```
# Executables
OSTYPE = $(shell uname -s)
JAVAC = javac
JAVA = java
A2PS = a2ps
GHOSTVIEW = gv
DOCP = javadoc
ARCH = zip
PS2PDF = ps2pdf -dPDFX=true -sPAPERSIZE=a4
DATE = $(shell date +%Y-%m-%d)
# ExÃ©cution de commandes dans un nouveau terminal (changer en fct de l'OS)
TERM = xterm
# Options de compilation
#CFLAGS = -verbose
CFLAGS =
CLASSPATH=.

JAVAOPTIONS = --verbose

PROJECT=Chat_Client_Serveur
# nom du fichier d'impression
OUTPUT = $(PROJECT)
# nom du rÃ©pertoire ou se situera la documentation
DOC = doc
# lien vers la doc en ligne du JDK
WEBLINK = "https://docs.oracle.com/javase/8/docs/api/"
# lien vers la doc locale du JDK
LOCALLINK = "file:///Users/davidroussel/Documents/docs/java/api/"
# nom de l'archive
ARCHIVE = $(PROJECT)
# format de l'archive pour la sauvegarde
ARCHFMT = zip
# RÃ©pertoire source
SRC = src
# RÃ©pertoire bin
BIN = bin
# RÃ©pertoire Listings
LISTDIR = listings
# RÃ©pertoire Archives
ARCHDIR = archives
# RÃ©pertoire Figures
FIGDIR = graphics
# noms des fichiers sources
MAIN = examples/RunRunnableExample \
examples/RunExampleFrame \
examples/RunListFrame \
RunChatServer \
RunChatClient
SOURCES = $(SRC)/AbstractRunChat.java \
$(SRC)/RunChatClient.java \
$(SRC)/RunChatServer.java \
$(SRC)/chat/client/ChatClient.java \
$(SRC)/chat/client/package-info.java \
$(SRC)/chat/client/ServerHandler.java \
$(SRC)/chat/client/UserHandler.java \
$(SRC)/chat/Failure.java \
$(SRC)/chat/package-info.java \
$(SRC)/chat/server/ChatServer.java \
$(SRC)/chat/server/ClientHandler.java \
$(SRC)/chat/server/InputClient.java \
$(SRC)/chat/server/InputOutputClient.java \
$(SRC)/chat/server/package-info.java \
$(SRC)/chat/UserOutputType.java \
$(SRC)/chat/Vocabulary.java \
$(SRC)/examples/package-info.java \
$(SRC)/examples/RunExampleFrame.java \
$(SRC)/examples/RunListFrame.java \
$(SRC)/examples/RunnableExample.java \
$(SRC)/examples/RunRunnableExample.java \
$(SRC)/examples/widgets/ExampleFrame.java \
$(SRC)/examples/widgets/ListExampleFrame.java \
$(SRC)/logger/LoggerFactory.java \
$(SRC)/logger/package-info.java \
$(SRC)/models/Message.java \
$(SRC)/models/NameSetListModel.java \
$(SRC)/models/AuthorListFilter.java \
$(SRC)/models/package-info.java \
$(SRC)/widgets/AbstractClientFrame.java \
$(SRC)/widgets/ClientFrame.java \
$(SRC)/widgets/ClientFrame2.java \
$(SRC)/widgets/package-info.java \
$(foreach name, $(MAIN), $(SRC)/$(name).java)

OTHER = readme.txt \
reponses.txt \
Sujet.pdf \
$(SRC)/examples/icons/add_user-16.png \
$(SRC)/examples/icons/add_user-32.png \
$(SRC)/examples/icons/bg_blue-16.png \
$(SRC)/examples/icons/bg_blue-32.png \
```



```
$(SRC)/examples/icons/bg_color-32.png \
$(SRC)/examples/icons/bg_red-16.png \
$(SRC)/examples/icons/bg_red-32.png \
$(SRC)/examples/icons/delete_sign-16.png \
$(SRC)/examples/icons/delete_sign-32.png \
$(SRC)/examples/icons/erase-16.png \
$(SRC)/examples/icons/erase-32.png \
$(SRC)/examples/icons/remove_user-16.png \
$(SRC)/examples/icons/remove_user-32.png \
$(SRC)/icons/cancel-16.png \
$(SRC)/icons/cancel-32.png \
$(SRC)/icons/clock-16.png \
$(SRC)/icons/clock-32.png \
$(SRC)/icons/delete_database-16.png \
$(SRC)/icons/delete_database-32.png \
$(SRC)/icons/disconnected-16.png \
$(SRC)/icons/disconnected-32.png \
$(SRC)/icons/erase-16.png \
$(SRC)/icons/erase-32.png \
$(SRC)/icons/erase2-16.png \
$(SRC)/icons/erase2-32.png \
$(SRC)/icons/filled_filter-16.png \
$(SRC)/icons/filled_filter-32.png \
$(SRC)/icons/gender_neutral_user-16.png \
$(SRC)/icons/gender_neutral_user-32.png \
$(SRC)/icons/logout-16.png \
$(SRC)/icons/logout-32.png \
$(SRC)/icons/remove_user-16.png \
$(SRC)/icons/remove_user-32.png \
$(SRC)/icons/select_all-16.png \
$(SRC)/icons/select_all-32.png \
$(SRC)/icons/sent-16.png \
$(SRC)/icons/sent-32.png

.PHONY : doc ps

# Les targets de compilation
# pour gÃ©nÃ©rer l'application
all : $(foreach name, $(MAIN), $(BIN)/$(name).class)

#rÃ¨gle de compilation gÃ©nÃ©rique
$(BIN)/%.class : $(SRC)/%.java
	$(JAVAC) -sourcepath $(SRC) -classpath $(BIN):$(CLASSPATH) -d $(BIN) $(CFLAGS) $<

# Edition des sources $(EDITOR) doit Ãªtre une variable d'environnement
edit :
	$(EDITOR) $(SOURCES) Makefile &

# nettoyer le rÃ©pertoire
clean :
	find bin/ -type f -name "*.class" -exec rm -f {} \;
	rm -rf *~ *.log* $(DOC)/* $(LISTDIR)/*

#realclean : clean
#	rm -f $(ARCHDIR)/*.$(ARCHFMT)

# gÃ©nÃ©rer le listing
$(LISTDIR) :
	mkdir $(LISTDIR)

ps : $(LISTDIR)
	$(A2PS) -2 --file-align=fill --line-numbers=1 --font-size=10 \
	--chars-per-line=100 --tabsize=4 --pretty-print \
	--highlight-level=heavy --prologue="gray" \
	-o$(LISTDIR)/$(OUTPUT).ps Makefile $(SOURCES)

pdf : ps
	$(PS2PDF) $(LISTDIR)/$(OUTPUT).ps $(LISTDIR)/$(OUTPUT).pdf

# gÃ©nÃ©rer le listing lisible pour GÃ©rard
bigps :
	$(A2PS) -1 --file-align=fill --line-numbers=1 --font-size=10 \
	--chars-per-line=100 --tabsize=4 --pretty-print \
	--highlight-level=heavy --prologue="gray" \
	-o$(LISTDIR)/$(OUTPUT).ps Makefile $(SOURCES)

bigpdf : bigps
	$(PS2PDF) $(LISTDIR)/$(OUTPUT).ps $(LISTDIR)/$(OUTPUT).pdf

# voir le listing
preview : ps
	$(GHOSTVIEW) $(LISTDIR)/$(OUTPUT); rm -f $(LISTDIR)/$(OUTPUT) $(LISTDIR)/$(OUTPUT)~

# gÃ©nÃ©rer la doc avec javadoc
doc : $(SOURCES)
	$(DOCP) -private -d $(DOC) -author -link $(LOCALLINK) $(SOURCES)
#	$(DOCP) -private -d $(DOC) -author -linkoffline $(WEBLINK) $(LOCALLINK) $(SOURCES)

# gÃ©nÃ©rer une archive de sauvegarde
$(ARCHDIR) :
```

```
181     mkdir $(ARCHDIR)
182
183 archive : pdf $(ARCHDIR)
184     $(ARCH) $(ARCHDIR)/$(ARCHIVE).$(ARCHFMT) $(SOURCES) $(LISTDIR)/*.pdf \
185     $(FIGDIR)/*.pdf $(OTHER) $(BIN) Makefile $(FIGDIR)/*.pdf
186
187 # exÃ©cution des programmes de test
188 run : all
189     $(foreach name, $(MAIN), $(TERM) -e $(JAVA) -classpath $(BIN):$(CLASSPATH) $(name) $(JAVAOPTIONS
    ) & )
190
191 # Lancement d'un serveur
192 runserver : all
193     $(TERM)  -title server -e $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatServer --verbose --noqui
    t &
194
195 # Lancement d'un client console
196 runclient : all
197     $(TERM) -title "Zebulon" -e $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatClient --name ZÃ©bulon
    --verbose &
198
199 # Lancement d'un client graphique version 1
200 rungui1 : all
201     $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatClient --name TÃ©nÃ©phore --gui 1 --verbose
202
203 # Lancement d'un client graphique version 2
204 rungui2 : all
205     $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatClient --name ZÃ©phirine --gui 2 --verbose
206
207 # Lancement d'un serveur, puis de 2 clients (l'un console, l'autre graphique)
208 rundemo : all
209     $(TERM)  -title server -e $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatServer & \
210     sleep 1;
211     $(TERM) -title "Zebulon" -e $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatClient --name Zebulon &
     \
212     $(JAVA) -classpath $(BIN):$(CLASSPATH) RunChatClient --name Tenephore --gui 1;
```



```
1  import java.io.IOException;
2  import java.util.logging.Level;
3  import java.util.logging.Logger;
4
5  import chat.Failure;
6  import logger.LoggerFactory;
7
8  /**
9   * Classe abstraite de base pour lancer un client ou un serveur de chat
10  * @author davidroussel
11  */
12 public abstract class AbstractRunChat
13 {
14     /**
15      * Port Ã  utiliser pour les connnections entre clients et serveur
16      */
17     protected int port;
18
19     /**
20      * numero de port de communication par dÃ©faut
21      */
22     public static final int DEFAULTPORT = 1394;
23
24     /**
25      * Etat de verbose. Si true les messages de debug seront
26      * affichÃ©s. Si false les messages de debug ne seront pas affichÃ©s
27      */
28     protected boolean verbose;
29
30     /**
31      * Le logger utilisÃ© pour afficher (ou pas) les messages d'infos et
32      * d'erreurs.
33      */
34     protected Logger logger;
35
36     /**
37      * Constructeur d'un client ou d'un serveur de chat d'aprÃ¨s les arguments
38      * fournis au programme principal
39      * @param args les arguments fournis au programme principal en vue de
40      * mettre en place certaines options particuliÃ¨re Ã  un client ou un serveur
41      * Recherche des valeurs pour {@link #port} et {@link #verbose} dans les
42      * chaÃ®nes de caractÃ¨res fournis en arguments
43      */
44     protected AbstractRunChat(String[] args)
45     {
46         setAttributes(args);
47     }
48
49     /**
50      * Mise en place des valeurs des attributs et parsing des arguments
51      * @param args les arguments fournis au programme principal en vue de
52      * mettre en place certaines options particuliÃ¨re Ã  un client ou un serveur
53      * Recherche des valeurs pour {@link #port} et {@link #verbose} dans les
54      * chaÃ®nes de caractÃ¨res fournis en arguments
55      */
56     protected void setAttributes(String[] args)
57     {
58         /*
59          * On met d'abord les attributs locaux Ã  leur valeur par dÃ©faut
60          */
61         port = DEFAULTPORT;
62         verbose = false;
63
64         /*
65          * parsing des arguments
66          *  -v | --verbose : si verbose affichage des messages dans la console
67          *      sinon affichage des messages dans un fichier de log portant
68          *      le nom de la classe qui l'instancie.log
69          *  -p | --port : port Ã  utiliser pour la serverSocket
70          */
71         for (int i=0; i < args.length; i++)
72         {
73             if (args[i].startsWith("-")) // option argument
74             {
75                 if (args[i].equals("--verbose") ∨ args[i].equals("-v"))
76                 {
77                     System.out.println("Setting verbose on");
78                     verbose = true;
79                 }
80                 if (args[i].equals("--port") ∨ args[i].equals("-p"))
81                 {
82                     System.out.print("Setting port to: ");
83                     if (i < (args.length - 1))
84                     {
85                         // recherche du numÃ©ro de port dans le prochain argument
86                         Integer portInteger = readInt(args[++i]);
87                         if (portInteger ≠ null)
88                         {
89                             int readPort = portInteger.intValue();
90                             if (readPort ≥ 1024)
```

```
                                {
                                    port = readPort;
                                }
                                else
                                {
                                    System.err.println(Failure.INVALID_PORT);
                                    System.exit(Failure.INVALID_PORT.toInteger());
                                }
                            }
                            System.out.println(port);
                        }
                        else
                        {
                            System.out.println("nothing, invalid value");
                        }
                    }
                }
            }

            /*
             * CrÃ©ation du logger
             */
            logger = null;
            Class<?> runningClass = getClass();
            String logFilename =
                (verbose ? null : runningClass.getSimpleName() + ".log");
            Logger parent = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);
            Level level = (verbose ? Level.ALL : Level.WARNING);
            try
            {
                logger = LoggerFactory.getLogger(runningClass,
                                                 verbose,
                                                 logFilename,
                                                 false,
                                                 parent,
                                                 level);
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
                System.exit(Failure.OTHER.toInteger());
            }
        }

        /**
         * Une fois le client ou le serveur prÃªt, on lance son exÃ©cution
         */
        protected abstract void launch();

        /**
         * Lecture d'un entier Ã  partir d'une chaÃ®ne de caractÃ¨res
         * @param s la chaine Ã  lire
         * @return l'entier parsÃ© dans la chaine de caractÃ¨re ou bien null
         * s'il s'est produit une erreur de parsing
         */
        protected Integer readInt(String s)
        {
            try
            {
                Integer value = new Integer(Integer.parseInt(s));
                return value;
            }
            catch (NumberFormatException e)
            {
                // System.err.println("readInt: " + s + " is not a number");
                logger.warning("readInt: " + s + " is not a number");
                return null;
            }
        }
}
```



```
import java.awt.EventQueue;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Vector;

import chat.Failure;
import chat.UserOutputType;
import chat.client.ChatClient;
import widgets.AbstractClientFrame;
import widgets.ClientFrame;

/**
 * Lanceur d'un client de chat.
 *
 * @author davidroussel
 */
public class RunChatClient extends AbstractRunChat
{
    /**
     * HÃ´te sur lequel se trouve le serveur de chat
     */
    private String host;

    /**
     * Nom d'utilisateur Ã  utiliser pour se connecter au serveur. Si le nom
     * n'est pas fournit
     */
    private String name;

    /**
     * Flux d'entrÃ©e sur lequel lire les messages tapÃ©s par l'utilisateur
     */
    private InputStream userIn;

    /**
     * Flux de sortie sur lequel envoyer les messages vers l'utilisateur
     */
    private OutputStream userOut;

    /**
     * Indique si le client Ã  crÃ©er est un GUI ou pas
     */
    private boolean gui;

    /**
     * La version de l'interface graphique Ã  lancer:
     * <ul>
     *  <li>version 1 correspond Ã  l'utilisation d'une ClientFrame</li>
     *  <li>version 2 correspond Ã  l'untilisation d'une SuperClientFrame</li>
     * </ul>
     */
    private int guiVersion;

    /**
     * Ensemble des threads des clients.
     * Il faudra attendre la fin de ces threads pour terminer l'exÃ©cution
     * principal.
     */
    private Vector<Thread> threadPool;

    /**
     * Constructeur d'un lanceur de client d'aprÃ¨s les arguments du programme
     * principal
     *
     * @param args les arguments du programme principal
     */
    protected RunChatClient(String[] args)
    {
        super(args);

        /*
         * Initialisation des flux d'I/O utilisateur Ã  null
         * ils dÃ©pendront du client Ã  crÃ©er (console ou GUI)
         */
        userIn = null;
        userOut = null;

        /*
         * Initialisation du pool de thread des clients
         */
        threadPool = new Vector<Thread>();
    }

    /**
     * Mise en place des attributs du client de chat en fonction des arguments
     * utilisÃ©s dans la ligne de commande
     * @param args les arguments fournis au programme principal.
```

```
91      */
92      @Override
93      protected void setAttributes(String[] args)
94      {
95          /*
96           * parsing des arguments communs aux clients et serveur
97           * -v | --verbose
98           * -p | --port : port Ã  utiliser pour la serverSocket
99           */
100         super.setAttributes(args);
101
102         /*
103          * On met d'abord les attributs locaux Ã  leur valeur par dÃ©faut
104          */
105         host = null;
106         name = null;
107         gui = false;
108
109         /*
110          * parsing des arguments spÃ©cifique au client
111          * -h | --host : nom ou adresse IP du serveur
112          * -n | --name : nom d'utilisateur
113          * -g | --gui : pour lancer le client GUI
114          */
115         for (int i = 0; i < args.length; i++)
116         {
117             if (args[i].equals("--host") ∨ args[i].equals("-h"))
118             {
119                 if (i < (args.length - 1))
120                 {
121                     // parse next arg for in port value
122                     host = args[++i];
123                     logger.fine("Setting host to " + host);
124                 }
125                 else
126                 {
127                     logger.warning("Setting host to: nothing, invalid value");
128                 }
129             }
130             else if (args[i].equals("--name") ∨ args[i].equals("-n"))
131             {
132                 if (i < (args.length - 1))
133                 {
134                     // parse next arg for in port value
135                     name = args[++i];
136                     logger.fine("Setting user name to: " + name);
137                 }
138                 else
139                 {
140                     logger.warning("Setting user name to: nothing, invalid value");
141                 }
142             }
143             if (args[i].equals("--gui") ∨ args[i].equals("-g"))
144             {
145                 gui = true;
146                 if (i < (args.length - 1))
147                 {
148                     // parse next arg for gui version
149                     try
150                     {
151                         guiVersion = Integer.parseInt(args[++i]);
152                         if (guiVersion < 1)
153                         {
154                             guiVersion = 1;
155                         }
156                         else if (guiVersion > 2)
157                         {
158                             guiVersion = 2;
159                         }
160                     }
161                     catch (NumberFormatException nfe)
162                     {
163                         logger.warning("Invalid gui number, revert to 1");
164                         guiVersion = 1;
165                     }
166                     logger.fine("Setting gui to " + guiVersion);
167                 }
168                 else
169                 {
170                     logger.warning("ReSetting gui version to 1, invalid value");
171                     guiVersion = 1;
172                 }
173             }
174         }
175
176         if (host ≡ null) // on va chercher local host
177         {
178             try
179             {
180                 host = InetAddress.getLocalHost().getHostName();
```



```
181             }
182             catch (UnknownHostException e)
183             {
184                 logger.severe(Failure.NO_LOCAL_HOST.toString());
185                 logger.severe(e.getLocalizedMessage());
186                 System.exit(Failure.NO_LOCAL_HOST.toInteger());
187             }
188         }
189
190         if (name ≡ null) // on va chercher le nom de l'utilisateur
191         {
192             try
193             {
194                 // Try LOGNAME on unix type systems
195                 name = System.getenv("LOGNAME");
196             }
197             catch (NullPointerException npe)
198             {
199                 logger.warning("no LOGNAME found, trying USERNAME");
200                 try
201                 {
202                     // Try USERNAME on other systems
203                     name = System.getenv("USERNAME");
204                 }
205                 catch (NullPointerException npe2)
206                 {
207                     logger.severe(Failure.NO_USER_NAME + " abort");
208                     System.exit(Failure.NO_USER_NAME.toInteger());
209                 }
210             }
211             catch (SecurityException se)
212             {
213                 logger.severe(Failure.NO_ENV_ACCESS + " !");
214                 System.exit(Failure.NO_ENV_ACCESS.toInteger());
215             }
216         }
217     }
218
219     /**
220      * Lancement du ChatClient
221      */
222     @Override
223     protected void launch()
224     {
225         /*
226          * Create and Launch client
227          */
228         logger.info("Creating client to " + host + " at port " + port
229                 + " with verbose " + (verbose ? "on" : "off ... "));
230
231         Boolean commonRun;
232
233         if (gui)
234         {
235             if (System.getProperty("os.name").startsWith("Mac OS"))
236             {
237                 // Met en place le menu en haut de l'Ã©cran plutÃ´t que dans l'application
238                 System.setProperty("apple.laf.useScreenMenuBar", "true");
239                 System.setProperty("com.apple.mrj.application.apple.menu.about.name", "Name");
240             }
241
242             /*
243              * On a besoin d'un commonRun entre la frame et les ServerHandler
244              * et UserHandler du client crÃ©Ã© plus bas.
245              */
246             commonRun = Boolean.TRUE;
247
248             /*
249              * CrÃ©ation de la fenÃªtre de chat
250              * TODO Ã  customizer lorsrque vous aurez crÃ©Ã© la classe
251              * ClientFrame2
252              */
253             final AbstractClientFrame frame = new ClientFrame(name, host, commonRun, logger);
254
255             /*
256              * TODO CrÃ©ation du flux de sortie vers le GUI : userOut Ã  partir du
257              * flux d'entrÃ©e de la frame (ClientFrame#getInPipe())
258              *  - Creation d'un PipedOutputStream Ã  connecter sur
259              *  - le PipedInputStream de la frame
260              */
261             try
262             {
263                 // userOut = TODO Complete ...
264                 throw new IOException(); // TODO Remove when done
265             }
266             catch (IOException e)
267             {
268                 logger.severe(Failure.USER_OUTPUT_STREAM
269                         + " unable to get piped out stream");
270                 logger.severe(e.getLocalizedMessage());
```

```
271                 System.exit(Failure.USER_OUTPUT_STREAM.toInteger());
272             }
273
274             /*
275              * TODO CrÃ©ation du flux d'entrÃ©e depuis le GUI : userIn Ã  partir du
276              * flux de sortie de la frame (ClientFrame#getOutPipe())
277              *  - CrÃ©ation d'un PipedInputStream Ã  connecter sur
278              *  - le PipedOutputStream de la frame
279              */
280             try
281             {
282                 // userIn = TODO Complete ...
