Université Bordeaux I
U.F.R. Mathématiques et Informatique

IUP MIAGe 3^{eme} année
Année 2003-2004

Lundi 12 janvier 2004, 10 heures - 11 heures 30
Tous documents autorisés.

Programmation par Objets
Alain Griffault

SUJET + CORRIGE

Introduction

Une personne naît, puis meurt. Durant sa vie, elle a pu se marier, divorcer et devenir veuf ou veuve. La figure 1 représente toutes les évolutions possibles au cours de l'existence d'une personne.

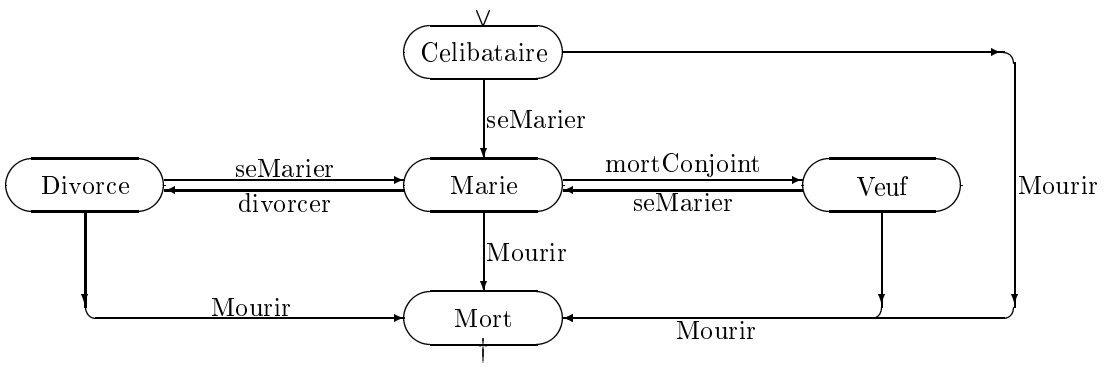

FIG. 1 – Le cycle de la vie

Le sujet porte sur la gestion (simplifiée) d'un groupe de personnes. L'étude fait évoluer une première version par l'utilisation de modèles de conception, puis de `Thread`.

1 Les étapes de la vie

Quatre personnes naissent, puis on affiche éventuellement trois étapes de leurs vies.
La classe `Personne` :

```
1   import java.util.*;
2   import java.text.*;
3   public class Personne{
4       private static int numero = 0;
5       private static Random alea = new Random();
6       private static SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
7       private int ident;
8       private Date naissance, deces;
9       private boolean masculin;
10      private boolean celibataire, marie, divorce, veuf, mort;
11
12      public Personne(boolean garcon){
13          ident = numero++;
14          naissance = new Date();
15          deces = null;
16          masculin = garcon;
```

```
17          celibataire = true;
18          marie = false;
19          divorce = false;
20          veuf = false;
21          mort = false;
22      }
23
24      public String toString(){
25          String res ="";
26          if (masculin) {
27              res +="Monsieur "+ident+", ne le ";
28              res += sdf.format(naissance)+", est";
29              if (celibataire) res +=" celibataire.\n";
30              if (marie)       res +=" marie.\n";
31              if (divorce)     res +=" divorce.\n";
32              if (veuf)        res +=" veuf.\n";
33              if (mort)        res +=" mort le "+sdf.format(deces)+".\n";
34          }
35          else {
36              if (celibataire){
37                  res +="Mademoiselle "+ident+", est nee le ";
38                  res += sdf.format(naissance)+".\n";
39              }
40              else{
41                  res +="Madame "+ident+", nee le ";
42                  res += sdf.format(naissance)+", est";
43              }
44              if (marie)       res +=" mariee.\n";
45              if (divorce)     res +=" divorcee.\n";
46              if (veuf)        res +=" veuve.\n";
47              if (mort)        res +=" morte le "+sdf.format(deces)+".\n";
48          }
49          return res;
50      }
51
52      public void mourir() throws Exception{
53          if (mort)
54              throw new Exception("Impossible");
55          else {
56              deces = new Date();
57              celibataire = false;
58              marie = false;
59              divorce = false;
60              veuf = false;
61              mort = true;
62          }
63      }
64
65      public void seMarier() throws Exception{
66          if (mort || marie)
67              throw new Exception("Impossible");
68          else {
69              celibataire = false;
70              marie = true;
71              divorce = false;
72              veuf = false;
73              mort = false;
```

```
74          }
75      }
76
77      public void divorcer() throws Exception{
78          if (mort || celibataire || veuf || divorce)
79              throw new Exception("Impossible");
80          else {
81              celibataire = false;
82              marie = false;
83              divorce = true;
84              veuf = false;
85              mort = false;
86          }
87      }
88
89      public void mortConjoint() throws Exception{
90          if (mort || celibataire || veuf || divorce)
91              throw new Exception("Impossible");
92          else {
93              celibataire = false;
94              marie = false;
95              divorce = false;
96              veuf = true;
97              mort = false;
98          }
99      }
100
101     public void vieillir() {
102         try {Thread.sleep(2000);}       // 2 secondes d'endormissement
103         catch(InterruptedException ie){}
104         finally {
105             int a = alea.nextInt(4);
106             try {
107                 switch (a) {
108                 case 0 : mourir(); break;
109                 case 1 : seMarier(); break;
110                 case 2 : divorcer(); break;
111                 case 3 : mortConjoint();
112                 }
113                 System.out.print(this);
114             }
115             catch (Exception e){} // Exceptions lors de chgt de situation
116         }
117     }
118
119 }
```

