Compte-rendu

Thomas Guzik & Adam Morel

Partie 1

Le code est fait pour être simple. Je crois qu'il n'y a pas grand chose à dire.

Partie 2

Le code de cette partie s'ajoute graduellement au fur et à mesure des questions. On parlera donc du code fini, c'est à dire de celui du chat. Les explications sont simplifiés volontairement. Il s'agit de donner un aperçu du code.
Les 2 codes (client & serveur) commence par vérifier les arguments du main, cette vérification est assurée par un bloc de try cache. Si les arguments de la fonction main contiennent des erreurs, le nom du client et son charset sont fixés par défault à "Default_name" et "UTF-8".
Coté serveur, on doit fixer le charset et le nombre de thread possible. Ici par défault, ça sera "UTF-8" et 5.

```
try {
name = args(0);
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Error name is not in parameters\nName set as: Default_name");
name = "Default_name";
}

try {
charset = Charset.forName(args(1));
} catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Error charset not defined in parameters\nCharset set as: UTF-8");
} catch(UnsupportedCharsetException e) {
System.out.println("Unsupported charset\nCharset set as: UTF-8");
} finally {
charset = Charset.forName("UTF-8");
}
```

Code client

```
try {
		cs = Charset.forName(args(0));
	} catch(ArrayIndexOutOfBoundsException e) {
		System.out.println("Error charset not defined in parameters\nCharset set as: U
		cs = Charset.forName("UTF-8");
	}

	try {
		nbThreads = Integer.parseInt(args(1));
		if(nbThreads < 1 || nbThreads > 64) {
			throw new Exception();
		}
	} catch(Exception e) { // NumberFormatException, ArrayIndexOutOfBoundsExcep
```

```
                System.out.println("Invalid pool of threads, set the number as 5");
                nbThreads = 5;
            }
```

Code serveur

Coté serveur, on ouvre une ServerSocket sur le port 9999. La méthode accept renvoie la socket cliente. À chaque nouvelle socket cliente, on crée un nouveau thread pour la traiter.

```
try {
ServerSocket socketServeur = new ServerSocket(portEcoute);
        try {
                System.out.println("Server open with ip: " + socketServeur.getInetAddress() + " port:
                        + socketServeur.getLocalPort());

                Executor service = Executors.newFixedThreadPool(nbThreads);
                while (true) {
                        Socket socketVersUnClient = socketServeur.accept();
                        service.execute(new TraiteUnClient(socketVersUnClient, cs));
                }
        } finally {
                socketServeur.close();
        }
} catch (IOException e) {
e.printStackTrace();
}
```

Le code pour traiter un client est le suivant. On crée un buffer (BufferedReader) permettant de recevoir les messages et une liste de printer (PrinterWriter) permettant d'envoyer des messages. À la fin, tous ces instances sont fermées.

```
protected static void traiterSocketCliente(Socket socketVersUnClient, Charset cs) throws IO
        PrintWriter printer = creerPrinter(cs, socketVersUnClient);
        BufferedReader reader = creerReader(cs, socketVersUnClient);
        String name = "???";
        try {
                String msg;
                name = avoirNom(reader);
                if (name == null) {
                        envoyerMessage(printer, "Erreur envoi du nom invalide");
                } else {
                        System.out.println("New client: " + name);
                        ajouterPrinterSocketActives(printer);
                        envoyerAToutesLesSocketsActive("Bienvenue a " + name);
                        while ((msg = recevoirMessage(reader)) != null && (!msg.equalsIgnoreCase(o
                                System.out.println("from: " + name + " > Msg received: " + msg);
                                if (!msg.equalsIgnoreCase(outMsg))
                                        envoyerAToutesLesSocketsActive(name + "> " + msg);
                        }
                }
        } catch (IOException e) {
                System.out.println("Client closed");
```

```
            e.printStackTrace();
        } finally {
            socketVersUnClient.close();
            System.out.println("Client closed");
            enleverPrinterSocketActives(printer);
            envoyerAToutesLesSocketsActive("--->" + name + " Left");
            printer.close();
            reader.close();
        }
        socketVersUnClient.close();
        printer.close();
        reader.close();
    }
```

Retour coté client, nous demandons l'ip du serveur grâce à cette boucle

```
while (!checkip(ip)) {
        System.out.println(" IP du serveur ?");
        ip = lireMessageAuClavier();
}
```