283                 throw new IOException(); // TODO Remove when done
284             }
285             catch (IOException e)
286             {
287                 logger.severe(Failure.USER_INPUT_STREAM
288                         + " unable to get user piped in stream");
289                 logger.severe(e.getLocalizedMessage());
290                 System.exit(Failure.USER_INPUT_STREAM.toInteger());
291             }
292
293             /*
294              * Insertion de la frame dans la file des Ã©vÃ¨nements GUI
295              * grÃ¢ce Ã  un Runnable anonyme
296              */
297             EventQueue.invokeLater(new Runnable()
298             {
299                 @Override
300                 public void run()
301                 {
302                     try
303                     {
304                         frame.pack();
305                         frame.setVisible(true);
306                     }
307                     catch (Exception e)
308                     {
309                         logger.severe(e.getLocalizedMessage());
310                     }
311                 }
312             });
313
314             /*
315              * CrÃ©ation et lancement du thread de la frame
316              */
317             Thread guiThread = new Thread(frame);
318             threadPool.add(guiThread);
319             guiThread.start();
320
321         }
322         else // client console
323         {
324             // lecture depuis la console
325             userIn = System.in;
326             // Ã©criture vers la console
327             userOut = System.out;
328             // On a pas besoin d'un commonRun avec le client console
329             commonRun = null;
330         }
331
332         /*
333          * Lancement du ChatClient
334          */
335         UserOutputType outType = UserOutputType.fromInteger(guiVersion);
336         ChatClient client = new ChatClient(host,        // hÃ´te du serveur
337                                            port,        // port tcp
338                                            name,        // nom d'utilisateur
339                                            userIn,      // entrÃ©es utilisateur
340                                            userOut,     // sorties utilisateur
341                                            outType,     // Type sortie utilisateur
342                                            commonRun,   // commonRun avec le GUI
343                                            logger);     // parent logger
344         if (client.isReady())
345         {
346             Thread clientThread = new Thread(client);
347             threadPool.add(clientThread);
348
349             clientThread.start();
350
351             logger.fine("client launched");
352
353             // attente de l'ensemble des threads du threadPool pour terminer
354             for (Thread t : threadPool)
355             {
356                 try
357                 {
358                     t.join();
359                     logger.fine("client thread end");
360                 }
```



```
361                 catch (InterruptedException e)
362                 {
363                     logger.severe("interrupted");
364                     logger.severe(e.getLocalizedMessage());
365                 }
366             }
367         }
368         else
369         {
370             logger.severe(Failure.CLIENT_NOT_READY + " abort ...");
371             System.exit(Failure.CLIENT_NOT_READY.toInteger());
372         }
373     }
374
375     /**
376      * Programme principal de lancement d'un client de chat
377      * @param args argument du programme
378      * <ul>
379      * <li>--host <host address> : set host to connect to</li>
380      * <li>--port <port number> : set host connection port</li>
381      * <li>--name <user name> : user name to use to connect</li>
382      * <li>--verbose : set verbose on</li>
383      * <li>--gui <1 or 2>: use graphical interface rather than console interface
384      * </li>
385      * </ul>
386      */
387     public static void main(String[] args)
388     {
389
390         RunChatClient client = new RunChatClient(args);
391
392         client.launch();
393     }
394 }
```

```java
import java.io.IOException;
import java.net.SocketException;

import chat.Failure;
import chat.server.ChatServer;

/**
 * Classe/programme qui lance un serveur de chat
 * @author davidroussel
 */
public class RunChatServer extends AbstractRunChat
{
    /**
     * Time out de la server socket avant qu'elle ne recommence Ã  attendre
     * des connections des Ã©ventuels clients
     */
    private int timeout;

    /**
     * Flag permettant (ou pas) de quitter le serveur lorsque le dernier
     * client se dÃ©logue
     */
    private boolean quitOnLastclient;

    /**
     * Default time out to wait for client connection : 5 seconds
     */
    public static final int DEFAULTTIMEOUT = 5000;

    /**
     * Constructeur d'un lanceur de serveur d'aprÃ¨s les arguments du programme
     * principal
     * @param args les arguments du programme principal
     */
    protected RunChatServer(String[] args)
    {
        super(args);
    }

    /**
     * Mise en place des attributs du serveur de chat en fonction des arguments
     * utilisÃ©s dans la ligne de commande
     * @param args les arguments fournis au programme principal.
     */
    @Override
    protected void setAttributes(String[] args)
    {
        /*
         * On met d'abord les attributs locaux Ã  leur valeur par dÃ©faut
         */
        timeout = DEFAULTTIMEOUT;
        quitOnLastclient = true;

        /*
         * parsing des arguments communs aux clients et serveur
         *  -v | --verbose
         *  -p | --port : port Ã  utiliser pour la serverSocket
         */
        super.setAttributes(args);

        /*
         * parsing des arguments spÃ©cifique au serveur
         *  -t | --timeout : timeout d'attente de la server socket
         */
        for (int i=0; i < args.length; i++)
        {
            if (args[i].equals("--timeout") ∨ args[i].equals("-t"))
            {
                if (i < (args.length - 1))
                {
                    // parse next arg for in port value
                    Integer timeInteger = readInt(args[++i]);
                    if (timeInteger ≠ null)
                    {
                        timeout = timeInteger.intValue();
                    }
                    logger.info("Setting timeout to " + timeout);
                }
                else
                {
                    logger.warning("invalid timeout value");
                }
            }
            if (args[i].equals("--quit") ∨ args[i].equals("-q"))
            {
                quitOnLastclient = true;
                logger.info("Setting quit on last client to true");
            }
            if (args[i].equals("--noquit") ∨ args[i].equals("-n"))
            {
```



```java
                quitOnLastclient = false;
                logger.info("Setting quit on last client to false");
            }
        }
    }

    /**
     * Lancement du serveur de chat
     */
    @Override
    protected void launch()
    {
        /*
         * Create and Launch server on local ip adress with port number and verbose
         * status
         */
        logger.info("Creating server on port " + port + " with timeout "
                + timeout + " ms and verbose " + (verbose ? "on" : "off"));

        ChatServer server = null;
        try
        {
            server = new ChatServer(port, timeout, quitOnLastclient, logger);
        }
        catch (SocketException se)
        {
            logger.severe(Failure.SET_SERVER_SOCKET_TIMEOUT + ", abort ...");
            logger.severe(se.getLocalizedMessage());
            System.exit(Failure.SET_SERVER_SOCKET_TIMEOUT.toInteger());
        }
        catch (IOException e)
        {
            logger.severe(Failure.CREATE_SERVER_SOCKET + ", abort ...");
            e.printStackTrace();
            System.exit(Failure.CREATE_SERVER_SOCKET.toInteger());
        }

        // Wait for serverThread to stop
        Thread serverThread = null;
        if (server ≠ null)
        {
            serverThread = new Thread(server);
            serverThread.start();

            logger.info("Waiting for server to terminate ... ");
            try
            {
                serverThread.join();
                logger.fine("Server terminated, program end.");
            }
            catch (InterruptedException e)
            {
                logger.severe("Server Thread Join interrupted");
                logger.severe(e.getLocalizedMessage());
            }
        }
    }

    /**
     * Programme principal
     * @param args les arguments
     * <ul>
     *  <li>--port <port number> : set host connection port</li>
     *  <li>--verbose : set verbose on</li>
     *  <li>--timeout <timeout in ms> : server socket waiting time out</li>
     * </ul>
     */
    public static void main(String[] args)
    {
        RunChatServer server = new RunChatServer(args);

        server.launch();
    }
}
```

```
1  package chat.client;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.io.OutputStream;
6  import java.io.PrintWriter;
7  import java.net.Socket;
8  import java.net.UnknownHostException;
9  import java.util.logging.Logger;
10
11 import chat.Failure;
12 import chat.UserOutputType;
13 import logger.LoggerFactory;
14
15 /**
16  * Classe Principale d'un client de chat.
17  * Instancie :
18  *  - la socket pour commnuniquer avec le serveur
19  *  - le UserHandler pour traiter les messages de l'utilisateur
20  *  - le ServerHandler pour traiter les messages du serveur
21  * @author davidroussel
22  */
23 public class ChatClient implements Runnable
24 {
25     /**
26      * Nom d'utilisateur utilisÃ© pour se connecter
27      */
28     private String userName;
29
30     /**
31      * Socket du client
32      */
33     private Socket clientSocket;
34
35     /**
36      * Flux d'entrÃ©e depuis le serveur
37      */
38     private InputStream serverIn;
39
40     /**
41      * Flux de sortie vers le serveur
42      */
43     private OutputStream serverOut;
44
45     /**
46      * Ecrivain vers le flux de sortie vers le serveur. UtilisÃ© temporairement
47      * pour envoyer notre nom d'utilisateur au serveur
48      */
49     private PrintWriter serverOutPW;
50
51     /**
52      * Flux d'entrÃ©e depuis l'utilisateur
53      */
54     private InputStream userIn;
55
56     /**
57      * Flux de sortie vers l'utilisateur
58      */
59     private OutputStream userOut;
60
61     /**
62      * Handler des donnÃ©es en provenance du serveur
63      *
64      * @uml.property name="serverHandler"
65      * @uml.associationEnd multiplicity="(1 1)" aggregation="composite"
66      */
67     private ServerHandler serverHandler = null;
68
69     /**
70      * Handler des donnÃ©es en provenance de l'utilisateur
71      *
72      * @uml.property name="userHandler"
73      * @uml.associationEnd multiplicity="(1 1)" aggregation="composite"
74      */
75     private UserHandler userHandler = null;
76
77     /**
78      * Etat d'exÃ©cution commun du {@link #userHandler} et du
79      * {@link #serverHandler}, lorsque l'un des deux Runnable se termine, il met
80      * commonRun Ã  faux ce qui force l'autre Ã  se terminer.
81      */
82     private Boolean commonRun;
83
84     /**
85      * Etat du client. true si la socket ainsi que les diffÃ©rents flux
86      * d'entrÃ©e/sortie ont Ã©tÃ© crÃ©Ã©s
87      *
88      * @uml.property name="ready"
89      */
90     private boolean ready;
```



```
91
92     /**
93      * Le logger utilisÃ© pour afficher les messages d'infos|erreurs|warnings
94      */
95     private Logger logger;
96
97     /**
98      * Constructeur d'un client de chat
99      *
100     * @param host l'adresse du serveur
101     * @param port le port Ã  utiliser pour communiquer avec le serveur
102     * @param name le nom d'utilisateur utilisÃ©
103     * @param in le flux d'entrÃ©e depuis l'utilisateur
104     * @param out le flux de sortie vers l'utilisateur
105     * @param outType le type de donnÃ©es attendues dans le flux de sortie vers
106     * le client (texte ou objets)
107     * @param l'Ã©tat d'exÃ©cution commun avec un autre runnable. ou bien null
108     *            s'il n'y a pas d'autre runnable Ã  synchroniser avec ceux
109     *            lancÃ©s dans le ChatClient
110     * @param verbose niveau de debug pour les messages
111     */
112    public ChatClient(String host,
113                      int port,
114                      String name,
115                      InputStream in,
116                      OutputStream out,
117                      UserOutputType outType,
118                      Boolean commonRun,
119                      Logger parentLogger)
120    {
121        userName = name;
122        ready = false;
123
124        // CrÃ©ation du logger
125        logger = LoggerFactory.getParentLogger(getClass(),
126                                               parentLogger,
127                                               parentLogger.getLevel());
128
129        /*
130         * TODO CrÃ©ation de la socket vers host/port
131         */
132        clientSocket = null;
133        try
134        {
135            clientSocket = new Socket(host, port);
136            logger.info("ChatClient: socket created");
137        }
138        catch (UnknownHostException e)
139        {
140            /*
141             * TODO Notez bien cette faÃ§on de faire, vous devrez la reproduire
142             * par la suite
143             */
144            logger.severe("ChatClient: " + Failure.UNKNOWN_HOST + ": " + host);
145            logger.severe(e.getLocalizedMessage());
146            System.exit(Failure.UNKNOWN_HOST.toInteger());
147        }
148        catch (IOException e)
149        {
150            logger.severe("ChatClient: " + Failure.CLIENT_CONNECTION
151                    + " to: \"" + host + "\" at port \"" + port + "\"");
152            logger.severe(e.getLocalizedMessage());
153            System.exit(Failure.CLIENT_CONNECTION.toInteger());
154        }
155
156        /*
157         * TODO Obtention du flux de sortie vers le serveur (serverOut) Ã  partir
158         * de la clientSocket.
159         * avec utilisation du logger pour afficher la progression ou les erreurs
160         *  - logger.info("ChatClient: got client output stream to server"); si le serverOut est non
 null
161         *  - logger.severe("ChatClient: null server out" + Failure.CLIENT_INPUT_STREAM); si le serv
erOut est null
162         *  - logger.severe("ChatClient: " + Failure.CLIENT_OUTPUT_STREAM); si une IOException survi
ent
163         * les "severe" doivent Ãªtre suivi d'un System.exit(...) comme ci-dessus;
164         */
165        serverOut = null;
166        try
167        {
168            // TODO serverOut = ...
169            if (serverOut ≠ null)
170            {
171                logger.info("ChatClient: got client output stream to server");
172            }
173            else
174            {
175                logger.severe("ChatClient: null server out" + Failure.CLIENT_INPUT_STREAM);
176                System.exit(Failure.CLIENT_OUTPUT_STREAM.toInteger());
177            }
```

```
178             throw new IOException(); // TODO Remove this line when serverOut is obtained
179         }
180         catch (IOException e)
181         {
182             logger.severe("ChatClient: " + Failure.CLIENT_OUTPUT_STREAM);
183             logger.severe(e.getLocalizedMessage());
184             System.exit(Failure.CLIENT_OUTPUT_STREAM.toInteger());
185         }
186 
187         /*
188          * TODO CrÃ©ation PrintWriter temporaire sur le serverOut
189          * (avec autoFlush): serverOutPW
190          * et envoi de notre nom d'utilisateur au serveur (avec un println)
191          * afin qu'il puisse crÃ©er un thread dÃ©diÃ© Ã  notre traitement
192          * ajout d'un message d'info au logger pour la crÃ©ation du serverOutPW
193          * et d'un warning si celui ci a des erreurs aprÃ¨s l'envoi du nom au
194          * serveur.
195          */
196         if (serverOut ≠ null)
197         {
198             // serverOutPW = // TODO Complete ...
199             logger.info("ChatClient: sending name to server ... ");
200 
201             serverOutPW.println(userName);
202             if (serverOutPW.checkError())
203             {
204                 logger.warning("ChatClient: serverOutPw has errors");
205             }
206         }
207 
208         /*
209          * TODO Obtention du flux d'entrÃ©e depuis le serveur (serverIn) Ã  partir
210          * de la clientSocket.
211          * Si une IOException
212          *  - ajout d'un "severe" au logger avec Failure.CLIENT_INPUT_STREAM
213          *  - System.exit(...);
214          */
215         serverIn = null;
216         try
217         {
218             // TODO serverIn = ...
219             throw new IOException(); // TODO Remove this line when serverIn is obtained
220         }
221         catch (IOException e)
222         {
223             logger.severe("ChatClient: " + Failure.CLIENT_INPUT_STREAM);
224             logger.severe(e.getLocalizedMessage());
225             System.exit(Failure.CLIENT_INPUT_STREAM.toInteger());
226         }
227 
228         // obtention des flux de l'utilisateur
229         userIn = in;
230         userOut = out;
231 
232         // Etat d'exÃ©cution commun
233         if (commonRun ≡ null)
234         {
235             this.commonRun = new Boolean(true);
236         }
237         else
238         {
239             this.commonRun = commonRun;
240         }
241 
242         // CrÃ©ation du user handler
243         userHandler = new UserHandler(userIn,
244                                       serverOut,
245                                       this.commonRun,
246                                       logger);
247 
248         // crÃ©ation du server handler
249         serverHandler = new ServerHandler(userName,
250                                           serverIn,
251                                           userOut,
252                                           outType,
253                                           this.commonRun,
254                                           logger);
255 
256         ready = true;
257     }
258 
259     /**
260      * AccÃ¨s en lecture de l'Ã©tat du client
261      *
262      * @return the ready
263      * @uml.property name="ready"
264      */
265     public boolean isReady()
266     {
267         return ready;
```



```
268     }
269 
270     /*
271      * (non-Javadoc)
272      * @see java.lang.Runnable#run()
273      */
274     @Override
275     public void run()
276     {
277         /*
278          * Tant que ce que l'on lit depuis l'utilisateur n'est pas null (avec un
279          * ctrl-D par exemple), on envoie ce que l'on a lu au serveur et on
280          * attends que celui ci nous rÃ©ponde pour afficher ce qu'il nous envoie.
281          * On a donc deux boucles d'attente : d'une part l'utilisateur, d'autre
282          * part le serveur. Chaque boucle est donc traitÃ©e dans son propre
283          * thread UserHandler traite les entrÃ©es de l'utilisateur ServerHandler
284          * traite les entrÃ©es du serveur et on attends la fin des deux threads
285          * pour terminer le client Les deux threads partagent une variable
286          * "commonRun" lorsque l'un des deux threads se termine il met cette
287          * vatiable Ã  false. A chaque tour de boucle de chacun des threads ils
288          * consultent (de maniÃ¨re atomique) cette variable afin de savoir s'ils
289          * peuvent continuer
290          */
291 
292         Thread[] threads = new Thread[2];
293 
294         // CrÃ©ation du thread du UserHandler
295         threads[0] = new Thread(userHandler);
296 
297         // CrÃ©ation du thread du ServerHandler
298         threads[1] = new Thread(serverHandler);
299 
300         // Lancement des threads
301         for (int i = 0; i < threads.length; i++)
302         {
303             threads[i].start();
304         }
305 
306         // Attente de la fin des 2 threads
307         for (int i = 0; i < threads.length; i++)
308         {
309             try
310             {
311                 threads[i].join();
312             }
313             catch (InterruptedException e)
314             {
315                 logger.warning("Join thread " + i + " interrupted");
316             }
317         }
318 
319         logger.info("ChatClient: All threads terminated");
320 
321         cleanup();
322     }
323 
324     /**
325      * Nettoyage du client : fermeture des flux d'entrÃ©e/sortie et fermeture de
326      * la socket
327      */
328     public void cleanup()
329     {
330         // Cleanup du #userHandler
331         userHandler.cleanup();
332 
333         // Cleanup du #serverHandler
334         serverHandler.cleanup();
335 
336         // fermeture du flux temporaire de sortie vers le serveur
337         logger.info("ChatClient: closing server output stream ... ");
338         serverOutPW.close();
339 
340         // fermeture de la socket
341         logger.info("ChatClient: closing client socket ... ");
342         try
343         {
344             clientSocket.close();
345         }
346         catch (IOException e)
347         {
348             logger.severe("ChatClient: closing client socket failed");
349             logger.severe(e.getLocalizedMessage());
350         }
351     }
352 }
```

```java
package chat.client;

/**
 * Sous-package contenant les classes relative Ã  la partie client du
 * client/serveur de chat
 */
```



```java
package chat.client;

import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.logging.Logger;

import chat.Failure;
import chat.UserOutputType;
import logger.LoggerFactory;
import models.Message;

/**
 * Server Handler. Classe s'occupant de lire le flux de messages en provenance
 * du serveur et de le transmettre sur le flux de sortie du client.
 * Un client peut accepter soit
 *  - du texte uniquement (c'est le cas du client console et du 1er client GUI)
 *  - des messages (comme ceux envoyÃ©s par le serveur) Ã  travers un ObjectStream
 *
 * @author davidroussel
 */
class ServerHandler implements Runnable
{
    /**
     * Flux d'entrÃ©e objet en provenance du serveur
     */
    private ObjectInputStream serverInOS;

    /**
     * Le type de flux Ã  utiliser pour envoyer les message au client.
     * Si le type de flux est {@link TEX}
     */
    private UserOutputType userOutType;

    /**
     * Ecrivain vers le flux de sortie texte vers l'utilisateur
     */
    private PrintWriter userOutPW;

    /**
     * Flux de sortie objet vers l'utilisateur
     */
    private ObjectOutputStream userOutOS;

    /**
     * Etat d'exÃ©cution commun du ServerHandler et du {@link UserHandler}
     */
    private Boolean commonRun;

    /**
     * Logger utilisÃ© pour afficher (ou pas) les messages d'erreurs
     */
    private Logger logger;

    /**
     * Constructeur d'un ServerHandler
     * @param name notre nom d'utilisateur sur le serveur
     * @param in le flux d'entrÃ©e en provenance du serveur
     * @param out le flux de sortie vers l'utilisateur
     * @param commonRun l'Ã©tat d'exÃ©cution commun du {@link ServerHandler} et du
     *            {@link UserHandler}
     * @param parentLogger logger parent pour affichage des messages de debug
     */
    public ServerHandler(String name,
                         InputStream in,
                         OutputStream out,
                         UserOutputType outType,
                         Boolean commonRun,
                         Logger parentLogger)
    {
        logger = LoggerFactory.getParentLogger(getClass(),
                                               parentLogger,
                                               parentLogger.getLevel());
        /*
         * On vÃ©rifie que l'InputStream est non null et on crÃ©e notre serverInOS
         * sur cet InputStream Sinon on quitte avec la valeur
         * Failure.CLIENT_INPUT_STREAM
         */
        if (in ≠ null)
        {
            logger.info("ServerHandler: creating server input reader ... ");
            /*
             * TODO CrÃ©ation du ObjectInputStream Ã  partir du flux d'entrÃ©e
             * en provenance du serveur, si une IOException survient,
             * on quitte avec la valeur Failure.CLIENT_INPUT_STREAM
             */
            serverInOS = null;
```

```
91          }
92          else
93          {
94              logger.severe("ServerHandler: " + Failure.CLIENT_INPUT_STREAM);
95              System.exit(Failure.CLIENT_INPUT_STREAM.toInteger());
96          }
97
98          /*
99           * On vÃ©rifie que l'OutputStream est non null et on crÃ©e notre userOutPW
100          * ou bien notre userOutOS sur cet OutputStream. Sinon on quitte avec
101          * la valeur Failure.USER_OUTPUT_STREAM
102          */
103         if (out ≠ null)
104         {
105             logger.info("ServerHandler: creating user output ... ");
106             /*
107              * TODO En fonction du outType, crÃ©ation d'un PrintWriter sur le
108              * flux de sortie vers l'utilisateur, ou bien d'un ObjectOutputStream
109              */
110             userOutType = outType;
111             switch (userOutType)
112             {
113                 case OBJECT:
114                     userOutPW = null;
115                     // userOutOS = TODO Complete ...
116                     break;
117                 case TEXT:
118                 default:
119                     userOutOS = null;
120                     // userOutPW = TODO Complete ...
121                     break;
122             }
123         }
124         else
125         {
126             logger.severe("ServerHandler: " + Failure.USER_OUTPUT_STREAM);
127             System.exit(Failure.USER_OUTPUT_STREAM.toInteger());
128         }
129
130         /*
131          * On vÃ©rifie que le commonRun passÃ© en argument est non null avant de
132          * le copier dans notre commonRun. Sinon on quitte avec la valeur
133          * Failure.OTHER
134          */
135         if (commonRun ≠ null)
136         {
137             this.commonRun = commonRun;
138         }
139         else
140         {
141             logger.severe("ServerHandler: null common run " + Failure.OTHER);
142             System.exit(Failure.OTHER.toInteger());
143         }
144     }
145
146     /**
147      * ExÃ©cution d'un ServerHandler. Ã‚-^Icoute les entrÃ©es en provenance du serveur
148      * et les envoient sur la sortie vers l'utilisateur
149      *
150      * @see java.lang.Runnable#run()
151      */
152     @Override
153     public void run()
154     {
155         /*
156          * Boucle principale de lecture des messages en provenance du serveur:
157          * tantque commonRun est vrai on lit une ligne depuis le serverInBR dans
158          * serverInput Si cette ligne est non nulle, on l'envoie dans le
159          * userOutPW Toute erreur ou exception dans cette boucle nous fait
160          * quitter cette boucle A la fin de la boucle on passe le commonRun Ã 
161          * false de maniÃ¨re synchronisÃ©e (atomique) afin que le UserHandler
162          * s'arrÃªte aussi.
163          */
164         while (commonRun.booleanValue())
165         {
166             /*
167              * TODO lecture d'un message du serveur avec le serverInOS
168              * Si une Exception intervient
169              *  - Ajout d'un warning au logger
170              *  - on quitte la boucle while (commonRun...
171              */
172             Message message = null;
173
174             if ((message ≠ null))
175             {
176                 /*
177                  * TODO Affichage du message vers l'utilisateur avec
178                  *  - le userOutPW si le client attends du texte
179                  *  - le userOutOS si le client attends des objet (des Message)
180                  * vÃ©rification de l'Ã©tat d'erreur du userOutPW
```



```
181                  * avec ajout d'un warning au logger si c'est le cas
182                  */
183                 boolean error = false;
184                 switch (userOutType)
185                 {
186                     case OBJECT:
187                         // TODO userOutOS...
188                         error = true;
189                         break; // Break this switch
190                     case TEXT:
191                     default:
192                         // TODO userOutPW...
