the average number of interventions for the tickets
    starting on
181      * given date. window. If the resolved flag is true will only show
    resolved tickets.
182      *
183      * @param resolved true if you only want the resolved tickets
184      * @param begin Begining date
185      *
186      * @return average Average number of intervention
187      */
188     public float averageInterventions(boolean resolved, Date begin)
189     {
190       return averageInterventions(resolved, begin, Calendar.getInstance
    ().getTime());
191     }
192
193     /**
194      * Returns the average number of interventions for the tickets in
    the given
195      * time window. If the resolved flag is true will only show
    resolved tickets.
196      *
197      * @param resolved true if you only want the resolved tickets
198      * @param begin Begining date of the window
199      * @param end Ending date of the window
200      *
201      * @return average Average number of intervention in the given window
202      */
203     public float averageInterventions(boolean resolved, Date begin, Date
     end)
204     {
205       int count = 0;
206       int total = 0;
207
208       // avoid doing useless work
209       if (this.isEmpty()) return 0;
210
211       for (int i = 0; i < this.size(); i++)
212       {
```

```
213         // not inclusive
214         if (this.get(i).date.after(begin) && this.get(i).date.before(end
    ))
215         {
216           if((resolved && this.get(i).isResolved()) || !resolved)
217           {
218             count += this.get(i).interventions.size();
219             total++;
220           }
221         }
222       }
223
224       return (float)count/(float)total;
225     }
226
227     public int loadFile()
228     {
229       SimpleDateFormat df = new SimpleDateFormat("EEE MMM dd HH:mm:ss
    zzz yyyy");
230
231       int readCount = 0;
232       String line;
233
234       try // Read tickets
235       {
236         FileInputStream fis = new FileInputStream(this.filename);
237         InputStreamReader in = new InputStreamReader(fis, "UTF-8");
238
239         BufferedReader iTickets = new BufferedReader(in);
240
241         while (null != (line = iTickets.readLine()))
242         {
243           // parce line
244           Map<String, String> properties = this.parseLine(line, 5, 7);
245           if (null == properties) continue; // invalid parse
246
247           try // property access
248           {
249             Ticket t = null;
250
251             if (properties.get("type").equals("system"))
252             {
253               t = new SystemTicket();
254
255               ((SystemTicket)t).description = properties.get(
    "description");
256               ((SystemTicket)t).station = posts.getById(Integer.valueOf(
    properties.get("station")));
257
258               // station can't be null
259               if (((SystemTicket)t).station == null) continue;
260             }
261             if (properties.get("type").equals("trainning"))
262             {
263               t = new TrainningTicket();
264
```

```
265               ((TrainningTicket)t).topic = properties.get("topic");
266             }
267
268             // common properties
269             t.id = Integer.valueOf(properties.get("id"));
270             t.author = users.getById(Integer.valueOf(properties.get(
    "author")));
271             if (t.author == null) continue;
272
273             try // Parse date
274             {
275               t.date = df.parse(properties.get("date"));
276             }
277             catch (ParseException e)
278             {
279               continue;
280             }
281
282             readCount++;
283             this.add(t);
284           }
285           catch (NumberFormatException e)
286           {
287             continue;
288           }
289           catch (NullPointerException e)
290           {
291             continue;
292           }
293         }
294         iTickets.close();
295       }
296       catch (FileNotFoundException e)
297       {
298         return 0;
299       }
300       catch (IOException e)
301       {
302         e.printStackTrace();
303       }
304
305       try // Read interventions
306       {
307         FileInputStream fis = new FileInputStream(this.interventionFile);
308         InputStreamReader in = new InputStreamReader(fis, "UTF-8");
309
310         BufferedReader iInterventions = new BufferedReader(in);
311
312         while (null != (line = iInterventions.readLine()))
313         {
314           // parce line
315           Map<String, String> properties = this.parseLine(line, 6, 6);
316           if (null == properties) continue; // invalid parse
317
318           try // access properties
319           {
```

