TAA (Master 2 Pro)

Olivier Barais/ François Fouquet

Spring et AOP

Lors de ce TP, nous vous proposons l'extrait de code suivant. Ce code est une application RMI de messagerie instantanée. L'interface du serveur est définie dans le fichier ChatRoom.java. La mise en œuvre du serveur est définie dans ChatRoomImpl.java. Chaque utilisateur lance un client particulier, une fenêtre d'authentification demande alors un login et un mot de passe (Figure 2). En cas de succès pour l'authentification, l'utilisateur doit donner son pseudo (Figure 3) puis il peut poster et recevoir des messages. Pour cela, il dispose d'une IHM minimaliste (`ChatUI.java`) (Figure 4). L'authentification est gérée à l'aide de l'API (JAAS Java Authentification Authorization Service)

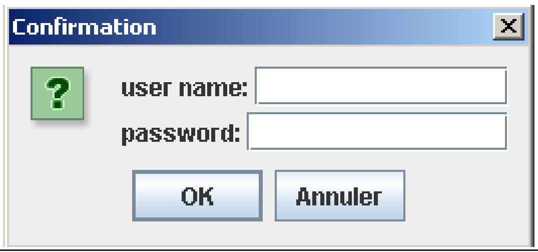


Figure 1

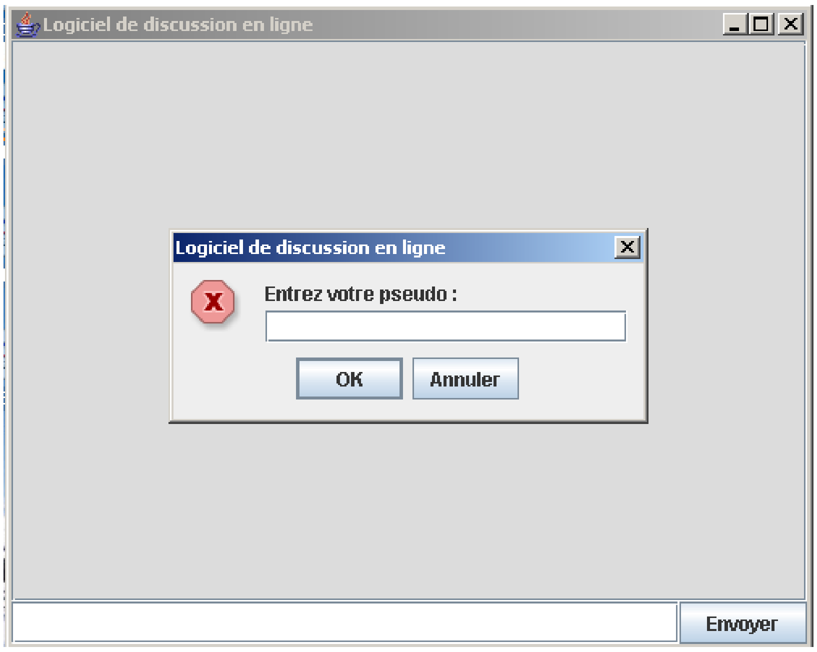
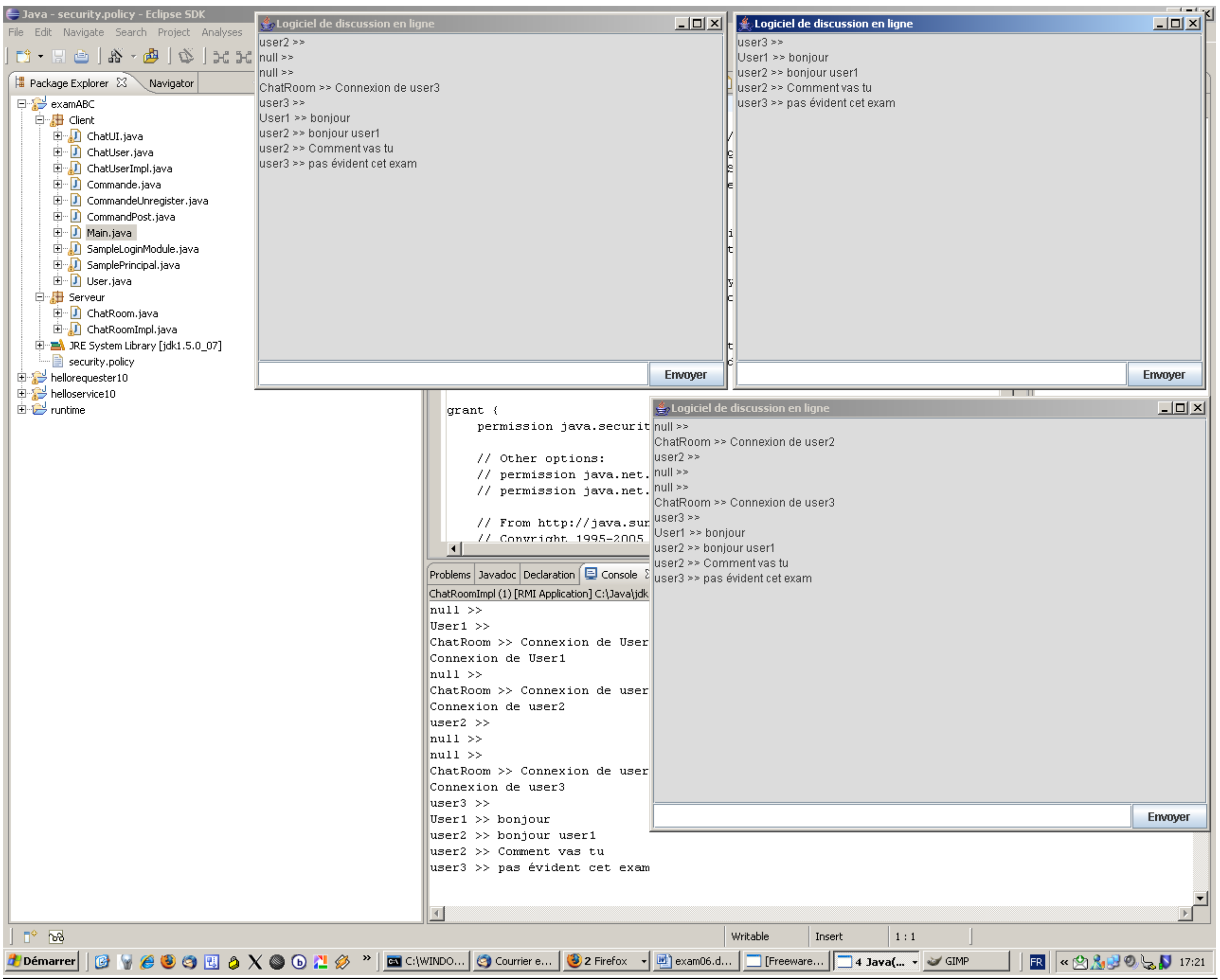

Logiciel de discussion en ligne
Logiciel de discussion en ligne
Entrez votre pseudo :
OK
Annuler
Envoyer

Figure 2



```
user2 >>
null >>
null >>
ChatRoom >> Connexion de user3
user3 >>
User1 >> bonjour
user2 >> bonjour user1
user2 >> Comment vas tu
user3 >> pas évident cet exam
```

```
user3 >>
User1 >> bonjour
user2 >> bonjour user1
user2 >> Comment vas tu
user3 >> pas évident cet exam
```

```
null >>
ChatRoom >> Connexion de user2
user2 >>
null >>
null >>
ChatRoom >> Connexion de user3
user3 >>
User1 >> bonjour
user2 >> bonjour user1
user2 >> Comment vas tu
user3 >> pas évident cet exam
```

```
grant {
    permission java.securit

    // Other options:
    // permission java.net.
    // permission java.net.

    // From http://java.sun
```

```
null >>
User1 >>
ChatRoom >> Connexion de User
Connexion de User1
null >>
ChatRoom >> Connexion de user
Connexion de user2
user2 >>
null >>
null >>
ChatRoom >> Connexion de user
Connexion de user3
user3 >>
User1 >> bonjour
user2 >> bonjour user1
user2 >> Comment vas tu
user3 >> pas évident cet exam
```

Figure 3

Objectif du TP

Proposez un certain nombre d'améliorations au niveau de la structure de votre application. Vous modifierez le code de vos classes afin de montrer une mise en œuvre possible à l'aide du *framework* Spring.