193                         error = true;
194                         break;
195                 }
196                 if (error)
197                 {
198                     break; // break this loop
199                 }
200             }
201             else
202             {
203                 logger.warning("ServerHandler: null input read");
204                 break;
205             }
206         }
207
208         if (commonRun.booleanValue())
209         {
210             logger.info("ServerHandler: changing run state at the end ... ");
211
212             synchronized (commonRun)
213             {
214                 commonRun = Boolean.FALSE;
215             }
216         }
217     }
218
219     /**
220      * Fermeture des flux
221      */
222     public void cleanup()
223     {
224         logger.info("ServerHandler: closing server input stream reader ... ");
225         /*
226          * fermeture du lecteur de flux d'entrÃ©e du serveur Si une
227          * IOException intervient ajout d'un severe au logger
228          */
229         try
230         {
231             serverInOS.close();
232         }
233         catch (IOException e)
234         {
235             logger.severe("ServerHandler: closing server input stream reader failed: " +
236                     e.getLocalizedMessage());
237         }
238
239         logger.info("ServerHandler: closing user output print writer ... ");
240
241         /*
242          * fermeture des flux de sortie vers l'utilisateur (si != null)
243          * Si une exception intervient, ajout d'un severe au logger
244          */
245         if (userOutPW ≠ null)
246         {
247             userOutPW.close();
248
249             if (userOutPW.checkError())
250             {
251                 logger.severe("ServerHandler: closed user text output has errors: ");
252             }
253         }
254
255         if (userOutOS ≠ null)
256         {
257             try
258             {
259                 userOutOS.close();
260             }
261             catch (IOException e)
262             {
263                 logger.severe("ServerHandler: closing user object output stream failed: "
264                         + e.getLocalizedMessage());
265             }
266         }
267     }
268 }
```

```java
package chat.client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.logging.Logger;

import chat.Failure;
import logger.LoggerFactory;

/**
 * User Handler Classe s'occupant de rÃ©cupÃ©rer ce que tape l'utilisateur et de
 * l'envoyer au serveur de chat
 *
 * @author davidroussel
 */
class UserHandler implements Runnable
{
    /**
     * Lecteur du flux d'entrÃ©e depuis l'utilisateur
     */
    private BufferedReader userInBR;

    /**
     * Ecrivain vers le flux de sortie vers le serveur
     */
    private PrintWriter serverOutPW;

    /**
     * Etat d'exÃ©cution commun du UserHandler et du {@link ServerHandler}
     */
    private Boolean commonRun;

    /**
     * Logger utilisÃ© pour afficher (ou pas) les messafes d'erreurs
     */
    private Logger logger;

    /**
     * Constructeur d'un UserHandler
     *
     * @param in Le flux d'entrÃ©e de l'utilisateur pour les entrÃ©es utilisateur
     * @param out le flux de sortie vers le serveur
     * @param commonRun l'Ã©tat d'exÃ©cution commun du {@link UserHandler} et du
     *            {@link ServerHandler}
     * @param parentLogger le logger parent
     */
    public UserHandler(InputStream in, OutputStream out, Boolean commonRun,
            Logger parentLogger)
    {
        logger = LoggerFactory.getParentLogger(getClass(), parentLogger,
                parentLogger.getLevel());

        /*
         * CrÃ©ation du lecteur de flux d'entrÃ©e de l'utilisateur : userInBR sur
         * l'InputStream in si celui ci est non null. Sinon on quitte avec la
         * valeur Failure.USER_INPUT_STREAM
         */
        if (in ≠ null)
        {
            logger.info("UserHandler: creating user input buffered reader ... ");

            /*
             * TODO CrÃ©ation du BufferedReader sur un InputStreamReader Ã  partir
             * du flux d'entrÃ©e en provenance de l'utilisateur
             */
            // userInBR = TODO Complete ...
        }
        else
        {
            logger.severe("UserHandler: null input stream"
                    + Failure.USER_INPUT_STREAM);
            System.exit(Failure.USER_INPUT_STREAM.toInteger());
        }

        /*
         * CrÃ©ation de l'Ã©crivain vers le flux de sortie vers le serveur :
         * serverOutPW sur l'OutputStream out si celui ci est non null. Sinon,
         * on quitte avec la valeur Failure.CLIENT_OUTPUT_STREAM
         */
        if (out ≠ null)
        {
            logger.info("UserHandler: creating server output print writer ... ");

            /*
             * TODO CrÃ©ation du PrintWriter sur le flux de sortie vers le
             * serveur (en mode autoflush)
             */
```



```java
            // serverOutPW = TODO Complete ...
        }
        else
        {
            logger.severe("UserHandler: null output stream"
                    + Failure.CLIENT_OUTPUT_STREAM);
            System.exit(Failure.CLIENT_OUTPUT_STREAM.toInteger());
        }

        /*
         * On vÃ©rifie que le commonRun passÃ© en argument est non null avant de
         * le copier dans notre commonRun. Sinon on quitte avec la valeur
         * Failure.OTHER
         */
        if (commonRun ≠ null)
        {
            this.commonRun = commonRun;
        }
        else
        {
            logger.severe("ServerHandler: null common run " + Failure.OTHER);
            System.exit(Failure.OTHER.toInteger());
        }
    }

    /**
     * ExÃ©ction d'un UserHandler. Ã‰coute les entrÃ©es en provenance de
     * l'utilisateur et les envoie dans le flux de sortie vers le serveur
     *
     * @see java.lang.Runnable#run()
     */
    @Override
    public void run()
    {
        String userInput = null;

        /*
         * Boucle principale de lecture des messages en provenance de
         * l'utilisateur. tantque commonRun est vrai on lit une ligne depuis le
         * userInBR dans userInput Si cette ligne est non nulle, on l'envoie
         * dans serverOutPW
         */
        while (commonRun.booleanValue())
        {
            /*
             * TODO Lecture d'une ligne en provenance de l'utilisateur grÃ¢ce
             * au userInBR. Si une IOException intervient - Ajout d'un
             * severe au logger - On quitte la boucle
             */
            // userInput = TODO Complete ...

            if (userInput ≠ null)
            {
                /*
                 * TODO Envoi du texte au serveur grÃ¢ce au serverOutPW et
                 * vÃ©rification de l'Ã©tat d'erreur du serverOutPW avec ajout
                 * d'un warning au logger et break si c'est le cas.
                 */
                // TODO serverOutPW...

                /*
                 * TODO Si la commande Vocabulary.byeCmd a Ã©tÃ© tapÃ©e par
                 * l'utilisateur on quitte la boucle
                 */
            }
            else
            {
                logger.warning("UserHandler: null user input");
                break;
            }
        }

        if (commonRun.booleanValue())
        {
            logger.info("UserHandler: changing run state at the end ... ");

            synchronized (commonRun)
            {
                commonRun = Boolean.FALSE;
            }
        }
    }

    /**
     * Fermeture des flux
     */
    public void cleanup()
    {
        logger.info("UserHandler: closing user input stream reader ... ");
        /*
```

```
181          * fermeture du lecteur de flux d'entrÃ©e de l'utilisateur Si une
182          * IOException intervient : - Ajout d'un severe au logger
183          */
184         try
185         {
186             userInBR.close();
187         }
188         catch (IOException e)
189         {
190             logger.severe("UserHandler: closing server input stream reader failed");
191             logger.severe(e.getLocalizedMessage());
192         }
193 
194         logger.info("UserHandler: closing server output print writer ... ");
195         // fermeture de l'Ã©crivain vers le flux de sortie vers le serveur
196         serverOutPW.close();
197     }
198 }
```



```
1  package chat;
2  
3  import java.io.BufferedReader;
4  import java.io.InputStreamReader;
5  import java.io.ObjectOutputStream;
6  import java.io.PipedInputStream;
7  import java.io.PipedOutputStream;
8  
9  /**
10  * EnumÃ©ration de toutes les erreurs possibles dans le systÃ¨me client/serveur de
11  * chat.
12  *
13  * @author davidroussel
14  */
15 public enum Failure
16 {
17     /**
18      * Unable to get local host IP address
19      */
20     NO_LOCAL_HOST,
21     /**
22      * Invalid port, usr ports should be > 1024
23      */
24     INVALID_PORT,
25     /**
26      * Unable to determin log name or user name
27      */
28     NO_USER_NAME,
29     /**
30      * Unable to access system env to get user or log names
31      */
32     NO_ENV_ACCESS,
33     /**
34      * Unable to set timeout on server
35      */
36     SET_SERVER_SOCKET_TIMEOUT,
37     /**
38      * Unable to create server socket
39      */
40     CREATE_SERVER_SOCKET,
41     /**
42      * @uml.property name="uNKNOWN_HOST"
43      * @uml.associationEnd
44      */
45     UNKNOWN_HOST,
46     /**
47      * Unable to create client socket to host
48      */
49     CLIENT_CONNECTION,
50     /**
51      * Unable to obtain input stream from server on client
52      * OR to obtain input stream from client on server
53      */
54     CLIENT_INPUT_STREAM,
55     /**
56      * Unable to obtain output stream to server on client
57      * OR to obtain temporary print writer to server on client
58      * OR to obtain temporary print writer to client in server
59      * @uml.property name="cLIENT_OUTPUT_STREAM"
60      * @uml.associationEnd
61      */
62     CLIENT_OUTPUT_STREAM,
63     /**
64      * Unable to create {@link PipedInputStream} from qui client Out Pipe
65      * OR unable to create {@link BufferedReader} on {@link InputStreamReader}
66      * from user
67      */
68     USER_INPUT_STREAM,
69     /**
70      * Unable to create {@link PipedOutputStream} to GUI client In Pipe
71      * OR Unable to create {@link ObjectOutputStream} to user in client
72      */
73     USER_OUTPUT_STREAM,
74     /**
75      * Unable to accept new connection from client in server
76      */
77     SERVER_CONNECTION,
78     /**
79      * Not used yet
80      */
81     SERVER_INPUT_STREAM,
82     /**
83      * Not used yet
84      */
85     SERVER_OUTPUT_STREAM,
86     /**
87      * @uml.property name="nO_NAME_CLIENT"
88      * @uml.associationEnd
89      */
90     NO_NAME_CLIENT,
```

```
91      /**
92       * GUI Client lauch failed
93       */
94      CLIENT_NOT_READY,
95      /**
96       * Other
97       */
98      OTHER;
99
100     /**
101      * Affichage sous forme de texte des erreurs possibles
102      */
103     @Override
104     public String toString()
105     {
106         switch (this)
107         {
108         // RunChatClient Failures (3)
109             case NO_LOCAL_HOST:
110                 return new String("Unable to get local host name");
111             case INVALID_PORT:
112                 return new String("Port number should be > 1024");
113             case NO_USER_NAME:
114                 return new String("Empty user name");
115             case NO_ENV_ACCESS:
116                 return new String(
117                         "System does not allow access to environment variables");
118                 // RunChatServer Failures (2)
119             case SET_SERVER_SOCKET_TIMEOUT:
120                 return new String("Unable to set Server socket timeout");
121             case CREATE_SERVER_SOCKET:
122                 return new String("Unable to create Server socket");
123                 // Chat Client (4)
124             case UNKNOWN_HOST:
125                 return new String("Unkown host");
126             case CLIENT_CONNECTION:
127                 return new String("Couldn't get I/O for connection to host");
128             case CLIENT_INPUT_STREAM:
129                 return new String("Could not get intput stream from client");
130             case CLIENT_OUTPUT_STREAM:
131                 return new String("Could not get output stream to client");
132                 // ServerHandler (2)
133             case USER_INPUT_STREAM:
134                 return new String("Could not get input stream from user");
135                 // ServerHandler
136             case USER_OUTPUT_STREAM:
137                 return new String("Could not get output stream to user");
138                 // ChatServer#run (3)
139             case SERVER_CONNECTION:
140                 return new String("Client connection to server failed");
141             case SERVER_INPUT_STREAM:
142                 return new String("could not get input stream from server");
143             case SERVER_OUTPUT_STREAM:
144                 return new String("could not get output stream to server");
145             case NO_NAME_CLIENT:
146                 return new String("Unable to read client's name");
147                 // Client (1)
148             case CLIENT_NOT_READY:
149                 return new String("Main Client not ready");
150             case OTHER:
151                 return new String("Other cause");
152         }
153         throw new AssertionError("Failure: unknown op: " + this);
154     }
155
156     /**
157      * Conversion en entier du type d'erreur
158      *
159      * @return le numÃ©ro de l'erreur
160      * @code System.exit(Failure.CLIENT_NOT_READY.toInteger());
161      * @endcode
162      */
163     public int toInteger()
164     {
165         return ordinal() + 1;
166     }
167
168 }
```



```
1  package chat;
2
3  /**
4   * Package contenant les parties client et serveur d'un serveur de Chat ainsi
5   * que le vocabulaire de commandes spÃ©ciales et un enum de toutes les causes
6   * possible d'Ã©checs des programmes. Un serveur de chat permet Ã  plusieurs
7   * clients de se connecter au serveur et chaque ligne envoyÃ©e d'un client est
8   * rÃ©pÃ©tÃ©e Ã  l'ensemble des client prÃ©cÃ©dÃ©e par l'identifiant du client qui l'a
9   * envoyÃ©e.
10  */
```

```java
package chat.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.util.Vector;
import java.util.logging.Logger;

import chat.Failure;
import logger.LoggerFactory;

/**
 * Classe du serveur de chat Chaque message de chaque client doit Ãªtre renvoyÃ© Ã 
 * tous autres clients
 *
 * @author davidroussel
 */
public class ChatServer implements Runnable
{
    /**
     * La socket serveur
     */
    private ServerSocket serverSocket;

    /**
     * Le port par dÃ©faut utilisÃ©
     */
    public final static int DEFAULTPORT = 1394;

    /**
     * Temps d'attente (en ms) par dÃ©faut d'une connection d'un client. Au bout
     * de ce temps une {@link SocketTimeoutException} est qÃ©nÃ©rÃ©e et on peut
     * choisir de recommencer Ã  attendre (s'il reste des clients) ou bien
     * arrÃªter le serveur (s'il n'y a plus de clients)
     */
    public final static int DEFAULTTIMEOUT = 1000;

    /**
     * La liste des diffÃ©rents clients. Un client est constituÃ© :
     * <ul>
     * <li>d'une {@link Socket}</li>
     * <li>d'un nom : {@link String}</li>
     * <li>d'un flux d'entrÃ©e : {@link BufferedReader}</li>
     * <li>d'un flux de sortie {@link PrintWriter}</li>
     * </ul>
     * Cette liste devra Ãªtre accÃ©dÃ©e de maniÃ¨re synchrone par les diffÃ©rents
     * threads traitant les diffÃ©rents clients.
     *
     * @uml.property name="clients"
     * @uml.associationEnd multiplicity="(0 -1)" ordering="true"
     *                     aggregation="composite"
     *                     inverse="chatServer:chat.server.InputOutputClient"
     */
    private Vector<InputOutputClient> clients;

    /**
     * Liste des handlers de chaque client
     * @uml.property name="handlers"
     * @uml.associationEnd multiplicity="(0 -1)" ordering="true"
     *                     aggregation="composite"
     *                     inverse="chatServer:chat.server.ClientHandler"
     */
    private Vector<ClientHandler> handlers;

    /**
     * logger pour afficher les messages d'erreur
     */
    private Logger logger;

    /**
     * Etat d'Ã©coute du serveur. Cet Ã©tat est vrai au dÃ©part et passe Ã  false
     * lorsque le dernier client se dÃ©connecte.
     */
    private boolean listening;

    /**
     * Termine le serveur lorsque le dernier client se dÃ©logue
     */
    private final boolean quitOnLastClient;

    /**
     * Constructeur valuÃ© d'un serveur de chat. Celui ci initialise la
     * {@link ServerSocket},
     *
     * @param port le port sur lequel on Ã©coute les requÃ¨tes
     * @param verbose affiche les messages de dÃ©bug ou pas
```



```java
     * @param timeout temps d'attente de connection d'un client
     * @param quitOnLastClient quitte le serveur lorsque le dernier client
     * se dÃ©loque
     * @param parentLogger logger parent pour l'affichage des messages de
     * debug
     * @throws IOException Si une erreur intervient lors de la crÃ©ation de la
     *             {@link ServerSocket}
     */
    public ChatServer(int port,
                      int timeout,
                      boolean quitOnLastClient,
                      Logger parentLogger)
        throws IOException
    {
        this.quitOnLastClient = quitOnLastClient;
        logger = LoggerFactory.getParentLogger(getClass(),
                                               parentLogger,
                                               parentLogger.getLevel());

        logger.info("ChatServer::ChatServer(port = " + port + ", timeout = "
            + timeout + ", quit = " + (quitOnLastClient ? "true" : "false")
            + ")");

        serverSocket = new ServerSocket(port);
        if (serverSocket ≠ null)
        {
            serverSocket.setSoTimeout(timeout);
        }

        clients = new Vector<InputOutputClient>();
        handlers = new Vector<ClientHandler>();
    }

    /**
     * Constructeur valuÃ© d'un serveur de chat. Celui ci initialise la
     * {@link ServerSocket},
     *
     * @param port le port sur lequel on Ã©coute les requÃ¨tes
     * @param verbose affiche les messages de dÃ©bug ou pas
     * @param parentLogger logger parent pour l'affichage de messages de debug
     * @throws IOException Si une erreur intervient lors de la crÃ©ation de la
     *             {@link ServerSocket}
     */
    public ChatServer(int port, Logger parentLogger) throws IOException
    {
        this(port, DEFAULTTIMEOUT, true, parentLogger);
    }

    /**
     * Constructeur par dÃ©faut d'un serveur de chat. Celui ci initialise la
     * @param parentLogger logger parent pour l'affichage des messages de
     * debug
     * {@link ServerSocket}, Le port utilisÃ© par dÃ©faut est dÃ©fini par
     * {@link #DEFAULTPORT}
     *
     * @throws IOException Si une erreur intervient lors de la crÃ©ation de la
     *             {@link ServerSocket}
     * @see #DEFAULTPORT
     */
    public ChatServer(Logger parentLogger) throws IOException
    {
        this(DEFAULTPORT, parentLogger);
    }

    /**
     * Accesseur en lecture du {@link #quitOnLastClient}
     * @return la valeu du {@link #quitOnLastClient}
     */
    public boolean isQuitOnLastClient()
    {
        return quitOnLastClient;
    }

    /**
     * Change l'Ã©tat d'Ã©coute du serveur
     * @param value la nouvelle valeur
     */
    public synchronized void setListening(boolean value)
    {
        listening = value;
    }

    /**
     * ExÃ©cution du serveur de chat : - On attend la connection d'un client -
     * Lorsque celle ci se produit le client est traitÃ© dans un nouveau thread -
     * Lorsqu'un client envoie un message au serveur, celui ci le rediffuse Ã 
     * l'ensemble des autres clients
     *
     * @see java.lang.Runnable#run()
     */
```

```
181     @Override
182     public void run()
183     {
184         Vector<Thread> handlerThreads = new Vector<Thread>();
185         listening = true;
186
187         while (listening)
188         {
189             Socket clientSocket = null;
190             String clientName = null;
191
192             // acceptation de la socket du client
193             try
194             {
195                 // on attends ici une connection d'un nouveau client
196                 clientSocket = serverSocket.accept(); // --> IOException
197                 logger.fine("ChatServer: client connection accepted");
198
199             }
200             catch (SocketTimeoutException ste)
201             {
202                 // on re-attends
203                 logger.info("Socket timeout, rewaiting ...");
204                 continue;
205             }
206             catch (IOException e)
207             {
208
209                 logger.severe(Failure.SERVER_CONNECTION.toString()
210                     + ":" + e.getLocalizedMessage());
211                 System.exit(Failure.SERVER_CONNECTION.toInteger());
212             }
213
214             if (clientSocket ≠ null)
215             {
216                 // rÃ©cupÃ©ration du nom du client
217                 BufferedReader reader = null;
218                 logger.info("ChatServer: Creatingc client input stream to get client's name ... ");
219                 try
220                 {
221                     reader = new BufferedReader(new InputStreamReader(
222                             clientSocket.getInputStream()));
223                 }
224                 catch (IOException e1)
225                 {
226                     logger.severe("ChatServer: " + Failure.CLIENT_INPUT_STREAM);
227                     logger.severe(e1.getLocalizedMessage());
228                     System.exit(Failure.CLIENT_INPUT_STREAM.toInteger());
229                 }
230                 if (reader ≠ null)
231                 {
232                     logger.info("ChatServer: reading client's name: ");
233                     try
234                     {
235                         // Lecture du nom du client
236                         clientName = reader.readLine();
237                         logger.info("ChatServer: client name " + clientName);
238                     }
239                     catch (IOException e)
240                     {
241                         logger.severe("ChatServer: "+ Failure.NO_NAME_CLIENT);
242                         logger.severe(e.getLocalizedMessage());
243                         System.exit(Failure.NO_NAME_CLIENT.toInteger());
244                     }
245
246                     /*
247                      * On ne doit PAS fermer le client input stream car cela
248                      * revient Ã  fermer la socket
249                      */
250                 }
251
252                 // Avant d'enregister cette connection dans l'ensemble des
253                 // clients il faut vÃ©rifier qu'aucun client ne porte le mÃªme
254                 // nom
255                 if (searchClientByName(clientName) ≡ null)
256                 {
257                     // CrÃ©ation d'un nouveau client
258                     InputOutputClient newClient =
259                             new InputOutputClient(clientSocket,
260                                                   clientName,
261                                                   logger);
262
263                     // Ajout du nouveau client Ã  la liste des clients.
264                     synchronized (clients)
265                     {
266                         clients.add(newClient);
267                     }
268
269                     // CrÃ©ation et lancement d'un handler pour ce client
270                     ClientHandler handler = new ClientHandler(this,
```



```
271                                                   newClient,
272                                                   clients,
273                                                   logger);
274                     handlers.add(handler);
275                     Thread handlerThread = new Thread(handler);
276                     handlerThread.start();
277                     handlerThreads.add(handlerThread);
278                 }
279                 else // un client avec ce nom existe dÃ©jÃ 
280                 {
281                     // on notifie au client qu'il est refusÃ©
282                     try
283                     {
284                         PrintWriter out = new PrintWriter(
285                                 clientSocket.getOutputStream(), true);
286                         out.println("server > Sorry another client already use the name "
287                                 + clientName);
288                         out.println("Hit ^D to close your client and try another name");
289                         out.close();
290                     }
291                     catch (IOException e)
292                     {
293                         logger.severe("ChatServer: " + Failure.CLIENT_OUTPUT_STREAM);
294                         logger.severe(e.getLocalizedMessage());
295                     }
296                 }
297
298                 /*
299                  * Lorsqu'un ClientHandler se termine il lance la mÃ©thode
300                  * cleanup qui lorqu'il n'y a plus aucun thread modifie la
301                  * valeur de "listening" Ã  false
302                  */
303             }
304         } // while listening
305
306         // attente de la fin de tous les threads de ClientHandler
307         for (Thread t : handlerThreads)
308         {
309             try
310             {
311                 t.join();
312             }
313             catch (InterruptedException e)
314             {
315                 logger.severe("ChatServer::run: Client handlers join interrupted");
316                 logger.severe(e.getLocalizedMessage());
317             }
318         }
319
320         logger.info("ChatServer::run: all client handlers terminated");
321
322
323         handlerThreads.clear();
324         handlers.clear();
325         clients.clear();
326
327         // Fermeture de la socket du serveur
328         logger.info("ChatServer::run: Closing server socket ... ");
329         try
330         {
331             serverSocket.close();
332         }
333         catch (IOException e)
334         {
335             logger.severe("Close serversocket Failed !");
336             logger.severe(e.getLocalizedMessage());
337         }
338
339     }
340
341     /**
342      * MÃ©thode invoquÃ©e par les {@link ClientHandler} Ã  la fin de leur exÃ©cution
343      * pour Ã©ventuellement arrÃªter le serveur lorsqu'il n'y a plus de clients
344      */
345     protected synchronized void cleanup()
346     {
347         // s'il ne reste plus de threads on arrÃªte la boucle
348         int nbThreads = ClientHandler.getNbThreads();
349         if (nbThreads ≤ 0)
350         {
351             if (quitOnLastClient)
352             {
353                 listening = false;
354                 logger.info("ChatServer::run: no more threads.");
355             }
356         }
357         else
358         {
359             logger.info("ChatServer::run: still " + nbThreads +
360                     " threads remaining ...");
```

```
361             }
362         }
363 
364         /**
365          * Recherche parmis les clients dÃ©jÃ  enregistrÃ©s un client portant le mÃªme
366          * nom que l'argument
367          *
368          * @param clientName le nom du client Ã  rechercher parmis les clients dÃ©jÃ 
369          *                   enregistrÃ©s
370          * @return le client recherchÃ© s'il existe ou bien null s'il n'existe pas
371          */
372         protected InputOutputClient searchClientByName(String clientName)
373         {
374             /*
375              * La consultation de la liste des clients Ã  la recherche d'un nom doit
376              * Ãªtre atomique afin qu'aucun autre thread ne puisse modifier cette
377              * liste pendant qu'on la consulte : d'oÃ¹ le "synchronized"
378              */
379             synchronized (clients)
380             {
381                 for (InputOutputClient c : clients)
382                 {
383                     if (c.getName().equals(clientName))
384                     {
385                         return c;
386                     }
387                 }
388             }
389 
390             return null;
391         }
392 }
```



```
1   package chat.server;
2   
3   import java.io.IOException;
4   import java.io.InvalidClassException;
5   import java.io.NotSerializableException;
6   import java.io.ObjectOutputStream;
7   import java.util.Vector;
8   import java.util.logging.Logger;
9   
10  import chat.Vocabulary;
11  import logger.LoggerFactory;
12  import models.Message;
13  
14  /**
15   * Classe utilisÃ©e pour traiter chacune des connections des clients dans un
16   * nouveau thread
17   *
18   * @author davidroussel
19   */
20  public class ClientHandler implements Runnable
21  {
22      /**
23       * le ChatServer qui a lancÃ© ce thread
24       *
25       * @uml.property name="parent"
26       * @uml.associationEnd aggregation="shared"
27       */
28      private ChatServer parent;
29  
30      /**
31       * Le client principal de ce handler
32       *
33       * @uml.property name="mainClient"
34       * @uml.associationEnd aggregation="shared"
35       */
36      private InputClient mainClient;
37  
38      /**
39       * Les autres clients reliÃ©s au serveur.
