Développer d'autres classes
Corrigé

Exercice 1 : Compte bancaire simplifié
L'objectif de cet exercice est de modéliser en UML, puis en Java un compte bancaire simplifié. Un compte possède un solde qui peut être positif ou négatif. On peut créditer ou débiter un compte d'un certain montant, ce montant devant être strictement positif. Lors de l'ouverture d'un compte, un dépôt initial (strictement positif) doit être réalisé qui deviendra naturellement le solde initial du compte. Le solde, les montants et le dépôt initial sont exprimés en centimes et représentés par le type `int` de Java.

1.1. Dessiner le diagramme d'analyse UML de la classe `Compte`.

Solution :

CompteSimple
requêtes
solde : int
commandes
créditer(montant : int) débiter(montant : int)
constructeurs
CompteSimple(depotInitial : int)

1.2. Écrire en Java la classe `Compte`. On donnera les commentaires de documentation.

Solution :

```java
/** CompteSimple modélise un compte bancaire simple tenu en euros.  Il
 * est caractérisé un solde (positif ou négatif)
 * et autorise seulement les opérations de crédit et débit.
 * @author    Xavier Crégut
 * @version    1.6
 */
public class CompteSimple {

    /** Solde du compte exprimé en euros. */
    private int solde;

    /** Initialiser un compte.
    * @param depotInitial le montant initial du compte
    */
    //@ requires depotInitial >= 0;    // montant initial strictement positif
    //@ ensures getSolde() == depotInitial;    // solde initialisé
    public CompteSimple(int depotInitial) {
```

```
18          solde = depotInitial;
19      }
20
21      /** Solde du compte exprimé en euros. */
22      public /*@ pure @*/ int getSolde() {
23          return solde;
24      }
25
26      /** Créditer le compte du montant (exprimé en euros).
27      * @param montant montant déposé sur le compte en euros
28      */
29      //@ requires montant > 0;
30      //@ ensures getSolde() == \old(getSolde()) + montant; // montant crédité
31      public void crediter(int montant) {
32          solde = solde + montant;
33      }
34
35      /** Débiter le compte du montant (exprimé en euros).
36      * @param montant montant retiré du compte en euros
37      */
38      //@ requires montant > 0;
39      //@ ensures getSolde() == \old(getSolde()) - montant; // montant débité
40      public void debiter(int montant){
41          solde = solde - montant;
42      }
43
44      public String toString() {
45          return super.toString() + "+solde=" + getSolde();
46      }
47
48  }
```

1.3. Écrire un programme de test qui réalise les tests suivants sur un compte ouvert avec un dépôt initial de 100 euros :

1. créditer de 250 euros et vérifier que le solde est 350 euros,
2. débiter le compte de 150 euros et vérifier que le solde est -50 euros.

Solution :

```
1   import org.junit.*;
2   import static org.junit.Assert.*;
3
4   /** Tests unitaires JUnit pour la classe CompteSimple.
5     * @author    Xavier Crégut
6     * @version    1.3
7     */
8   public class CompteSimpleTest {
9
10      public static final double EPSILON = 1e-6;
11          // précision pour la comparaison entre réels.
12
13      protected CompteSimple c1;
14
```

```
15      @Before
16      public void setUp() {
17          this.c1 = new CompteSimple(100);
18      }
19
20      @Test
21      public void testerInitialisationC1() {
22          assertEquals(10000, this.c1.getSolde(), EPSILON);
23      }
24
25      @Test
26      public void testerCrediter() {
27          this.c1.crediter(25000);
28          assertEquals(25000, this.c1.getSolde(), EPSILON);
29      }
30
31      @Test
32      public void testerDebiter() {
33          this.c1.debiter(15000);
34          assertEquals(-5000, this.c1.getSolde(), EPSILON);
35      }
36
37      public static void main(String[] args) {
38          org.junit.runner.JUnitCore.main(CompteSimpleTest.class.getName());
39      }
40
41  }
```

Exercice 2 : Quelques opérations sur les dates

Nous souhaitons pouvoir disposer d'un type date permettant d'obtenir le jour, le mois et l'année d'une date. Nous souhaitons pouvoir afficher une date suivant le format jj/mm/année, le jour et le mois étant toujours affichés sur deux positions (par exemple 05/10/2010). On veut pouvoir savoir si une date est antérieure (strictement) à une autre ou (strictement) postérieure à une autre. On veut aussi pouvoir incrémenter une date (c'est-à-dire passer au lendemain) ou ajouter une durée (exprimée en nombre de jours) à une date.

2.1. Dessiner le diagramme d'analyse UML qui spécifie une date.

Solution :

Date
requêtes
jour : int mois : int année : int lt(autre : Date) : boolean gt(autre : Date) : boolean toString : String
commandes
afficher incrémenter ajouter(durée : int)
constructeurs
Date(jour : int, mois : int, année : int)

2.2. Choisir les attributs et dessiner le diagramme de classe correspondant.
Solution : On choisit ici jour, mois et année comme attributs. Est-ce le seul choix ? Non !