La classe `Test` :

```
1  import java.util.*;
2  public class Test{
3      public static void main(String[] args){
4          Personne[] personne = new Personne[4];
5          Random alea = new Random();
6          System.out.println("\tNaissances");
7          for (int i=0; i<personne.length; i++){
8              personne[i] = new Personne(alea.nextBoolean()); //Homme ou Femme
9              System.out.print(personne[i]);
10         }
```

```
11          System.out.println("\tQuelques etapes de la vie");
12          for (int time=0; time<3; time++)
13              for (int i=0; i<personne.length; i++)
14                  personne[i].vieillir();
15      }
16  }
```

et un exemple de sortie générée.

```
1           Naissances
2   Mademoiselle 0, est nee le 11/01/2004 17:02:19.
3   Mademoiselle 1, est nee le 11/01/2004 17:02:19.
4   Monsieur 2, ne le 11/01/2004 17:02:19, est celibataire.
5   Monsieur 3, ne le 11/01/2004 17:02:19, est celibataire.
6           Quelques etapes de la vie
7   Madame 0, nee le 11/01/2004 17:02:19, est mariee.
8   Madame 1, nee le 11/01/2004 17:02:19, est morte le 11/01/2004 17:02:31.
9   Monsieur 2, ne le 11/01/2004 17:02:19, est marie.
10  Monsieur 2, ne le 11/01/2004 17:02:19, est divorce.
11  Monsieur 3, ne le 11/01/2004 17:02:19, est mort le 11/01/2004 17:02:43.
```

2 Le modèle de conception *état*

Le modèle **Etat** permet à un objet de modifier son comportement, quand son état interne change. Tout se passera comme si l'objet changeait de classe.

