TE Génie Logiciel **Nom :**

Date : 21 Avril 2011

1	2	3	4	5	6	7	8	9	10	11
2.5	1	1	1.5	3	1	0,5	4	4	2	3.5
								4	5	14

30/39 — 4,8

EXERCICE 1 – 3.5 POINTS

6 — 8,5 — 15,5

- 1 POINT PAR RÉPONSE FAUSSE

La crise du logiciel est le résultat de l'avènement des ordinateurs de troisième génération, car ces nouvelles machines rendaient possible des applications jusqu'alors irréalisables. La crise fut révélée lors de la conférence de Garmish (conférence OTAN) en 1968. Quels étaient les symptômes de cette crise. Cocher la (les) proposition(s) pertinente(s).

2.5/3.5

- [x] Souvent, les logiciels réalisés ne correspondaient pas aux besoins des utilisateurs.
- [ ] Les ressources en programmeurs étaient insuffisantes.
- [x] Les délais de livraison n'étaient pas respectés.
- [x] Les coûts de développement étaient rarement prévisibles.
- [ ] Le matériel était trop cher.
- [x] La maintenance des logiciels était complexe et coûteuse.
- [x] Les logiciels n'étaient pas portables et difficilement transportables. -1

EXERCICE 2 – 1 POINT

Les gens du Génie Logiciel sont parfois tordus. Ils opèrent une distinction assez nette entre le fait de *vérifier* le logiciel et le fait de le *valider. Qu'est-ce que cela signifie ?*

1/1

- vérifier : tester pour corriger les bugs restants ✓
- valider : tester pour être sûr que ça correspond aux attentes du client ✓

EXERCICE 3 – 1 POINT

Que signifie « prendre en compte la gestion des risques, dans le cadre de la méthode UP ?

1/1

Lorsqu'une fonctionnalité présente un gros risque, elle est testée pour vérifier qu'elle soit réalisable.
Si elle n'est pas réalisable → client est averti le plus vite possible

OK, mais un peu vague quand même (parler plutôt de priorité donnée aux éléments à risque)

EXERCICE 4 – 1.5 POINT

- 1POINT PAR RÉPONSE FAUSSE

1.5/1.5

Quelles sont les différences entre le Cycle de Vie en Spirale (CVS) et le Modèle Itératif Incrémental (MII)? Cocher les propositions correctes.

- [x] Les incréments du MII sont plus petits que ceux du CVS
- [ ] Les incréments donnent lieu dans les deux cas à un produit exécutable
- [x] Avec le CVS, le client n'a pas l'opportunité de tester le produit avant qu'il ne soit réalisé entièrement, contrairement au MII. ✓

6

EXERCICE 5 – 3 POINTS

- 1POINT PAR RÉPONSE FAUSSE

Parmi les différents critères de qualités énumérés ci-dessous, indiquez s'il s'agit d'une qualité « externe », « interne » ou alors ayant trait au processus de développement. Une seule réponse possible par critère de qualité.

3/3

	Externe	Interne	Processus
Maintenabilité	o	✓	o
Evolutivité	o	✓	o
Correction	✓	o	o
Robustesse	✓	o	o
Respect des délais	o	o	✓
Réutilisation	o	✓	o

EXERCICE 6 – 1 POINT

Jouons sur les mots. L'objectif même de la phase d'analyse est-il de faire la « chose correcte » ou de faire les « choses correctement » ?

1/1

De faire la chose correcte → quoi?
Faire les choses correctement relève déjà de la conception ✓

EXERCICE 7 – 1 POINT

Le modèle MVC vise avant tout assurer la réutilisation de l'une des 3 couches Modèle, Vue et Contrôleur. Laquelle ? et pourquoi ?

