Programmation Java Avancée : Exercices

IUT de Villetaneuse — R&T 2ème année

Laure Petrucci

11 novembre 2015

1 Les exceptions et les fichiers texte

Exercice 1.1 : Une classe pour les réseaux

Le code des classes utilisées est fourni ci-après.

Question 1 : Expliquer ce que fait chacune des méthodes de la classe `Protocoles` en complétant les lignes de commentaires.

Question 2 : Tester le fonctionnement des classes qui sont fournies.

Question 3 : Quelles méthodes des classes `Machine`, `Routeur`, `Hub`, etc. devraient lever des exceptions ? Pourquoi ? Les modifier en conséquence.

Question 4 : Rajouter à la classe `Hub` :

- une méthode `enregistrerTexte` permettant de sauvegarder les caractéristiques du hub dans un fichier texte ;
- un constructeur recevant en paramètre le nom d'un fichier texte et reconstituant le hub à partir de ce fichier.

Le fichier contiendra sur la première ligne le nombre de ports du hub, puis sur chaque ligne le nom d'une machine, son adresse IP et son adresse MAC (et éventuellement ses autres adresses IP et MAC). Par exemple :

Listing 4 – hub.txt

```
8
r20003;140.10.3.4;01:01:01:01:01:01;120.10.3.1;02:02:02:02:02:02
r20101;140.10.5.2;03:03:03:03:03:03
r10204;140.10.2.8;04:04:04:04:04:04;130.10.1.1;05:05:05:05:05:05
```

java.util

Class ArrayList<E>

Constructor Summary

ArrayList()	Constructs an empty list with an initial capacity of ten.
ArrayList(Collection<? extends E> c) throws NullPointerException	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity) throws IllegalArgumentException	Constructs an empty list with the specified initial capacity.

Method Summary

boolean	add(E o) Appends the specified element to the end of this list.
void	add(int index, E element) throws IndexOutOfBoundsException Inserts the specified element at the specified position in this list.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified Collection into this list, starting at the specified position.
void	clear() Removes all of the elements from this list.
boolean	contains(Object elem) Returns true if this list contains the specified element.
E	get(int index) throws IndexOutOfBoundsException Returns the element at the specified position in this list.
int	indexOf(Object elem) Searches for the first occurence of the given argument, testing for equality using the equals method.
boolean	isEmpty() Tests if this list has no elements.
int	lastIndexOf(Object elem) Returns the index of the last occurrence of the specified object in this list.
E	remove(int index) throws IndexOutOfBoundsException Removes the element at the specified position in this list.
boolean	remove(Object o) Removes a single instance of the specified element from this list, if it is present (optional operation).
protected void	removeRange(int fromIndex, int toIndex) Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
E	set(int index, E element) throws IndexOutOfBoundsException Replaces the element at the specified position in this list with the specified element.
int	size() Returns the number of elements in this list.
Object[]	toArray() Returns an array containing all of the elements in this list in the correct order.
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

Class Protocoles

java.lang.Object
    Protocoles

---

public class **Protocoles** extends Object

Method Summary :

Modifier and Type	Method and Description
static char	calculerClasse(String adresseIP) calcule la classe (A, B ou C) à partir de l'adresse IP machine supposée correcte
static String	calculerIPReseau(String adresseIP) calcule l'adresse IP réseau à partir de l'adresse IP machine supposée correcte
static void	verifierIPMachine(String adresseIP) throws Exception vérifie que l'adresse IP d'une machine est une adresse IP valide de classe A, B ou C
static void	verifierIPReseau(String adresseIP) throws Exception vérifie que le paramètre représente une adresse réseau de classe A, B ou C
static void	verifierMac(String mac) throws Exception vérifie que le paramètre représente une adresse MAC