Question 1.

Faites tourner cette application sur deux machines différentes.

Question 2.

Proposez un refactoring du code pour en améliorer sa conception. (Utilisation de Design Pattern, Interface, …). Identifier les mauvaises pratiques dans ce code :

- recherche des applications possibles de design pattern « strategy »
- utilisation de pattern « command » pour découpler les traitements de l’ihm de ses déclenchements
- utilisation de pattern IoC pour rendre l’IHM configurable
- utilisation de pattern « command » pour gérer les stratégies de communication réseau

Question 3.
Retravailler cette application pour utiliser Spring et l'injection de dépendance.

Question 4.
Utiliser Spring AOP pour isoler convenablement le log et la sécurité (authentification)

Question 5. (Optionnelle) Rediriger le log faire un EJB MDB.

Bon courage.

Code de l'application.

```java
public class Main {
	public static void main(String[] args) throws RemoteException {
		new Thread(new ChatUserImpl("essai1")).start();

	}

}
```

```java
public class ChatUI {

	private Commande  unregister;
	private Commande  postMessage;


	private ChatUser u = null;
	private String title = "Logiciel de discussion en ligne";
	private JFrame window = new JFrame(this.title);
	private JTextArea txtOutput = new JTextArea();
	private JTextField txtMessage = new JTextField();
	private JButton btnSend = new JButton("Envoyer");

	public ChatUI(ChatUser u,Commande post, Commande unregister) {

		this.postMessage = post;
		this.unregister = unregister;
		JPanel panel = (JPanel) this.window.getContentPane();
		JScrollPane sclPane = new JScrollPane(txtOutput);
		panel.add(sclPane, BorderLayout.CENTER);
		JPanel southPanel = new JPanel(new BorderLayout());
		southPanel.add(this.txtMessage, BorderLayout.CENTER);
		southPanel.add(this.btnSend, BorderLayout.EAST);
		panel.add(southPanel, BorderLayout.SOUTH);

		// Gestion des évènements
		window.addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				window_windowClosing(e);
			}
		});
		btnSend.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				btnSend_actionPerformed(e);
			}
		});
		txtMessage.addKeyListener(new KeyAdapter() {
			public void keyReleased(KeyEvent event) {
```

```
                        if (event.getKeyChar() == '\n')
                              btnSend_actionPerformed(null);
                  }
            });

            // Initialisation des attributs
            this.txtOutput.setBackground(new Color(220, 220, 220));
            this.txtOutput.setEditable(false);
            this.window.setSize(500, 400);
            this.window.setVisible(true);
            this.txtMessage.requestFocus();
      }

      public void window_windowClosing(WindowEvent e) {
            try {
                  unregister.execute();
            } catch (Exception exc) {
                  System.err.println("Desinscription impossible");
            }
             System.exit(-1);
      }

      protected String message = null;
      public void btnSend_actionPerformed(ActionEvent e) {
            try {
                  message = this.txtMessage.getText();
                  postMessage.execute();
            } catch (Exception exception) {
                  System.err.println("Envoie message impossible");
            }
            this.txtMessage.setText("");
            this.txtMessage.requestFocus();
      }

    public void displayMessage(String message){
        this.txtOutput.append(message + "\n");
        this.txtOutput.moveCaretPosition(this.txtOutput.getText().length());
    }

    public String requestPseudo() {
         String pseudo = JOptionPane.showInputDialog(
                this.window, "Entrez votre pseudo : ",
                this.title,  JOptionPane.OK_OPTION
        );
        if (pseudo == null) System.exit(0);
        return pseudo;
    }

      public String getMessage() {
            return message;
      }
}
```