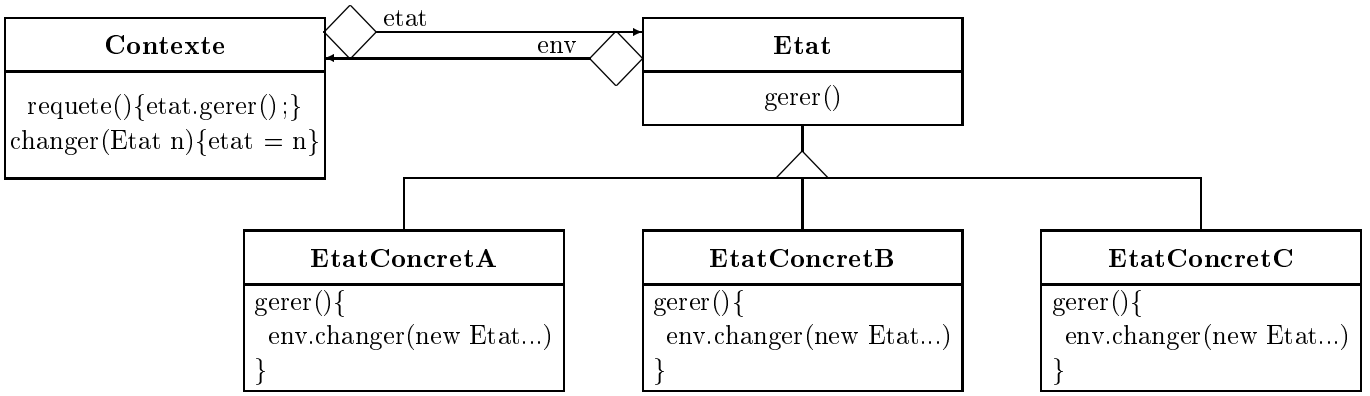

FIG. 2 – La structure du modèle État

L'objectif de l'exercice est que le même programme `test` génère une sortie équivalente.
Pour cela, le modèle **Etat** est implémenté non pas comme un interface, mais comme la classe `Situation` donnant les comportements par défauts.

```
1   import java.text.*;
2   public class Situation{
3       public static SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
4       Personne moi;
5
6       public Situation(Personne me){moi = me;}
7
8       public void mourir()       throws Exception{throw new Exception("Impossible");}
9       public void seMarier()     throws Exception{throw new Exception("Impossible");}
10      public void divorcer()     throws Exception{throw new Exception("Impossible");}
11      public void mortConjoint() throws Exception{throw new Exception("Impossible");}
12  }
```

Question 2.1 (4 points) *Complétez (lignes 12-16, 33-36 et 45) le code de la classe* `Personne`.

Réponse à la question 2.1:

```
1  import java.util.*;
2  public class Personne{
3      private static int numero = 0;
4      private static Random alea = new Random();
5
6      private int ident;
7      private Date naissance, deces;
8      private boolean masculin;
9      private Situation situation; // remplace les booleens.
10
11     public Personne(boolean garcon){
12         ident = numero++;
13         naissance = new Date();
14         deces = null;
15         masculin = garcon;
16         situation = new Celibataire(this);
17     }
18
19     public int     getIdent()    {return ident;}
20     public boolean isMasculin()  {return masculin;}
21     public Date    getNaissance(){return naissance;}
22     public Date    getDeces()    {return deces;}
23     public String  toString()    {return situation.toString();}
24     public void    setDeces()    {deces = new Date();}
25
26     public void vieillir() {
27         try {Thread.sleep(2000);}
28         catch(InterruptedException ie){}
29         finally {
30             int a = alea.nextInt(4);
31             try {
32                 switch (a) {
33                 case 0 : situation.mourir(); break;
34                 case 1 : situation.seMarier(); break;
35                 case 2 : situation.divorcer(); break;
36                 case 3 : situation.mortConjoint();
37                 }
38                 System.out.print(this);
39             }
40             catch (Exception e){} // Exceptions lors de chgt de situation
41         }
42     }
43
44     public void changerSituation(Situation sit){
45         situation = sit;
46     }
47
48 }
```

Question 2.2 (3 points) *Complétez (lignes 5, 21 et 25) le code de la classe* `Divorce`.

Réponse à la question 2.2:

```
1  import java.util.*;
2  import java.text.*;
3  public class Divorce extends Situation{
4      public Divorce(Personne me){
5          super(me);
6      }
7      public String toString(){
8          String res ="";
9          if (moi.isMasculin()){
10             res +="Monsieur "+moi.getIdent()+", ne le ";
11             res += sdf.format(moi.getNaissance())+", est divorce.\n";
12         }
13         else{
14             res +="Madame "+moi.getIdent()+", nee le ";
15             res += sdf.format(moi.getNaissance())+", est divorcee.\n";
16         }
17         return res;
18     }
19
20     public void mourir() throws Exception{
21         moi.changerSituation(new Mort(moi));
22     }
23
24     public void marier() throws Exception{
25         moi.changerSituation(new Marie(moi));
26     }
27
28 }
```

3 Le modèle de conception *état* avec historique

On veut en plus pouvoir afficher les situations des différentes personnes à une date donnée.
La classe `Test` est légèrement modifiée à cette fin (lignes 7 et 18-24).

```
1  import java.text.*;
2  import java.util.*;
3  public class Test{
4      public static void main(String[] args){
5          Personne[] personne = new Personne[4];
6          Random alea = new Random();
7          Date debut = new Date();
8          System.out.println("\tNaissances");
9          for (int i=0; i<personne.length; i++){
10             personne[i] = new Personne(alea.nextBoolean()); //Homme ou Femme
11             System.out.print(personne[i]);
12         }
13         System.out.println("\tQuelques etapes de la vie");
14         for (int time=0; time<3; time++){
15             for (int i=0; i<personne.length; i++)
16                 personne[i].vieillir();
17         }
18         Date fin = new Date();
19         Date milieu = new Date((debut.getTime()+fin.getTime())/2);
```

```
20          SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
21          System.out.println("\tSituation a la date "+sdf.format(milieu));
22          for (int i=0; i<personne.length; i++){
23              System.out.print(personne[i].getSituation(milieu));
24          }
25      }
26  }
```