Methods inherited from class java.lang.Object : clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Listing 6 – Protocoles.java

```
1  import java.util.*;
2
3  public class Protocoles{
4    // La méthode troisPoints...
5    //
6    //
7    private static boolean troisPoints(String ch){
8      int nbPoints = 0;
9      for (int i = 0; i < ch.length(); i++)
10       if (ch.charAt(i) == '.')
11         nbPoints++;
12     return (nbPoints == 3);
13   } // fin troisPoints
14
15   // La méthode cinqDeuxPoints...
16   //
17   //
18   private static boolean cinqDeuxPoints(String ch){
19     int nbDeuxPoints=0;
20     for (int i = 0; i < ch.length(); i++)
21       if (ch.charAt(i) == ':')
22         nbDeuxPoints++;
23     return (nbDeuxPoints == 5);
24   } // fin cinqDeuxPoints
25
26   // La méthode verifierIPMachine...
27   //
28   //
29   public static void verifierIPMachine(String adresseIP) throws Exception{
30     //
31     //
32     if (adresseIP.length() > 15  || adresseIP.length() < 7)
33       throw new Exception("adresse␣IP␣incorrecte");
34     //
35     //
36     if (!troisPoints(adresseIP))
37       throw new Exception("adresse␣IP␣incorrecte");
38     //
39     //
40     if ((adresseIP.charAt(0) == '.') ||
41         (adresseIP.charAt(adresseIP.length() - 1) == '.'))
42       throw new Exception("adresse␣IP␣incorrecte");
43     //
44     //
45     if (adresseIP.indexOf("..") != -1)
46       throw new Exception("adresse␣IP␣incorrecte");
47     //
48     //
49     StringTokenizer st = new StringTokenizer(adresseIP,".");
50     int premier = Integer.parseInt(st.nextToken());
51     //
52     //
```

```
53      if ((premier > 223) || (premier <= 0))
54        throw new Exception("classe incorrecte");
55      //
56      //
57      int compteur = 0;
58      int i = -1;
59      while (st.hasMoreTokens()){
60        i = Integer.parseInt(st.nextToken());
61        if ((i >= 255) || (i < 0))
62          throw new Exception("adresse IP incorrecte");
63        compteur++;
64      }
65      //
66      //
67      if (i == 0)
68        throw new Exception("adresse IP incorrecte");
69      if (compteur != 3)
70        throw new Exception("adresse IP incorrecte");
71    } // fin verifierIPMachine
72
73    // La méthode verifierMac...
74    //
75    //
76    public static void verifierMac(String mac) throws Exception{
77      //
78      //
79      if (mac.indexOf("::") != -1)
80        throw new Exception("adresse MAC incorrecte");
81      //
82      //
83      if (mac.startsWith(":") || mac.endsWith(":"))
84        throw new Exception("adresse MAC incorrecte");
85      //
86      //
87      StringTokenizer st = new StringTokenizer(mac,":");
88      int compteur = 0;
89      //
90      //
91      while(st.hasMoreTokens()){
92        String morceau = st.nextToken();
93        //
94        //
95        if (morceau.length() != 2)
96          throw new Exception("adresse MAC incorrecte");
97        //
98        //
99        int i = Integer.parseInt(morceau,16);
100       if ((i > 255) || (i < 0))
101         throw new Exception("adresse MAC incorrecte");
102       compteur++;
103     }
104     //
105     //
106     if (compteur!=6)
```

```
107         throw new Exception("adresse MAC incorrecte");
108     } // fin verifierMac
109
110     // La méthode calculerClasse...
111     //
112     //
113     public static char calculerClasse(String adresseIP){
114       StringTokenizer st = new StringTokenizer(adresseIP,".");
115       int premier = Integer.parseInt(st.nextToken());
116       if (premier < 127)
117         return('A');
118       if  ((premier > 127) && (premier < 192))
119         return('B');
120       return('C');
121     } // fin calculerClasse
122
123     // La méthode calculerIPReseau...
124     //
125     //
126     public static String calculerIPReseau(String adresseIP){
127       String adresseIPReseau = "";
128       char classe = Protocoles.calculerClasse(adresseIP);
129       StringTokenizer st = new StringTokenizer(adresseIP,".");
130       switch (classe){
131         case 'A': adresseIPReseau = st.nextToken()+".0.0.0";
132                   break;
133         case 'B': adresseIPReseau = st.nextToken()+"."+st.nextToken()+".0.0";
134                   break;
135         case 'C':  adresseIPReseau = st.nextToken()+"."+st.nextToken()+"."+
136                             st.nextToken()+".0";
137                   break;
138       }
139           return(adresseIPReseau);
140     } // fin calculerIPReseau
141
142     // La méthode verifierClasse...
143     //
144     //
145     public static void verifierClasse(String adresseIP) throws Exception{
146       StringTokenizer st = new StringTokenizer(adresseIP,".");
147       int premier = Integer.parseInt(st.nextToken());
148       if ((premier > 223) || (premier <= 0))
149         throw new Exception("classe incorrecte");
150     } // fin verifierClasse
151
152     // La méthode verifierIPReseau...
153     //
154     //
155     public static void verifierIPReseau(String adresseIP) throws Exception{
156       //
157       //
158       Protocoles.verifierClasse(adresseIP);
159       char classe =  Protocoles.calculerClasse(adresseIP);
160       String[] t = new String[4];
```

```
161       StringTokenizer st = new StringTokenizer(adresseIP,".");
162       for (int i = 0; i < 4; i++)
163         t[i] = st.nextToken();
164       //
165       //
166       if ((classe == 'C') && !t[3].equals("0"))
167         throw new Exception("adresse réseau classe C incorrecte");
168       //
169       //
170       if ((classe == 'B') && (!t[2].equals("0") || !t[3].equals("0")))
171         throw new Exception("adresse réseau classe B incorrecte");
172       //
173       //
174       if ((classe == 'A') && (!t[1].equals("0") || !t[2].equals("0") ||
175           !t[3].equals("0")))
176         throw new Exception("adresse réseau classe A incorrecte");
177     } // fin verifierIPReseau
178
179     // La méthode verifierIPReseau2...
180     //
181     //
182     public static void verifierIPReseau2(String adresseIP) throws Exception{
183       Protocoles.verifierClasse(adresseIP);
184       String ip = Protocoles.calculerIPReseau(adresseIP);
185       if (!ip.equals(adresseIP))
186         throw new Exception("adresse réseau incorrecte");
187     } // fin verifierIPReseau2
188 } // fin de la classe Protocoles
```

Listing 7 – Donnees.java

```
1  public class Donnees{
2    private String donnees;
3
4    public Donnees(String donnees){
5      this.donnees = donnees;
6    }
7
8    public String getDonnees(){
9      return(this.donnees);
10   }
11
12   public String toString(){
13     return(this.donnees);
14   }
15
16   public boolean equals(Object o){
17     if (!(o instanceof Donnees))
18       return false;
19     Donnees p = (Donnees)o;
20     return(this.donnees.equals(p.donnees));
21   }
22 } // fin de la classe Donnees
```

Listing 8 – Trame.java

```
1  public class Trame{
2    private String DA;
3    private String SA;
4    private String EType;
5    private Donnees donnees;
6
7    public Trame(String DA,String SA,String EType,Donnees donnees){
8      this.DA = DA;
9      this.SA = SA;
10     this.EType = EType;
11     this.donnees = donnees;
12   }
13
14   public String getDA(){
15     return(this.DA);
16   }
17
18   public String getSA(){
19     return(this.SA);
20   }
21
22   public String getEType(){
23     return(this.EType);
24   }
25
26   public Donnees getDonnees(){
27     return(this.donnees);
28   }
29
30   public String toString(){
31     return(this.DA + ";" + this.SA + ";" + this.EType + ";" +
32       this.donnees);
33   }
34 } // fin de la classe Trame
```

