

14 janvier 2015

**Contrôle d'informatique no 2**

Durée : 1 heure 45'

Nom : .....

Groupe : 

Prénom : .....

No	1.1	1.2	2
Total points	65 points	30 points	65 points

*Remarque générale : toutes les questions qui suivent se réfèrent au langage de programmation Java (à partir de Java SE 7) et les réponses doivent être rédigées à l'encre et d'une manière propre sur ces feuilles agrafées.*

**Sujet no 1.**

Le but de cet exercice est de réaliser une application autonome interactive qui aide l'utilisateur à traiter des problèmes de mécanique élémentaire faisant intervenir des ressorts. Cette application doit correspondre à un projet doté d'un package appelé **cms\_ctr2** et qui contient deux classes **publiques**, définies dans deux fichiers distincts et appelées **Ressort** et, respectivement, **CP\_Ctr2Exo1**. Par la suite, aux points **1.1** et **1.2**, on vous demande d'écrire le code complet de ces deux classes, en respectant les consignes précisées. Vous pouvez éventuellement répondre au point **1.2** en supposant le point **1.1** résolu correctement.

**1.1** Une instance de la classe **Ressort** est un objet de type **Ressort** qui regroupe (ou encapsule) des informations concernant un ressort, à savoir la constante d'élasticité **k** et la déformation **d**. De plus, la classe **Ressort** est munie des méthodes permettant de manipuler les objets de type **Ressort**.

Écrire le code complet de la classe **public** **Ressort** qui appartient au package **cms\_ctr2** et qui définit :

- a) un champ (d'instance) nommé **k** de type numérique réel, déclaré **privé** et sans valeur initiale explicite et qui sert à stocker la valeur de la constante d'élasticité **k** ;
- b) un champ (d'instance) nommé **d** de type numérique réel, déclaré **sans modificateur d'accès**, avec la valeur initiale explicite **1** et qui sert à stocker la valeur de la déformation **d** ;
- c) une méthode publique (d'instance) "**setter**" nommée en respectant la convention Java pour le nommage des setters et qui permet la modification de la valeur du champ privé **k** ; cette méthode doit respecter les consignes suivantes :
  - i. si la valeur de son argument noté lui aussi **k** est positive ou nulle, on copie la valeur de cet argument dans le champ **k** ;
  - ii. autrement, on stocke dans le champ **k** la valeur **10** ;
- d) une méthode publique (d'instance) "**getter**" nommée en respectant la convention Java pour le nommage des getters et qui retourne (sans aucune vérification) la valeur du champ privé **k** ;
- e) un constructeur **public** (surchargé) avec deux arguments de type numérique réel : le premier argument nommé **k** (comme le champ **k**) et le deuxième argument nommé **d** (comme le champ **d**) ; ce constructeur permet de créer un objet correspondant à un nouveau ressort (en proposant comme premier argument la valeur de la constante d'élasticité **k** et comme deuxième argument la valeur de la déformation **d**) et doit respecter les consignes suivantes :
  - i. la valeur de son premier argument **k** est "stockée" dans le champ **k** par un appel à la méthode setter correspondante ;
  - ii. si la valeur de son deuxième argument **d** est strictement positive, on copie la valeur de cet argument dans le champ **d** ;