40       *
41       * @uml.property name="allClients"
42       * @uml.associationEnd multiplicity="(1 -1)" ordering="true"
43       *                     aggregation="shared"
44       *                     inverse="clientHandler:chat.server.InputOutputClient"
45       */
46      private Vector<InputOutputClient> allClients;
47  
48      /**
49       * Compteur d'instances du nombre de threads crÃ©Ã©s pour traiter les
50       * connections
51       *
52       * @uml.property name="nbThreads"
53       */
54      private static int nbThreads = 0;
55  
56      /**
57       * Logger pour l'affichage des messages de debug
58       */
59      private Logger logger;
60  
61      /**
62       * Constructeur d'un handler de client
63       *
64       * @param parent le {@link ChatServer} qui a lancÃ© ce Runnable
65       * @param mainClient le client principal qu'il faut Ã©couter
66       * @param allClients les autres clients Ã  qui il faut redistribuer ce
67       *            qu'envoie le client principal
68       */
69      public ClientHandler(ChatServer parent,
70                           InputClient mainClient,
71                           Vector<InputOutputClient> allClients,
72                           Logger parentLogger)
73      {
74          this.parent = parent;
75          this.mainClient = mainClient;
76          this.allClients = allClients;
77          nbThreads++;
78          logger = LoggerFactory.getParentLogger(getClass(),
79                                                 parentLogger,
80                                                 parentLogger.getLevel());
81      }
82  
83      /**
84       * Accesseur en lecture du nombre de ClientHandler en activitÃ©
85       *
86       * @return the nbThreads
87       * @uml.property name="nbThreads"
88       */
89      public static int getNbThreads()
90      {
```

```
91          return nbThreads;
92      }
93
94      /**
95       * ExÃ©cution d'un handler de client. Consiste Ã  lire une ligne du client
96       * jusqu'Ã  ce que l'on reÃ§oive la commande bye, ou qu'une IOException
97       * intervienne si le flux est coupÃ©
98       *
99       * @see java.lang.Runnable#run()
100      */
101     @Override
102     public void run()
103     {
104         boolean loggedOut = false;
105         boolean killed = false;
106         String clientInput = null;
107
108
109         try
110         {
111             /*
112              * Attente d'une ligne de texte de la part d'un client (appel
113              * bloquant)
114              */
115             while (¬loggedOut ∧ ¬killed ∧
116                     ((clientInput = mainClient.getIn().readLine()) ≠ null))
117             {
118                 // Affiche ce qui est reÃ§u par le serveur dans la console
119                 System.out.println(mainClient.getName() + " > " + clientInput);
120
121                 // on vÃ©rifie que ce client n'a pas Ã©tÃ© banni par un super utilisateur
122                 if (mainClient.isBanned())
123                 {
124                     logger.info(mainClient.getName() + " is banned");
125                     loggedOut = true;
126                     break;
127                 }
128
129                 // On vÃ©rifie qu'il ne s'agit pas d'un message de contrÃ´le (kick ou bye)
130                 boolean controlMessage = false;
131                 for (String command : Vocabulary.commands)
132                 {
133                     if (clientInput.toLowerCase().startsWith(command))
134                     {
135                         controlMessage = true;
136                         break;
137                     }
138                 }
139
140                 StringBuffer messageContent = new StringBuffer();
141
142                 if (controlMessage)
143                 {
144                     // Le client veut nous quitter
145                     if (clientInput.toLowerCase().equals(Vocabulary.byeCmd))
146                     {
147                         messageContent.append(mainClient.getName() +
148                                                 " logged out");
149                         loggedOut = true;
150                     }
151                     // on vÃ©rifie si un kill est demandÃ© par le client
152                     else if (clientInput.toLowerCase().startsWith(Vocabulary.killCmd))
153                     {
154                         // on vÃ©rifie que le client est super-utilisateur
155                         // (ler de tous les clients)
156                         if (allClients.get(0) ≡ mainClient)
157                         {
158                             killed = true;
159                             parent.setListening(false);
160                             break;
161                         }
162                     }
163                     // on vÃ©rifie si un kick est demandÃ© par le client
164                     else if (clientInput.toLowerCase().startsWith(Vocabulary.kickCmd))
165                     {
166                         messageContent.append(Vocabulary.kickCmd);
167                         // On bloque l'accÃ¨s Ã  allClients tant que l'on traite
168                         // la commande du mainClient
169                         synchronized (allClients)
170                         {
171                             // on vÃ©rifie que le client est super-utilisateur
172                             // (ler de tous les clients)
173                             if (allClients.get(0) ≡ mainClient)
174                             {
175                                 // on recherche le nom du client Ã  kicker
176                                 String kickedName = null;
177                                 try
178                                 {
179                                     /*
180                                      * On recherche le nom du client Ã  kicker
```



```
181                                      * dans kick clientToKill
182                                      */
183                                     kickedName = clientInput.substring(
184                                             Vocabulary.kickCmd.length() + 1);
185                                 }
186                                 catch (IndexOutOfBoundsException iob)
187                                 {
188                                     logger.warning("ClientHandler: Error retreiving client name to kick");
189                                 }
190                                 if (kickedName ≠ null)
191                                 {
192                                     messageContent.append(" " + kickedName);
193                                     InputOutputClient kickedClient =
194                                         parent.searchClientByName(kickedName);
195                                     if (kickedClient ≠ null)
196                                     {
197                                         kickedClient.setBanned(true);
198                                         logger.info("Clienthandler["
199                                             + mainClient.getName() + "] client "
200                                             + kickedName + " banned");
201                                         messageContent.append(" [request granted by server]");
202                                     }
203                                     else
204                                     {
205                                         messageContent.append(" [client "
206                                             + kickedName + " does not exist]");
207                                     }
208                                 }
209                                 else
210                                 {
211                                     messageContent.append(" [no client name to kick]");
212                                 }
213                             }
214                             else
215                             {
216                                 int cmdL = Vocabulary.kickCmd.length();
217                                 messageContent.append(clientInput.substring(cmdL, (clientInput.lengt
h() - 1)));
218                                 messageContent.append(" [request denied by server]");
219                             }
220                             messageContent.append(" by " + mainClient.getName());
221                         }
222                     }
223                 }
224                 else
225                 {
226                     // Il s'agit d'un message ordinaire
227                     messageContent.append(clientInput);
228                 }
229
230                 /*
231                  * CrÃ©ation du message Ã  diffuser
232                  */
233                 Message message = null;
234                 if (controlMessage)
235                 {
236                     message = new Message(messageContent.toString());
237                 }
238                 else
239                 {
240                     message = new Message(messageContent.toString(),
241                                           mainClient.getName());
242                 }
243
244                 /*
245                  * Diffusion du message Ã  tous les clients.
246                  * allClients est un Vector qui est atomique donc a
247                  * priori on a pas besoin du "synchronized (allClients)",
248                  * NÃ©anmoins ce synchronized permet de bloquer l'accÃ¨s Ã 
249                  * l'ensemble des autres clients quand on diffuse le message de
250                  * notre mainClient Ã  tous les clients. Sans quoi on pourrait
251                  * diffuser le message Ã  un client, puis se faire interrompre
252                  * par un autre client, puis diffuser le message Ã  un autre
253                  * client, etc. A vÃ©rifier ...
254                  */
255                 synchronized (allClients)
256                 {
257                     for (InputOutputClient c : allClients)
258                     {
259                         if (c.isReady())
260                         {
261                             // rÃ©cupÃ©ration du flux de sortie et envoi du message
262                             ObjectOutputStream out = c.getOut();
263                             out.writeObject(message);
264                         }
265                         else
266                         {
267                             logger.warning("ClientHandler["
268                                     + mainClient.getName() + "]Client "
269                                     + c.getName() + " not ready");
```

```
                        }
                    }
                }
            }
        }
        catch (InvalidClassException ice)
        {
            logger.severe("ClientHandler["
                + mainClient.getName() + "]: write to client invalid class " +
                ice.getLocalizedMessage());
        }
        catch (NotSerializableException nse)
        {
            logger.severe(
                "ClientHandler[" + mainClient.getName()
                    + "]: write to not serializable exception "
                    + nse.getLocalizedMessage());
        }
        catch (IOException e)
        {
            logger.severe("ClientHandler[" + mainClient.getName()
                    + "]: received or write failed, Closing client " + this);
        }

        // remove current client from allClients (should be atomic)
        synchronized (allClients)
        {
            allClients.remove(mainClient);
        }
        // cleanup current client
        mainClient.cleanup();
        synchronized (parent)
        {
            // dÃ©crÃ©mentation du nombre de threads des clients
            nbThreads--;
            // Nettoyage du ChatServer parent (qui pourra evt s'arrÃªter s'il n'y a
            // plus de clients)
            parent.cleanup();
        }
    }

}
```



```
package chat.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.logging.Logger;

import logger.LoggerFactory;

/**
 * Classe stockant les caractÃ©ristiques d'un client traitÃ© par un
 * {@link ClientHandler}. Celui ci est caractÃ©risÃ© par
 * <ul>
 * <li>{@link #clientSocket} : {@link Socket} du client</li>
 * <li>{@link #name} : nom du client</li>
 * <li>{@link #inBR} : {@link BufferedReader} crÃ©Ã© Ã  partir d'un
 * {@link InputStreamReader} sur l'{@link InputStream} de la {@link Socket}
 * et permettant de lire le texte en provenance du client</li>
 * <li>{@link #ready} indique que l'{@link BufferedReader} a Ã©tÃ© crÃ©Ã© et que
 * l'on est prÃªt Ã  lire les lignes en provenance du client</li>
 * <li>{@link #banned} indique le statut de bannissement</li>
 * </ul>
 *
 * @author davidroussel
 */
public class InputClient
{
    /**
     * La socket du client
     */
    protected Socket clientSocket;

    /**
     * Le nom du client
     *
     * @uml.property name="name"
     */
    protected String name;

    /**
     * le flux d'entrÃ©e du client (celui sur lequel on lit ce qui vient du
     * client)
     */
    protected BufferedReader inBR;

    /**
     * Un Main client est "ready" lorsque sa clientSocket est non nulle et que
     * l'on a rÃ©ussi Ã  obtenir son input stream
     *
     * @uml.property name="ready"
     */
    protected boolean ready;

    /**
     * Etat de bannissement du client. IdÃ©e : le premier utilisateur du serveur
     * est considÃ©rÃ© comme le super-user (un MainClient). En consÃ©quence il a
     * le privilÃ¨ge de pouvoir kicker les autres clients.
     *
     * @uml.property name="banned"
     */
    protected boolean banned;

    /**
     * logger pour afficher les messages de debug
     */
    protected Logger logger;

    /**
     * Constructeur d'un MainClient
     * @param socket the client's socket
     * @param name the client's name
     * @param parentLogger logger parent pour l'affichage des messages de debug
     */
    public InputClient(Socket socket, String name, Logger parentLogger)
    {
        clientSocket = socket;
        this.name = name;
        inBR = null;
        ready = false;

        logger = LoggerFactory.getParentLogger(getClass(),
                                               parentLogger,
                                               parentLogger.getLevel());

        if (socket ≠ null)
        {
            logger.info("InputClient: Creating Input Stream ... ");
            try
            {
```

```
91                  inBR = new BufferedReader(new InputStreamReader(
92                          socket.getInputStream()));
93                  ready = true;
94              }
95              catch (IOException e)
96              {
97                  logger.severe("InputClient: unable to get client socket input stream");
98                  logger.severe(e.getLocalizedMessage());
99              }
100         }
101     }
102
103     /**
104      * Accesseur en lecture du nom du client
105      *
106      * @return the name
107      * @uml.property name="name"
108      */
109     public String getName()
110     {
111         return name;
112     }
113
114     /**
115      * Accesseur en lecture du flux d'entrÃ©e du client
116      *
117      * @return the input {@link BufferedReader}
118      */
119     public BufferedReader getIn()
120     {
121         return inBR;
122     }
123
124     /**
125      * Accesseur en lecture de l'Ã©tat du client
126      *
127      * @return the ready
128      * @uml.property name="ready"
129      */
130     public boolean isReady()
131     {
132         return ready;
133     }
134
135     /**
136      * Accesseur en lecture de l'Ã©tat de banissement
137      *
138      * @return l'Ã©tat de banissement
139      * @uml.property name="banned"
140      */
141     public boolean isBanned()
142     {
143         return banned;
144     }
145
146     /**
147      * Accesseur en Ã©criture de l'Ã©tat de banissement
148      *
149      * @param l'Ã©tat de banissement Ã  mettre en place
150      * @uml.property name="banned"
151      */
152     public void setBanned(boolean banned)
153     {
154         this.banned = banned;
155     }
156
157     /**
158      * Nettoyage d'un client principal : fermeture du flux d'entrÃ©e et fermeture
159      * de sa socket.
160      */
161     public void cleanup()
162     {
163         ready = false;
164         logger.info("MainClient::cleanup: closing input stream ... ");
165         try
166         {
167             inBR.close();
168         }
169         catch (IOException e)
170         {
171             logger.severe("MainClient::cleanup: unable to close input stream");
172             logger.severe(e.getLocalizedMessage());
173         }
174
175         logger.info("MainClient::cleanup: closing client socket ... ");
176         try
177         {
178             clientSocket.close();
179         }
180         catch (IOException e)
```



```
181         {
182             logger.severe("MainClient::cleanup: unable to close client socket");
183             logger.severe(e.getLocalizedMessage());
184         }
185     }
186 }
```

```java
package chat.server;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.logging.Logger;

import chat.Failure;


/**
 * Classe stockant les caractÃ©ristiques d'un client :
 * voir {@link InputClient}.
 * Un client "normal" ajoute aussi le flux de sortie sur lequel on Ã©crit les
 * messages vers le client
 * <ul>
 *  <li>out : {@link ObjectOutputStream}</li>
 * </ul>
 * @author davidroussel
 *
 */
public class InputOutputClient extends InputClient
{
    /**
     * Le flux de sortie vers le client (celui sur lequel on Ã©crit au client)
     */
    private ObjectOutputStream outOS;

    /**
     * Constructeur d'un client
     * @param socket la socket du client
     * @param name le nom du client
     * @param verbose niveau de debug pour les messages
     * @param parentLogger logger parent pour l'affichage des messages
     */
    public InputOutputClient(Socket socket, String name, Logger parentLogger)
    {
        super(socket, name, parentLogger);
        if (ready)
        {
            outOS = null;
            ready = false;

            if (clientSocket ≠ null)
            {
                logger.info("Client: Creating Output Stream ... ");
                try
                {
                    outOS = new ObjectOutputStream(clientSocket.getOutputStream());
                    ready = true;
                }
                catch (IOException e)
                {
                    logger.severe("Client: unable to get client output stream");
                    logger.severe(e.getLocalizedMessage());
                }
            }
        }
        else
        {
            logger.severe("Client: " + Failure.CLIENT_NOT_READY + ", abort...");
            System.exit(Failure.CLIENT_NOT_READY.toInteger());
        }
    }

    /**
     * Accesseur en lecture du flux de sortie d'un client
     * @return the out
     */
    public ObjectOutputStream getOut()
    {
        return outOS;
    }

    /**
     * Nettoyage d'un client : fermeture du flux de sortie et super.cleanup()
     */
    @Override
    public void cleanup()
    {
        logger.info("Client::cleanup: closing output stream ... ");
        try
        {
            outOS.close();
        }
        catch (IOException e)
        {
            logger.severe("Client: unable to close client output stream");
            logger.severe(e.getLocalizedMessage());
        }
        super.cleanup();
```



```java
    }
}
```

17 nov 14 17:44 **package-info.java** Page 1/1

```java
package chat.server;

/**
 * Sous-package contenant les classes relatives Ã  la partie serveur du
 * client/serveur de chat
 */
```

10 avr 16 19:39 **UserOutputType.java** Page 1/1

```java
package chat;

/**
 * Les diffÃ©rents types de de donnÃ©es attendues dans le flux de sortie
 * vers le client pour afficher les message en provenance du serveur.
 */
public enum UserOutputType
{
    /**
     * Le client attends des donnÃ©es sous forme texte
     */
    TEXT,
    /**
     * Le client attends des donnÃ©es sous forme d'objets (en l'occurrence
     * des Message ou des UserMessage)
     */
    OBJECT;

    /**
     * Affichage sous forme de texte des erreurs possibles
     */
    @Override
    public String toString()
    {
        switch (this)
        {
            case TEXT:
                return new String("Text output type");
            case OBJECT:
                return new String("Object output type");
        }
        throw new AssertionError("UserOutputType: unknown type: " + this);
    }

    /**
     * Conversion en entier du type sortie vers l'utilisateur
     *
     * @return le numÃ©ro correspondant au type de sortie vers l'utilisateur
     * <ul>
     *  <li>TEXT = 1</li>
     *  <li>OBJECT = 2</li>
     * </ul>
     */
    public int toInteger()
    {
        return ordinal() + 1;
    }

    public static UserOutputType fromInteger(int value)
    {
        int controlValue;
        if (value < 1)
        {
            controlValue = 1;
        }
        else if (value > 2)
        {
            controlValue = 2;
        }
        else
        {
            controlValue = value;
        }
        switch (controlValue)
        {
            default:
            case 1:
                return TEXT;
            case 2:
                return OBJECT;
        }
    }

}
```

```java
package chat;
/**
 * Interface contenant le vocabulaire spÃ©cial utilisÃ© dans le serveur de chat
 * @author davidroussel
 */
public interface Vocabulary
{
    /**
     * Mot clÃ© utilisÃ© par un client pour se dÃ©loguer du serveur
     */
    public final static String byeCmd="bye";

    /**
     * Mot clÃ© utilisÃ© par un super user pour terminer le serveur
     */
    public final static String killCmd="kill";

    /**
     * Mot clÃ© spÃ©cial utilisÃ© par un super user pour dÃ©loguer de force un
     * client : kick <username>
     */
    public final static String kickCmd="kick";

    /**
     * Sauts de ligne du systÃ¨me d'exploitation (utilisÃ© dans le texte)
     */
    public final static String newLine = System.getProperty("line.separator");

    /**
     * Un tableau contenant l'ensemble des commandes du serveur afin de pouvoir
     * le parcourir
     */
    public final static String[] commands = {byeCmd, kickCmd, killCmd};

}
```



```java
/**
 * Package contenant des exemples de
 * <ul>
 *  <li>{@link JFrame} illustrant une fenÃªtre et son contenu (et en particuluer
 *  lorsqu'un container contient un {@link JScrollPane} qui lui mÃªme contient
 *  un {@link JTextPane} qui lui mÃªme contient un {@link StyledDocument} dans
 *  lequel on peut ajouter du texte riche.</li>
 *  <li>{@link Runnable}</li>
 * </ul>
 * @author davidroussel
 */
package examples;
```

```java
package examples;
import java.awt.EventQueue;

import examples.widgets.ExampleFrame;


/**
 * Programme principal lanÃ§ant une {@link ExampleFrame}
 * @author davidroussel
 *
 */
public class RunExampleFrame
{
    /**
     * Programme principal
     * @param args
     */
    public static void main(String[] args)
    {
        if (System.getProperty("os.name").startsWith("Mac OS"))
        {
            // Met en place le menu en haut de l'Ã©cran plutÃ´t que dans l'application
            System.setProperty("apple.laf.useScreenMenuBar", "true");
            System.setProperty("com.apple.mrj.application.apple.menu.about.name", "Name");
        }

        // Insertion de la frame dans la file des Ã©vÃ¨nements GUI
        EventQueue.invokeLater(new Runnable()
        {
            @Override
            public void run()
            {
                try
                {
                    ExampleFrame frame = new ExampleFrame();
                    frame.pack();
                    frame.setVisible(true);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
}
```



```java
package examples;
import java.awt.EventQueue;

import javax.swing.JFrame;

import examples.widgets.ExampleFrame;
import examples.widgets.ListExampleFrame;


/**
 * Programme principal lanÃ§ant une {@link ExampleFrame}
 * @author davidroussel
 *
 */
public class RunListFrame
{
    /**
     * Programme principal
     * @param args
     */
    public static void main(String[] args)
    {
        if (System.getProperty("os.name").startsWith("Mac OS"))
        {
            // Met en place le menu en haut de l'Ã©cran plutÃ´t que dans l'application
            System.setProperty("apple.laf.useScreenMenuBar", "true");
            System.setProperty("com.apple.mrj.application.apple.menu.about.name", "Name");
        }

        // Insertion de la frame dans la file des Ã©vÃ¨nements GUI
        EventQueue.invokeLater(new Runnable()
        {
            @Override
            public void run()
            {
                try
                {
                    JFrame frame = new ListExampleFrame();
                    frame.pack();
                    frame.setVisible(true);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
1  package examples;
2  import java.util.ArrayList;
3  import java.util.Collection;
4
5  /**
6   * Exemple de classe implÃ©mentant un Runnable et lancÃ© dans un Thread
7   *
8   * @author davidroussel
9   */
10 public class RunnableExample
11 {
12     /**
13      * Classe interne reprÃ©sentant un simple compteur Ã  exÃ©cuter dans un thread.
14      * Le compteur compte de 0 Ã  une valeur max. Lorsque le compteur atteint la
15      * valeur max le compteur s'arrÃªte.
16      * @author davidroussel
17      */
18     protected static class Counter implements Runnable
19     {
20         /**
21          * Nombre de compteurs instanciÃ©s
22          */
23         private static int CounterNumber = 0;
24
25         /**
26          * Le numÃ©ro de compteur
27          */
28         private int number;
29         /**
30          * Le compteur proprement dit
31          */
32         private int count;
33
34         /**
35          * La valeur max du compteur
36          */
37         private int max;
38
39         /**
40          * Constructeur valuÃ© du compteur
41          * @param max la valeur max du compteur Ã  laquelle il s'arrÃªte
42          */
43         public Counter(int max)
44         {
45             number = ++CounterNumber;
46             count = 0;
47             this.max = max;
48         }
49
50
51         /* (non-Javadoc)
52          * @see java.lang.Object#finalize()
53          */
54         @Override
55         protected void finalize() throws Throwable
56         {
57             CounterNumber--;
58         }
59
60         /**
61          * Boucle d'Ã©xÃ©cution principale du compteur : Tant que le compteur n'a
62          * pas atteint la valeur max le compteur incrÃ©mente son compteur de 1,
63          * affiche la valeur courante du compteur puis on demande au thread
64          * dans lequel il tourne de passer la main Ã  un autre thread (en
65          * espÃ©rant que ceux ci nous repassent la main un jour afin que l'on
66          * puisse continuer Ã  compter).
67          */
68         @Override
69         public void run()
70         {
71             while (count < max)
72             {
73                 count++;
74
75                 System.out.println(this); // utilisation du toString
76
77                 // passe la main Ã  d'autres threads (si besoin)
78                 Thread.yield();
79             }
80         }
81
82         /* (non-Javadoc)
83          * @see java.lang.Object#toString()
84          */
85         @Override
86         public String toString()
87         {
88             return new String("Counter #" + number + " = " + count);
89         }
90     }
```



```
91
92     /**
93      * Collection de compteurs Runnable Ã  lancer
94      */
95     protected Collection<Counter> counters;
96
97     /**
98      * Collection de threads dans lesquels on va vaire tourner les Counter.