Listing 9 – Hub.java

```
1  import java.util.*;
2
3  public class Hub{
4    private ArrayList <Machine> listeMachines;
5    private int nbMachines;
6
7    public Hub(int taille){
8      this.listeMachines = new ArrayList<Machine>();
9      for (int i = 0; i < taille; i++)
10       this.listeMachines.add(null);
11     this.nbMachines = 0;
12   }
13
14   public int size(){
15     return(this.listeMachines.size());
16   }
```

```
17
18    public int getNbMachines(){
19      return(this.nbMachines);
20    }
21
22    public ArrayList <Machine> getListeMachines(){
23      return(this.listeMachines);
24    }
25
26    public boolean estVide(){
27      return(this.nbMachines == 0);
28    }
29
30    public boolean estPlein(){
31      return(this.nbMachines == this.size());
32    }
33
34    // retourne la machine branchée sur le port numéro port
35    public Machine elementAt(int port){
36      if ((port < 0) || (port >= this.nbMachines))
37        return(null);
38      else return (this.listeMachines.get(port));
39    }
40
41    public boolean estOccupe(int port){
42      return(this.elementAt(port) != null);
43    }
44
45    // branche la machine m sur le port port sans précaution
46    // il n'y a pas besoin de vérifier le numéro de port
47    // parce que les méthodes qui utilisent setMachine font
48    // la vérification. Méthode utile pour passer de
49    // l'implémentation avec tableau à l'implémentation avec
50    // avec ArrayList
51    // la mettre en public pour pouvoir la tester ; private ensuite
52    private void setMachine(Machine m, int port){
53      this.listeMachines.set(port,m);
54    }
55
56    // initialisation interactive
57    // on peut brancher des machines ou des routeurs
58    // l'occupation des ports doit être continue
59    public void init(){
60      int reponse;
61      String encore;
62      Machine m;
63      Scanner sc = new Scanner(System.in);
64      do {
65        do {
66          System.out.print("1 : machine / 2 : routeur ");
67          reponse=sc.nextInt();
68        } while ((reponse < 1) || (reponse > 2));
69        if (reponse == 1)
70          m = new Machine();
```

```
71          else m = new Routeur();
72          m.init();
73          this.setMachine(m,this.nbMachines);
74          m.setHub(this);
75          this.nbMachines++;
76          System.out.print("encore ? : ");
77          encore = sc.next();
78        } while (encore.equals("o") && !this.estPlein());
79      }
80
81      public String toString(){
82        String s = "";
83        for (int i = 0; i < this.nbMachines; i++)
84          s = s + "\n" + this.elementAt(i);
85        return(s);
86      }
87
88      // retourne le numéro du port sur lequel la machine m est
89      // branchée ; retourne -1 si la machine m n'est pas branchée
90      public int positionMachine(Machine m){
91        return(this.listeMachines.lastIndexOf(m));
92      }
93
94      // retourne true si la machine m est branchée au hub
95      // et false sinon
96      public boolean contains(Machine m){
97        return (this.listeMachines.contains(m));
98      }
99
100     // on ne doit pas brancher deux fois la même machine
101     // on branche la machine sur le premier port libre
102     public boolean connecterMachine(Machine m){
103       if (this.estPlein() || this.contains(m))
104         return (false);
105       this.setMachine(m,this.nbMachines);
106       m.setHub(this);
107       this.nbMachines++;
108       return(true);
109     }
110
111     public boolean connecterRouteur(Routeur r,int eth){
112       if (this.estPlein() || this.contains(r))
113         return (false);
114       this.setMachine(r,this.nbMachines);
115       if (eth == 0)
116         r.setHub(this);
117       else r.setHub2(this);
118       this.nbMachines++;
119       return(true);
120     }
121
122     // on débranche la machine branchée sur le port port
123     // (si celui-ci est occupé), on la remplace par
124     // la dernière et on met cette dernière à null
```

```
125   public boolean debrancherMachine(int port){
126     if (this.estOccupe(port)){
127       this.elementAt(port).setHub(null);
128       Machine derniere = this.elementAt(this.nbMachines - 1);
129       this.setMachine(derniere,port);
130       this.setMachine(null,this.nbMachines - 1);
131       this.nbMachines--;
132       return(true);
133     } else return(false);
134   }
135
136   public boolean debrancherRouteur(int port,int eth){
137     if (this.estOccupe(port)){
138       if (eth == 0)
139         this.elementAt(port).setHub(null);
140       else ((Routeur)this.elementAt(port)).setHub2(null);
141       // on branche la dernière machine
142       // à la place de celle qu'on vient de débrancher
143       Machine derniere = this.elementAt(this.nbMachines - 1);
144       this.setMachine(derniere,port);
145       this.setMachine(null,this.nbMachines - 1);
146       this.nbMachines--;
147       return(true);
148     } else return(false);
149   }
150
151   // on débranche la machine m (si elle est branchée)
152   // on la remplace par la dernière et on met cette dernière à null
153   public boolean debrancherMachine(Machine m){
154     return(this.debrancherMachine(this.positionMachine(m)));
155   }
156
157   public boolean debrancherRouteur(Routeur r,int eth){
158     return(this.debrancherRouteur(this.positionMachine(r),eth));
159   }
160
161   // le hub émet la trame vers toutes les machines connectées
162   // sauf celle qui a émis la trame
163   public void emettreTrame(Machine machineEmettrice, Trame t){
164     for (int i = 0; i < this.nbMachines; i++)
165       if (this.elementAt(i) != machineEmettrice)
166         this.elementAt(i).recevoirTrame(t);
167   }
168
169   public String toStringComplet(){
170     String s = "taille du hub : " + this.size();
171     for (int i = 0; i < this.nbMachines; i++)
172       s = s + "\n" + this.elementAt(i).toStringComplet();
173     return(s);
174   }
175 } // fin de la classe Hub
```