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ChatUser extends Remote {
    public void displayMessage(String message) throws RemoteException;
}
```

```
import java.rmi.Naming;
import java.rmi.RemoteException;
```

```java
import java.rmi.server.UnicastRemoteObject;
import java.util.HashMap;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginException;
import Serveur.ChatRoom;
import com.sun.security.auth.callback.DialogCallbackHandler;

public class ChatUserImpl extends UnicastRemoteObject implements ChatUser,
            Runnable, User {
      private ChatRoom room = null;
      private String pseudo = null;
      private String name = null;
      private ChatUI ui;
      public ChatUserImpl(String name) throws RemoteException {
            super(); // Appel au constructeur de UnicastRemoteObject
            this.name = name;
            try {
                  this.room = (ChatRoom) Naming.lookup("rmi://coreff/ChatRoom");
            } catch (Exception e) {

                  e.printStackTrace();
                  System.exit(0);
            }

            this.createIHM();
            // this.requestPseudo();
      }

      public void createIHM() {
            Commande unreg = new CommandeUnregister(room);
            unreg.setUser(this);
            Commande post = new CommandPost(room);
            post.setUser(this);
            ui = new ChatUI(this, post, unreg);
            ((CommandPost) post).setUI(ui);
      }

      public void displayMessage(String message) throws RemoteException {
            ui.displayMessage(message);
      }

      public void run() {
            try {
                  SampleLoginModule lc = null;
                  try {
                        lc = new SampleLoginModule(room);
                        lc.initialize(new Subject(), new
                        DialogCallbackHandler(), null,
                                    new HashMap());

                  } catch (SecurityException se) {
                        System.err.println("Cannot create LoginContext. "
                                    + se.getMessage());
                        System.exit(-1);
                  }

                  // the user has 3 attempts to authenticate successfully
                  int i;
                  for (i = 0; i < 3; i++) {
                        try {

                              // attempt authentication
                              lc.login();
```

```
                    // if we return with no exception, authentication
succeeded
                    break;

                } catch (LoginException le) {

                    System.err.println("Authentication failed:");
                    System.err.println("  " + le.getMessage());
                    try {
                        Thread.currentThread().sleep(3000);
                    } catch (Exception e) {
                        // ignore
                    }

                }
            }

            // did they fail three times?
            if (i == 3) {
                System.out.println("Sorry");

                System.exit(-1);
            }

            System.out.println("Authentication succeeded!");
            this.pseudo = ui.requestPseudo();
            this.room.subscribe(this, this.pseudo);

        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }

    public String getPseudo() {
        return pseudo;
    }
}
```

```
public interface Commande {
    void execute();
    void setUser(User u);
}
```

```
public class CommandeUnregister implements Commande {
    ChatRoom room = null;
    User user = null;

    public CommandeUnregister(ChatRoom room) {
        this.room = room;
    }

    public void execute() {
        try {
            room.unsubscribe(user.getPseudo());
        } catch (RemoteException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void setUser(User user) {
```

```java
		this.user = user;
	}

}
```

```java
public class CommandPost implements Commande {

	public CommandPost(ChatRoom room) {
		this.room = room;
	}

	ChatRoom room = null;
	User user = null;
	ChatUI ui = null;

	public void execute() {
		try {
			room.postMessage(user.getPseudo(), ui.getMessage());
		} catch (RemoteException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void setUser(User user) {
		this.user = user;
	}

	public void setUI(ChatUI ui2) {
		this.ui=ui2;
	}

}
```

```java
public class SampleLoginModule implements LoginModule {

	// initial state
	private Subject subject;
	private CallbackHandler callbackHandler;
	private Map sharedState;
	private Map options;
	// configurable option
	private boolean debug = false;
	// the authentication status
	private boolean succeeded = false;
	private boolean commitSucceeded = false;
	// username and password
	private String username;
	private char[] password;
	// testUser's SamplePrincipal
	private SamplePrincipal userPrincipal;
	private ChatRoom room;
	public SampleLoginModule(ChatRoom room) {
		this.room = room;
	}


	public void initialize(Subject subject, CallbackHandler callbackHandler,
			Map sharedState, Map options) {


		this.subject = subject;
```

```
            this.callbackHandler = callbackHandler;
            this.sharedState = sharedState;
            this.options = options;