et un exemple de sortie générée.

```
1           Naissances
2   Mademoiselle 0, nee le 11/01/2004 19:39:49.
3   Mademoiselle 1, nee le 11/01/2004 19:39:49.
4   Monsieur 2, ne le 11/01/2004 19:39:49, est celibataire.
5   Mademoiselle 3, nee le 11/01/2004 19:39:49.
6           Quelques etapes de la vie
7   Madame 0, nee le 11/01/2004 19:39:49, est morte le 11/01/2004 19:39:51.
8   Madame 1, nee le 11/01/2004 19:39:49, est mariee depuis le 11/01/2004 19:39:53.
9   Monsieur 2, ne le 11/01/2004 19:39:49, est mort le 11/01/2004 19:39:55.
10  Madame 1, nee le 11/01/2004 19:39:49, est divorcee depuis le 11/01/2004 19:40:01.
11          Situation a la date 11/01/2004 19:40:01
12  Madame 0, nee le 11/01/2004 19:39:49, est morte le 11/01/2004 19:39:51.
13  Madame 1, nee le 11/01/2004 19:39:49, est mariee depuis le 11/01/2004 19:39:53.
14  Monsieur 2, ne le 11/01/2004 19:39:49, est mort le 11/01/2004 19:39:55.
15  Mademoiselle 3, nee le 11/01/2004 19:39:49.
```

Il faut gérer pour chaque personne un historique des ses situations. Une solution consiste à associer à chaque nouvelle situation une date.

Question 3.1 (3 points) *Complétez (lignes 9-10, 14 et 18) le code de la classe* `Situation`.

Réponse à la question 3.1:

```
1   import java.text.*;
2   import java.util.*;
3   public class Situation{
4       public static SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
5       Date debut;  // pour memoriser le debut de cette situation
6       Personne moi;
7
8       public Situation(Personne me){
9           moi = me;
10          debut = new Date();
11      }
12
13      public Date getDebut(){
14          return debut;
15      }
16
17      public Date getDeces(){
18          return null;
19      }
20
21      public void mourir()       throws Exception{throw new Exception("Impossible");}
22      public void seMarier()     throws Exception{throw new Exception("Impossible");}
23      public void divorcer()     throws Exception{throw new Exception("Impossible");}
24      public void mortConjoint() throws Exception{throw new Exception("Impossible");}
25  }
```

Question 3.2 (4 points) *Complétez (lignes 12-15, 31-34, 43-44 et 48-54) le code de la classe* `Personne`.

Réponse à la question 3.2:

```
1  import java.util.*;
2  public class Personne{
3      private static int numero = 0;
4      private static Random alea = new Random();
5
6      private int ident;
7      private boolean masculin;
8      private List vie = new ArrayList(); // pour gerer l'historique
9      private Situation situation;
10
11     public Personne(boolean garcon){
12         ident = numero++;
13         masculin = garcon;
14         situation = new Celibataire(this);
15         vie.add(situation);
16     }
17
18     public int     getIdent()    {return ident;}
19     public boolean isMasculin()  {return masculin;}
20     public Date    getNaissance(){return ((Situation)vie.get(0)).getDebut();}
21     public Date    getDeces()    {return situation.getDeces();}
22     public String  toString()    {return situation.toString();}
23
24     public void vieillir() {
25         try {Thread.sleep(2000);}
26         catch(InterruptedException ie){}
27         finally {
28             int a = alea.nextInt(4);
29             try {
30                 switch (a) {
31                 case 0 : situation.mourir(); break;
32                 case 1 : situation.seMarier(); break;
33                 case 2 : situation.divorcer(); break;
34                 case 3 : situation.mortConjoint();
35                 }
36                 System.out.print(this);
37             }
38             catch (Exception e){} // Exceptions lors de chgt de situation
39         }
40     }
41
42     public void changerSituation(Situation sit){
43         situation = sit;
44         vie.add(situation);
45     }
46
47     public Situation getSituation(Date evt){
48         int i=0;
49         while (i<vie.size() && ((Situation)vie.get(i)).getDebut().before(evt))
50             i++;
51         if (i>0)
52             return (Situation)vie.get(i-1);
53         else
```

```
54              return null;
55      }
56  }
```

Question 3.3 (1 point) *Complétez (ligne 10) le code de la classe* `Mort`.