0,5/1

Oui, mais le justificatif devrait plutôt se baser sur le Le modèle car il représente l'accès aux données. Une vue différente, par exemple, doit pouvoir s'insérer auprès de ce modèle et que sa fonctionne

EXERCICE 8 - 4 POINTS

- 1POINT PAR RÉPONSE FAUSSE

Méthode UP. Indiquer par une croix dans quelle phase l'intensité de chaque discipline/activité est maximum (choix exclusif !)
Init : Initialisation – **Elab** : Elaboration – **Constr** : Construction – **Trans** : Transition

4/4

	Init	Elab	Constr	Trans
Réalisation de l'architecture centrale	o	✓	o	o
Etude de faisabilité	o	✓	o	o
Modélisation de domaine	✓	o	o	o
Rédaction des cas d'utilisation	o	✓	o	o
Livraison finale	o	o	o	✓
Réalisation d'un module	o	o	✓	o
Planification des itérations	o	✓	o	o
Génération d'un sous-ensemble exécutable	o	o	✓	o

✓

8,5

d'indépendance des couches par rapport à l'application

EXERCICE 9 – 4 POINTS

Soit le code source suivant :

```
public class MultiCompteur {
    private int registre = 0;
    private int iteration;
    private Object obj;
    public void effectue() {
        Runnable r1 = new Runnable() { synchronized(obj) {
            public void run() {
                for(int i=0; i<iteration; i++) {
                    int tmp = registre;
                    tmp = tmp + 1;
                    registre = tmp;
                }}
            }
        };
        Runnable r2 = new Runnable() { synchronized(obj) {
            public void run() {
                for(int i=0; i<iteration; i++) {
                    int tmp = registre;
                    tmp = tmp - 1;
                    registre = tmp;
                }}
            }
        };
        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);

        t1.start();
        t2.start();
        System.out.println("Nous avons au final: "+registre);
    }
    public MultiCompteur(int iteration) { this.iteration = iteration; }
    public static void main(String args[]) {
        MultiCompteur mc = new MultiCompteur(100000000);
        mc.effectue();
    }
}
```

Répondre aux deux questions suivantes :

- Le concepteur du programme aurait voulu qu'à la fin de l'exécution des deux méthodes `run`, le registre ait la valeur 0. Or ce n'est probablement pas le cas. Pourquoi ? Comment y remédier ? (proposer des modifications du code). Parce que les sections critiques peuvent être atteintes en même temps. Il faut faire une synchronisation pour qu'il n'y ait plus de préemption.
- A la fin de la méthode `effectue()`, nous avons un affichage avec l'appel à `System.out.println()`. Cette instruction produira de toute façon un résultat aberrant (indépendamment du point précédent). Pourquoi ?

4/4

Car l'ordonnanceur peut redonner la main au main avant la fin de l'exécution des threads.

Comment y remédier ? (proposer des modifications du code).

Il faut ajouter :

```
t1.join();
t2.join();
```

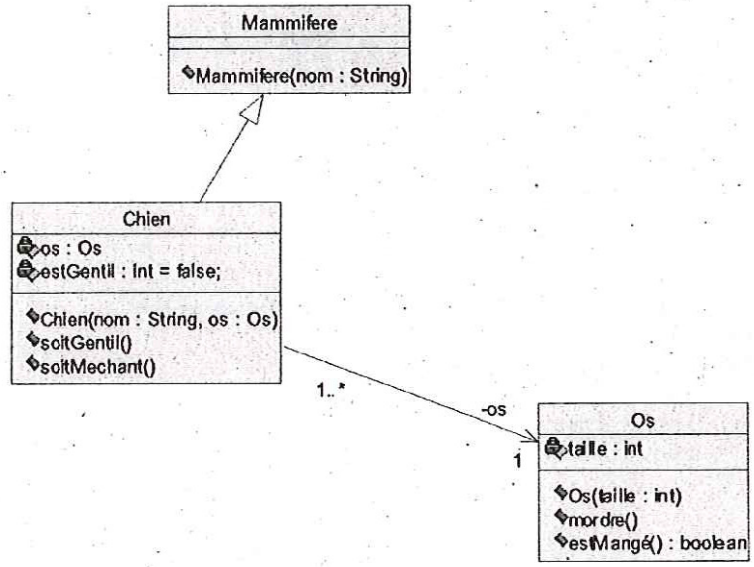
juste avant System.out.println(...);

EXERCICE 10 – 8 POINTS

Des chiens et un os

On aimerait modifier la Classe «Chien» de la hiérarchie «Animal».