```
320             Ticket t = this.getById(Integer.valueOf(properties.get(
    "ticket")));
321
322             // invalid ticket
323             if (t == null) continue;
324             Intervention iv = new Intervention();
325
326             try
327             {
328               iv.date = df.parse(properties.get("date"));
329             }
330             catch (ParseException e)
331             {
332               continue;
333             }
334
335             iv.technician = techs.getById(Integer.valueOf(properties.get
    ("technician")));
336             if (iv.technician == null) continue;
337
338             iv.duration = Integer.valueOf(properties.get("duration"));
339             iv.resolved = Boolean.valueOf(properties.get("resolved"));
340             iv.description = properties.get("description");
341
342             t.interventions.add(iv);
343           }
344           catch (NumberFormatException e)
345           {
346             continue;
347           }
348           catch (NullPointerException e)
349           {
350             continue;
351           }
352         }
353         iInterventions.close();
354       }
355       catch (FileNotFoundException e)
356       {
357         System.out.println("Aviso: Ficheiro das intervenções não foi
    encontrado.");
358         return readCount;
359       }
360       catch (IOException e)
361       {
362         e.printStackTrace();
363       }
364
365       return readCount;
366     }
367
368     public int saveFile()
369     {
370       int writeCount = 0;
371
372       try
```

```
373       {
374         FileOutputStream fos1 = new FileOutputStream(this.filename);
375         FileOutputStream fos2 = new FileOutputStream(this.
      interventionFile);
376
377         OutputStreamWriter out1 = new OutputStreamWriter(fos1, "UTF-8");
378         OutputStreamWriter out2 = new OutputStreamWriter(fos2, "UTF-8");
379
380         Writer oTickets = new BufferedWriter(out1);
381         Writer oInterventions = new BufferedWriter(out2);
382
383         for (int i = 0; i < this.size(); i++)
384         {
385           // writes the ticket line
386           oTickets.write(this.get(i).toString() + "\n");
387           writeCount++;
388
389           if (this.get(i).interventions.isEmpty()) continue;
390           for (int j = 0; j < this.get(i).interventions.size(); j++)
391           {
392             // writes the interventions
393             oInterventions.write("ticket: " + this.get(i).id + "| " +
394                 this.get(i).interventions.get(j).toString() + "\n");
395           }
396
397         }
398
399         oTickets.close();
400         oInterventions.close();
401       }
402       catch (Exception e)
403       {
404         e.printStackTrace();
405         return 0;
406       }
407       return writeCount;
408     }
409
410     public void clPrintAll()
411     {
412       String formating = "%-3s|%-10s|%-30s|%-4s|%-30s|%-10s|%-10s|%s\n";
413       Ticket t;
414
415       // prints header
416       if (this.isEmpty())
417       {
418         System.out.println("Nao a tickets.");
419       }
420       else
421       {
422         System.out.printf(formating,
423             "id",
424             "tipo",
425             "autor",
426             "int",
427             "topico/descrição",
```

```
428             "posto",
429             "resolvido",
430             "data"
431           );
432       }
433
434       for (int i = 0; i < this.size(); i++)
435       {
436         t = this.get(i);
437
438         String extended = "";
439         String post = "";
440
441         if (t.type.equals("system"))
442         {
443           extended = ((SystemTicket)t).description;
444           post = String.valueOf(((SystemTicket)t).station.id);
445         }
446         if (t.type.equals("trainning"))
447         {
448           extended = ((TrainningTicket)t).topic;
449         }
450
451         System.out.printf(formating,
452             t.id,
453             t.type,
454             t.author.name,
455             t.interventions.size(),
456             extended,
457             post,
458             String.valueOf(t.isResolved()),
459             t.date
460           );
461       }
462       System.out.println("");
463     }
464
465     public void clPrint(int id)
466     {
467       int pos;
468       if (0 <= (pos = this.findId(id)))
469       {
470         Ticket t = this.get(pos);
471
472         System.out.println("id: " + t.id);
473         System.out.println("tipo: " + t.type);
474         System.out.println("autor: " + t.author.name);
475         System.out.println("data: " + t.date);
476
477         if (t.type.equals("system"))
478         {
479           System.out.println("descrição: " + ((SystemTicket)t).
    description);
480           System.out.println("posto: " + ((SystemTicket)t).station.name);
481         }
482         if (t.type.equals("trainning"))
```