Réponse à la question 3.3:

```
1   import java.util.*;
2   import java.text.*;
3   public class Mort extends Situation{
4
5       public Mort(Personne me){
6           super(me);
7       }
8
9       public Date getDeces(){
10          return debut;
11      }
12
13      public String toString(){
14          String res ="";
15          if (moi.isMasculin()){
16              res +="Monsieur "+moi.getIdent()+", ne le ";
17              res += sdf.format(moi.getNaissance())+", est mort le ";
18          }
19          else{
20              res +="Madame "+moi.getIdent()+", nee le ";
21              res += sdf.format(moi.getNaissance())+", est morte le ";
22          }
23          res += sdf.format(debut)+".\n";
24          return res;
25      }
26
27  }
```

Question 3.4 (1 point) *Expliquez comment vous pourriez réaliser une méthode* `undo` *à partir de la solution précédente.*

Réponse à la question 3.4:
Il est assez facile d'ajouter une méthode `undo()` dans la classe `Personne`. Cette méthode remet la personne dans sa situation antérieure, et détruit sa dernière situation (on ne demande pas d'instruction `redo()`).

```
57      public void undo(){
58          if (vie.size()>1)
59              vie.remove(vie.size()-1);
60          situation = (Situation)vie.get(vie.size()-1);
61      }
62
63      public Date getDateLastEvent(){
64          return situation.getDebut();
65      }
66
67  }
```

Une méthode `undo()` agissant sur un groupe de personnes est plus délicat, car il faut comparer les dates des dernières actions de chacun pour savoir l'action qu'il faut *défaire*. La classe `Test` doit faire ce travail en utilisant la méthode `getDateLastEvent()` de la classe `Personne`.

```
27      private static void undo(Personne[] personne){
28          SortedMap dates = new TreeMap();
29          for (int i=0; i<personne.length; i++)
30              dates.put(personne[i].getDateLastEvent(),new Integer(i));
31          int indice = ((Integer)dates.get(dates.lastKey())).intValue();
32          System.out.print("Avant undo "+personne[indice]);
33          personne[indice].undo();
34          System.out.print("Apres undo "+personne[indice]);
35      }
36  }
```

et voici un exemple de sortie générée.

```
1           Naissances
2   Mademoiselle 0, nee le 13/01/2004 12:13:32.
3   Monsieur 1, ne le 13/01/2004 12:13:32, est celibataire.
4   Monsieur 2, ne le 13/01/2004 12:13:32, est celibataire.
5   Monsieur 3, ne le 13/01/2004 12:13:32, est celibataire.
6           Quelques etapes de la vie
7   Avant undo Monsieur 3, ne le 13/01/2004 12:13:32, est celibataire.
8   Apres undo Monsieur 3, ne le 13/01/2004 12:13:32, est celibataire.
9   Madame 0, nee le 13/01/2004 12:13:32, est mariee depuis le 13/01/2004 12:13:34.
10  Monsieur 1, ne le 13/01/2004 12:13:32, est mort le 13/01/2004 12:13:36.
11  Avant undo Monsieur 1, ne le 13/01/2004 12:13:32, est mort le 13/01/2004 12:13:36.
12  Apres undo Monsieur 1, ne le 13/01/2004 12:13:32, est celibataire.
13  Madame 0, nee le 13/01/2004 12:13:32, est veuve depuis le 13/01/2004 12:13:42.
14  Monsieur 1, ne le 13/01/2004 12:13:32, est marie depuis le 13/01/2004 12:13:44.
15  Monsieur 2, ne le 13/01/2004 12:13:32, est marie depuis le 13/01/2004 12:13:46.
16  Monsieur 3, ne le 13/01/2004 12:13:32, est mort le 13/01/2004 12:13:48.
17  Avant undo Monsieur 3, ne le 13/01/2004 12:13:32, est mort le 13/01/2004 12:13:48.
18  Apres undo Monsieur 3, ne le 13/01/2004 12:13:32, est celibataire.
19  Monsieur 1, ne le 13/01/2004 12:13:32, est divorce depuis le 13/01/2004 12:13:52.
20  Monsieur 2, ne le 13/01/2004 12:13:32, est mort le 13/01/2004 12:13:54.
21          Situation a la date 13/01/2004 12:13:44
22  Madame 0, nee le 13/01/2004 12:13:32, est veuve depuis le 13/01/2004 12:13:42.
23  Monsieur 1, ne le 13/01/2004 12:13:32, est celibataire.
24  Monsieur 2, ne le 13/01/2004 12:13:32, est celibataire.
25  Monsieur 3, ne le 13/01/2004 12:13:32, est celibataire.
```