99      */
100    protected Collection<Thread> threads;
101
102    /**
103     * Constructeur d'un RunnableExample.
104     * CrÃ©e un certain nombre de compteur (Runnable), puis crÃ©e le mÃªme nombre
105     * de threads dans lesquels on place ces compteurs
106     */
107    public RunnableExample(int nbCounters)
108    {
109        counters = new ArrayList<Counter>(nbCounters);
110        threads = new ArrayList<Thread>(nbCounters);
111
112        for (int i = 0; i < nbCounters; i++)
113        {
114            Counter c = new Counter(10);
115            counters.add(c);
116
117            Thread t = new Thread(c);
118            threads.add(t);
119        }
120    }
121
122    /**
123     * Lancement de tous les threads (contenant les compteurs)
124     */
125    public void launch()
126    {
127        for (Thread t : threads)
128        {
129            t.start();
130        }
131    }
132
133    /**
134     * attente de la fin de tous les threads pour terminer le thread principal
135     */
136    public void terminate()
137    {
138        for (Thread t : threads)
139        {
140            try
141            {
142                t.join();
143            }
144            catch (InterruptedException e)
145            {
146                System.err.println("Thread" + t + " join interrupted");
147                e.printStackTrace();
148            }
149        }
150
151        System.out.println("All threads terminated");
152    }
153
154    /**
155     * Programme principal.
156     * Lancement de plusieurs Counters
157     *
158     * @param args arguments du programme pour y lire le nombre de compteurs Ã 
159     * lancer
160     */
161    public static void main(String[] args)
162    {
163        int nbCounters = 3;
164        // on lit le nombre de counters dans le premier argument du programme
165        if (args.length > 0)
166        {
167            int value;
168            try
169            {
170                value = Integer.parseInt(args[0]);
171                if (value > 0)
172                {
173                    nbCounters = value;
174                }
175            }
176            catch (NumberFormatException nfe)
177            {
178                System.err.println("Error reading number of counters");
179            }
180        }
```

```
181
182             RunnableExample runner = new RunnableExample(nbCounters);
183
184             runner.launch();
185
186             System.out.println("All threads launched");
187
188             runner.terminate();
189         }
190     }
```



```
1   package examples;
2   import java.util.ArrayList;
3   import java.util.Collection;
4
5   /**
6    * Exemple de classe implÃ©mentant un Runnable et lancÃ© dans un Thread
7    *
8    * @author davidroussel
9    */
10  public class RunRunnableExample
11  {
12      /**
13       * Classe interne reprÃ©sentant un simple compteur Ã  exÃ©cuter dans un thread.
14       * Le compteur compte de 0 Ã  une valeur max. Lorsque le compteur atteint la
15       * valeur max le compteur s'arrÃªte.
16       * @author davidroussel
17       */
18      protected static class Counter implements Runnable
19      {
20          /**
21           * Nombre de compteurs instanciÃ©s
22           */
23          private static int CounterNumber = 0;
24
25          /**
26           * Le numÃ©ro de compteur
27           */
28          private int number;
29          /**
30           * Le compteur proprement dit
31           */
32          private int count;
33
34          /**
35           * La valeur max du compteur
36           */
37          private int max;
38
39          /**
40           * Constructeur valuÃ© du compteur
41           * @param max la valeur max du compteur Ã  laquelle il s'arrÃªte
42           */
43          public Counter(int max)
44          {
45              number = ++CounterNumber;
46              count = 0;
47              this.max = max;
48          }
49
50          /**
51           * Nettoyage lors de la destruction
52           * @see java.lang.Object#finalize()
53           */
54          @Override
55          protected void finalize() throws Throwable
56          {
57              CounterNumber--;
58          }
59
60          /**
61           * Boucle d'Ã©xÃ©cution principale du compteur : Tant que le compteur n'a
62           * pas atteint la valeur max le compteur incrÃ©mente son compteur de 1,
63           * affiche la valeur courante du compteur puis on demande au thread
64           * dans lequel il tourne de passer la main Ã  un autre thread (en
65           * espÃ©rant que ceux ci nous repassent la main un jour afin que l'on
66           * puisse continuer Ã  compter).
67           */
68          @Override
69          public void run()
70          {
71              while (count < max)
72              {
73                  count++;
74
75                  System.out.println(this); // utilisation du toString
76
77                  // passe la main Ã  d'autres threads (si besoin)
78                  Thread.yield();
79              }
80          }
81
82          /**
83           * ReprÃ©sentation sous forme de chaine de caractÃ¨res
84           * @see java.lang.Object#toString()
85           */
86          @Override
87          public String toString()
88          {
89              return new String("Counter #" + number + " = " + count);
90          }
```

```
91      }
92
93      /**
94       * Collection de compteurs Runnable Ã  lancer
95       */
96      protected Collection<Counter> counters;
97
98      /**
99       * Collection de threads dans lesquels on va vaire tourner les Counter.
100      */
101     protected Collection<Thread> threads;
102
103     /**
104      * Constructeur d'un RunnableExample.
105      * CrÃ©e un certain nombre de compteur (Runnable), puis crÃ©e le mÃªme nombre
106      * de threads dans lesquels on place ces compteurs
107      */
108     public RunRunnableExample(int nbCounters)
109     {
110         counters = new ArrayList<Counter>(nbCounters);
111         threads = new ArrayList<Thread>(nbCounters);
112
113         for (int i = 0; i < nbCounters; i++)
114         {
115             Counter c = new Counter(10);
116             counters.add(c);
117
118             Thread t = new Thread(c);
119             threads.add(t);
120         }
121     }
122
123     /**
124      * Lancement de tous les threads (contenant les compteurs)
125      */
126     public void launch()
127     {
128         for (Thread t : threads)
129         {
130             t.start();
131         }
132     }
133
134     /**
135      * attente de la fin de tous les threads pour terminer le thread principal
136      */
137     public void terminate()
138     {
139         for (Thread t : threads)
140         {
141             try
142             {
143                 t.join();
144             }
145             catch (InterruptedException e)
146             {
147                 System.err.println("Thread" + t + " join interrupted");
148                 e.printStackTrace();
149             }
150         }
151
152         System.out.println("All threads terminated");
153     }
154
155     /**
156      * Programme principal.
157      * Lancement de plusieurs Counters
158      *
159      * @param args arguments du programme pour y lire le nombre de compteurs Ã 
160      * lancer
161      */
162     public static void main(String[] args)
163     {
164         int nbCounters = 3;
165         // on lit le nombre de counters dans le premier argument du programme
166         if (args.length > 0)
167         {
168             int value;
169             try
170             {
171                 value = Integer.parseInt(args[0]);
172                 if (value > 0)
173                 {
174                     nbCounters = value;
175                 }
176             }
177             catch (NumberFormatException nfe)
178             {
179                 System.err.println("Error reading number of counters");
180             }
```



```
181             }
182
183             RunRunnableExample runner = new RunRunnableExample(nbCounters);
184
185             runner.launch();
186
187             System.out.println("All threads launched");
188
189             runner.terminate();
190         }
191     }
```

```
1  package examples.widgets;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Component;
6  import java.awt.Dimension;
7  import java.awt.HeadlessException;
8  import java.awt.event.ActionEvent;
9  import java.awt.event.InputEvent;
10 import java.awt.event.KeyEvent;
11
12 import javax.swing.AbstractAction;
13 import javax.swing.Action;
14 import javax.swing.Box;
15 import javax.swing.ImageIcon;
16 import javax.swing.JButton;
17 import javax.swing.JFrame;
18 import javax.swing.JMenu;
19 import javax.swing.JMenuBar;
20 import javax.swing.JMenuItem;
21 import javax.swing.JScrollPane;
22 import javax.swing.JSeparator;
23 import javax.swing.JTextPane;
24 import javax.swing.JToolBar;
25 import javax.swing.KeyStroke;
26 import javax.swing.text.BadLocationException;
27 import javax.swing.text.Style;
28 import javax.swing.text.StyleConstants;
29 import javax.swing.text.StyledDocument;
30
31 /**
32  * Exemple simple de fenÃªtre graphique
33  * @author davidroussel
34  */
35 public class ExampleFrame extends JFrame
36 {
37     /**
38      * ChaÃ®ne de caractÃ¨re pour passer Ã  la ligne
39      */
40     protected static String newline = System.getProperty("line.separator");
41
42     /**
43      * Bouton "Red"
44      */
45     private JButton redButton;
46
47     /**
48      * Bouton "Blue"
49      */
50     private JButton blueButton;
51
52     /**
53      * Bouton "Clear"
54      */
55     private JButton clearButton;
56
57     /**
58      * Document dans lequel Ã©crire (Ã  extraire du JTextPane avec
59      * {@link JTextPane.getStyledDocument()})
60      */
61     protected StyledDocument document;
62
63     /**
64      * Style Ã  appliquer lors de l'Ã©criture dans le document
65      */
66     protected Style style;
67
68     /**
69      * Couleur par dÃ©faut lors de l'Ã©criture dans le document
70      */
71     protected Color defaultColor;
72
73     /**
74      * Action Ã  rÃ©aliser lorsque l'on cliquera sur le bouton "Red" ou lorsque
75      * l'on tapera "Crtl-R" dans le JTextPane
76      */
77     private final Action redAction;
78
79     /**
80      * Action Ã  rÃ©aliser lorsque l'on cliquera sur le bouton "Blue" ou lorsque
81      * l'on tapera "Crtl-B" dans le JTextPane
82      */
83     private final Action blueAction;
84
85     /**
86      * Action Ã  rÃ©aliser lorsque l'on cliquera sur le bouton "Clear" ou lorsque
87      * l'on tapera "Crtl-L" dans le JTextPane
88      */
89     private final Action clearAction;
90
```



```
91     /**
92      * CrÃ©ation d'une fenÃªtre graphique simple
93      * @throws HeadlessException
94      */
95     public ExampleFrame() throws HeadlessException
96     {
97         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
98         setTitle("Red Blue Example");
99         redAction = new RedAction();
100        blueAction = new BlueAction();
101        clearAction = new ClearAction();
102
103        setPreferredSize(new Dimension(400, 200));
104
105        JMenuBar menuBar = new JMenuBar();
106        setJMenuBar(menuBar);
107
108        JMenu menuActions = new JMenu("Actions");
109        menuBar.add(menuActions);
110
111        JMenuItem menuItemRed = new JMenuItem(redAction);
112        menuActions.add(menuItemRed);
113
114        JMenuItem menuItemBlue = new JMenuItem(blueAction);
115        menuActions.add(menuItemBlue);
116
117        JSeparator separator = new JSeparator();
118        menuActions.add(separator);
119
120        JMenuItem menuItemClear = new JMenuItem(clearAction);
121        menuActions.add(menuItemClear);
122
123        JToolBar toolBar = new JToolBar();
124        toolBar.setFloatable(false);
125        getContentPane().add(toolBar, BorderLayout.NORTH);
126
127        redButton = new JButton(redAction);
128        toolBar.add(redButton);
129
130        blueButton = new JButton(blueAction);
131        toolBar.add(blueButton);
132
133        Component horizontalGlue = Box.createHorizontalGlue();
134        toolBar.add(horizontalGlue);
135
136        clearButton = new JButton(clearAction);
137        toolBar.add(clearButton);
138
139        JScrollPane scrollPane = new JScrollPane();
140        getContentPane().add(scrollPane, BorderLayout.CENTER);
141
142        JTextPane textPane = new JTextPane();
143
144        document = textPane.getStyledDocument();
145        style = textPane.addStyle("New Style", null);
146        defaultColor = StyleConstants.getForeground(style);
147
148        scrollPane.setViewportView(textPane);
149    }
150
151    /**
152     * Ajoute du texte avec une couleur spÃ©cifique Ã  la fin du document
153     * @param text le texte Ã  ajouter
154     * @param color la couleur dans laquelle ajouter le texte
155     */
156    public void appendToDocument(String text, Color color)
157    {
158        StyleConstants.setForeground(style, color);
159
160        try
161        {
162            document.insertString(document.getLength(), text
163                    + newline, style);
164        }
165        catch (BadLocationException ex)
166        {
167            System.err.println("write at bad location");
168            ex.printStackTrace();
169        }
170
171        StyleConstants.setForeground(style, defaultColor);
172    }
173
174    // ------------------------------------------------------------------------
175    // Actions de l'application
176    //  On utilise des actions lorsque celles ci doivent pouvoir Ãªtre invoquÃ©es
177    //  depuis divers Ã©lements de l'interface graphique: p.ex. menu ET bouton.
178    //  Sinon un simple ActionListener sur un bouton par exemple suffirait.
179    // ------------------------------------------------------------------------
180
```

```
181     /**
182      * Action listener interne Ã  la classe ExampleFrame pour executer les
183      * instructions requises lorsque l'on clique sur le bouton "blue"
184      */
185     private class BlueAction extends AbstractAction
186     {
187         /**
188          * Constructeur de BlueAction: met en place le nom et la description de
189          * l'action ainsi que son raccourci clavier
190          */
191         public BlueAction()
192         {
193             putValue(MNEMONIC_KEY, KeyEvent.VK_B);
194             putValue(SMALL_ICON, new ImageIcon(ExampleFrame.class.getResource("/examples/icons/bg_blue-16.p
    ng")));
195             putValue(LARGE_ICON_KEY, new ImageIcon(ExampleFrame.class.getResource("/examples/icons/bg_blue
    -32.png")));
196             putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_B, InputEvent.META_MASK));
197             putValue(NAME, "Blue");
198             putValue(SHORT_DESCRIPTION, "Prints \"Blue\" in blue in the document");
199         }
200
201         /**
202          * Action Ã  rÃ©aliser lorsque le BlueAction est sollicitÃ©
203          * @param e l'action event associÃ©
204          */
205         @Override
206         public void actionPerformed(ActionEvent e)
207         {
208             /*
209              * BlueAction Ã©tant une classe interne (non static) elle a
210              * donc accÃ¨s aux membres de la classe ExampleFrame
211              * Change la couleur du texte en bleu et affiche un message
212              */
213             appendToDocument("Blue", Color.BLUE);
214         }
215     }
216
217     /**
218      * Listener lorsque le bouton #btnClear est activÃ©.
219      * Efface le contenu du {@link #document}
220      */
221     private class ClearAction extends AbstractAction
222     {
223         /**
224          * Constructeur de ClearAction: met en place le nom et la description de
225          * l'action ainsi que son raccourci clavier
226          */
227         public ClearAction()
228         {
229             putValue(MNEMONIC_KEY, KeyEvent.VK_L);
230             putValue(SMALL_ICON, new ImageIcon(ExampleFrame.class.getResource("/examples/icons/erase-16.png
    ")));
231             putValue(LARGE_ICON_KEY, new ImageIcon(ExampleFrame.class.getResource("/examples/icons/erase-3
    2.png")));
232             putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_L, InputEvent.META_MASK));
233             putValue(NAME, "Clear");
234             putValue(SHORT_DESCRIPTION, "Clears the document");
235         }
236
237         /**
238          * OpÃ©rations Ã  rÃ©aliser lorsque #clearAction est sollicitÃ©e
239          * @param e l'Ã©vÃ¨nement Ã  l'origine du dÃ©clenchement de l'action
240          */
241         @Override
242         public void actionPerformed(ActionEvent e)
243         {
244             try
245             {
246                 document.remove(0, document.getLength());
247             }
248             catch (BadLocationException ex)
249             {
250                 System.err.println("ClientFrame: clear doc: bad location");
251                 ex.printStackTrace();
252             }
253         }
254     }
255
256     /**
257      * Action interne Ã  la classe ExampleFrame pour executer les
258      * instructions requises lorsque l'on clique sur le bouton "red"
259      */
260     private class RedAction extends AbstractAction
261     {
262         /**
263          * Constructeur de RedAction: met en place le nom et la description de
264          * l'action ainsi que son raccourci clavier
265          */
266         public RedAction()
```



```
267         {
268             putValue(MNEMONIC_KEY, KeyEvent.VK_R);
269             putValue(SMALL_ICON, new ImageIcon(ExampleFrame.class.getResource("/examples/icons/bg_red-16.pn
    g")));
270             putValue(LARGE_ICON_KEY, new ImageIcon(ExampleFrame.class.getResource("/examples/icons/bg_red-
    32.png")));
271             putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_R, InputEvent.META_MASK));
272             putValue(NAME, "Red");
273             putValue(SHORT_DESCRIPTION, "Prints \"Red\" in red in the document");
274         }
275
276         /**
277          * OpÃ©rations Ã  rÃ©aliser lorsque #redAction est sollicitÃ©e
278          * @param e l'Ã©vÃ¨nement Ã  l'origine du dÃ©clenchement de l'action
279          */
280         @Override
281         public void actionPerformed(ActionEvent e)
282         {
283             /*
284              * Change la couleur du texte en rouge et affiche "Red" dans le
285              * document
286              */
287             appendToDocument("Red", Color.RED);
288         }
289     }
290 }
```

```
1  package examples.widgets;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Component;
6  import java.awt.Dimension;
7  import java.awt.HeadlessException;
8  import java.awt.event.ActionEvent;
9  import java.awt.event.InputEvent;
10 import java.awt.event.KeyEvent;
11 import java.awt.event.MouseAdapter;
12 import java.awt.event.MouseEvent;
13 import java.util.Stack;
14
15 import javax.swing.AbstractAction;
16 import javax.swing.Action;
17 import javax.swing.DefaultListModel;
18 import javax.swing.ImageIcon;
19 import javax.swing.JButton;
20 import javax.swing.JFrame;
21 import javax.swing.JLabel;
22 import javax.swing.JList;
23 import javax.swing.JMenuItem;
24 import javax.swing.JOptionPane;
25 import javax.swing.JPanel;
26 import javax.swing.JPopupMenu;
27 import javax.swing.JScrollPane;
28 import javax.swing.JSeparator;
29 import javax.swing.JTextArea;
30 import javax.swing.KeyStroke;
31 import javax.swing.ListCellRenderer;
32 import javax.swing.ListSelectionModel;
33 import javax.swing.UIManager;
34 import javax.swing.event.ListSelectionEvent;
35 import javax.swing.event.ListSelectionListener;
36
37 /**
38  * Exemple de fenÃ¨tre contenant une liste d'Ã©lÃ©ments
39  *
40  * @author davidroussel
41  */
42 public class ListExampleFrame extends JFrame
43 {
44     /**
45      * ChaÃ®ne de caractÃ¨re pour passer Ã  la ligne
46      */
47     private static String newline = System.getProperty("line.separator");
48
49     /**
50      * Liste des Ã©lÃ©ments Ã  afficher dans la JList.
51      * Les ajouts et retraits effectuÃ©s dans cette ListModel seront alors
52      * automatiquement transmis au JList contenant ce ListModel
53      */
54     private DefaultListModel<String> elements = new DefaultListModel<String>();
55
56     /**
57      * Le modÃ¨le de sÃ©lection de la JList.
58      * Conserve les indices des Ã©lÃ©ments sÃ©lectionnÃ©s de {@link #elements} dans
59      * la JList qui affiche ces Ã©lÃ©ments.
60      */
61     private ListSelectionModel selectionModel = null;
62
63     /**
64      * La text area oÃ¹ afficher les messages
65      */
66     private JTextArea output = null;
67
68     /**
69      * Action Ã  rÃ©aliser lorsque l'on souhaite supprimer les Ã©lÃ©ments
70      * sÃ©lectionnÃ©s de la liste
71      */
72     private final Action removeAction = new RemoveItemAction();
73
74     /**
75      * Action Ã  rÃ©aliser lorsque l'on souhaite dÃ©selctionner tous les Ã©lements de la liste
76      */
77     private final Action clearSelectionAction = new ClearSelectionAction();
78
79     /**
80      * Action Ã  rÃ©aliser lorsque l'on souhaite ajouter un Ã©lÃ©ment Ã  la liste
81      */
82     private final Action addAction = new AddAction();
83
84     /**
85      * @throws HeadlessException
86      */
87     public ListExampleFrame() throws HeadlessException
88     {
89         super(); // dÃ©jÃ  implicite
90         elements.addElement("TÃ©nÃ©phore");
```



```
91         elements.addElement("ZÃ©bulon");
92         elements.addElement("ZÃ©phirine");
93         elements.addElement("Uriel");
94         elements.addElement("PhilomÃ¨ne");
95
96         setPreferredSize(new Dimension(200, 100));
97         getContentPane().setLayout(new BorderLayout(0, 0));
98
99         JScrollPane textScrollPane = new JScrollPane();
100        getContentPane().add(textScrollPane, BorderLayout.CENTER);
101
102        output = new JTextArea();
103        textScrollPane.setViewportView(output);
104
105        JPanel leftPanel = new JPanel();
106        leftPanel.setPreferredSize(new Dimension(200, 10));
107        getContentPane().add(leftPanel, BorderLayout.WEST);
108        leftPanel.setLayout(new BorderLayout(0, 0));
109
110        JButton btnClearSelection = new JButton("Clear Selection");
111        btnClearSelection.setAction(clearSelectionAction);
112        leftPanel.add(btnClearSelection, BorderLayout.NORTH);
113
114        JScrollPane listScrollPane = new JScrollPane();
115        leftPanel.add(listScrollPane, BorderLayout.CENTER);
116
117        JList<String> list = new JList<String>(elements);
118        listScrollPane.setViewportView(list);
119        list.setName("Elements");
120        list.setBorder(UIManager.getBorder("EditorPane.border"));
121        list.setSelectedIndex(0);
122        list.setCellRenderer(new ColorTextRenderer());
123
124        JPopupMenu popupMenu = new JPopupMenu();
125        addPopup(list, popupMenu);
126
127        JMenuItem mntmAdd = new JMenuItem(addAction);
128        mntmAdd.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D, InputEvent.META_MASK));
129        popupMenu.add(mntmAdd);
130
131        JMenuItem mntmRemove = new JMenuItem(removeAction);
132        popupMenu.add(mntmRemove);
133
134        JSeparator separator = new JSeparator();
135        popupMenu.add(separator);
136
137        JMenuItem mntmClearSelection = new JMenuItem(clearSelectionAction);
138        popupMenu.add(mntmClearSelection);
139
140        selectionModel = list.getSelectionModel();
141        selectionModel.addListSelectionListener(new ListSelectionListener()
142        {
143            @Override
144            public void valueChanged(ListSelectionEvent e)
145            {
146                ListSelectionModel lsm = (ListSelectionModel) e.getSource();
147
148                int firstIndex = e.getFirstIndex();
149                int lastIndex = e.getLastIndex();
150                boolean isAdjusting = e.getValueIsAdjusting();
151                /*
152                 * isAdjusting remains true while events like drag n drop are
153                 * still processed and becomes false afterwards.
154                 */
155                if (¬isAdjusting)
156                {
157                    output.append("Event for indexes " + firstIndex + " - "
158                        + lastIndex + "; selected indexes:");
159
160                    if (lsm.isSelectionEmpty())
161                    {
162                        removeAction.setEnabled(false);
163                        clearSelectionAction.setEnabled(false);
164                        output.append(" <none>");
165                    }
166                    else
167                    {
168                        removeAction.setEnabled(true);
169                        clearSelectionAction.setEnabled(true);
170                        // Find out which indexes are selected.
171                        int minIndex = lsm.getMinSelectionIndex();
172                        int maxIndex = lsm.getMaxSelectionIndex();
173                        for (int i = minIndex; i ≤ maxIndex; i++)
174                        {
175                            if (lsm.isSelectedIndex(i))
176                            {
177                                output.append(" " + i);
178                            }
179                        }
180                    }
```

```
181                     output.append(newline);
182                 }
183                 else
184                 {
185                     // Still adjusting ...