            // initialize any configured options
            debug = "true".equalsIgnoreCase((String) options.get("debug"));
      }

      /**
       * Authenticate the user by prompting for a user name and password.
       */
      public boolean login() throws LoginException {

            // prompt for a user name and password
            if (callbackHandler == null)
                  throw new LoginException("Error: no CallbackHandler available
"     + "to garner authentication information from the user");

            Callback[] callbacks = new Callback[2];
            callbacks[0] = new NameCallback("user name: ");
            callbacks[1] = new PasswordCallback("password: ", false);

            try {
                  callbackHandler.handle(callbacks);
                  username = ((NameCallback) callbacks[0]).getName();
                  char[] tmpPassword = ((PasswordCallback) callbacks[1])
                              .getPassword();
                  if (tmpPassword == null) {
                        // treat a NULL password as an empty password
                        tmpPassword = new char[0];
                  }
                  password = new char[tmpPassword.length];
                  System.arraycopy(tmpPassword, 0, password, 0, tmpPass-
word.length);
                  ((PasswordCallback) callbacks[1]).clearPassword();

            } catch (java.io.IOException ioe) {
                  throw new LoginException(ioe.toString());
            } catch (UnsupportedCallbackException uce) {
                  throw new LoginException("Error: " + uce.getCallback().to-
String()
                              + " not available to garner authentication inform-
ation "   + "from the user");
            }

            // print debugging information
            if (debug) {
                  System.out.println("\t\t[SampleLoginModule] "
                              + "user entered user name: " + username);
                  System.out.print("\t\t[SampleLoginModule] "
                              + "user entered password: ");
                  for (int i = 0; i < password.length; i++)
                        System.out.print(password[i]);
                  System.out.println();
            }

            try {
                  return room.authentification(username, password);
            } catch (RemoteException e) {
                  e.printStackTrace();
                  return false;
            }

      }
```

```java
/**
 * <p>
 * This method is called if the LoginContext's overall authentication
 * succeeded (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL
 * LoginModules succeeded).
 */
public boolean commit() throws LoginException {
      if (succeeded == false) {
            return false;
      } else {
            // add a Principal (authenticated identity)
            // to the Subject

            // assume the user we authenticated is the SamplePrincipal
            userPrincipal = new SamplePrincipal(username);
            if (!subject.getPrincipals().contains(userPrincipal))
                  subject.getPrincipals().add(userPrincipal);

            if (debug) {
                  System.out.println("\t\t[SampleLoginModule] "
                              + "added SamplePrincipal to Subject");
            }

            // in any case, clean out state
            username = null;
            for (int i = 0; i < password.length; i++)
                  password[i] = ' ';
            password = null;

            commitSucceeded = true;
            return true;
      }
}

/**
 * <p>
 * This method is called if the LoginContext's overall authentication
 * failed. (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL
 * LoginModules did not succeed).
 *
 */
public boolean abort() throws LoginException {
      if (succeeded == false) {
            return false;
      } else if (succeeded == true && commitSucceeded == false) {
            // login succeeded but overall authentication failed
            succeeded = false;
            username = null;
            if (password != null) {
                  for (int i = 0; i < password.length; i++)
                        password[i] = ' ';
                  password = null;
            }
            userPrincipal = null;
      } else {
            // overall authentication succeeded and commit succeeded,
            // but someone else's commit failed
            logout();
      }
      return true;
}

/**
```

```
         * Logout the user.
         *
         */
        public boolean logout() throws LoginException {

              subject.getPrincipals().remove(userPrincipal);
              succeeded = false;
              succeeded = commitSucceeded;
              username = null;
              if (password != null) {
                    for (int i = 0; i < password.length; i++)
                          password[i] = ' ';
                    password = null;
              }
              userPrincipal = null;
              return true;
        }

        public Subject getSubject() {
              return subject;
        }
}
```

```
/**
 * <p> This class implements the <code>Principal</code> interface
 * and represents a Sample user.
 */
public class SamplePrincipal implements Principal, java.io.Serializable {