Considérons le diagramme de classes représenté ci-après :




```
//-------------------------------------------------------------------------
class Mammifere {
  public Mammifere (String nom) {
    super(nom);
  }
}
//-------------------------------------------------------------------------
class Chien extends Mammifere {

  private boolean estGentil = false;
  private Os os;              // os associé
  public Chien (String nom, Os os) {
     super (nom);
     this.os = os;
  }

  public void soitGentil() { estGentil = true; }
  public void soitMechant () { estGentil = false; }
}
//-------------------------------------------------------------------------
class Os {
  private int taille;

  public Os (int taille) {this.taille = taille;}

  public void mordre() {
   // Une fois mordu, la taille de l'os diminue d'une unité
   // Si l'os est déjà mangé, une exception est levée
     if (taille <= 0) throw new RuntimeException("Os déjà mangé");
     taille--;
  }
  public boolean estMangé() {return taille == 0; }
}
```

Syntaxe: 2/3

Synchro: 0/2

2/5

implements Runnable

private Thread mange = new Thead();

mange.start();

this

-1

public void run() {
while(!os.estMangé())
if (!os.estMangé()) {
synchronized() {
while (!os.estMangé()) {
try { wait(); } catch (InterruptedException) {}
os.mordre();...

synchronized (os)

et notifyAll()

J'imagine bien qu'il y a une histoire de wait() pour que les deux chiens mangent l'os à tour de rôle, mais je n'arrive pas à l'implémenter de manière cohérente.

Non, pas compris!

//----------

On aimerait que toute instance de la classe «Chien» soit un **objet actif** : sa tâche, démarrée dès que le chien est créé, doit passer son temps, une fois par seconde, à mordre dans l'os qui lui a été associé à la naissance.

Ainsi, toutes les secondes, le message «os.mordre()» sera envoyé à l'os associé, et ceci **jusqu'à ce que l'os soit entièrement mangé.**

Voici, en gros, **l'idée** de l'algorithme :

```
while (! os.estMangé()) {
  os.mordre() ;
  try {Thread.sleep(1000) ;}
  catch (InterruptedException e) {}
}
```

☞ Modifiez en conséquence les classes «Chien» et «Os» (éventuellement).

Pour gagner du temps, vous pouvez « oublier » les méthodes «soitGentil()» et «soitMechant()», qui ne sont pas concernées par la modification.

Contraintes!

On aimerait qu'il soit possible de créer deux chiens, **associés au même os**, auquel ils s'attaqueront en même temps:

```
Os unOs = new Os(10) ;  // Os de taille 10
Chien ch1 = new Chien("Medor", unOs) ;
Chien ch2 = new Chien("toutou", unOs) ;
```

La levée d'exception, au cas où le chien mord dans un os déjà mangé, doit continuer à être levée. Toutefois, si votre solution est correcte, cette dernière ne doit jamais être levée (en effet, le test de la boucle while du chien devrait l'assurer).

COMMENTEZ VOTRE SOLUTION : notamment, vos décisions de mise en œuvre doivent être expliquées !!

EXERCICE 11 – 14 POINTS

Considérez les 4 classes suivantes :

```
class MaPile {
      private ArrayList liste = new ArrayList();

      public MaPile() {}

      public void empiler (int element) {
            if (liste.size() < 5) {
                  liste.add(element);
            }
      }
      public int depiler () {
            if (liste.size() != 0) {
                  int sommet =
                        ((Integer)(liste.get(liste.size()-1))).intValue();
                  liste.remove(liste.size()-1);
                  return sommet;
            }
            return 0;
      }
}

class Debordement {

      private ArrayList liste = new ArrayList();

      public Debordement() {}

      private void addElement(Object element) {
            liste.add (arg);
      }

}

class VueMessage extends JPanel {

      private JLabel label = new JLabel();
      private String texte;
      public VueMessage (String texte) {
            this.texte=texte;
            add (label);
      }
      private void setTexte() {label.setText(texte);}
      private void clearTexte() {label.setText("");}
}

class Controleur {
      public Controleur() {
            MaPile pile = new MaPile();
            Debordement debordement = new Debordement();
            VueMessage v1 = new VueMessage("Pile vide");
            VueMessage v2 = new VueMessage("Pile pleine");
            VueMessage v3 = new VueMessage("Debordement");
      }
}
```

Que faire ??