186                     output.append("Processing ..." + newline);
187                 }
188             }
189         });
190     }
191
192     /**
193      * Color Text renderer for drawing list's elements in colored text
194      * @author davidroussel
195      */
196     public static class ColorTextRenderer extends JLabel
197         implements ListCellRenderer<String>
198     {
199         private Color color = null;
200
201         /**
202          * Customized rendering for a ListCell with a color obtained from
203          * the hashCode of the string to display
204          * @see
205          * javax.swing.ListCellRenderer#getListCellRendererComponent(javax.swing
206          * .JList, java.lang.Object, int, boolean, boolean)
207          */
208         @Override
209         public Component getListCellRendererComponent(
210             JList<? extends String> list, String value, int index,
211             boolean isSelected, boolean cellHasFocus)
212         {
213             color = list.getForeground();
214             if (value ≠ null)
215             {
216                 if (value.length() > 0)
217                 {
218                     color = new Color(value.hashCode()).darker();
219                 }
220             }
221             setText(value);
222             if (isSelected)
223             {
224                 setBackground(color);
225                 setForeground(list.getSelectionForeground());
226             }
227             else
228             {
229                 setBackground(list.getBackground());
230                 setForeground(color);
231             }
232             setEnabled(list.isEnabled());
233             setFont(list.getFont());
234             setOpaque(true);
235             return this;
236         }
237     }
238
239     /**
240      * Adds a popup menu to a component
241      * @param component the parent component of the popup menu
242      * @param popup the popup menu to add
243      */
244     private static void addPopup(Component component, final JPopupMenu popup)
245     {
246         component.addMouseListener(new MouseAdapter()
247         {
248             @Override
249             public void mousePressed(MouseEvent e)
250             {
251                 if (e.isPopupTrigger())
252                 {
253                     showMenu(e);
254                 }
255             }
256
257             @Override
258             public void mouseReleased(MouseEvent e)
259             {
260                 if (e.isPopupTrigger())
261                 {
262                     showMenu(e);
263                 }
264             }
265
266             private void showMenu(MouseEvent e)
267             {
268                 popup.show(e.getComponent(), e.getX(), e.getY());
269             }
270         });
```



```
271     }
272
273     private class RemoveItemAction extends AbstractAction
274     {
275         public RemoveItemAction()
276         {
277             putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_R, InputEvent.META_MASK));
278             putValue(SMALL_ICON, new ImageIcon(ListExampleFrame.class.getResource("/examples/icons/remove
    _user-16.png")));
279             putValue(LARGE_ICON_KEY, new ImageIcon(ListExampleFrame.class.getResource("/examples/icons/re
    move_user-32.png")));
280             putValue(NAME, "Remove");
281             putValue(SHORT_DESCRIPTION, "Removes item from list");
282         }
283
284         @Override
285         public void actionPerformed(ActionEvent e)
286         {
287             output.append("Remove action triggered for indexes : ");
288             int minIndex = selectionModel.getMinSelectionIndex();
289             int maxIndex = selectionModel.getMaxSelectionIndex();
290             Stack<Integer> toRemove = new Stack<Integer>();
291             for (int i = minIndex; i ≤ maxIndex; i++)
292             {
293                 if (selectionModel.isSelectedIndex(i))
294                 {
295                     output.append(" " + i);
296                     toRemove.push(new Integer(i));
297                 }
298             }
299             output.append(newline);
300             while (¬toRemove.isEmpty())
301             {
302                 int index = toRemove.pop().intValue();
303                 output.append("removing element: "
304                     + elements.getElementAt(index) + newline);
305                 elements.remove(index);
306             }
307         }
308     }
309
310     private class ClearSelectionAction extends AbstractAction
311     {
312         public ClearSelectionAction()
313         {
314             putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_X, InputEvent.META_MASK));
315             putValue(LARGE_ICON_KEY, new ImageIcon(ListExampleFrame.class.getResource("/examples/icons/de
    lete_sign-32.png")));
316             putValue(SMALL_ICON, new ImageIcon(ListExampleFrame.class.getResource("/examples/icons/delete_s
    ign-16.png")));
317             putValue(NAME, "Clear selection");
318             putValue(SHORT_DESCRIPTION, "Unselect selected items");
319         }
320
321         @Override
322         public void actionPerformed(ActionEvent e)
323         {
324             output.append("Clear selection action triggered" + newline);
325             selectionModel.clearSelection();
326         }
327     }
328
329     private class AddAction extends AbstractAction
330     {
331         public AddAction()
332         {
333             putValue(ACCELERATOR_KEY, KeyStroke.getKeyStroke(KeyEvent.VK_A, InputEvent.META_MASK));
334             putValue(SMALL_ICON, new ImageIcon(ListExampleFrame.class.getResource("/examples/icons/add_use
    r-16.png")));
335             putValue(LARGE_ICON_KEY, new ImageIcon(ListExampleFrame.class.getResource("/examples/icons/ad
    d_user-32.png")));
336             putValue(NAME, "Add...");
337             putValue(SHORT_DESCRIPTION, "Add item");
338         }
339
340         @Override
341         public void actionPerformed(ActionEvent e)
342         {
343             output.append("Add action triggered" + newline);
344             String inputValue = JOptionPane.showInputDialog("New item name");
345             if (inputValue ≠ null)
346             {
347                 if (inputValue.length() > 0)
348                 {
349                     elements.addElement(inputValue);
350                 }
351             }
352         }
353     }
354 }
```

```
1   package logger;
2
3   import java.io.IOException;
4   import java.util.logging.FileHandler;
5   import java.util.logging.Handler;
6   import java.util.logging.Level;
7   import java.util.logging.Logger;
8   import java.util.logging.SimpleFormatter;
9
10  /**
11   * Logger Factory
12   * @author davidroussel
13   */
14  public class LoggerFactory
15  {
16      /**
17       * Factory simple pour un logger de console
18       * @param client la classe cliente du logger. utilisÃ©e pour donner un nom au
19       * logger
20       * @param le niveau de log
21       * @return un logger simple utilisant la console
22       * @throws IOException
23       */
24      public static <E> Logger getConsoleLogger(Class<E> client, Level level)
25      {
26          Logger logger = null;
27          try
28          {
29              logger = getLogger(client, true, null, false, null, level);
30
31          }
32          catch (IOException e)
33          {
34              System.err.println("getConsoleLogger: impossible file IO error");
35              e.printStackTrace();
36              System.exit(e.hashCode());
37          }
38
39          return logger;
40      }
41
42      /**
43       * Factory pour obtenir un logger ayant un parent spÃ©cifique
44       * @param client la classe cliente du logger. utilisÃ©e pour donner un nom au
45       * logger
46       * @param parentLogger le logger parent
47       * @param level le niveau de log
48       * @return un logger ayant pour parent le parentLogger
49       */
50      public static <E> Logger getParentLogger(Class<E> client,
51                                               Logger parentLogger,
52                                               Level level)
53      {
54          Logger logger = null;
55          try
56          {
57              logger = getLogger(client, true, null, false, parentLogger, level);
58          }
59          catch (IOException e)
60          {
61              System.err.println("getParentLogger: impossible file IO error");
62              e.printStackTrace();
63              System.exit(e.hashCode());
64          }
65
66          return logger;
67      }
68
69      /**
70       * Factory pour obtenir un logger dans un fichier de log
71       * @param client la classe cliente du logger. utilisÃ©e pour donner un nom au
72       * logger
73       * @param fileName nom du fichier de log
74       * @param xmlFormat formattage du fichier de log en XML
75       * @param level le niveau de log
76       * @return un nouveau logger vers un fichier de log
77       * @throws IOException si l'on arrive pas Ã  ouvrir le fichier de log
78       */
79      public static <E> Logger getFileLogger(Class<E> client,
80                                             String fileName,
81                                             boolean xmlFormat,
82                                             Level level)
83          throws IOException
84      {
85          return getLogger(client, false, fileName, xmlFormat, null, level);
86      }
87
88      /**
89       * Factory gÃ©nÃ©rale pour obtenir un logger
90       * @param client la classe cliente du logger. utilisÃ©e pour donner un nom au
```



```
91       * logger
92       * @param verbose affichage des logs dans la console
93       * @param logFileName fichier de log (pas de fichier de log si null)
94       * @param xmlFormat formattage du fichier de log en XML
95       * @param parentLogger parent logger. Si le parent logger est non null
96       * l'argument verbose n'est pas pris en compte
97       * @param level le niveau de log
98       * @return un nouveau logger si les paramÃ¨tres le permettent ou bien null si
99       * ce n'est pas le cas
100      * @throws IOException si l'on arrive pas Ã  ouvrir le fichier de log
101      */
102     public static <E> Logger getLogger(Class<E> client,
103                                        boolean verbose,
104                                        String logFileName,
105                                        boolean xmlFormat,
106                                        Logger parentLogger,
107                                        Level level)
108         throws IOException
109     {
110         Logger logger = null;
111
112         if (verbose ∨ (logFileName ≠ null) ∨ (parentLogger ≠ null))
113         {
114             if (client ≠ null)
115             {
116                 String canonicalName = client.getCanonicalName();
117                 logger = Logger.getLogger(canonicalName);
118
119                 if (parentLogger ≠ null)
120                 {
121                     logger.setParent(parentLogger);
122                 }
123                 else
124                 {
125                     if (¬verbose)
126                     {
127                         /*
128                          * On ne veut pas que les messages de log aillent dans
129                          * la console.
130                          */
131                         logger.setUseParentHandlers(false);
132                     }
133                 }
134
135                 if (logFileName ≠ null)
136                 {
137                     String filename = logFileName;
138                     if (xmlFormat)
139                     {
140                         if (¬logFileName.contains(new String("xml")))
141                         {
142                             filename = logFileName + ".xml";
143                         }
144                     }
145
146                     // Ajout d'un fileHandler au logger
147                     try
148                     {
149                         Handler handler = new FileHandler(filename);
150                         if (¬xmlFormat)
151                         {
152                             // par dÃ©faut le formattage fichier sera en XML
153                             // il faut donc remettre en place un formatteur
154                             // simple
155                             handler.setFormatter(new SimpleFormatter());
156                         }
157
158                         // Ajout de ce filehandler au logger
159                         logger.addHandler(handler);
160                         logger.info("log file created");
161                     }
162                     catch (IllegalArgumentException e)
163                     {
164                         String message = "Empty log file name";
165                         logger.severe(message);
166                         logger.severe(e.getLocalizedMessage());
167                         throw e;
168                     }
169                     catch (SecurityException e)
170                     {
171                         String message =
172                             "Do not have privileges to open log file "
173                                 + logFileName;
174                         logger.warning(message);
175                         logger.warning(e.getLocalizedMessage());
176                     }
177                     catch (IOException e)
178                     {
179                         String message = "Error opening file " + logFileName;
180                         logger.severe(message);
```

```
181                         logger.severe(e.getLocalizedMessage());
182                         throw e;
183                     }
184                 }
185             }
186             else
187             {
188                 if (parentLogger ≠ null)
189                 {
190                     logger = parentLogger;
191                 }
192             }
193         }
194 
195         if (logger ≠ null)
196         {
197             logger.info("Logger ready");
198             logger.setLevel(level);
199         }
200 
201         return logger;
202     }
203 }
```



```
1 /**
2  * Classe contenant une factory permettant d'instancier plusieurs types de loggers
3  * Un logger permet d'envoyer des messagde de logs (soit dans la console, soit
4  * dans une fichier).
5  * @author davidroussel
6  */
7 package logger;
```

```java
package models;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.EnumSet;
import java.util.Iterator;
import java.util.Set;

/**
 * Classe contenant un message envoyÃ© par le serveur.
 * Un message d'un utilisateur est caractÃ©risÃ© par :
 * <ul>
 *  <li>la date d'arrivÃ©e du message</li>
 *  <li>le contenu du message></li>
 *  <li>(eventuellement) un author</li>
 * </ul>
 * Les message peuvent Ãªtre comparÃ©s entre eux pour obtenir l'ordre des messages
 * avec la mÃ©thode compareTo(Message m). Les critÃ¨re d'ordre des messages
 * peuvent Ãªtre customizÃ©s.
 * @author davidroussel
 */
public class Message implements Serializable, Comparable<Message>
{
    public enum MessageOrder
    {
        AUTHOR,
        DATE,
        CONTENT;

        /**
         * Affichage d'un critÃ¨re d'ordre
         * @return une chaine de caractÃ¨re reprÃ©sentant un critÃ¨re d'ordre
         */
        @Override
        public String toString()
        {
            switch (this)
            {
                case AUTHOR:
                    return new String("Author");
                case DATE:
                    return new String("Date");
                case CONTENT:
                    return new String("Content");
            }
            throw new AssertionError("UserMessage::Order: unknown type: " + this);
        }
    }

    /**
     * Ensemble des critÃ¨res de tri [InitialisÃ© Ã  la date seule]
     * Les critÃ¨res de tri peuvent contenir une et une seule instance
     * des diffÃ©rents Ã©lÃ©ments de {@link MessageOrder} dans n'importe quel
     * ordre.
     */
    protected static Set<MessageOrder> orders = EnumSet.of(MessageOrder.DATE);

    /**
     * Ajout d'un critÃ¨re de tri aux critÃ¨res de tri
     * @param o le critÃ¨re Ã  ajouter
     * @return true si le critÃ¨re de tri n'Ã©tait pas dÃ©jÃ  prÃ©sent dans
     * l'ensemble et qu'il a pu Ãªtre ajoutÃ©, false sinon.
     */
    public static boolean addOrder(MessageOrder o)
    {
        return orders.add(o);
    }

    /**
     * Retrait d'un critÃ¨re de tri aux critÃ¨res de tri
     * @param o le critÃ¨re de tri Ã  retirer
     * @return true si le criÃ¨re de tri Ã©tait prÃ©sent dans l'ensemble des
     * critÃ¨res et qu'il a Ã©tÃ© retirÃ©, false sinon.
     */
    public static boolean removeOrder(MessageOrder o)
    {
        return orders.remove(o);
    }

    /**
     * Effacement de l'ensemble des critÃ¨res de tri
     */
    public static void clearOrders()
    {
        orders.clear();
    }

    /**
```



```java
     * La date d'arrivÃ©e du message
     */
    private Date date;

    /**
     * Le contenu du message
     */
    private String content;

    /**
     * L'auteur du message (optionnel).
     * Un message du serveur peut Ã©ventuellement ne pas avoir d'auteur
     */
    private String author;

    /**
     * Formatteur pour l'affichage de la date des messages
     */
    protected static SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

    /**
     * Constructeur valuÃ© d'un message
     * @param date la date d'arrivÃ©e du message
     * @param content le contenu du message
     * @param author l'auteur du message
     */
    public Message(Date date, String content, String author)
    {
        // date ne doit pas Ãªtre null
        this.date = (date ≠ null ? date : Calendar.getInstance().getTime());
        // content ne doit pas Ãªtre null
        this.content = (content ≠ null ? content : new String());
        this.author = author;
    }

    /**
     * Constructeur valuÃ© d'un message
     * @param date la date d'arrivÃ©e du message
     * @param content le contenu du message
     */
    public Message(Date date, String content)
    {
        this(date, content, null);
    }

    /**
     * Constructeur valuÃ© d'un message.
     * La date d'arrivÃ©e est implicitement initialisÃ©e Ã  "maintenant" en
     * utilisant le calendrier
     * @param content le contenu du message
     * @param author l'auteur du message
     * @see Calendar#getInstance()
     * @see Calendar#getTime()
     */
    public Message(String content, String author)
    {
        this(null, content, author);
    }

    /**
     * Constructeur valuÃ© d'un message.
     * La date d'arrivÃ©e est implicitement initialisÃ©e Ã  "maintenant" en
     * utilisant le calendrier
     * @param content le contenu du message
     * @see Calendar#getInstance()
     * @see Calendar#getTime()
     */
    public Message(String content)
    {
        this(content, null);
    }

    /**
     * Accesseur en lecture de la date du message
     * @return la date du message
     */
    public Date getDate()
    {
        return date;
    }

    /**
     * Accesseur en lecture de la chaÃ®ne formattÃ©e de la date du message
     * @return la chaÃ®ne formattÃ©e de la date du message
     */
    public String getFormattedDate()
    {
        return dateFormat.format(date);
    }

```

```
181     /**
182      * Accesseur en lecture du contenu du message
183      * @return le contenu du message
184      */
185     public String getContent()
186     {
187         return content;
188     }
189
190     /**
191      * Accesseur en lecture de l'auteur du message
192      * @return l'auteur du message ou bien null s'il s'agit d'un
193      * message direct du serveur
194      */
195     public String getAuthor()
196     {
197         return author;
198     }
199
200     /**
201      * Indique si un message Ã  un auteur (ce qui n'est le cas que pour les
202      * messages envoyÃ©s par les utilisateurs au serveur, les messages de
203      * contrÃ´le diffusÃ©s par le serveur n'ont pas d'auteurs.)
204      * @return true si le message a un auteur, false autrement
205      */
206     public boolean hasAuthor()
207     {
208         return author ≠ null;
209     }
210
211     /**
212      * Accesseur en lecture du formatteur de date des messages
213      * @return le formateur de date des messages
214      */
215     public static SimpleDateFormat getDateFormat()
216     {
217         return dateFormat;
218     }
219
220     /**
221      * @return le hashcode du message basÃ© sur le hashcode de sa date, de son
222      * auteur et de son contenu (evt utilisÃ© dans un hashset de messages)
223      */
224     @Override
225     public int hashCode()
226     {
227         final int prime = 31;
228         int hash = date.hashCode();
229         hash = (prime * hash) + content.hashCode();
230         if (author ≠ null)
231         {
232             hash = (prime * hash) + author.hashCode();
233         }
234         return hash;
235     }
236
237     /**
238      * Comparaison binaire avec un autre objet
239      * @param obj l'autre objet Ã  comparer
240      * @return true si l'autre objet est un message avec les mÃªmes attributs
241      * @note on peut utiliser la comparaison 3-way pour effectivement comparer
242      * deux messages;
243      */
244     @Override
245     public boolean equals(Object obj)
246     {
247         if (obj ≡ null)
248         {
249             return false;
250         }
251
252         if (obj ≡ this)
253         {
254             return true;
255         }
256
257         if (obj instanceof Message)
258         {
259             Message m = (Message) obj;
260
261             if (date.equals(m.date))
262             {
263                 if (content.equals(m.content))
264                 {
265                     if (author ≠ null)
266                     {
267                         return author.equals(m.author);
268                     }
269                     else
270                     {
```



```
271                         return m.author ≡ null;
272                     }
273                 }
274             }
275         }
276
277         return false;
278     }
279
280     /**
281      * Affichage du message sous forme de chaÃ®ne de caractÃ¨res
282      * @return une chaÃ®ne de caractÃ¨re reprÃ©sentant le message sous la forme
283      * [yyyy/mm/dd HH:MM:SS] author > message content
284      */
285     @Override
286     public String toString()
287     {
288         StringBuffer sb = new StringBuffer("[");
289
290         sb.append(dateFormat.format(date));
291         sb.append("] ");
292         if (author ≠ null)
293         {
294             sb.append(author);
295             sb.append(" > ");
296         }
297         sb.append(content);
298
299         return sb.toString();
300     }
301
302     @Override
303     public int compareTo(Message m)
304     {
305         int compare = 0;
306         if (orders.isEmpty())
307         {
308             // l'ordre par dÃ©faut est la date du message
309             compare = date.compareTo(m.date);
310         }
311         else
312         {
313             for (Iterator<MessageOrder> it = orders.iterator(); it.hasNext();)
314             {
315                 MessageOrder criterium = it.next();
316                 switch (criterium)
317                 {
318                     case AUTHOR:
319                         if (author ≠ null)
320                         {
321                             if(m.author ≠ null)
322                             {
323                                 compare = author.compareTo(m.author);
324                             }
325                             else
326                             {
327                                 /*
328                                  * Un message avec auteur sera considÃ©rÃ© comme
329                                  * supÃ©rieur Ã  un message sans auteur
330                                  */
331                                 compare = 1;
332                             }
333                         }
334                         else // author == null
335                         {
336                             if (m.author ≠ null)
337                             {
338                                 // un message sans auteur sera considÃ©rÃ© comme
339                                 // infÃ©rieur Ã  un message avec auteur
340                                 compare = -1;
341                             }
342                             else
343                             {
344                                 compare = 0;
345                             }
346                         }
347                         break;
348                     case DATE:
349                         compare = date.compareTo(m.date);
350                         break;
351                     case CONTENT:
352                         compare = content.compareTo(m.content);
353                     default:
354                         break;
355                 }
356                 // Si le critÃ¨re courant permet de diffÃ©rentier les messages
357                 // on renvoie sa valeur tout de suite.
358                 if (compare ≠ 0)
359                 {
360                     break;
```

```
361                     }
362                 }
363                 // On a terminÃ© la boucle sans avoir renvoyÃ© une valeur != 0,
364                 // tous les critÃ¨res de comparaison ont Ã©tÃ© 0 (valeurs Ã©gales)
365             }
366             return compare;
367         }
368 }
```



```
1  package models;
2
3  import javax.swing.AbstractListModel;
4
5  /**
6   * ListModel contenant des noms uniques (toujours triÃ© grÃ¢ce Ã  un TreeSet par
7   * exemple).
8   * L'accÃ¨s Ã  la liste de noms est thread safe (c'Ã  d : plusieurs threads peuvent
9   * accÃ©der concurrentiellement Ã  la liste de noms sans que celle ci se retrouve
10  * dans un Ã©tat incohÃ©rent) : Les modifications du Set interne se font
11  * toujours dans un bloc synchronized.
12  * L'ajout ou le retrait d'un Ã©lÃ©ment dans l'ensemble de nom est accompagnÃ©
13  * d'un fireContentsChanged sur l'ensemble des Ã©lÃ©ments de la liste (Ã  cause
14  * du tri implicite des Ã©lÃ©ments) ce qui permet au List Model de notifier
15  * tout widget dans lequel serait contenu ce ListModel.
16  * @see {@link javax.swing.AbstractListModel}
17  */
18 public class NameSetListModel extends AbstractListModel<String>
19 {
20     /**
21      * Ensemble de noms triÃ©s
22      */
23     // TODO nameSet ...
24
25     /**
26      * Constructeur
27      */
28     public NameSetListModel()
29     {
30         // TODO nameSet = ...
31     }
32
33     /**
34      * Ajout d'un Ã©lÃ©ment
35      * @param value la valeur Ã  ajouter
36      * @return true si l'Ã©lÃ©ment Ã  ajouter est non null et qu'il n'Ã©tait pas
37      * dÃ©jÃ  prÃ©sent dans l'ensemble et false sinon.
38      */
39     public boolean add(String value)
40     {
41         // TODO Replace with implementation ...
42         return false;
43     }
44
45     /**
46      * Teste si l'ensemble de noms contient le nom passÃ© en argument
47      * @param value le nom Ã  rechercher
48      * @return true si l'ensemble de noms contient "value", false sinon.
49      */
50     public boolean contains(String value)
51     {
52         // TODO Replace with implementation ...
53         return false;
54     }
55
56     /**
57      * Retrait de l'Ã©lÃ©ment situÃ© Ã  l'index index
58      * @param index l'index de l'Ã©lÃ©ment Ã  supprimer
59      * @return true si l'Ã©lÃ©ment a Ã©tÃ© supprimÃ©, false sinon
60      */
61     public boolean remove(int index)
62     {
63         // TODO Replace with implementation ...
64         return false;
65     }
66
67     /* (non-Javadoc)
68      * @see javax.swing.ListModel#getSize()
69      */
70     @Override
71     public int getSize()
72     {
73         // TODO Replace with implementation ...
74         return 0;
75     }
76
77     /* (non-Javadoc)
78      * @see javax.swing.ListModel#getElementAt(int)
79      */
80     @Override
81     public String getElementAt(int index)
82     {
83         // TODO Replace with implementation ...
84         return null;
85     }
86 }
```

```java
package models;

/**
 * Sous-package contenant les classes des modÃ¨les de donnÃ©es manipulÃ©s.