Listing 10 – Machine.java

```
1  import java.util.*;
2
3  public class Machine{
4    private String nom;
5    private String adresseIP;
6    private String adresseMAC;
7    private Hub hub;
8    private ArrayList <Trame> tramesRecues;
9    private ArrayList <Trame> tramesEnvoyees;
10
11   public Machine(String nom, String adresseIP, String adresseMAC,
12       Hub hub){
13     this.nom = nom;
14     this.adresseIP = adresseIP;
15     this.adresseMAC = adresseMAC;
16     if (hub != null)
17       hub.connecterMachine(this);
18     this.tramesRecues = new ArrayList <Trame>();
19     this.tramesEnvoyees = new ArrayList <Trame>();
20   }
21
22   public Machine(){
23     this("","","",null);
24   }
25
26   public Machine(String nom, String adresseIP, String adresseMAC){
27     this(nom,adresseIP,adresseMAC,null);
28   }
29
30   public Machine(Machine m){
31     this(m.nom,m.adresseIP,m.adresseMAC,m.hub);
32   }
33
34   public void init(){
35     Scanner sc = new Scanner(System.in);
36     System.out.print("nom ? : ");
37     this.nom = sc.nextLine();
38     System.out.print("adresse IP ? : ");
39     this.adresseIP = sc.nextLine();
40     System.out.print("adresse MAC ? : ");
41     this.adresseMAC = sc.nextLine();
42   }
43
44   public String getNom(){
45     return(this.nom);
46   }
47
48   public String getAdresseIP(){
49     return(this.adresseIP);
50   }
51
52   public String getAdresseMAC(){
```

```
53      return(this.adresseMAC);
54    }
55
56    public void setHub(Hub hub){
57      this.hub = hub;
58    }
59
60    public boolean estConnectee(){
61      return(this.hub != null);
62    }
63
64    // retourne un tableau de quatre entiers, chacun d'eux correspondant
65    // à un nombre de l'adresse IP de la machine
66    public int[] getTableauIP(){
67      int tab[] = new int[4];
68      StringTokenizer st = new StringTokenizer(this.adresseIP,".");
69      for (int i = 0; i < 4; i++)
70        tab[i] = Integer.parseInt(st.nextToken());
71      return(tab);
72    }
73
74    // retourne A, B ou C selon que l'adresse IP de la machine
75    // est de classe A, B ou C
76    public char getClasse(){
77      int tab[] = this.getTableauIP();
78      int premier = tab[0];
79      if (premier < 127)
80        return('A');
81      if ((premier > 127) && (premier < 192))
82        return('B');
83      return('C');
84    }
85
86    // calcule l'adresse IP du réseau à partir de celle de la machine
87    public String getIPReseau(){
88      String adresseIPReseau = "";
89      char classe = this.getClasse();
90      StringTokenizer st = new StringTokenizer(this.adresseIP,".");
91      switch (classe){
92        case 'A': adresseIPReseau = st.nextToken() + ".0.0.0";
93                  break;
94        case 'B': adresseIPReseau = st.nextToken() + "." +
95                    st.nextToken() + ".0.0";
96                  break;
97        case 'C': adresseIPReseau = st.nextToken() + "." +
98                    st.nextToken() + "." + st.nextToken()+".0";
99                  break;
100     }
101     return(adresseIPReseau);
102   }
103
104   // permet de savoir si la machine fournissant la méthode
105   // appartient au même réseau que la machine passée en paramètre
106   public boolean estCompatible(Machine m){
```

```
107       return(this.getIPReseau().equals(m.getIPReseau()));
108     }
109
110     public String toString(){
111       return(new String(this.nom + ";" + this.adresseIP + ";" +
112         this.adresseMAC + ";" + this.tramesRecues + ";" +
113         this.tramesEnvoyees));
114     }
115
116     public String toStringComplet(){
117       String s = "nom : " + this.nom + "\n";
118       s = s + "adresse IP : " + this.adresseIP + "\n";
119       s = s + "adresse Mac : " + this.getAdresseMAC() + "\n";
120       s = s +"trames reçues : " + this.tramesRecues + "\n";
121       s = s +"trames envoyées : " + this.tramesEnvoyees+ "\n";
122       return(s);
123     }
124
125     public boolean equals(Object o){
126       if (!(o instanceof Machine))
127         return false;
128       Machine m = (Machine)o;
129       return(this.nom.equals(m.nom) &&
130         this.adresseIP.equals(m.adresseIP) &&
131         this.adresseMAC.equals(m.adresseMAC));
132     }
133
134     // renvoie -1 si la machine a une adresse IP plus petite
135     // que celle de la machine passée en paramètre,
136     // 1 si elle est plus grande et 0 si les deux machines ont
137     // la même adresse IP
138     public int compareTo(Machine m){
139       int tab1[] = this.getTableauIP();
140       int tab2[] = m.getTableauIP();
141       int i = 0;
142       while ((i < 4) && tab1[i] == tab2[i])
143         i++;
144       if (i == 4)
145         return(0);
146       if (tab1[i] < tab2[i])
147         return(-1);
148       return(1);
149     }
150
151     // connecte la machine au hub passé en paramètre
152     public boolean connecterAuHub(Hub hub){
153       if (hub.contains(this))
154         return(false);
155       this.debrancherDuHub();
156       return(hub.connecterMachine(this));
157     }
158
159     // débranche la machine du hub auquel elle est branchée
160     public boolean debrancherDuHub(){
```

```
161       if (this.estConnectee())
162         return this.hub.debrancherMachine(this);
163       else return(false);
164     }
165
166     // la machine fabrique la trame t à partir des données
167     // puis demande au hub auquel elle est branchée d'émettre
168     // la trame vers les autres machines connectées au hub.
169     // La trame est rajoutée à la liste des trames envoyées
170     public boolean emettreTrame(String DA,String EType,
171         Donnees donnees){
172       if (!this.estConnectee())
173         return(false);
174       Trame t = new Trame(DA,this.getAdresseMAC(),EType,donnees);
175       System.out.println("la machine " + this.nom +
176         " emet vers le hub la trame : " + t);
177       this.hub.emettreTrame(this,t);
178       this.tramesEnvoyees.add(t);
179       return(true);
180     }
181
182     // la machine affiche la trame que lui a envoyé le hub
183     // La trame est rajoutée à la liste des trames reçues
184     public void recevoirTrame(Trame t){
185       System.out.println("trame reçue par la machine " +
186         this.nom + " : " + t);
187       this.tramesRecues.add(t);
188     }
189   } // fin de la classe Machine
```

Listing 11 – Routeur.java

```
1  import java.util.*;
2
3  public class Routeur extends Machine{
4    private String adresseIP2;
5    private String adresseMAC2;
6    private Hub hub2;
7
8    public Routeur(){
9      super();
10     this.adresseIP2 = "";
11     this.adresseMAC2 = "";
12   }
13
14   public Routeur(String nom, String adresseIP, String adresseMAC,
15       String adresseIP2, String adresseMAC2, Hub hub, Hub hub2){
16     super(nom,adresseIP,adresseMAC,hub);
17     this.adresseIP2 = adresseIP2;
18     this.adresseMAC2 = adresseMAC2;
19     this.setHub2(hub2);
20   }
21
22   public Routeur(String nom, String adresseIP, String adresseMAC,
23       String adresseIP2, String adresseMAC2){
```

```
24      this( nom,adresseIP , adresseMAC,adresseIP2 ,adresseMAC2,null,null);
25    }
26
27    public Routeur(Machine m, String adresseIP2 , String adresseMAC2, Hub hub2){
28      super(m);
29      this.adresseIP2 = adresseIP2;
30      this.adresseMAC2 = adresseMAC2;
31      this.setHub2(hub2);
32    }
33
34    public Routeur(Routeur r){
35      super(r);
36      this.adresseIP2 = r.adresseIP2;
37      this.adresseMAC2 = r.adresseMAC2;
38      this.setHub2(r.hub2);
39    }
40
41    public void init(){
42      super.init();
43      Scanner sc = new Scanner(System.in);
44      System.out.print("deuxième␣adresse␣IP␣?␣:␣");
45      this.adresseIP2 = sc.next();
46      System.out.print("deuxième␣adresse␣MAC␣?␣:␣");
47      this.adresseMAC2 = sc.next();
48    }
49
50    public String getAdresseIP2(){
51      return(this.adresseIP2);
52    }
53
54    public String getAdresseMAC2(){
55      return(this.adresseMAC2);
56    }
57
58    public void setHub2(Hub hub){
59      this.hub2 = hub;
60    }
61
62    // retourne un tableau de quatre entiers, chacun de ces
63    // entiers correspondant à un nombre de la deuxième
64    // adresse IP du routeur
65    public int[] getTableauIP2(){
66      return(new Machine("",this.adresseIP2,this.adresseMAC2).
67          getTableauIP());
68    }
69
70    // retourne A, B ou C selon que la deuxième adresse IP
71    // du routeur est de classe A, B ou C.
72    public char getClasse2(){
73      return(new Machine("",this.adresseIP2,this.adresseMAC2).
74        getClasse());
75    }
76
77    public String getIPReseau2(){
```

```
78        return(new Machine("",this.adresseIP2,this.adresseMAC2).
79          getIPReseau());
80      }
81
82      // permet de savoir si le routeur fournissant la méthode
83      // appartient au même réseau que la machine passée en paramètre
84      public boolean estCompatible(Machine m){
85        return(super.estCompatible(m) ||
86          new Machine("",this.adresseIP2,this.adresseMAC2).
87            estCompatible(m));
88      }
89
90      public String toString(){
91        return (super.toString()+ ";" + this.adresseIP2 + ";" +
92          this.adresseMAC2);
93      }
94
95      public boolean equals (Object o){
96        if (!(o instanceof Routeur))
97          return false;
98        Routeur r = (Routeur)o;
99        return(super.equals(o) &&
100         this.adresseIP2.equals(r.adresseIP2) &&
101         this.adresseMAC2.equals(r.adresseMAC2));
102     }
103
104     // renvoie -1 si le routeur a la plus petite des quatre adresses IP,
105     // 1 si c'est le routeur reçu en paramètre qui a la plus petite des
106     // quatre adresses IP, et 0 si les quatre adresses IP sont égales
107     public int compareTo(Routeur r){
108       int tabMin1[];
109       int tabMin2[];
110       int tab1[] = this.getTableauIP();
111       int tab2[] = this.getTableauIP2();
112       int tab3[] = r.getTableauIP();
113       int tab4[] = r.getTableauIP2();
114       // recherche de la plus petite adresse IP de this
115       int i = 0;
116       while ((i < 4) && (tab1[i] == tab2[i]))
117         i++;
118       if ((i == 4) || (tab1[i] < tab2[i]))
119         tabMin1 = tab1;
120       else tabMin1 = tab2;
121       // recherche de la plus petite adresse IP de r
122       i = 0;
123       while ((i < 4) && (tab3[i] == tab4[i]))
124         i++;
125       if ((i == 4) || (tab3[i] < tab4[i]))
126         tabMin2 = tab3;
127       else tabMin2 = tab4;
128       // comparaison des deux tabMin
129       i = 0;
130       while ((i < 4) && (tabMin1[i] == tabMin2[i]))
131         i++;
```

```
132       if (i == 4)
133         return(0);
134       if (tabMin1[i] < tabMin2[i])
135         return(-1);
136       else return(1);
137     }
138   } // fin de la classe Routeur
```