4 Le modèle de conception *état*, historique et Thread

Le défaut majeur de la solution précédente est la gestion du vieillissement par la classe `Test`. On souhaite maintenant que chaque personne soit assez indépendante dans son rythme de vie.
Une solution consiste à considérer les objets `Personne` comme des processus indépendants, c'est à dire des `Thread` java, toutes les autres classes étant inchangées. L'avantage immédiat est la possibilité pour la classe `Test` d'attendre la mort des personnes au lieu de limiter à priori le nombre d'étapes dans la vie de chacun.
La nouvelle classe `Test` (lignes 13-19 modifiées).

```
1  import java.text.*;
2  import java.util.*;
3  public class Test{
4      public static void main(String[] args){
5          Personne[] personne = new Personne[4];
6          Random alea = new Random();
7          Date debut = new Date();
8          System.out.println("\tNaissances");
9          for (int i=0; i<personne.length; i++){
10             personne[i] = new Personne(alea.nextBoolean()); //Homme ou Femme
11             System.out.print(personne[i]);
12         }
13         System.out.println("\tToutes les etapes de la vie");
14         for (int i=0; i<personne.length; i++)
15             personne[i].start();                //demarrer les Thread
16         for (int i=0; i<personne.length; i++){
17             try {personne[i].join();}           //attendre la fin des Thread
18             catch(InterruptedException ie){}
19         }
20         Date fin = new Date();
21         Date milieu = new Date((debut.getTime()+fin.getTime())/2);
22         SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
23         System.out.println("\tSituation a la date "+sdf.format(milieu));
24         for (int i=0; i<personne.length; i++){
25             System.out.print(personne[i].getSituation(milieu));
26         }
27     }
28 }
```

et un exemple de sortie générée.

```
1          Naissances
2  Monsieur 0, ne le 11/01/2004 21:41:44, est celibataire.
3  Mademoiselle 1, nee le 11/01/2004 21:41:44.
4  Monsieur 2, ne le 11/01/2004 21:41:44, est celibataire.
5  Monsieur 3, ne le 11/01/2004 21:41:44, est celibataire.
6          Toutes les etapes de la vie
7  Madame 1, nee le 11/01/2004 21:41:44, est morte le 11/01/2004 21:41:46.
8  Monsieur 2, ne le 11/01/2004 21:41:44, est marie depuis le 11/01/2004 21:41:46.
9  Monsieur 3, ne le 11/01/2004 21:41:44, est mort le 11/01/2004 21:41:46.
10 Monsieur 2, ne le 11/01/2004 21:41:44, est veuf depuis le 11/01/2004 21:41:48.
11 Monsieur 0, ne le 11/01/2004 21:41:44, est marie depuis le 11/01/2004 21:41:50.
12 Monsieur 0, ne le 11/01/2004 21:41:44, est divorce depuis le 11/01/2004 21:41:52.
13 Monsieur 2, ne le 11/01/2004 21:41:44, est mort le 11/01/2004 21:41:54.
14 Monsieur 0, ne le 11/01/2004 21:41:44, est mort le 11/01/2004 21:41:56.
15         Situation a la date 11/01/2004 21:41:50
16 Monsieur 0, ne le 11/01/2004 21:41:44, est celibataire.
17 Madame 1, nee le 11/01/2004 21:41:44, est morte le 11/01/2004 21:41:46.
18 Monsieur 2, ne le 11/01/2004 21:41:44, est veuf depuis le 11/01/2004 21:41:48.
19 Monsieur 3, ne le 11/01/2004 21:41:44, est mort le 11/01/2004 21:41:46.
```

Question 4.1 (4 points) *Complétez (lignes 12-15, 25, 32-35, 41, 45-46 et 50-56) le code de la classe* `Personne`.