 * En l'occurrence
 * <ul>
 * <li>{@link models.Message} une classe reprÃ©sentant les messages envoyÃ©s
 * par les utilisateurs</li>
 * <li>{@link models.NameSetListModel} une classe reprÃ©sentant des noms
 * d'utilisateurs uniques et toujours triÃ©s dans une liste d'utilisateurs (par
 * exemple une {@link javax.swing.JList})</li>
 * <li> {@link models.AuthorListFilter} une classe permettant de filtrer
 * un flux de messages en vÃ©rifiant si un message particulier contient un
 * auteur qui fait partie de la liste des auteurs rÃ©fÃ©rencÃ©s dans ce filtre</li>
 * </ul>
 */
```



```java
package widgets;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.HeadlessException;
import java.io.IOException;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.io.PrintWriter;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.swing.JFrame;
import javax.swing.JTextPane;
import javax.swing.text.Style;
import javax.swing.text.StyledDocument;

import logger.LoggerFactory;

public abstract class AbstractClientFrame extends JFrame implements Runnable
{
    /**
     * Etat d'exÃ©cution du run pour Ã©couter les messages en provenance du
     * serveur
     */
    protected Boolean commonRun;

    /**
     * Flux d'entrÃ©e pour lire les messages du serveur
     */
    protected final PipedInputStream inPipe;

    /**
     * Ecrivain vers le flux de sortie Ecrit le contenu du {@link #txtFieldSend}
     * dans le {@link #outPipe}
     */
    protected final PrintWriter outPW;

    /**
     * Flux de sortie pour envoyer le contenu du message
     */
    protected final PipedOutputStream outPipe;

    /**
     * Logger pour afficher les messages ou les rediriger dans un fichier de log
     */
    protected Logger logger;

    /**
     * Le document sous-jacent d'un {@link JTextPane} dans lequel on Ã©crira
     * les messages
     */
    protected StyledDocument document;

    /**
     * Le style du document {@link #document}
     */
    protected Style documentStyle;

    /**
     * La couleur par dÃ©faut du texte {@link #documentStyle}
     */
    protected Color defaultColor;

    /**
     * Constructeur [protÃ©gÃ©] de la fenÃªtre de chat abstraite
     * @param name le nom de l'utilisateur
     * @param host l'hÃ´te sur lequel on est connectÃ©
     * @param commonRun Ã©tat d'exÃ©cution des autres threads du client
     * @param parentLogger le logger parent pour les messages
     * @throws HeadlessException
     */
    protected AbstractClientFrame(String name,
                                  String host,
                                  Boolean commonRun,
                                  Logger parentLogger)
        throws HeadlessException
    {
        // ----------------------------------------------------------------
        // Logger
        //-----------------------------------------------------------------
        logger = LoggerFactory.getParentLogger(getClass(),
                                               parentLogger,
                                               (parentLogger == null ?
                                                Level.WARNING :
                                                parentLogger.getLevel()));

        // ----------------------------------------------------------------
        // Common run avec d'autres threads
```

```
91          //-------------------------------------------------------------------
92          if (commonRun ≠ null)
93          {
94              this.commonRun = commonRun;
95          }
96          else
97          {
98              this.commonRun = Boolean.TRUE;
99          }
100
101         // -------------------------------------------------------------------
102         // Flux d'IO
103         //-------------------------------------------------------------------
104         inPipe = new PipedInputStream();
105
106         outPipe = new PipedOutputStream();
107         outPW = new PrintWriter(outPipe, true);
108         if (outPW.checkError())
109         {
110             logger.warning("ClientFrame: Output PrintWriter has errors");
111         }
112
113         // -------------------------------------------------------------------
114         // Window setup
115         //-------------------------------------------------------------------
116         if (name ≠ null)
117         {
118             setTitle(name);
119         }
120
121         setPreferredSize(new Dimension(400, 200));
122         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
123
124         document = null;
125         documentStyle = null;
126         defaultColor = Color.BLACK;
127     }
128
129     /**
130      * Envoi d'un message. Envoi d'un message dans le {@link #outPipe} (si celui
131      * ci est non null) en utilisant le {@link #outPW}
132      * @param le message Ã  envoyer
133      */
134     protected void sendMessage(String message)
135     {
136         logger.info("ClientFrame::sendMessage writing out: "
137             + (message ≡ null ? "NULL" : message));
138         /*
139          * TODO envoi du message dans le outPW et vÃ©rification du statut
140          * d'erreur du #outPW (si c'est le cas on ajoute un warning au logger).
141          */
142         if (message ≠ null)
143         {
144             outPW.println(message);
145             if (outPW.checkError())
146             {
147                 logger.warning("ClientFrame::sendMessage: error writing");
148             }
149         }
150     }
151
152     /**
153      * Couleur d'un texte d'aprÃ¨s le contenu du texte.
154      * @param name le texte
155      * @return un couleur alÃ©atoire initialisÃ©e avec le hashCode du texte ou
156      * bien null si name est vide ou null
157      */
158     protected Color getColorFromName(String name)
159     {
160         /*
161          * TODO crÃ©er et renvoyer une couleur (pas trop claire) d'aprÃ¨s le nom
162          * fourni en argument. Calcule une couleur en utilisant le hashCode du
163          * texte pour initialiser un Random, le nextInt de ce Random nous
164          * fournira alors un entier utilisÃ© pour crÃ©er une Color. On pourra
165          * Ã©ventuellement utiliser la mÃ©thode darker() sur cette couleur pour
166          * Ã©viter les couleurs trop claires qui se voient mal sur fond blanc.
167          */
168         Random rand;
169         if (name ≠ null)
170         {
171             if (name.length() > 0)
172             {
173                 rand = new Random(name.hashCode());
174                 return new Color(rand.nextInt()).darker();
175                 // return new Color(name.hashCode()).darker();
176             }
177         }
178
179         return null;
180     }
```



```
181
182     /**
183      * Accesseur en lecture de l' {@link #inPipe} pour y connecter un
184      * {@link PipedOutputStream}
185      * @return l'inPipe sur lequel on lit
186      */
187     public PipedInputStream getInPipe()
188     {
189         return inPipe;
190     }
191
192     /**
193      * Accesseur en lecture de l' {@link #outPipe} pour y connecter un
194      * {@link PipedInputStream}
195      * @return l'outPipe sur lequel on Ã©crit
196      */
197     public PipedOutputStream getOutPipe()
198     {
199         return outPipe;
200     }
201
202     /**
203      * Fermeture de la fenÃ¨tre et des flux Ã  la fin de l'exÃ©cution
204      */
205     public void cleanup()
206     {
207         logger.info("ClientFrame::cleanup: closing window ... ");
208         dispose();
209
210         logger.info("ClientFrame::cleanup: closing output print writer ... ");
211         outPW.close();
212
213         logger.info("ClientFrame::cleanup: closing output stream ... ");
214         try
215         {
216             outPipe.close();
217         }
218         catch (IOException e)
219         {
220             logger.warning("ClientFrame::cleanup: failed to close output stream"
221                 + e.getLocalizedMessage());
222         }
223
224         logger.info("ClientFrame::cleanup: closing input stream ... ");
225         try
226         {
227             inPipe.close();
228         }
229         catch (IOException e)
230         {
231             logger.warning("ClientFrame::cleanup: failed to close input stream"
232                 + e.getLocalizedMessage());
233         }
234     }
235 }
```

```
1   package widgets;
2
3   import java.awt.BorderLayout;
4   import java.awt.Color;
5   import java.awt.Component;
6   import java.awt.HeadlessException;
7   import java.awt.event.ActionEvent;
8   import java.awt.event.InputEvent;
9   import java.awt.event.KeyEvent;
10  import java.awt.event.WindowAdapter;
11  import java.awt.event.WindowEvent;
12  import java.io.BufferedReader;
13  import java.io.IOException;
14  import java.io.InputStreamReader;
15  import java.util.logging.Logger;
16
17  import javax.swing.AbstractAction;
18  import javax.swing.Box;
19  import javax.swing.ImageIcon;
20  import javax.swing.JButton;
21  import javax.swing.JFrame;
22  import javax.swing.JLabel;
23  import javax.swing.JMenu;
24  import javax.swing.JMenuBar;
25  import javax.swing.JMenuItem;
26  import javax.swing.JPanel;
27  import javax.swing.JScrollPane;
28  import javax.swing.JSeparator;
29  import javax.swing.JTextField;
30  import javax.swing.JTextPane;
31  import javax.swing.JToolBar;
32  import javax.swing.KeyStroke;
33  import javax.swing.text.BadLocationException;
34  import javax.swing.text.DefaultCaret;
35  import javax.swing.text.StyleConstants;
36
37  import chat.Vocabulary;
38
39  /**
40   * FenÃªtre d'affichae de la version GUI texte du client de chat.
41   * @author davidroussel
42   */
43  public class ClientFrame extends AbstractClientFrame
44  {
45      /**
46       * Lecteur de flux d'entrÃ©e. Lit les donnÃ©es texte du {@link #inPipe} pour
47       * les afficher dans le {@link #document}
48       */
49      private BufferedReader inBR;
50
51      /**
52       * Le label indiquant sur quel serveur on est connectÃ©
53       */
54      protected final JLabel serverLabel;
55
56      /**
57       * La zone du texte Ã  envoyer
58       */
59      protected final JTextField sendTextField;
60
61      /**
62       * Actions Ã  rÃ©aliser lorsque l'on veut effacer le contenu du document
63       */
64      private final ClearAction clearAction;
65
66      /**
67       * Actions Ã  rÃ©aliser lorsque l'on veut envoyer un message au serveur
68       */
69      private final SendAction sendAction;
70
71      /**
72       * Actions Ã  rÃ©aliser lorsque l'on veut envoyer un message au serveur
73       */
74      protected final QuitAction quitAction;
75
76      /**
77       * RÃ©fÃ©rence Ã  la fenÃªtre courante (Ã  utiliser dans les classes internes)
78       */
79      protected final JFrame thisRef;
80
81      /**
82       * Constructeur de la fenÃªtre
83       * @param name le nom de l'utilisateur
84       * @param host l'hÃ´te sur lequel on est connectÃ©
85       * @param commonRun Ã©tat d'exÃ©cution des autres threads du client
86       * @param parentLogger le logger parent pour les messages
87       * @throws HeadlessException
88       */
89      public ClientFrame(String name,
90                         String host,
```



```
91                         Boolean commonRun,
92                         Logger parentLogger)
93          throws HeadlessException
94      {
95          super(name, host, commonRun, parentLogger);
96          thisRef = this;
97
98          // ------------------------------------------------------------------
99          // Flux d'IO
100         //------------------------------------------------------------------
101         /*
102          * Attention, la crÃ©ation du flux d'entrÃ©e doit (Ã©ventuellement) Ãªtre
103          * reportÃ©e jusqu'au lancement du run dans la mesure oÃ¹ le inPipe
104          * peut ne pas encore Ãªtre connectÃ© Ã  un PipedOutputStream
105          */
106
107         // ------------------------------------------------------------------
108         // CrÃ©ation des actions send, clear et quit
109         // ------------------------------------------------------------------
110
111         sendAction = new SendAction();
112         clearAction = new ClearAction();
113         quitAction = new QuitAction();
114
115
116         /*
117          * Ajout d'un listener pour fermer correctement l'application lorsque
118          * l'on ferme la fenÃªtre. WindowListener sur this
119          */
120         addWindowListener(new FrameWindowListener());
121
122         // ------------------------------------------------------------------
123         // Widgets setup (handled by Window builder)
124         // ------------------------------------------------------------------
125
126         JToolBar toolBar = new JToolBar();
127         toolBar.setFloatable(false);
128         getContentPane().add(toolBar, BorderLayout.NORTH);
129
130         JButton quitButton = new JButton(quitAction);
131         toolBar.add(quitButton);
132
133         JButton clearButton = new JButton(clearAction);
134         toolBar.add(clearButton);
135
136         Component toolBarSep = Box.createHorizontalGlue();
137         toolBar.add(toolBarSep);
138
139         serverLabel = new JLabel(host == null ? "" : host);
140         toolBar.add(serverLabel);
141
142         JPanel sendPanel = new JPanel();
143         getContentPane().add(sendPanel, BorderLayout.SOUTH);
144         sendPanel.setLayout(new BorderLayout(0, 0));
145         sendTextField = new JTextField();
146         sendTextField.setAction(sendAction);
147         sendPanel.add(sendTextField);
148         sendTextField.setColumns(10);
149
150         JButton sendButton = new JButton(sendAction);
151         sendPanel.add(sendButton, BorderLayout.EAST);
152
153         JScrollPane scrollPane = new JScrollPane();
154         getContentPane().add(scrollPane, BorderLayout.CENTER);
155
156         JTextPane textPane = new JTextPane();
157         textPane.setEditable(false);
158         // autoscroll textPane to bottom
159         DefaultCaret caret = (DefaultCaret) textPane.getCaret();
160         caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
161
162         scrollPane.setViewportView(textPane);
163
164         JMenuBar menuBar = new JMenuBar();
165         setJMenuBar(menuBar);
166
167         JMenu actionsMenu = new JMenu("Actions");
168         menuBar.add(actionsMenu);
169
170         JMenuItem sendMenuItem = new JMenuItem(sendAction);
171         actionsMenu.add(sendMenuItem);
172
173         JMenuItem clearMenuItem = new JMenuItem(clearAction);
174         actionsMenu.add(clearMenuItem);
175
176         JSeparator separator = new JSeparator();
177         actionsMenu.add(separator);
178
179         JMenuItem quitMenuItem = new JMenuItem(quitAction);
180         actionsMenu.add(quitMenuItem);
```

```
181
182         // ----------------------------------------------------------------------
183         // Documents
184         // rÃ©cupÃ©ration du document du textPane ainsi que du documentStyle et du
185         // defaultColor du document
186         //----------------------------------------------------------------------
187         document = textPane.getStyledDocument();
188         documentStyle = textPane.addStyle("New Style", null);
189         defaultColor = StyleConstants.getForeground(documentStyle);
190
191
192     }
193
194     /**
195      * Affichage d'un message dans le {@link #document}, puis passage Ã  la ligne
196      * (avec l'ajout de {@link Vocabulary#newLine})
197      * La partie "[yyyy/MM/dd HH:mm:ss]" correspond Ã  la date/heure courante
198      * obtenue grÃ¢ce Ã  un Calendar et est affichÃ©e avec la defaultColor alors
199      * que la partie "utilisateur > message" doit Ãªtre affichÃ©e avec une couleur
200      * dÃ©terminÃ©e d'après le nom d'utilisateur avec
201      * {@link #getColorFromName(String)}, le nom d'utilisateur est quant Ã  lui
202      * dÃ©terminÃ© d'après le message lui mÃªme avec {@link #parseName(String)}.
203      * @param message le message Ã  afficher dans le {@link #document}
204      * @throws BadLocationException si l'Ã©criture dans le document Ã©choue
205      * @see {@link examples.widgets.ExampleFrame#appendToDocument(String, Color)}
206      * @see java.text.SimpleDateFormat#SimpleDateFormat(String)
207      * @see java.util.Calendar#getInstance()
208      * @see java.util.Calendar#getTime()
209      * @see javax.swing.text.StyleConstants
210      * @see javax.swing.text.StyledDocument#insertString(int, String,
211      * javax.swing.text.AttributeSet)
212      */
213     protected void writeMessage(String message) throws BadLocationException
214     {
215         /*
216          * ajout du message "[yyyy/MM/dd HH:mm:ss] utilisateur > message" Ã 
217          * la fin du document avec la couleur dÃ©terminÃ©e d'après "utilisateur"
218          * (voir AbstractClientFrame#getColorFromName)
219          */
220         StringBuffer sb = new StringBuffer();
221
222         sb.append(message);
223         sb.append(Vocabulary.newLine);
224
225         // source et contenu du message avec la couleur du message
226         String source = parseName(message);
227         if ((source ≠ null) ∧ (source.length() > 0))
228         {
229             /*
230              * Changement de couleur du texte
231              */
232             StyleConstants.setForeground(documentStyle,
233                                          getColorFromName(source));
234         }
235
236         document.insertString(document.getLength(),
237                               sb.toString(),
238                               documentStyle);
239
240         // Retour Ã  la couleur de texte par dÃ©faut
241         StyleConstants.setForeground(documentStyle, defaultColor);
242
243     }
244
245     /**
246      * Recherche du nom d'utilisateur dans un message de type
247      * "utilisateur > message".
248      * parseName est utilisÃ© pour extraire le nom d'utilisateur d'un message
249      * afin d'utiliser le hashCode de ce nom pour crÃ©er une couleur dans
250      * laquelle
251      * sera affichÃ© le message de cet utilisateur (ainsi tous les messages d'un
252      * mÃªme utilisateur auront la mÃªme couleur).
253      * @param message le message Ã  parser
254      * @return le nom d'utilisateur s'il y en a un sinon null
255      */
256     protected String parseName(String message)
257     {
258         /*
259          * renvoyer la chaine correspondant Ã  la partie "utilisateur" dans
260          * un message contenant "utilisateur > message", ou bien null si cette
261          * partie n'existe pas.
262          */
263         if (message.contains(">") ∧ message.contains("]"))
264         {
265             int pos1 = message.indexOf(']');
266             int pos2 = message.indexOf('>');
267             try
268             {
269                 return new String(message.substring(pos1 + 1, pos2 - 1));
270             }
```



```
271             catch (IndexOutOfBoundsException iobe)
272             {
273                 logger.warning("ClientFrame::parseName: index out of bounds");
274                 return null;
275             }
276         }
277         else
278         {
279             return null;
280         }
281     }
282
283     /**
284      * Recherche du contenu du message dans un message de type
285      * "utilisateur > message"
286      * @param message le message Ã  parser
287      * @return le contenu du message s'il y en a un sinon null
288      */
289     protected String parseContent(String message)
290     {
291         if (message.contains(">"))
292         {
293             int pos = message.indexOf('>');
294             try
295             {
296                 return new String(message.substring(pos + 1, message.length()));
297             }
298             catch (IndexOutOfBoundsException iobe)
299             {
300                 logger
301                     .warning("ClientFrame::parseContent: index out of bounds");
302                 return null;
303             }
304         }
305         else
306         {
307             return message;
308         }
309     }
310
311     /**
312      * Listener lorsque le bouton #btnClear est activÃ©. Efface le contenu du
313      * {@link #document}
314      */
315     protected class ClearAction extends AbstractAction
316     {
317         /**
318          * Constructeur d'une ClearAction : met en place le nom, la description,
319          * le raccourci clavier et les small|Large icons de l'action
320          */
321         public ClearAction()
322         {
323             putValue(SMALL_ICON,
324                     new ImageIcon(ClientFrame.class
325                         .getResource("/icons/erase-16.png")));
326             putValue(LARGE_ICON_KEY,
327                     new ImageIcon(ClientFrame.class
328                         .getResource("/icons/erase-32.png")));
329             putValue(ACCELERATOR_KEY,
330                     KeyStroke.getKeyStroke(KeyEvent.VK_L,
331                                            InputEvent.META_MASK));
332             putValue(NAME, "Clear");
333             putValue(SHORT_DESCRIPTION, "Clear document content");
334         }
335
336         /**
337          * OpÃ©rations rÃ©alisÃ©es lorsque l'action est sollicitÃ©e
338          * @param e Ã©vÃ¨nement Ã  l'origine de l'action
339          * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
340          */
341         @Override
342         public void actionPerformed(ActionEvent e)
343         {
344             /*
345              * Effacer le contenu du document
346              */
347             try
348             {
349                 document.remove(0, document.getLength());
350             }
351             catch (BadLocationException ex)
352             {
353                 logger.warning("ClientFrame: clear doc: bad location");
354                 logger.warning(ex.getLocalizedMessage());
355             }
356         }
357     }
358
359     /**
360      * Action rÃ©alisÃ©e pour envoyer un message au serveur
```

```
361     */
362    protected class SendAction extends AbstractAction
363    {
364        /**
365         * Constructeur d'une SendAction : met en place le nom, la description,
366         * le raccourci clavier et les small|Large icons de l'action
367         */
368        public SendAction()
369        {
370            putValue(SMALL_ICON,
371                    new ImageIcon(ClientFrame.class
372                        .getResource("/icons/logout-16.png")));
373            putValue(LARGE_ICON_KEY,
374                    new ImageIcon(ClientFrame.class
375                        .getResource("/icons/logout-32.png")));
376            putValue(ACCELERATOR_KEY,
377                    KeyStroke.getKeyStroke(KeyEvent.VK_S,
378                                           InputEvent.META_MASK));
379            putValue(NAME, "Send");
380            putValue(SHORT_DESCRIPTION, "Send text to server");
381        }
382
383        /**
384         * OpÃ©rations rÃ©alisÃ©es lorsque l'action est sollicitÃ©e
385         * @param e Ã©vÃ¨nement Ã  l'origine de l'action
386         * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
387         */
388        @Override
389        public void actionPerformed(ActionEvent e)
390        {
391            /*
392             * rÃ©cupÃ©ration du contenu du textfield et envoi du message au
393             * serveur (ssi le message n'est pas vide), puis effacement du
394             * contenu du textfield.
395             */
396            // Obtention du contenu du sendTextField
397            String content = sendTextField.getText();
398
399            // logger.fine("Le contenu du textField etait = " + content);
400
401            // envoi du message
402            if (content ≠ null)
403            {
404                if (content.length() > 0)
405                {
406                    sendMessage(content);
407
408                    // Effacement du contenu du textfield
409                    sendTextField.setText("");
410                }
411            }
412        }
413    }
414
415    /**
416     * Action rÃ©alisÃ©e pour se dÃ©logguer du serveur
417     */
418    private class QuitAction extends AbstractAction
419    {
420        /**
421         * Constructeur d'une QuitAction : met en place le nom, la description,
422         * le raccourci clavier et les small|Large icons de l'action
423         */
424        public QuitAction()
425        {
426            putValue(SMALL_ICON,
427                    new ImageIcon(ClientFrame.class
428                        .getResource("/icons/cancel-16.png")));
429            putValue(LARGE_ICON_KEY,
430                    new ImageIcon(ClientFrame.class
431                        .getResource("/icons/cancel-32.png")));
432            putValue(ACCELERATOR_KEY,
433                    KeyStroke.getKeyStroke(KeyEvent.VK_Q,
434                                           InputEvent.META_MASK));
435            putValue(NAME, "Quit");
436            putValue(SHORT_DESCRIPTION, "Disconnect from server and quit");
437        }
438
439        /**
440         * OpÃ©rations rÃ©alisÃ©es lorsque l'action "quitter" est sollicitÃ©e
441         * @param e Ã©vÃ¨nement Ã  l'origine de l'action
442         * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
443         */
444        @Override
445        public void actionPerformed(ActionEvent e)
446        {
447            logger.info("QuitAction: sending bye ... ");
448
449            serverLabel.setText("");
450            thisRef.validate();
```



```
451
452            try
453            {
454                Thread.sleep(1000);
455            }
456            catch (InterruptedException e1)
457            {
458                return;
459            }
460
461            sendMessage(Vocabulary.byeCmd);
462        }
463    }
464
465    /**
466     * Classe qÃ©rant la fermeture correcte de la fenÃªtre. La fermeture correcte
467     * de la fenÃ¨tre implique de lancer un cleanup
468     */
469    protected class FrameWindowListener extends WindowAdapter
470    {
471        /**
472         * MÃ©thode dÃ©clenchÃ©e Ã  la fermeture de la fenÃªtre. Envoie la commande
473         * "bye" au serveur
474         */
475        @Override
476        public void windowClosing(WindowEvent e)
477        {
478            logger.info("FrameWindowListener::windowClosing: sending bye ... ");
479            /*
480             * appeler actionPerformed de quitAction si celle ci est
481             * non nulle
482             */
483            if (quitAction ≠ null)
484            {
485                quitAction.actionPerformed(null);
486            }
487        }
488    }
489
490    /**
491     * ExÃ©cution de la boucle d'exÃ©cution. La boucle d'exÃ©cution consiste Ã  lire
492     * une ligne sur le flux d'entrÃ©e avec un BufferedReader tant qu'une erreur
493     * d'IO n'intervient pas indiquant que le flux a Ã©tÃ© coupÃ©. Auquel cas on
494     * quitte la boucle principale et on ferme les flux d'I/O avec #cleanup()
495     */
496    @Override
497    public void run()
498    {
499        inBR = new BufferedReader(new InputStreamReader(inPipe));
500
501        String messageIn;
502
503        while (commonRun.booleanValue())
504        {
505            messageIn = null;
506            /*
507             * - Lecture d'une ligne de texte en provenance du serveur avec inBR
508             * Si une exception survient lors de cette lecture on quitte la
509             * boucle.