Listing 12 – TestMenu.java

```
1  import java.util.Scanner;
2
3  public class TestMenu{
4    public static void main (String[] args){
5      Hub hub = new Hub(5);
6      Machine [] tabMachines = new Machine[5];
7      int reponse,position;
8      int nbMachines; // nombre de machines dans le tableau du main
9      Scanner sc = new Scanner(System.in);
10     System.out.println(
11       "1/ initialiser le hub vide et utiliser des machines prédéfinies");
12     System.out.println("2/ initialiser le hub à partir de la méthode init");
13     do {
14       System.out.print("votre choix ? : ");
15       reponse=sc.nextInt();
16     } while ((reponse < 1) || (reponse > 2));
17     if (reponse==1){
18       // on utilise 5 machines prédéfinies
19       tabMachines[0] = new Machine("menelas","1","1");
20       tabMachines[1] = new Machine("polux","2","2");
21       tabMachines[2] = new Machine("ajax","3","3");
22       tabMachines[3] = new Machine("enee","4","4");
23       tabMachines[4] = new Machine("cresus","5","5");
24       nbMachines = 5;
25     } else {
26       // sinon, on fait construire le réseau à l'utilisateur
27       // en commençant par créer un hub
28       hub.init();
29       for (int i = 0;i < hub.getNbMachines(); i++)
30         tabMachines[i] = hub.elementAt(i);
31       nbMachines = hub.getNbMachines();
32     }
33     do {
34       System.out.println("1/ tester si le hub est vide ");
35       System.out.println("2/ tester si le hub est plein ");
36       System.out.println(
37         "3/ tester si une machine donnée est branchée sur le hub");
38       System.out.println(
39         "4/ brancher une machine à partir de la classe Machine");
40       System.out.println(
41         "5/ brancher une machine à partir de la classe Hub");
42       System.out.println(
43         "6/ créer et connecter une nouvelle machine à partir de la classe Machine");
44       System.out.println(
45         "7/ débrancher une machine à partir de la classe Machine");
```

```
46          System.out.println(
47            "8/ débrancher une machine à partir de la classe Hub");
48          System.out.println(
49            "9/ débrancher une machine d'un port précis du hub");
50          System.out.println("10/ émettre une trame sur le réseau");
51          System.out.println("11/ afficher le contenu du hub ");
52          System.out.println("0/ terminer : ");
53          do {
54            System.out.print("votre choix ? : ");
55            reponse = sc.nextInt();
56          } while ((reponse < 0) || (reponse > 11));
57          switch (reponse){
58            case 1 :  if (hub.estVide())
59                        System.out.println("hub vide");
60                      else System.out.println("hub non vide");
61                      break;
62            case 2 :  if (hub.estPlein())
63                        System.out.println("hub plein");
64                      else System.out.println("hub non plein");
65                      break;
66            case 3 :  do {
67                        System.out.print(
68                          "position de la machine dans le tableau local ? : ");
69                        position = sc.nextInt();
70                      } while ((position < 0) || (position > nbMachines - 1));
71                      if (hub.contains(tabMachines[position]))
72                        System.out.println("la machine est branchée sur le hub");
73                      else System.out.println(
74                             "la machine n'est pas branchée sur le hub");
75                      break;
76            case 4 :  do {
77                        System.out.print(
78                          "position de la machine dans le tableau local ? : ");
79                        position = sc.nextInt();
80                      } while ((position < 0) || (position > nbMachines - 1));
81                      if (tabMachines[position].connecterAuHub(hub))
82                        System.out.println("la connexion a réussi");
83                      else System.out.println("la connexion a échoué");
84                      break;
85            case 5 :  do {
86                        System.out.print(
87                          "position de la machine dans le tableau local ? : ");
88                        position = sc.nextInt();
89                      } while ((position < 0) || (position > nbMachines - 1));
90                      if (hub.connecterMachine(tabMachines[position]))
91                        System.out.println("la connexion a réussi");
92                      else System.out.println("la connexion a échoué");
93                      break;
94            case 6 :  if (nbMachines == tabMachines.length)
95                        System.out.println("tableau plein");
96                      else {
97                        Machine m = new Machine();
98                        m.init();
99                        if (m.connecterAuHub(hub)){
```

```
100                         System.out.println("la connexion a réussi");
101                         tabMachines[nbMachines] = m;
102                         nbMachines++;
103                       } else System.out.println("la connexion a échoué");
104                     }
105                     break;
106           case 7 :  do {
107                       System.out.print(
108                         "position de la machine dans le tableau local ? : ");
109                       position = sc.nextInt();
110                     } while ((position < 0) || (position > nbMachines - 1));
111                     if (tabMachines[position].debrancherDuHub())
112                       System.out.println("la machine a été débranchée");
113                     else System.out.println("problème");
114                     break;
115           case 8 :  do {
116                       System.out.print(
117                         "position de la machine dans le tableau local ? : ");
118                       position=sc.nextInt();
119                     } while ((position < 0) || (position > nbMachines - 1));
120                     if (hub.debrancherMachine(tabMachines[position]))
121                       System.out.println("la machine a été débranchée");
122                     else System.out.println("problème");
123                     break;
124           case 9 :  System.out.print("position du port ? : ");
125                     position = sc.nextInt();
126                     if (hub.debrancherMachine(position))
127                       System.out.println("la machine a été débranchée");
128                     else System.out.println("problème");
129                     break;
130           case 10 : System.out.print(
131             "numéro de port de la machine emettrice : ");
132                     position = sc.nextInt();
133                     if (hub.estOccupe(position)){
134                       System.out.print("adresse MAC poste destination : ");
135                       String adresseMAC = sc.next();
136                       System.out.print("message : ");
137                       String message = sc.next();
138                       Donnees donnees = new Donnees(message);
139                       hub.elementAt(position).
140                         emettreTrame(donnees,adresseMAC,"etype");
141                     } else  System.out.println(
142                               "aucune machine connectée sur ce port");
143                     break;
144           case 11 : System.out.println(hub);
145                     break;
146         } // fin switch
147       } while (reponse != 0);
148     }  // fin main
149   } // fin de la classe TestMenu
```