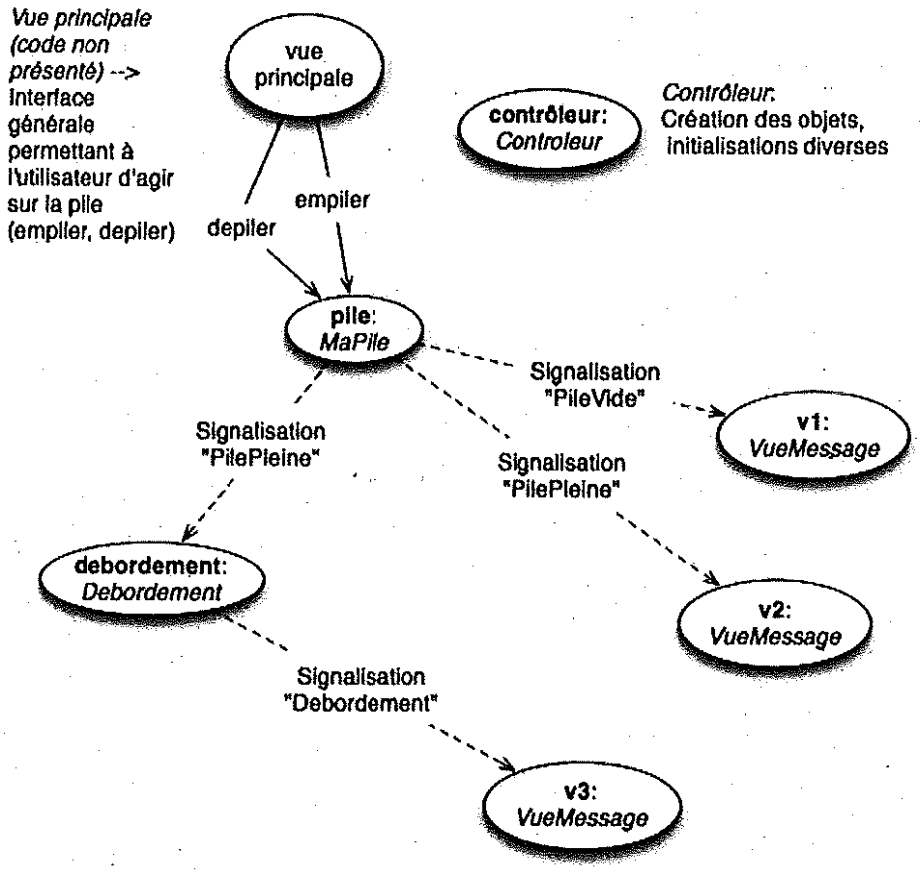
Vous allez **compléter** et/ou **adaptez** chacune de ces 4 classes, l'objectif étant d'obéir à la structure présentée ci-dessous en exploitant au mieux le design pattern « Observable-Observé » pour la signalisation des événements. Pour des raisons liées à l'efficacité, vous devez chercher à éviter qu'un objet reçoive un signal « `update` » pour un événement qui ne le concerne pas à priori.

Compléments d'informations :

La vue générale, dont le code n'est pas représenté ici, permet à l'utilisateur d'agir sur la pile : dépiler ou empiler un élément.

Si l'utilisateur demande à la pile de se dépiler alors que cette dernière est vide, la pile réagit simplement en signalant cet événement (Pile vide) à ses observateurs. A la réception de ce signal, la vue `v1` réagit en affichant le message « `Pile Vide` » pendant 3 secondes.

Si l'utilisateur demande à la pile d'empiler un élément alors que cette dernière est pleine (contient déjà 5 éléments), la pile réagit simplement en signalant cet événement (Pile pleine). A la réception de ce signal, la vue `v2` réagit en affichant le message « `Pile Pleine` » pendant 3 secondes. L'objet « `debordement` » réagit quant à lui en stockant l'élément quil fallait empiler dans sa liste, et en signalant l'événement « Debordement». A la réception du signal de débordement, la vue `v3` réagit en affichant le message « `Debordement` » pendant 3 secondes.