510             * - Si cette ligne de texte n'est pas nulle on affiche le message
511             * dans le document avec le format voulu en utilisant
512             * #writeMessage(String)
513             * - AprÃ¨s la fin de la boucle on change commonRun Ã  false de
514             * maniÃ¨re synchronisÃ©e afin que les autres threads utilisant ce
515             * commonRun puissent s'arrÃªter eux aussi :
516             * synchronized(commonRun)
517             * {
518             * commonRun = Boolean.FALSE;
519             * }
520             * Dans toutes les Ã©tapes si un problÃ¨me survient (erreur,
521             * exception, ...) on quitte la boucle en ayant au prÃ©alable ajoutÃ©
522             * un "warning" ou un "severe" au logger (en fonction de l'erreur
523             * rencontrÃ©e) et mis le commonRun Ã  false (de maniÃ¨re synchronisÃ©).
524             */
525            try
526            {
527                /*
528                 * read from input (doit Ãªtre bloquant)
529                 */
530                messageIn = inBR.readLine();
531            }
532            catch (IOException e)
533            {
534                logger.warning("ClientFrame: I/O Error reading");
535                break;
536            }
537
538            if (messageIn ≠ null)
539            {
540                // Ajouter le message Ã  la fin du document avec la couleur
```

```
541                 // voulue
542                 try
543                 {
544                     writeMessage(messageIn);
545                 }
546                 catch (BadLocationException e)
547                 {
548                     logger.warning("ClientFrame: write at bad location: "
549                         + e.getLocalizedMessage());
550                 }
551             }
552             else // messageIn == null
553             {
554                 break;
555             }
556         }
557
558         if (commonRun.booleanValue())
559         {
560             logger
561                 .info("ClientFrame::cleanup: changing run state at the end ... ");
562             synchronized (commonRun)
563             {
564                 commonRun = Boolean.FALSE;
565             }
566         }
567
568         cleanup();
569     }
570
571     /**
572      * Fermeture de la fenÃ¨tre et des flux Ã  la fin de l'exÃ©cution
573      */
574     @Override
575     public void cleanup()
576     {
577         logger.info("ClientFrame::cleanup: closing input buffered reader ... ");
578         try
579         {
580             inBR.close();
581         }
582         catch (IOException e)
583         {
584             logger.warning("ClientFrame::cleanup: failed to close input reader"
585                 + e.getLocalizedMessage());
586         }
587
588         super.cleanup();
589     }
590 }
```



```
1  /**
2   * Package contenant les classes de l'interface graphique
3   */
4  package widgets;
```

```java
package examples;
import java.util.ArrayList;
import java.util.Collection;

/**
 * Exemple de classe implÃ©mentant un Runnable et lancÃ© dans un Thread
 *
 * @author davidroussel
 */
public class RunRunnableExample
{
    /**
     * Classe interne reprÃ©sentant un simple compteur Ã  exÃ©cuter dans un thread.
     * Le compteur compte de 0 Ã  une valeur max. Lorsque le compteur atteint la
     * valeur max le compteur s'arrÃªte.
     * @author davidroussel
     */
    protected static class Counter implements Runnable
    {
        /**
         * Nombre de compteurs instanciÃ©s
         */
        private static int CounterNumber = 0;

        /**
         * Le numÃ©ro de compteur
         */
        private int number;
        /**
         * Le compteur proprement dit
         */
        private int count;

        /**
         * La valeur max du compteur
         */
        private int max;

        /**
         * Constructeur valuÃ© du compteur
         * @param max la valeur max du compteur Ã  laquelle il s'arrÃªte
         */
        public Counter(int max)
        {
            number = ++CounterNumber;
            count = 0;
            this.max = max;
        }

        /**
         * Nettoyage lors de la destruction
         * @see java.lang.Object#finalize()
         */
        @Override
        protected void finalize() throws Throwable
        {
            CounterNumber--;
        }

        /**
         * Boucle d'Ã©xÃ©cution principale du compteur : Tant que le compteur n'a
         * pas atteint la valeur max le compteur incrÃ©mente son compteur de 1,
         * affiche la valeur courante du compteur puis on demande au thread
         * dans lequel il tourne de passer la main Ã  un autre thread (en
         * espÃ©rant que ceux ci nous repassent la main un jour afin que l'on
         * puisse continuer Ã  compter).
         */
        @Override
        public void run()
        {
            while (count < max)
            {
                count++;

                System.out.println(this); // utilisation du toString

                // passe la main Ã  d'autres threads (si besoin)
                Thread.yield();
            }
        }

        /**
         * ReprÃ©sentation sous forme de chaine de caractÃ¨res
         * @see java.lang.Object#toString()
         */
        @Override
        public String toString()
        {
            return new String("Counter #" + number + " = " + count);
        }
    }
```



```java
    }

    /**
     * Collection de compteurs Runnable Ã  lancer
     */
    protected Collection<Counter> counters;

    /**
     * Collection de threads dans lesquels on va vaire tourner les Counter.
     */
    protected Collection<Thread> threads;

    /**
     * Constructeur d'un RunnableExample.
     * CrÃ©e un certain nombre de compteur (Runnable), puis crÃ©e le mÃªme nombre
     * de threads dans lesquels on place ces compteurs
     */
    public RunRunnableExample(int nbCounters)
    {
        counters = new ArrayList<Counter>(nbCounters);
        threads = new ArrayList<Thread>(nbCounters);

        for (int i = 0; i < nbCounters; i++)
        {
            Counter c = new Counter(10);
            counters.add(c);

            Thread t = new Thread(c);
            threads.add(t);
        }
    }

    /**
     * Lancement de tous les threads (contenant les compteurs)
     */
    public void launch()
    {
        for (Thread t : threads)
        {
            t.start();
        }
    }

    /**
     * attente de la fin de tous les threads pour terminer le thread principal
     */
    public void terminate()
    {
        for (Thread t : threads)
        {
            try
            {
                t.join();
            }
            catch (InterruptedException e)
            {
                System.err.println("Thread" + t + " join interrupted");
                e.printStackTrace();
            }
        }

        System.out.println("All threads terminated");
    }

    /**
     * Programme principal.
     * Lancement de plusieurs Counters
     *
     * @param args arguments du programme pour y lire le nombre de compteurs Ã 
     * lancer
     */
    public static void main(String[] args)
    {
        int nbCounters = 3;
        // on lit le nombre de counters dans le premier argument du programme
        if (args.length > 0)
        {
            int value;
            try
            {
                value = Integer.parseInt(args[0]);
                if (value > 0)
                {
                    nbCounters = value;
                }
            }
            catch (NumberFormatException nfe)
            {
                System.err.println("Error reading number of counters");
            }
```

```
181             }
182
183             RunRunnableExample runner = new RunRunnableExample(nbCounters);
184
185             runner.launch();
186
187             System.out.println("All threads launched");
188
189             runner.terminate();
190         }
191     }
```



```
1   package examples;
2   import java.awt.EventQueue;
3
4   import examples.widgets.ExampleFrame;
5
6
7   /**
8    * Programme principal lanÃ§ant une {@link ExampleFrame}
9    * @author davidroussel
10   *
11   */
12  public class RunExampleFrame
    {
14      /**
15       * Programme principal
16       * @param args
17       */
18      public static void main(String[] args)
19      {
20          if (System.getProperty("os.name").startsWith("Mac OS"))
21          {
22              // Met en place le menu en haut de l'Ã©cran plutÃ´t que dans l'application
23              System.setProperty("apple.laf.useScreenMenuBar", "true");
24              System.setProperty("com.apple.mrj.application.apple.menu.about.name", "Name");
25          }
26
27          // Insertion de la frame dans la file des Ã©vÃ¨nements GUI
28          EventQueue.invokeLater(new Runnable()
29          {
30              @Override
31              public void run()
32              {
33                  try
34                  {
35                      ExampleFrame frame = new ExampleFrame();
36                      frame.pack();
37                      frame.setVisible(true);
38                  }
39                  catch (Exception e)
40                  {
41                      e.printStackTrace();
42                  }
43              }
44          });
45      }
46  }
```

```java
package examples;
import java.awt.EventQueue;

import javax.swing.JFrame;

import examples.widgets.ExampleFrame;
import examples.widgets.ListExampleFrame;


/**
 * Programme principal lanÃ§ant une {@link ExampleFrame}
 * @author davidroussel
 *
 */
public class RunListFrame
{
    /**
     * Programme principal
     * @param args
     */
    public static void main(String[] args)
    {
        if (System.getProperty("os.name").startsWith("Mac OS"))
        {
            // Met en place le menu en haut de l'Ã©cran plutÃ´t que dans l'application
            System.setProperty("apple.laf.useScreenMenuBar", "true");
            System.setProperty("com.apple.mrj.application.apple.menu.about.name", "Name");
        }

        // Insertion de la frame dans la file des Ã©vÃ¨nements GUI
        EventQueue.invokeLater(new Runnable()
        {
            @Override
            public void run()
            {
                try
                {
                    JFrame frame = new ListExampleFrame();
                    frame.pack();
                    frame.setVisible(true);
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
}
```



```java
import java.io.IOException;
import java.net.SocketException;

import chat.Failure;
import chat.server.ChatServer;

/**
 * Classe/programme qui lance un serveur de chat
 * @author davidroussel
 */
public class RunChatServer extends AbstractRunChat
{
    /**
     * Time out de la server socket avant qu'elle ne recommence Ã  attendre
     * des connections des Ã©ventuels clients
     */
    private int timeout;

    /**
     * Flag permettant (ou pas) de quitter le serveur lorsque le dernier
     * client se dÃ©logue
     */
    private boolean quitOnLastclient;

    /**
     * Default time out to wait for client connection : 5 seconds
     */
    public static final int DEFAULTTIMEOUT = 5000;

    /**
     * Constructeur d'un lanceur de serveur d'aprÃ¨s les arguments du programme
     * principal
     * @param args les arguments du programme principal
     */
    protected RunChatServer(String[] args)
    {
        super(args);
    }

    /**
     * Mise en place des attributs du serveur de chat en fonction des arguments
     * utilisÃ©s dans la ligne de commande
     * @param args les arguments fournis au programme principal.
     */
    @Override
    protected void setAttributes(String[] args)
    {
        /*
         * On met d'abord les attributs locaux Ã  leur valeur par dÃ©faut
         */
        timeout = DEFAULTTIMEOUT;
        quitOnLastclient = true;

        /*
         * parsing des arguments communs aux clients et serveur
         *  -v | --verbose
         *  -p | --port : port Ã  utiliser pour la serverSocket
         */
        super.setAttributes(args);

        /*
         * parsing des arguments spÃ©cifique au  serveur
         *  -t | --timeout : timeout d'attente de la server socket
         */
        for (int i=0; i < args.length; i++)
        {
            if (args[i].equals("--timeout") ∨ args[i].equals("-t"))
            {
                if (i < (args.length - 1))
                {
                    // parse next arg for in port value
                    Integer timeInteger = readInt(args[++i]);
                    if (timeInteger ≠ null)
                    {
                        timeout = timeInteger.intValue();
                    }
                    logger.info("Setting timeout to " + timeout);
                }
                else
                {
                    logger.warning("invalid timeout value");
                }
            }
            if (args[i].equals("--quit") ∨ args[i].equals("-q"))
            {
                quitOnLastclient = true;
                logger.info("Setting quit on last client to true");
            }
            if (args[i].equals("--noquit") ∨ args[i].equals("-n"))
            {
```

```
                quitOnLastclient = false;
                logger.info("Setting quit on last client to false");
            }
        }
    }

    /**
     * Lancement du serveur de chat
     */
    @Override
    protected void launch()
    {
        /*
         * Create and Launch server on local ip adress with port number and verbose
         * status
         */
        logger.info("Creating server on port " + port + " with timeout "
                + timeout + " ms and verbose " + (verbose ? "on" : "off"));

        ChatServer server = null;
        try
        {
            server = new ChatServer(port, timeout, quitOnLastclient, logger);
        }
        catch (SocketException se)
        {
            logger.severe(Failure.SET_SERVER_SOCKET_TIMEOUT + ", abort ...");
            logger.severe(se.getLocalizedMessage());
            System.exit(Failure.SET_SERVER_SOCKET_TIMEOUT.toInteger());
        }
        catch (IOException e)
        {
            logger.severe(Failure.CREATE_SERVER_SOCKET + ", abort ...");
            e.printStackTrace();
            System.exit(Failure.CREATE_SERVER_SOCKET.toInteger());
        }

        // Wait for serverThread to stop
        Thread serverThread = null;
        if (server ≠ null)
        {
            serverThread = new Thread(server);
            serverThread.start();

            logger.info("Waiting for server to terminate ... ");
            try
            {
                serverThread.join();
                logger.fine("Server terminated, program end.");
            }
            catch (InterruptedException e)
            {
                logger.severe("Server Thread Join interrupted");
                logger.severe(e.getLocalizedMessage());
            }
        }
    }

    /**
     * Programme principal
     * @param args les arguments
     * <ul>
     *   <li>--port <port number> : set host connection port</li>
     *   <li>--verbose : set verbose on</li>
     *   <li>--timeout <timeout in ms> : server socket waiting time out</li>
     * </ul>
     */
    public static void main(String[] args)
    {
        RunChatServer server = new RunChatServer(args);

        server.launch();
    }
}
```



```
import java.awt.EventQueue;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Vector;

import chat.Failure;
import chat.UserOutputType;
import chat.client.ChatClient;
import widgets.AbstractClientFrame;
import widgets.ClientFrame;

/**
 * Lanceur d'un client de chat.
 *
 * @author davidroussel
 */
public class RunChatClient extends AbstractRunChat
{
    /**
     * HÃ´te sur lequel se trouve le serveur de chat
     */
    private String host;

    /**
     * Nom d'utilisateur Ã  utiliser pour se connecter au serveur. Si le nom
     * n'est pas fournit
     */
    private String name;

    /**
     * Flux d'entrÃ©e sur lequel lire les messages tapÃ©s par l'utilisateur
     */
    private InputStream userIn;

    /**
     * Flux de sortie sur lequel envoyer les messages vers l'utilisateur
     */
    private OutputStream userOut;

    /**
     * Indique si le client Ã  crÃ©er est un GUI ou pas
     */
    private boolean gui;

    /**
     * La version de l'interface graphique Ã  lancer:
     * <ul>
     *   <li>version 1 correspond Ã  l'utilisation d'une ClientFrame</li>
     *   <li>version 2 correspond Ã  l'untilisation d'une SuperClientFrame</li>
     * </ul>
     */
    private int guiVersion;

    /**
     * Ensemble des threads des clients.
     * Il faudra attendre la fin de ces threads pour terminer l'exÃ©cution
     * principal.
     */
    private Vector<Thread> threadPool;

    /**
     * Constructeur d'un lanceur de client d'aprÃ¨s les arguments du programme
     * principal
     *
     * @param args les arguments du programme principal
     */
    protected RunChatClient(String[] args)
    {
        super(args);

        /*
         * Initialisation des flux d'I/O utilisateur Ã  null
         * ils dÃ©pendront du client Ã  crÃ©er (console ou GUI)
         */
        userIn = null;
        userOut = null;

        /*
         * Initialisation du pool de thread des clients
         */
        threadPool = new Vector<Thread>();
    }

    /**
     * Mise en place des attributs du client de chat en fonction des arguments
     * utilisÃ©s dans la ligne de commande
     * @param args les arguments fournis au programme principal.
```

```
91      */
92     @Override
93     protected void setAttributes(String[] args)
94     {
95         /*
96          * parsing des arguments communs aux clients et serveur
97          * -v | --verbose
98          * -p | --port : port Ã  utiliser pour la serverSocket
99          */
100        super.setAttributes(args);
101
102        /*
103         * On met d'abord les attributs locaux Ã  leur valeur par dÃ©faut
104         */
105        host = null;
106        name = null;
107        gui = false;
108
109        /*
110         * parsing des arguments spÃ©cifique au client
111         * -h | --host : nom ou adresse IP du serveur
112         * -n | --name : nom d'utilisateur
113         * -g | --gui : pour lancer le client GUI
114         */
115        for (int i = 0; i < args.length; i++)
116        {
117            if (args[i].equals("--host") ∨ args[i].equals("-h"))
118            {
119                if (i < (args.length - 1))
120                {
121                    // parse next arg for in port value
122                    host = args[++i];
123                    logger.fine("Setting host to " + host);
124                }
125                else
126                {
127                    logger.warning("Setting host to: nothing, invalid value");
128                }
129            }
130            else if (args[i].equals("--name") ∨ args[i].equals("-n"))
131            {
132                if (i < (args.length - 1))
133                {
134                    // parse next arg for in port value
135                    name = args[++i];
136                    logger.fine("Setting user name to: " + name);
137                }
138                else
139                {
140                    logger.warning("Setting user name to: nothing, invalid value");
141                }
142            }
143            if (args[i].equals("--gui") ∨ args[i].equals("-g"))
144            {
145                gui = true;
146                if (i < (args.length - 1))
147                {
148                    // parse next arg for gui version
149                    try
150                    {
151                        guiVersion = Integer.parseInt(args[++i]);
152                        if (guiVersion < 1)
153                        {
154                            guiVersion = 1;
155                        }
156                        else if (guiVersion > 2)
157                        {
158                            guiVersion = 2;
159                        }
160                    }
161                    catch (NumberFormatException nfe)
162                    {
163                        logger.warning("Invalid gui number, revert to 1");
164                        guiVersion = 1;
165                    }
166                    logger.fine("Setting gui to " + guiVersion);
167                }
168                else
169                {
170                    logger.warning("ReSetting gui version to 1, invalid value");
171                    guiVersion = 1;
172                }
173            }
174        }
175
176        if (host ≡ null) // on va chercher local host
177        {
178            try
179            {
180                host = InetAddress.getLocalHost().getHostName();
```



```
181             }
182             catch (UnknownHostException e)
183             {
184                 logger.severe(Failure.NO_LOCAL_HOST.toString());
185                 logger.severe(e.getLocalizedMessage());
186                 System.exit(Failure.NO_LOCAL_HOST.toInteger());
187             }
188         }
189
190         if (name ≡ null) // on va chercher le nom de l'utilisateur
191         {
192             try
193             {
194                 // Try LOGNAME on unix type systems
195                 name = System.getenv("LOGNAME");
196             }
197             catch (NullPointerException npe)
198             {
199                 logger.warning("no LOGNAME found, trying USERNAME");
200                 try
201                 {
202                     // Try USERNAME on other systems
203                     name = System.getenv("USERNAME");
204                 }
205                 catch (NullPointerException npe2)
206                 {
207                     logger.severe(Failure.NO_USER_NAME + " abort");
208                     System.exit(Failure.NO_USER_NAME.toInteger());
209                 }
210             }
211             catch (SecurityException se)
212             {
213                 logger.severe(Failure.NO_ENV_ACCESS + " !");
214                 System.exit(Failure.NO_ENV_ACCESS.toInteger());
215             }
216         }
217     }
218
219     /**
220      * Lancement du ChatClient
221      */
222     @Override
223     protected void launch()
224     {
225         /*
226          * Create and Launch client
227          */
228         logger.info("Creating client to " + host + " at port " + port
229                 + " with verbose " + (verbose ? "on" : "off ... "));
230
231         Boolean commonRun;
232
233         if (gui)
234         {
235             if (System.getProperty("os.name").startsWith("Mac OS"))
236             {
237                 // Met en place le menu en haut de l'Ã©cran plutÃ´t que dans l'application
238                 System.setProperty("apple.laf.useScreenMenuBar", "true");
239                 System.setProperty("com.apple.mrj.application.apple.menu.about.name", "Name");
240             }
241
242             /*
243              * On a besoin d'un commonRun entre la frame et les ServerHandler
244              * et UserHandler du client crÃ©Ã© plus bas.
245              */
246             commonRun = Boolean.TRUE;
247
248             /*
249              * CrÃ©ation de la fenÃªtre de chat
250              * TODO Ã  customizer lorsrque vous aurez crÃ©Ã© la classe
251              * ClientFrame2
252              */
253             final AbstractClientFrame frame = new ClientFrame(name, host, commonRun, logger);
254
255             /*
256              * TODO CrÃ©ation du flux de sortie vers le GUI : userOut Ã  partir du
257              * flux d'entrÃ©e de la frame (ClientFrame#getInPipe())
258              *  - Creation d'un PipedOutputStream Ã  connecter sur
259              *  - le PipedInputStream de la frame
260              */
261             try
262             {
263                 // userOut = TODO Complete ...
264                 throw new IOException(); // TODO Remove when done
265             }
266             catch (IOException e)
267             {
268                 logger.severe(Failure.USER_OUTPUT_STREAM
269                         + " unable to get piped out stream");
270                 logger.severe(e.getLocalizedMessage());
```

```
271                 System.exit(Failure.USER_OUTPUT_STREAM.toInteger());
272             }
273
274             /*
275              * TODO CrÃ©ation du flux d'entrÃ©e depuis le GUI : userIn Ã  partir du
276              * flux de sortie de la frame (ClientFrame#getOutPipe())
277              *  - CrÃ©ation d'un PipedInputStream Ã  connecter sur
278              *  - le PipedOutputStream de la frame
279              */
280             try
281             {
282                 // userIn = TODO Complete ...
283                 throw new IOException(); // TODO Remove when done
284             }
285             catch (IOException e)
286             {
287                 logger.severe(Failure.USER_INPUT_STREAM
288                         + " unable to get user piped in stream");
289                 logger.severe(e.getLocalizedMessage());
290                 System.exit(Failure.USER_INPUT_STREAM.toInteger());
291             }
292
293             /*
294              * Insertion de la frame dans la file des Ã©vÃ¨nements GUI
295              * grÃ¢ce Ã  un Runnable anonyme
296              */
297             EventQueue.invokeLater(new Runnable()
298             {
299                 @Override
300                 public void run()
301                 {
302                     try
303                     {
304                         frame.pack();
305                         frame.setVisible(true);
306                     }
307                     catch (Exception e)
308                     {
309                         logger.severe(e.getLocalizedMessage());
310                     }
311                 }
312             });
313
314             /*
315              * CrÃ©ation et lancement du thread de la frame
316              */
317             Thread guiThread = new Thread(frame);
318             threadPool.add(guiThread);
319             guiThread.start();
320
321         }
322         else // client console
323         {
324             // lecture depuis la console
325             userIn = System.in;
326             // Ã©criture vers la console
327             userOut = System.out;
328             // On a pas besoin d'un commonRun avec le client console
329             commonRun = null;
330         }
331
332         /*
333          * Lancement du ChatClient
334          */
335         UserOutputType outType = UserOutputType.fromInteger(guiVersion);
336         ChatClient client = new ChatClient(host,        // hÃ´te du serveur
337                                            port,        // port tcp
338                                            name,        // nom d'utilisateur
339                                            userIn,      // entrÃ©es utilisateur
340                                            userOut,     // sorties utilisateur
341                                            outType,     // Type sortie utilisateur
342                                            commonRun,   // commonRun avec le GUI
343                                            logger);     // parent logger
344         if (client.isReady())
345         {
346             Thread clientThread = new Thread(client);
347             threadPool.add(clientThread);
348
349             clientThread.start();
350
351             logger.fine("client launched");
352
353             // attente de l'ensemble des threads du threadPool pour terminer
354             for (Thread t : threadPool)
355             {
356                 try
357                 {
358                     t.join();
359                     logger.fine("client thread end");
360                 }
```



```
361                 catch (InterruptedException e)
362                 {
363                     logger.severe("interrupted");
364                     logger.severe(e.getLocalizedMessage());
365                 }
366             }
367         }
368         else
369         {
370             logger.severe(Failure.CLIENT_NOT_READY + " abort...");
371             System.exit(Failure.CLIENT_NOT_READY.toInteger());
372         }
373     }
374
375     /**
376      * Programme principal de lancement d'un client de chat
377      * @param args argument du programme
378      * <ul>
379      * <li>--host <host address> : set host to connect to</li>
380      * <li>--port <port number> : set host connection port</li>
381      * <li>--name <user name> : user name to use to connect</li>
382      * <li>--verbose : set verbose on</li>
383      * <li>--gui <1 or 2>: use graphical interface rather than console interface
384      * </li>
385      * </ul>
386      */
387     public static void main(String[] args)
388     {
389
390         RunChatClient client = new RunChatClient(args);
391
392         client.launch();
393     }
394 }
```