Par la suite, nous créons un BufferedReader in, nous permettant de recevoir des messages, un PrintWriter out, et nous envoyer notre nom au serveur avec envoyerNom(out, name).
Nous commencons un nouveau thread chargé d'écouter les messages. Nous détaillerons le code de ce thread après. Le thread main s'occupe quand à lui de récupérer les messages au clavier et de les envoyer avec lireMessageAuClavier et envoyerMessage;

```
socket = new Socket(InetAddress.getByName(ip), port);

try {
in = creerReader(charset, socket);
PrintWriter out = creerPrinter(charset, socket);
String msg = "";
envoyerNom(out, name);

ReceiveRunnable recvRun = new ReceiveRunnable();
Thread t1 = new Thread(recvRun);
t1.start();

while (!msg.equalsIgnoreCase("fin")) {
        msg = lireMessageAuClavier();
        envoyerMessage(out, msg);
}

recvRun.quit();
t1.join();
end(socket, in, out);
} finally {
socket.close();
}
```

Le code du thread est le suivant. La variable recvOn permet l'activation ou la désactivation de la boucle.

```
private static class ReceiveRunnable implements Runnable {
private boolean recvOn;

public void run() {
	try {
		recvOn = true;
		while(recvOn) {
			System.out.println(recevoirMessage(in));
		}
	} catch(IOException e) {
		e.printStackTrace();
	}
}

public void quit() {
	recvOn = false;
}
}
```

Partie 3

Le code de cette troisième partie est similaire à celui de la partie précédente. Le thread main s'occupe d'envoyer les messages lu au clavier tandis qu'un nouveau thread s'occupe de recevoir les messages.

```
public static void main(String[] args) {
	ReceiveRunnable recvRun = new ReceiveRunnable();
	Thread t1 = new Thread(recvRun);
	t1.start();
	try {
		MulticastSocket s = new MulticastSocket(9999);

		System.out.println("Name ?");
		String name = lireMessageAuClavier();
		try {
			String msg = lireMessageAuClavier();
			while(!msg.equals("fin")) {
				envoyerMessage(s, name + ">" + msg);
				msg = lireMessageAuClavier();
			}

			System.out.println("Closing... time needed: 1s");
			recvRun.quit();
			t1.join();
			envoyerMessage(s, name + " quit the chat");
			s.close();
		}
		finally {
			recvRun.quit();
			s.close();
		}
	} catch (UnknownHostException e) {
		e.printStackTrace();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("FIN");
        System.out.println("Statut du thread " + t1.getName() + " = " +t1.getState());
    }
```

Le code du thread est celui ci :

```
    private static class ReceiveRunnable implements Runnable {
        private MulticastSocket recvSocket = null;
        private InetAddress group;

        public void run() {
            String recvMsg;
            recvOn = true;
            try {
                recvSocket = new MulticastSocket(9999);
                group = InetAddress.getByName("225.0.4.7");
                try {
                    recvSocket.joinGroup(group);
                    while(recvOn) {
                        recvMsg = recevoirMessage(recvSocket);
                        System.out.println(recvMsg);
                    }
                } finally {
                    quit();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public void quit() {
            if(recvSocket != null) {
                recvOn = false;
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                recvSocket.close();
            }
        }
    }
```

Une précision sur la fermeture. Pour recevoir un message nous utilisons la méthode MulticastSocket.receive(). Or cette fonction est bloquante, elle peut bloquer indéfinemment le thread. Le seul moyen de la rendre débloquante est de lui fixer un timeout. Ici le choix a

été fait de lui donner un timeout de 1s de pouvoir arreter le thread au besoin.
L'implémentation est donc celle ci et recvOn joue le role de dire si elle doit être bloquante ou non.

```
public static String recevoirMessage(MulticastSocket s) throws IOException {
	// bloque tant qu'un message n'est pas reçu
	byte() buf = new byte(256);
	DatagramPacket packet = new DatagramPacket(buf, buf.length);
	s.setSoTimeout(1000);
	while(recvOn && packet.getAddress() == null ) {
		try {
			s.receive(packet);
		} catch(SocketTimeoutException e) {/* On fait rien */}
	}
	return new String(packet.getData());
}
```