```
483         {
484           System.out.println("topico: " + ((TrainningTicket)t).topic);
485         }
486
487         System.out.println("Intervenções:");
488         this.clPrintAllInterventions(pos);
489       }
490     }
491
492     public boolean clNew()
493     {
494       Ticket t;
495       boolean done;
496
497       if (users.isEmpty() || techs.isEmpty())
498       {
499         System.out.println("Precisa de adicional utilizadores e/ou
    tecnicos primeiro");
500         return false;
501       }
502
503       System.out.print("Tipo de pedido (s)istema/(f)ormação: ");
504       String opt = User.readString();
505
506       if (opt.length() != 1)
507       {
508         System.out.println("Tipo de pedido desconhecido");
509         return false;
510       }
511       if (!(opt.charAt(0) == 's' || opt.charAt(0) == 'f'))
512       {
513         System.out.println("Tipo de pedido desconhecido");
514         return false;
515       }
516
517       if (opt.charAt(0) == 's')
518       {
519         if (posts.isEmpty())
520         {
521           System.out.println("Precisa de addicionar postos primeiro");
522           return false;
523         }
524
525         t = new SystemTicket();
526       }
527       else
528       {
529         t = new TrainningTicket();
530       }
531
532       // current date
533       t.date = new Date(Calendar.getInstance().getTimeInMillis());
534
535       do
536       {
537         done = true;
```

```
538         users.clPrintAll();
539         System.out.print("id do autor: ");
540         int id = User.readInt();
541 
542         if (null == (t.author = users.getById(id)))
543         {
544           System.out.println("Id invalida");
545           done = false;
546           continue;
547         }
548       }
549       while (!done);
550 
551       // trainning only fields
552       if (opt.charAt(0) == 'f')
553       {
554         System.out.print("topico: ");
555         ((TrainningTicket)t).topic = User.readString();
556       }
557       else
558       {
559         // system only fields
560         System.out.print("descrição: ");
561         ((SystemTicket)t).description = User.readString();
562 
563         do
564         {
565           done = true;
566           posts.clPrintAll();
567           System.out.print("id do posto: ");
568           int id = User.readInt();
569 
570           if (null == (((SystemTicket)t).station = posts.getById(id)))
571           {
572             System.out.println("Id invalida");
573             done = false;
574             continue;
575           }
576         }
577         while (!done);
578       }
579 
580       this.add(t);
581       return true;
582     }
583 
584     public boolean clEdit(int id)
585     {
586       int pos;
587       if (0 > (pos = this.findId(id)))
588       {
589         System.out.println("Id não existe.");
590         return false;
591       }
592 
593       String eop;
```

```
594       do
595       {
596         this.clPrint(id);
597         System.out.print("Opcoes: editar (p)edido, Intervenções:
      (n)ova, ");
598         System.out.println("(e)ditar, (a)pagar, (v)oltar ao menu.");
599         eop = User.readString();
600
601         // makes sure there's something there
602         if (eop.length() != 1) eop = "i";
603
604         switch(eop.charAt(0))
605         {
606         case 'n':
607           this.clNewIntervention(pos);
608           break;
609         case 'e':
610           this.clEditIntervention(pos);
611           break;
612         case 'a':
613           this.clDeleteIntervention(pos);
614           break;
615
616         case 'p':
617           this.clEditTicket(pos);
618           break;
619
620         case 'v':
621           break;
622
623         default:
624           System.out.println("Opcao invalida.");
625         }
626
627       }
628       while (eop.charAt(0) != 'v');
629       return true;
630     }
631
632     /**
633      * Prompts in the command line for updated fields
634      * of the element with the given id.
635      *
636      * @param index of the Ticket
637      * @return true if sucessfull. false on error
638      */
639     public boolean clEditTicket(int index)
640     {
641       Ticket t = this.get(index);
642
643       boolean done;
644       do
645       {
646         done = true;
647         users.clPrintAll();
648         System.out.print("id do autor[" +  t.author.id + "]: ");
```