    /**
     * @serial
     */
    private String name;


    public SamplePrincipal(String name) {
      if (name == null)
          throw new NullPointerException("illegal null input");

      this.name = name;
    }


    public String getName() {
      return name;
    }


    public String toString() {
      return("SamplePrincipal:  " + name);
    }


    public boolean equals(Object o) {
      if (o == null)
          return false;

        if (this == o)
            return true;

        if (!(o instanceof SamplePrincipal))
            return false;
        SamplePrincipal that = (SamplePrincipal)o;
      if (this.getName().equals(that.getName()))
          return true;
      return false;
    }


    public int hashCode() {
      return name.hashCode();
    }
}
```

```
public interface User {
       public String getPseudo();
}
```

```
public interface ChatRoom extends Remote {
    public void subscribe(ChatUser user, String pseudo) throws RemoteException;
    public void unsubscribe(String pseudo) throws RemoteException;
    public void postMessage(String pseudo, String message) throws
       RemoteException;
    public boolean authentification(String username, char[] password) throws
      FailedLoginException, RemoteException
}
```

```
public class ChatRoomImpl extends UnicastRemoteObject implements ChatRoom {
      private Hashtable<String, ChatUser> users = new Hashtable<String,
ChatUser>();

      private Hashtable<String, char[]> alloweduser = new Hashtable<String,
char[]>();

      // configurable option
      private boolean debug = false;

      public ChatRoomImpl() throws RemoteException {
            super();
            alloweduser.put("toto", "passtoto".toCharArray());
            alloweduser.put("titi", "passtiti".toCharArray());
            alloweduser.put("testUser", "testPassword".toCharArray());

      }

      public void subscribe(ChatUser user, String pseudo) throws RemoteException
{
            String message = "Connexion de " + pseudo;
            this.postMessage("ChatRoom", message);
            System.out.println(message);
            this.users.put(pseudo, user);
      }

      public void unsubscribe(String pseudo) throws RemoteException {
            String message = "Deconnexion de " + pseudo;
            System.out.println(message);
            this.users.remove(pseudo);
            this.postMessage("ChatRoom", message);
      }

      public void postMessage(String pseudo, String message)
                  throws RemoteException {
            String fullMessage = pseudo + " >> " + message;
```

```
			System.out.println(fullMessage);

			for (ChatUser user : users.values()) {
				user.displayMessage(fullMessage);
			}
		}

		public boolean authentification(String username, char[] password)
				throws FailedLoginException, RemoteException {
			// verify the username/password
			boolean usernameCorrect = false;
			boolean passwordCorrect = false;
			if (this.alloweduser.containsKey(username)) {
				usernameCorrect = true;
				if (password.length == this.alloweduser.get(username).length
						&& testPassword(this.alloweduser.get(username),
password)) {

					// authentication succeeded!!!
					passwordCorrect = true;
					if (debug)
						System.out.println("\t\t[SampleLoginModule] "
								+ "authentication succeeded");
					return true;
				}

			}

			// authentication failed -- clean out state
			if (debug)
				System.out.println("\t\t[SampleLoginModule] "
						+ "authentication failed");
			for (int i = 0; i < password.length; i++)
				password[i] = ' ';
			if (!usernameCorrect) {
				throw new FailedLoginException("User Name Incorrect");
			} else {
				throw new FailedLoginException("Password Incorrect");

			}
		}

		private boolean testPassword(char[] cs, char[] password) {
			boolean result = true;
			int i = 0;
			while (i < cs.length && result) {
				if (cs[i] != password[i])
					result = false;
				i++;
			}
			return result;
		}

		public static void main(String[] args) throws Exception {
			try {
				LocateRegistry.createRegistry(1099);
			} catch (Exception e) {
				System.err.println("A registry has already been started");
			}

			ChatRoomImpl room = new ChatRoomImpl();
			Naming.rebind("ChatRoom", room);
		}
}
```