2 Classes abstraites

Exercice 2.1 : Analyse de trame dans la classe `Machine`

Rappel du format d'une trame Ethernet II :

Préambule	SFD	DA	SA	EType	données	Bourrage	FCS
7*o*	1*o*	6*o*	6*o*	2*o*	0–1500*o*	0–46*o*	4*o*

Dans ce qui suit, nous ne considérons ni le préambule, ni le SFD, ni le FCS. Nous nous affranchirons aussi de la nécessité de rajouter du bourrage dans le cas d'une longueur de trame inférieure à 64 octets.

Le type des données (LPDU) est déterminé par le `EType`. Il peut s'agir par exemple de données de type ARP, IP, ...

Dans ces conditions, on pourrait définir le type de trame par :

```
1 public class Trame{
2   private String DA;
3   private String SA;
4   private String EType;
5   private DonneesARP donnees;
6   ...
7 }
```

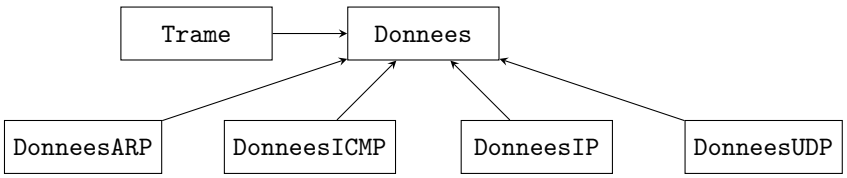
ou bien

```
1 public class Trame{
2   private String DA;
3   private String SA;
4   private String EType;
5   private DonneesIP donnees;
6   ...
7 }
```

ou encore d'autres, avec d'autres types de données.

De plus, certains types de données peuvent être considérés comme *terminaux* (par exemple les données ARP), tandis que d'autres (par exemple IP) contiennent eux aussi des données dont le type est déterminé par la valeur du champ protocole. Si le paquet IP contient des données de type UDP par exemple, ces données contiennent elles-mêmes des données dont le type sera fixé soit par la valeur du port source soit par celle du port destination.

Pour remédier à ce problème, on définit une *classe abstraite* `Donnees` dont hériteront les classes réelles `DonneesARP`, `DonneesIP`, ...



La classe `Trame` devient alors :

```
1 public class Trame{
2   private String DA;
3   private String SA;
4   private String EType;
5   private Donnees donnees;
6   ...
7 }
```

Lors de l'appel au constructeur de la classe `Trame`, les données seront du bon type : `DonneesARP`, `DonneesIP`, ...

Pour résoudre le problème des classes *non terminales*, la classe `Donnees` contiendra une variable d'instance nommée `donneeNiveauSuperieur` qui sera de type `Donnees` :

```
1 public abstract class Donnees{
2   private Donnees donneeNiveauSuperieur;
3   ...
4 }
```

En pratique, dans la classe `DonneesIP`, cette variable d'instance sera de type `DonneesICMP` ou `DonneesUSB` ou ...

Question 1 : Écrire un programme de test dans lequel vous instancerez un `Hub` et quelques `Machine`s que vous connecterez au `Hub`.

Question 2 : Compléter le programme en utilisant les classes de type `Donnees` pour simuler un `ARP.request` depuis la machine A, concernant la machine B puis la réponse de la machine A à la machine B.

Question 3 : Simuler un `ping` entre les machines C et D (`ECHO.request` suivi de `ECHO.reply`).

Question 4 : Rajouter à la classe `Machine` une méthode :

```
public void envoyerArpRequest(String adresseIPDestination)
    throws Exception
```

Question 5 : Rajouter à la classe `Machine` une méthode :

```
public void envoyerEchoRequest(String adresseIPDestination,
    String adresseMACDestination) throws Exception
```

Question 6 : Modifier le classe `Machine` afin que celle-ci devienne capable d'analyser les trames qu'elle reçoit et qu'éventuellement elle y réponde. Rédiger la méthode `analyserARP` : lorsque la machine reçoit un `ARP.request` et que c'est elle qui est concernée, elle répond par un `ARP.reply` ; lorsquelle reçoit un `ARP.reply` et qu'elle est destinatrice de la trame, elle affiche l'adresse MAC recherchée.

La méthode `recevoirTrame` devient :

```
1 public void recevoirTrame(Trame t){
2   System.out.println("Trame reçue par la machine" + this.nom + ":" + t);
3   this.tramesRecues.add(t);
4   try {
5         this.AnalyserTrameEthernet(t);
6   } catch(Exception e){System.out.println(e);}
7 }
```

La méthode `analyserTrameEthernet` se base sur le `EType` de la trame :

```
1  public void analyserTrameEthernet(Trame t) throws Exception{
2    String eType = t.getEType();
3    if (eType.equals("0806")){ // ARP
4      DonneesARP donneesARP = (DonneesARP)t.getDonnees();
5          this.analyserARP(donneesARP);
6    }
7    if (eType.equals("0800")){ // IP
8      DonneeIP donneeIP = (DonneesIP)t.getDonnees();
9          this.analyserIP(donneesIP,t.getSA());
10   }
11   ...
12 }
```

Class Donnees

java.lang.Object
    Donnees

Direct Known Subclasses:
    DonneesARP, DonneesICMP, DonneesIP, DonneesUDP

---

public abstract class Donnees extends java.lang.Object

Field Summary

Modifier and Type	Field and Description
private Donnees	donneesNiveauSuperieur

Constructor Summary

Constructor and Description
Donnees()
Donnees(Donnees donneesNiveauSuperieur)

Method Summary

Modifier and Type	Method and Description
boolean	equals(java.lang.Object o)
Donnees	getDonneesNiveauSuperieur()