```
649         int id = User.readInt();
650 
651         if (null == users.getById(id))
652         {
653           System.out.println("Id invalida");
654           done = false;
655           continue;
656         }
657         else
658         {
659           t.author = users.getById(id);
660         }
661       }
662       while (!done);
663 
664       // trainning only fields
665       if (t.type.equals("trainning"))
666       {
667         System.out.print("topico [" + ((TrainningTicket)t).topic + "]: "
    );
668         ((TrainningTicket)t).topic = User.readString();
669       }
670       if (t.type.equals("system"))
671       {
672         // system only fields
673         System.out.print("descrição [" + ((SystemTicket)t).description +
     "]: ");
674         ((SystemTicket)t).description = User.readString();
675 
676         do
677         {
678           done = true;
679           posts.clPrintAll();
680           System.out.print("id do posto [" + ((SystemTicket)t).station.
    id + "]: ");
681           int id = User.readInt();
682 
683           if (null == posts.getById(id))
684           {
685             System.out.println("Id invalida");
686             done = false;
687             continue;
688           }
689           else
690           {
691             ((SystemTicket)t).station = posts.getById(id);
692           }
693         }
694         while (!done);
695       }
696       return true;
697     }
698 
699     /**
700      * Prints all the interventions of a given Ticket
701      *
```

```
702      * @param index of the Ticket
703      */
704     public void clPrintAllInterventions(int index)
705     {
706       String formating = "%-3s|%-15s|%-10s|%-30s|%-10s|%s\n";
707 
708       Ticket t = this.get(index);
709 
710       if (t.interventions.isEmpty())
711       {
712         System.out.println("Nao a intervenções.");
713       }
714       else
715       {
716         System.out.printf(formating,
717             "id",
718             "tecnico",
719             "duração",
720             "descrição",
721             "resolvido",
722             "data"
723           );
724       }
725 
726       for (int i = 0; i < t.interventions.size(); i++)
727       {
728         Intervention iv = t.interventions.get(i);
729 
730         System.out.printf(formating,
731             i,
732             iv.technician.name,
733             iv.duration,
734             iv.description,
735             iv.resolved,
736             iv.date
737           );
738       }
739       System.out.println("");
740     }
741 
742     /**
743      * Prompts in the command line for a new element
744      * on the list and adds it.
745      *
746      * @param index of the Ticket
747      * @return true if sucessfull. false on error
748      */
749     public boolean clNewIntervention(int index)
750     {
751       Ticket t = this.get(index);
752       Intervention iv = new Intervention();
753       boolean done;
754 
755       // date
756       iv.date = Calendar.getInstance().getTime();
757 
```

```
758       do
759       {
760         done = true;
761         techs.clPrintAll();
762         System.out.print("id do tecnico: ");
763         int id = User.readInt();
764 
765         if (null == (iv.technician = techs.getById(id)))
766         {
767           System.out.println("Id invalida");
768           done = false;
769           continue;
770         }
771       }
772       while (!done);
773 
774       System.out.print("Duração (min): ");
775       iv.duration = User.readInt();
776 
777       System.out.print("Descrição: ");
778       iv.description = User.readString();
779 
780       do
781       {
782         done = true;
783         System.out.print("Resolvido? (s/n): ");
784         String rep = User.readString();
785 
786         if (!(rep.equals("s") || rep.equals("n")))
787         {
788           System.out.println("resposta errada.");
789           done = false;
790           continue;
791         }
792         else
793         {
794           iv.resolved = (rep.equals("s"))? true : false;
795         }
796       }
797       while(!done);
798       return t.interventions.add(iv);
799     }
800 
801     /**
802      * Prompts in the command line for updated fields
803      * of the element with the given id.
804      *
805      * @param index of the Ticket
806      * @return true if sucessfull. false on error
807      */
808     public boolean clEditIntervention(int index)
809     {
810       Ticket t = this.get(index);
811       Intervention iv = null;
812       boolean done;
813 
```