Date
− jour : int − mois : int − année : int
+ jour : int + mois : int + année : int + lt(autre : Date) : boolean + gt(autre : Date) : boolean + toString : String + afficher + incrémenter + ajouter(durée : int)
+ Date(jour : int, mois : int, année : int)

2.3. Définir un ou plusieurs constructeurs sur la classe Date.
2.4. Définir les opérations pour comparer deux dates (antérieure et postérieure).
2.5. Définir l'opération pour afficher une date.
2.6. Définir l'opération qui incrémente une date.
2.7. Ajouter une durée (exprimée en nombre de jours) à une date.
Solution :

```
1  /**
2   * Définition d'une date dans le calendrier grégorien.
3   *
4   * @author    Xavier Crégut
5   * @version    1.5
```

```java
 */
public class Date {

    /** le numéro du jour dans le mois */
    private int jour;

    /** le numéro du mois dans l'année */
    private int mois;

    /** l'année */
    private int annee;

    /** Construire une date à partir du numéro du jour, du numéro du mois et de
        * l'année (qui doivent former une date valide).
        * @param j le numéro du jour dans le mois m
        * @param m le numéro du mois
        * @param a l'année
        */
    public Date(int j, int m, int a) {
        this.set(j, m, a);
    }

    /** Changer la date à partir du numéro du jour, du numéro du mois et de
     * l'année.
     * @param j le numéro du jour dans le mois m
     * @param m le numéro du mois
     * @param a l'année
     */
    private void set(int j, int m, int a) {
        this.jour = j;
        this.mois = m;
        this.annee = a;
    }

    /** Obtenir le numéro du jour dans le mois.
     * @return le numéro du jour dans le mois
     */
    public int getJour() {
        return this.jour;
    }

    /** Obtenir le numéro du mois dans l'année.
     * @return le numéro du mois dans l'année
     */
    public int getMois() {
        return this.mois;
    }

    /** Obtenir l'année.
     * @return l'année
     */
    public int getAnnee() {
        return this.annee;
```

```
59      }
60
61      /** Afficher la date sous la forme jour/mois/année.  */
62      public void afficher() {
63          System.out.println(this);
64      }
65
66      // Méthode définie dans objet.  Je n'utilise pas un commentaire javadoc car
67      // le commentaire défini dans la classe parente est le commentaire à
68      // utiliser.
69      public String toString() {
70          return "" + int2String(this.jour)
71                          + '/' + int2String(this.mois)
72                          + '/' + this.annee;
73                  // le "" intial est là pour forcer la conversion en String,
74                  // les caractères sont en effet compatibles avec les entiers !
75      }
76
77      /** Obtenir la représentation d'un entier sous la forme d'une chaîne de
78       * caractères avec au moins 2 caractères.
79       * @param entier l'entier à convertir
80       * @return  la chaîne de caractères correspondant à entier
81       */
82      private static String int2String(int entier) {
83          String prefixe = (entier >= 0 && entier < 10) ? "0" : "";
84          return  prefixe + entier;
85      }
86
87      /** Incrémenter cette date. */
88      public void incrementer() {
89          if (this.jour < nbJoursDansMois(this.mois, this.annee)) {
90              // changer de jour
91              this.jour++;
92          } else {
93              // changer de mois
94              this.jour = 1;
95              if (this.mois < 12) {
96                  // passer au mois suivant
97                  this.mois++;
98              } else {
99                  // changer d'année
100                 this.mois = 1;
101                 this.annee++;
102             }
103         }
104     }
105
106     /** L'année a est-elle bissextile ?
107      * @param a l'année
108      * @return vrai si l'année est bissextile
109      */
110     public static boolean estBissextile(int a) {
111         return (a % 4 == 0)           // divisible par 4
```

```
112                 && ((a % 100 != 0)        // et non divible par 100
113                     || (a % 400 == 0));    // sauf si divisible par 400
114     }
115
116     /** Nombre de jours dans le mois m de l'année a.
117      * @param m le numéro du mois
118      * @param a l'année
119      * @return le nombre de jours dans le mois m de l'année a
120      */
121     public static int nbJoursDansMois(int m, int a) {
122         int resultat = 0;
123         switch (m) {
124             case  1:    // mois à 31 jours
125             case  3:
126             case  5:
127             case  7:
128             case  8:
129             case 10:
130             case 12:
131                 resultat = 31;
132                 break;
133
134             case  4:    // mois à 30 jours
135             case  6:
136             case  9:
137             case 11:
138                 resultat = 30;
139                 break;
140
141             default:    // février
142                 assert m == 2;
143                 resultat = estBissextile(a) ? 29 : 28;
144         }
145         return resultat;
146     }
147
148
149 //    Relation d'ordre sur les dates
150 //    ------------------------------
151
152     /** Déterminer si une date est antérieure strictement à une autre date.
153      * @param autre l'autre date (non nulle)
154      * @return cette date est-elle strictement antérieure à l'autre date ?
155      */
156     public boolean lt(Date autre) {
157         if (this.getAnnee() != autre.getAnnee()) {
158             return this.getAnnee() < autre.getAnnee();
159         } else if (this.getMois() != autre.getMois()) {
160             return  this.getMois() < autre.getMois();
161         } else {
162             return  this.getJour() < autre.getJour();
163         }
164     }
```

```java

    /** Déterminer si une date est postérieure ou égale à une autre date.
     * @param autre l'autre date (non nulle)
     * @return cette date est-elle postérieure ou égale à l'autre date
     */
    public boolean ge(Date autre) {
        return ! this.lt(autre);
    }

    /** Déterminer si une date est strictement postérieure à une autre date.
     * @param autre l'autre date (non nulle)
     * @return cette date est-elle strictement postérieure à l'autre date
     */
    public boolean gt(Date autre) {
        return autre.lt(this);
                // Utilisation obligatoire de this (on ne peut pas s'en passer).
    }

    /** Déterminer si une date est antérieure ou égale à une autre date.
     * @param autre l'autre date (non nulle)
     * @return cette date est-elle antérieure ou égale à l'autre date
     */
    public boolean le(Date autre) {
        return  ! autre.lt(this);
    }

    /** Déterminer si deux dates sont égales.
     * @param autre l'autre date
     * @return cette date est égale à l'autre date
     */
    public boolean egale(Date autre) {
        return autre != null
                && this.getAnnee() == autre.getAnnee()
                && this.getMois() == autre.getMois()
                && this.getJour() == autre.getJour();
    }

}
```