Methods inherited from class java.lang.Object :

clone, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class DonneesARP

java.lang.Object
    Donnees
        DonneesARP

3

public class DonneesARP extends Donnees

Field Summary

Modifier and Type	Field and Description
private int	opCode
private java.lang.String	SHA
private java.lang.String	SIPA
private java.lang.String	THA
private java.lang.String	TIPA

Constructor Summary

Constructor and Description
DonneesARP(int opCode, java.lang.String SHA, java.lang.String SIPA, java.lang.String THA, java.lang.String TIPA) throws Exception

Method Summary

Modifier and Type	Method and Description
int	getOpCode()
java.lang.String	getSHA()
java.lang.String	getSIPA()
java.lang.String	getTHA()
java.lang.String	getTIPA()
java.lang.String	toString()

Methods inherited from class Donnees

equals, getDonneesNiveauSuperieur

Methods inherited from class java.lang.Object

clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class DonneesIP

java.lang.Object
    Donnees
        DonneesIP

---

public class DonneesIP extends Donnees

Field Summary

Modifier and Type	Field and Description
private int	protocole
private java.lang.String	SIP
private java.lang.String	TIP
private int	version

Constructor Summary

Constructor and Description
DonneesIP(int version, int protocole, java.lang.String SIP, java.lang.String TIP, Donnees donneesNiveauSup) throws Exception

Method Summary

Modifier and Type	Method and Description
int	getProtocole()
java.lang.String	getSIP()
java.lang.String	getTIP()
int	getVersion()
java.lang.String	toString()

Methods inherited from class Donnees

equals, getDonneesNiveauSuperieur

Methods inherited from class java.lang.Object

clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Class DonneesICMP

java.lang.Object
    Donnees
        DonneesICMP

---

public class DonneesICMP
extends Donnees

Field Summary

Modifier and Type	Field and Description
private int	code
private int	type

Constructor Summary

Constructor and Description
DonneesICMP(int type, int code)

Method Summary

Modifier and Type	Method and Description
int	getCode()
int	getType()
java.lang.String	toString()

Methods inherited from class Donnees

equals, getDonneesNiveauSuperieur

Methods inherited from class java.lang.Object

clone, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

3 Client/Serveur UDP

Exercice 3.1 : Services

Question 1 : Que contient le fichier `/etc/services` ? À quoi sert-il ?

Question 2 : `xinetd` est un *méta-serveur* qui peut écouter sur plusieurs ports, lancer les démons associés lorsque c'est nécessaire et les arrêter quand ils ne servent plus. Regardez le contenu des fichiers `echo-udp`, `daytime-udp`, `echo` et `daytime` dans le répertoire `/etc/xinetd.d`. Que peut-on en conclure quant à l'activation des ports 7 (`echo`) et 13 (`daytime`) en `udp` et `tcp` ? Faites en sorte que les quatre services soient activés sur votre machine en modifiant si nécessaire les fichiers de configuration.

Question 3 : Relancez `xinetd` :

```
/etc/init.d/xinetd restart
```

Vérifez le bon fonctionnement des service 7 et 13 sur votre machine puis sur celle de votre voisin, en utilisant le client `nc` en UDP puis en TCP, puis avec le client TCP `telnet`.

Exercice 3.2 : Programmation java Client/Serveur

Question 1 : Quelles sont les deux méthodes fondamentales que doivent fournir un client ou un serveur ?

Question 2 : En déduire une première version d'une interface `ClientServeur` qui sera complétée par la suite, et implémentée par tous les clients et tous les serveurs.

Exercice 3.3 : Client/Serveur UDP

Question 1 : Rédigez une classe `ClientUDP` implémentant l'interface `ClientServeur`. Les listings 21 et 22 en donnent deux exemples d'utilisation.

Listing 21 – TestClientUDPEcho.java

```
1  public class TestClientUDPEcho{
2    public static void main(String[] args){
3      if (args.length != 2){
4        System.out.println("usage : java TestClientUDPEcho <serveur> <message>");
5        System.exit(0);
6      }
7      try{
8        ClientUDP client1 = new ClientUDP(args[0],7);
9        client1.envoyerMessage(args[1]);
10       System.out.println(client1.recevoirMessage());
```

```
11          client1.fermer();
12          ClientUDP client2 = new ClientUDP();
13          client2.envoyerMessage(args[1],args[0],7);
14          System.out.println(client2.recevoirMessage());
15          client2.fermer();
16        } catch(Exception e){
17          System.out.println(e);
18        }
19      }
20    } // fin de la classe TestClientUDPEcho
```

Listing 22 – TestClientUDPDaytime.java

```
1   public class TestClientUDPDaytime{
2     public static void main(String[] args){
3       if (args.length != 1){
4         System.out.println("usage : java TestClientUDPDaytime <serveur>");
5         System.exit(0);
6       }
7       try{
8         ClientUDP client = new ClientUDP(args[0],13);
9         client.envoyerMessage("");
10        System.out.println(client.recevoirMessage());
11        client.fermer();
12      } catch(Exception e){
13        System.out.println(e);
14      }
15    }
16  } // fin de la classe TestClientUDPDaytime
```

Question 2 : Utilisez ces programmes avec votre machine comme serveur puis avec celle de votre voisin comme serveur.

Question 3 : Rédigez la classe abstraite `ServeurUDP` implémentant l'interface `ClientServeur` puis la classe `ServeurUDP7777` fournissant sur le port 7777 le même service que `udp echo` sur le port 7.

Question 4 : Testez votre serveur avec votre client UDP dans deux fenêtres différentes, puis avec le client de votre voisin.

Remarque : Pour éviter les conflits sur les numéros de port, vous devez choisir un numéro de port différent de ceux déjà utilisés par le système. Il faut vérifier qu'il n'est pas déjà utilisé dans `/etc/services` et ne fait pas non plus partie des numéros attribués dynamiquement aux serveurs RPC (Remote Procedure Call), par la commande `rpcinfo -p` (ou `rpc_dump`), ni déjà utiilisés par divers processus (`netstat -an`). Dans tous les cas choisissez un numéro supérieur à 1024.