```
814         System.out.print("id da intervenção: ");
815         int pos = User.readInt();
816         try
817         {
818           iv = t.interventions.get(pos);
819         }
820         catch (IndexOutOfBoundsException e)
821         {
822           System.out.println("Intervenção não existe.");
823           return false;
824         }
825
826         do
827         {
828           done = true;
829           techs.clPrintAll();
830           System.out.print("id do tecnico ["+ iv.technician.id + "]: ");
831           int id = User.readInt();
832
833           if (null == techs.getById(id))
834           {
835             System.out.println("Id invalida");
836             done = false;
837             continue;
838           }
839           else
840           {
841             iv.technician = techs.getById(id);
842           }
843         }
844         while (!done);
845
846         System.out.print("Duração (min) ["+ iv.duration +": ");
847         iv.duration = User.readInt();
848
849         System.out.print("Descrição [" + iv.description + "]: ");
850         iv.description = User.readString();
851
852         do
853         {
854           done = true;
855           System.out.print("Resolvido? (s/n) ["+ iv.resolved + "]: ");
856           String rep = User.readString();
857
858           if (!(rep.equals("s") || rep.equals("n")))
859           {
860             System.out.println("resposta errada.");
861             done = false;
862             continue;
863           }
864           else
865           {
866             iv.resolved = (rep.equals("s"))? true : false;
867           }
868         }
869         while(!done);
```

```
870       return false;
871     }
872
873     /**
874      * Prompts in the command line for comfirmation
875      * and deletes the element.
876      *
877      * @param index of the Ticket
878      * @return true if sucessfull. false on error
879      */
880     public boolean clDeleteIntervention(int index)
881     {
882       Ticket t = this.get(index);
883
884       System.out.print("id da intervenção: ");
885       int id = User.readInt();
886
887       try
888       {
889         t.interventions.remove(id);
890       }
891       catch (IndexOutOfBoundsException e)
892       {
893         System.out.println("Intervenção não existe.");
894         return false;
895       }
896
897       return true;
898     }
899
900     /**
901      * Parces a line for a properties.
902      *
903      * @param line to parse
904      * @param minCount Minimal number of properties expected
905      * @param maxCount Maximum amound of properties expected
906      * @return Map of properties or null if amound of elements found is
     out of bounds
907      */
908     private Map<String, String> parseLine(String line, int minCount, int
      maxCount)
909     {
910       StringTokenizer st = new StringTokenizer(line, "|");
911       Map<String, String> dic = new HashMap<String, String>();
912
913       // out of bounds
914       if (!(st.countTokens() >= minCount && st.countTokens() <= maxCount
     )) return null;
915
916       while (st.hasMoreTokens())
917       {
918         String key = "";
919         String value = "";
920
921         StringTokenizer item = new StringTokenizer(st.nextToken(), ":");
922         if (item.countTokens() < 2) continue; // invalid item
```

```
923
924         key = item.nextToken();
925         value = item.nextToken();
926
927         // in case there's more ':' after the first
928         while (item.hasMoreTokens())
929         {
930           value += ":" + item.nextToken();
931         }
932
933         // trims white spaces
934         key = key.trim();
935         value = value.trim();
936
937         dic.put(key, value);
938       }
939
940       // out of bounds
941       if (!(dic.size() >= minCount && dic.size() <= maxCount)) return
    null;
942       return dic;
943     }
944   }
945
```