Pile - 10
---
1 Observable
1 Observation des événements Pile Vide par vue1
1 Observée des événements Pile Pleine par vue2
1 Observée des événements Pile Pleine par débordement
1 Notifie "pile pleine"
1 Notifie "pile vide"
4 Distinction des deux types d'événements "pile pleine" & "pile vide"

Debordement - 6
---
1 Observable
1 Observateur (implements update)
1 Observe la pile
1 Observé par vue3
1 Récupère l'élément renvoyé par la pile
1 Notifie "Debordement"

VueMessage - 5
---
1 Observateur (implements update)
1 Affiche son texte quand notifiée
1 L'efface après 3 secondes
2 Dans un thread séparé pour libérer update

14

21

```
 1 /* ......... */
 2 class MultiCompteur {
 3     private int registre = 0;
 4     private int iteration;
 5     private Object jeton = new Object();
 6
 7     public void effectue() {
 8         Runnable r1 = new Runnable() {
 9             public void run() {
10                 for(int i=0; i<iteration; i++) {
11                   synchronized (jeton) {
12                   int tmp = registre;
13                   tmp = tmp + 1;
14                   registre = tmp;
15                     }
16                 }
17             }
18         };
19         Runnable r2 = new Runnable() {
20             public void run() {
21                 for(int i=0; i<iteration; i++) {
22                     synchronized (jeton) {
23                   int tmp = registre;
24                   tmp = tmp - 1;
25                   registre = tmp;
26                   }
27                 }
28             }
29         };
30         Thread t1 = new Thread(r1);
31         Thread t2 = new Thread(r2);
32
33         t1.start();
34         t2.start();
35         try {t1.join();} catch (InterruptedException e) {}
36         try {t2.join();} catch (InterruptedException e) {}
37         System.out.println("Nous avons au final: "+registre);
38     }
39     public MultiCompteur(int iteration) {
40       this.iteration = iteration;
41
42     }
43     public static void main(String args[]) {
44         MultiCompteur mc = new MultiCompteur(100000000);
45         mc.effectue();
46     }
47 }
48
```

```
 1 import java.awt.*;
 2 import java.awt.event.*;
 3 import javax.swing.*;
 4 import java.util.ArrayList;
 5 import java.util.*;
 6
 7 //-------------------------------------------------------------
 8 class Glb {
 9 // Constantes globales
10    public static int TAILLE_X = 400;
11    public static int TAILLE_Y = 400;
12
13    public static void pause (int msec) {
14       try {Thread.sleep(msec);}
15       catch (InterruptedException e) {}
16    }
17    public static void emettre(String texte) {
18       System.out.println (texte);
19    }
20 }
21 //-------------------------------------------------------------
22 class MaPile {
23
24    private ArrayList liste = new ArrayList();
25    private PilePleineEvent ppe = new PilePleineEvent();
26    private PileVideEvent pve =   new PileVideEvent();
27
28    class PilePleineEvent extends Observable {
29       public void notifier(int element) {
30          this.setChanged();
31          this.notifyObservers(element);
32       }
33    }
34
35    class PileVideEvent extends Observable {
36       public void notifier() {
37          this.setChanged();
38          this.notifyObservers();
39       }
40    }
41
42    public MaPile() {}
43
44    public void empiler (int element) {
45       if (liste.size() >= 5) {
46          ppe.notifier(element);
47
48       }
49       else {
50          liste.add(element);
51       }
52
53    }
54
55    public int depiler () {
56       if (liste.size() == 0) {
57          pve.notifier();
58       }
59       else {
60          int sommet = ((Integer)(liste.get(liste.size()-1))).intValue();
61          liste.remove(liste.size()-1);
62          return sommet;
63       }
64       return 0;
65    }
```

```
66
67    public void addPilePleineObserver(Observer o) {
68       ppe.addObserver(o);
69    }
70
71    public void addPileVideObserver(Observer o) {
72       pve.addObserver(o);
73    }
74
75 }
76
77 //-------------------------------------------------------
78 class Debordement extends Observable implements Observer {
79
80    private ArrayList liste = new ArrayList();
81
82    public Debordement() {}
83
84    public void update (Observable o, Object arg) {
85       liste.add (arg);