Réponse à la question 4.1:

```
1  import java.util.*;
2  public class Personne extends Thread{
3      private static int numero = 0;
4      private static Random alea = new Random();
5
6      private int ident;
7      private boolean masculin;
8      private List vie = new ArrayList(); // pour gerer l'historique
9      private Situation situation;
10
11     public Personne(boolean garcon){
12         ident = numero++;
13         masculin = garcon;
14         situation = new Celibataire(this);
15         vie.add(situation);
16     }
17
18     public int     getIdent()    {return ident;}
19     public boolean isMasculin()  {return masculin;}
20     public Date    getNaissance(){return ((Situation)vie.get(0)).getDebut();}
21     public Date    getDeces()    {return situation.getDeces();}
22     public String  toString()    {return situation.toString();}
23
24     public void run(){ // Remplace vieillir()
25         while (getDeces()==null) {
26             try {Thread.sleep(2000);}
27             catch(InterruptedException ie){}
28             finally {
29                 int a = alea.nextInt(4);
30                 try {
31                     switch (a) {
32                     case 0 : situation.mourir(); break;
33                     case 1 : situation.seMarier(); break;
34                     case 2 : situation.divorcer(); break;
35                     case 3 : situation.mortConjoint();
36                     }
37                     System.out.print(this);
38                 }
39                 catch(Exception e){}
40             }
41         }
42     }
43
44     public void changerSituation(Situation sit){
45         situation = sit;
46         vie.add(situation);
47     }
48
49     public Situation getSituation(Date evt){
50         int i=0;
51         while (i<vie.size() && ((Situation)vie.get(i)).getDebut().before(evt))
52             i++;
53         if (i>0)
```

```
54              return (Situation)vie.get(i-1);
55          else
56              return null;
57      }
58  }
```

Question 4.2 (1 point) *Expliquez pourquoi, cette solution ne fonctionne pas si on l'implémente directement avec la méthode* `vieillir()`.

Réponse à la question 4.2:
Si la classe `Personne` n'hérite plus de la classe `Thread`, et qu'elle n'implémente pas l'interface `Runnable`, ses objets ne sont plus des *processus* indépendants. L'application ne contient plus qu'un seul *flot d'instructions* ordonnancé par la méthode `main()` de la classe `Test`.
Toute exécution fera donc d'abord vieillir `personne[0]`, puis `personne[1]`,...
Voici par exemple une sortie possible :

```
1           Naissances
2   Mademoiselle 0, nee le 12/01/2004 17:17:18.
3   Mademoiselle 1, nee le 12/01/2004 17:17:18.
4   Mademoiselle 2, nee le 12/01/2004 17:17:18.
5   Mademoiselle 3, nee le 12/01/2004 17:17:18.
6           Toutes les etapes de la vie
7   Madame 0, nee le 12/01/2004 17:17:18, est mariee depuis le 12/01/2004 17:17:20.
8   Madame 0, nee le 12/01/2004 17:17:18, est veuve depuis le 12/01/2004 17:17:24.
9   Madame 0, nee le 12/01/2004 17:17:18, est morte le 12/01/2004 17:17:50.
10  Madame 1, nee le 12/01/2004 17:17:18, est mariee depuis le 12/01/2004 17:17:56.
11  Madame 1, nee le 12/01/2004 17:17:18, est morte le 12/01/2004 17:17:58.
12  Madame 2, nee le 12/01/2004 17:17:18, est morte le 12/01/2004 17:18:00.
13  Madame 3, nee le 12/01/2004 17:17:18, est mariee depuis le 12/01/2004 17:18:02.
14  Madame 3, nee le 12/01/2004 17:17:18, est divorcee depuis le 12/01/2004 17:18:04.
15  Madame 3, nee le 12/01/2004 17:17:18, est morte le 12/01/2004 17:18:14.
16          Situation a la date 12/01/2004 17:17:46
17  Madame 0, nee le 12/01/2004 17:17:18, est veuve depuis le 12/01/2004 17:17:24.
18  Mademoiselle 1, nee le 12/01/2004 17:17:18.
19  Mademoiselle 2, nee le 12/01/2004 17:17:18.
20  Mademoiselle 3, nee le 12/01/2004 17:17:18.
```