Interface ClientServeur

All Known Implementing Classes : ClientUDP, ServeurUDP, ServeurUDP7777

---

public interface ClientServeur

Method Summary :

Modifier and Type	Method and Description
void	envoyerDatas(byte[] datas) throws Exception
byte[]	recevoirDatas() throws Exception

Class ClientUDP

java.lang.Object
    ClientUDP

All Implemented Interfaces : ClientServeur

---

public class ClientUDP extends Object implements ClientServeur

Field Summary :

Modifier and Type	Field and Description
private InetAddress	adresseIPDistante
private int	portDistant
private DatagramSocket	socket

Constructor Summary :

Constructor and Description
ClientUDP() throws Exception
ClientUDP(String nomServeur, int portServeur) throws Exception

Method Summary :

Modifier and Type	Method and Description
void	envoyerDatas(byte[] datas) throws Exception
void	envoyerDatas(byte[] datas, InetAddress adresseIPDistante, int portDistant) throws Exception
void	envoyerMessage(String message) throws Exception
void	envoyerMessage(String message, String adresseIPDistante, int portDistant) throws Exception
void	fermer()
byte[]	recevoirDatas() throws Exception
String	recevoirMessage() throws Exception

Methods inherited from class java.lang.Object :

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class ServeurUDP

java.lang.Object
ServeurUDP

All Implemented Interfaces : ClientServeur
Direct Known Subclasses : ServeurUDP7777

---

public abstract class ServeurUDP extends Object implements ClientServeur

Field Summary :

Modifier and Type	Field and Description
protected InetAddress	adresseIPDistante
private int	portDistant
private DatagramSocket	socket

Constructor Summary :

Constructor and Description
ServeurUDP(int port) throws Exception

Method Summary :

Modifier and Type	Method and Description
void	envoyerDatas(byte[] datas) throws Exception
void	envoyerMessage(String message) throws Exception
void	fermer()
abstract void	fonctionner() throws Exception
byte[]	recevoirDatas() throws Exception
String	recevoirMessage() throws Exception

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Class ServeurUDP7777

java.lang.Object
    ServeurUDP
        ServeurUDP7777

All Implemented Interfaces : ClientServeur

---

public class ServeurUDP7777 extends ServeurUDP

Field Summary :

Fields inherited from class ServeurUDP :

adresseIPDistante

Constructor Summary :

Constructor and Description
ServeurUDP7777() throws Exception

Method Summary :

Modifier and Type	Method and Description
void	fonctionner() throws Exception

Methods inherited from class ServeurUDP :

envoyerDatas, envoyerMessage, fermer, recevoirDatas, recevoirMessage

Methods inherited from class java.lang.Object :

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

4 Client/Serveur TCP

Exercice 4.1 : Client TCP

Question 1 : Rédigez une classe `ClientTCP` implémentant l'interface `ClientServeur`. Contrairement à la classe `DatagramSocket`, la classe `Socket` utilise deux fichiers texte, l'un en entrée et l'autre en sortie. Dans le cas d'un client TCP, le fichier en sortie ira vers le serveur, celui en entrée arrivera du serveur.

Les listings 28 et 29 donnent deux exemples d'utilisation de cette classe.

Listing 28 – TestClientTCPEcho.java

```
1  public class TestClientTCPEcho{
2    public static void main(String[] args){
3      if (args.length != 2){
4        System.out.println("usage : java TestClientTCPEcho <serveur> <message>");
5        System.exit(0);
6      }
7      try{
8        ClientTCP client = new ClientTCP(args[0],7);
9        client.envoyerMessage(args[1]);
10       System.out.println(client.recevoirMessage());
11       client.envoyerMessage("bis : " + args[1]);
12       System.out.println(client.recevoirMessage());
13       client.fermer();
14     } catch(Exception e){
15       System.out.println(e);
16     }
17   }
18 } // fin de la classe TestClientTCPEcho
```

Listing 29 – TestClientTCPDaytime.java

```
1  public class TestClientTCPDaytime{
2    public static void main(String[] args){
3      if (args.length != 1){
4        System.out.println("usage : java TestClientTCPDaytime <serveur>");
5        System.exit(0);
6      }
7      try{
8        ClientTCP client = new ClientTCP(args[0],13);
9        client.envoyerMessage("");
10       System.out.println(client.recevoirMessage());
11       client.fermer();
12     } catch(Exception e){
13       System.out.println(e);
14     }
15   }
16 } // fin de la classe TestClientTCPDaytime
```

Question 2 : Utilisez ces programmes avec votre machine comme serveur puis avec celle de votre voisin comme serveur.

Question 3 : Utilisez les clients TCP `telnet` ou `nc` pour :

- récupérer le contenu d'un fichier d'extension `.html` de votre machine puis de la machine de votre voisin ;
- récupérer la réponse du script `php` du listing exécuté sur votre machine puis sur celle de votre voisin.

```
1 <script language="php">
2   if (isset($_GET['x']))
3     print($_GET['x'] * $_GET['x']);
4   else print("pb paramètres");
5 </script>
```

Question 4 : Rédigez une classe `ClientTCPWeb` permettant d'envoyer des requêtes http à un serveur web et d'afficher sa réponse.

Question 5 : Utilisez une instance de cette classe pour faire la même chose qu'à la question 3.

Exercice 4.2 : Serveur TCP

Question 1 : Rédigez une classe `MultiServeurTCPEcho`. Son constructeur recevra en paramètre un numéro de port (supérieur à 1024) sur lequel le serveur sera en écoute. Elle possèdera une méthode `run` (rédigée sur le modèle de celle des threads) dans laquele elle se mettra indéfiniment en attente des connexions des clients. Elle déléguera le traitement de chaque connexion à une instance de la classe `ServeurTCPEcho` (fichier `.class` fourni). Chaque instance de `ServeurTCPEcho` retourne au client le message que celui-ci lui a envoyé. Plusieurs communications client/serveur peuvent avoir lieu simultanément.

Question 2 : Rédigez une classe `LancerMultiServeurTCPEcho` dont la méthode `main` démarrera dans une première console le multi-serveur en écoute sur un port passé en paramètre.

Question 3 : Rédigez une classe de test `TestClientMultiServeur` qui utilisera un client TCP dans une seconde console pour envoyer un message au multi-serveur et afficher la réponse de `ServeurTCPEcho`.

Question 4 : Rédigez votre propre classe `ServeurTCPEcho` Cette classe présente de nombreuses analogies avec la classe `ClientTCP`. Son constructeur reçoit la connexion client de `MultiServeurTCPEcho`.

public class **MultiServeurTCPEcho**

Field Summary	
`private java.net.ServerSocket`	ss

Constructor Summary	
MultiServeurTCPEcho `(int port)` throws Exception	

Method Summary	
`void`	demarrer `()` throws Exception méthode qui s'exécute tant que le serveur fonctionne : traite plusieurs connexions en en déléguant le traitement à des threads ou plus exactement à des instances de la classe ServeurTCPEcho qui implémente un thread

---

public class **ServeurTCPEcho** implements java.lang.Runnable

Field Summary	
private java.io.BufferedReader	in
private java.io.PrintWriter	out
private java.net.Socket	socket

Constructor Summary	
ServeurTCPEcho(java.net.Socket socket) throws IOException	

Method Summary	
void	envoyerMessage(java.lang.String message) throws IOException
void	fermer() throws IOException
java.lang.String	recevoirMessage() throws IOException
void	demarrer() //invoque la méthode start du thread qui elle-même déclanche la méthode run()

Methods inherited from interface java.lang.Runnable
void run()