86       setChanged();
87       notifyObservers(arg);
88    }
89
90 }
91
92 //-------------------------------------------------------
93 class VueMessage extends JPanel implements Observer, Runnable {
94    private JLabel label = new JLabel("            ");
95    private String texte;
96    public VueMessage (String texte) {
97       this.texte=texte;
98       add (label);
99    }
100   public void update (Observable o, Object arg) {
101      Thread activite = new Thread (this);
102      activite.start();
103   }
104   public void run() {
105      label.setText(texte);
106      try {Thread.sleep(1000);}catch (Exception e) {}
107      label.setText("");
108   }
109 }
110
111
112 //-------------------------------------------------------
113 class VuePrincipale extends JPanel {
114
115    private MaPile pile;
116    private JTextField textField = new JTextField(10);
117
118
119    public VuePrincipale (MaPile p,
120                          VueMessage v1,
121                          VueMessage v2,
122                          VueMessage v3) {
123       this.pile = p;
124       JButton bDepiler = new JButton ("Depiler");
125       bDepiler.addActionListener(new ActionListener() {
126          public void actionPerformed(ActionEvent e) {
127             pile.depiler();
128          }
129       });
130       this.add(bDepiler);
```

```
131
132         add(textField);
133         JButton bEmpiler = new JButton ("Empiler");
134         bEmpiler.addActionListener(new ActionListener() {
135            public void actionPerformed(ActionEvent e) {
136               int valeur=0;
137               try {valeur=Integer.parseInt(textField.getText());}
138               catch (Exception x) {}
139
140               pile.empiler(valeur);
141            }
142         });
143         this.add(bEmpiler);
144         this.add(v1);
145         this.add(v2);
146         this.add(v3);
147      }
148
149      public void paintComponent(Graphics g) {
150         super.paintComponent(g);
151      }
152
153
154      public Dimension getPreferredSize() {
155         return new Dimension (Glb.TAILLE_X, Glb.TAILLE_Y);
156      }
157      public synchronized void f() {}
158 }
159 //-------------------------------------------------------
160 class Controleur {
161    public Controleur() {
162       MaPile pile = new MaPile();
163       VueMessage v1 = new VueMessage("Pile vide");
164       pile.addPileVideObserver (v1);
165       VueMessage v2 = new VueMessage("Pile pleine");
166       pile.addPilePleineObserver (v2);
167       Debordement debordement = new Debordement();
168       pile.addPilePleineObserver(debordement);
169       VueMessage v3 = new VueMessage("Debordement");
170       debordement.addObserver (v3);
171
172       JFrame f = new JFrame ("Titre fenetre");
173       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
174       f.getContentPane().add(new VuePrincipale(pile, v1, v2, v3),
175                              BorderLayout.CENTER);
176       f.pack();
177       f.setVisible(true);
178
179    }
180 }
181 //-------------------------------------------------------
182 public class Amorce  {
183    public static void main (String [] args) {
184       new Controleur();
185    }
186 }
```

```
 1
 2 class Chien implements Runnable {
 3
 4   private Os os;            // os associé
 5   private Thread activite;
 6   private String nom;
 7
 8   public Chien (String nom, Os os) {
 9     this.nom = nom;
10     this.os = os;
11     activite = new Thread(this);
12     activite.start();
13   }
14
15   public void run() {
16     boolean osTermine = false;
17     while (! osTermine) {
18       synchronized (os) {
19         if (!os.estMange()) {
20           os.mordre();
21           System.out.println (nom + "Gloups");
22         }
23         else {
24           osTermine = true;
25         }
26       }
27       if (!osTermine){
28         try { Thread.sleep(1000); }
29         catch (InterruptedException e) {}
30       }
31     }
32
33   }
34
35
36 }
37
38 class Os {
39   private int taille;
40   public Os (int taille) {this.taille = taille;}
41
42   public synchronized void mordre() {
43     if (taille <= 0)
44        throw new RuntimeException("Os deja mange");
45     taille--;
46   }
47   public boolean estMange() {return taille == 0; }
48 }
49
```