- f) un constructeur **public** (surchargé) avec un argument de type **Ressort** (et qui assure que le nouveau ressort créé a les mêmes caractéristiques que le ressort argument) ; ce constructeur doit respecter les consignes suivantes :
- i. si la valeur de son argument est la valeur spéciale **null** (c'est-à-dire si l'adresse correspondant à son argument n'est pas définie), on stocke dans les champs **k** et **d** de l'objet créé la valeur zéro ;
  - ii. autrement, on copie la valeur du champ **k** de l'objet argument dans le champ **k** de l'objet créé et la valeur du champ **d** de l'objet argument dans le champ **d** de l'objet créé ;
- g) une méthode **publique d'instance** nommée **corriger** qui n'a pas d'argument, ne retourne pas de résultat et respecte les consignes suivantes : si la valeur du champ **d** de l'objet appelant est strictement négative, sa nouvelle valeur doit devenir l'ancienne valeur avec le signe changé ;
- h) une méthode **publique d'instance** nommée **calculer** qui n'a pas d'argument et retourne l'adresse d'un tableau de type numérique réel ; en fait, cette méthode calcule et retourne la norme de la force élastique et la valeur de l'énergie potentielle de déformation associées au ressort correspondant à l'objet appelant et doit être définie en respectant les consignes suivantes :
- i. on crée un (nouveau) tableau de deux éléments de type numérique réels et on stocke son adresse dans une variable locale créée à cet effet ;
  - ii. on stocke dans le premier élément du tableau la norme de la force élastique **F** du ressort correspondant à l'objet appelant, en sachant que  $F = k*d$  ;
  - iii. on stocke dans le deuxième élément du tableau la valeur de l'énergie potentielle de déformation **E** du ressort correspondant à l'objet appelant (en sachant que  $E = k*d^2/2$ ) ;
  - iv. on retourne l'adresse du tableau créé par la méthode ;
- i) une méthode **publique** et **statique** nommée **serialiser** qui permet de calculer les caractéristiques d'un ressort équivalent à deux ressorts couplés en série (en sachant que le montage en série de deux ressorts de constantes d'élasticité **k1** et **k2** et, respectivement, de déformations **d1** et **d2** peut être remplacé par un seul ressort équivalent de constante d'élasticité équivalente  $ke=(k1*k2)/(k1+k2)$  et dont la déformation équivalente vaut  $de=d1+d2$ ) ; plus précisément, cette méthode a deux arguments de type **Ressort** (qui

correspondent aux deux ressorts couplés en série), retourne un résultat de type **Ressort** (qui correspond au ressort équivalent) et doit être écrite en respectant les consignes suivantes :

- i. on crée une variable locale de type numérique réel nommée **ke** et on y stocke la constante d'élasticité équivalente au montage en série des deux ressorts arguments (conformément aux formules ci-dessus) ;
  - ii. on crée une variable locale de type numérique réel nommée **de** et on y stocke la déformation équivalente au montage en série des deux ressorts arguments (conformément aux formules ci-dessus) ;
  - iii. on crée un nouvel objet de type **Ressort** correspondant à un objet dont les champs d'instance valent **ke** et **de**, et on retourne l'adresse de ce nouvel objet ;
- j) une méthode **publique** et **statique** nommée **paralleliser** qui permet de calculer les caractéristiques d'un ressort équivalent à deux ressorts couplés en parallèle (en sachant que le montage en parallèle de deux ressorts de constantes d'élasticité **k1** et **k2** et dont les déformations **d1** et **d2** sont égales peut être remplacé par un seul ressort équivalent de constante d'élasticité équivalente **ke=k1+k2** et dont la déformation équivalente vaut **de=d1=d2**) ; plus précisément, cette méthode a deux arguments de type **Ressort** (qui correspondent aux deux ressorts couplés en parallèle), retourne un résultat de type **Ressort** (qui correspond au ressort équivalent) et doit être écrite en respectant les consignes suivantes :
- i. on crée une variable locale de type numérique réel nommée **ke** et on lui donne la valeur initiale **0** ;
  - ii. on crée une variable locale de type numérique réel nommée **de** et on lui donne la valeur initiale **0** ;
  - iii. si les champs **d** des deux objets arguments ont la même valeur :
    - i. on stocke dans la variable **ke** la constante d'élasticité équivalente au montage en parallèle des deux ressorts arguments (conformément aux formules ci-dessus) ;
    - ii. on copie dans la variable **de** la valeur du champ **d** de l'objet correspondant au premier argument ;
  - iv. (dans tous les cas) on crée un nouvel objet de type **Ressort** correspondant à un objet dont les champs d'instance valent **ke** et **de**, et on retourne l'adresse de ce nouvel objet ;

[illegible]

This image shows a full page of a handwriting practice worksheet. It consists of approximately 20 horizontal rows. Each row is defined by two parallel dotted lines, creating a series of uniform gaps for writing. The lines are evenly spaced across the entire page, providing a guide for letter height and placement. There is no text or other markings on the page.

[illegible]

This image shows a full page of white paper with horizontal dotted lines, typical of primary school writing paper. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.



**1.2** Ecrire le *code complet* de la classe publique **CP\_Ctr2Exo1** qui appartient au package **cms\_ctr2** et qui contient la méthode **main**. Cette classe réalise la partie interactive du projet (voir aussi l'exemple d'affichage à l'exécution présenté plus loin) et utilise la classe **Ressort**.

Plus précisément, dans la méthode **main** :

- a) on déclare (et, le cas échéant, on initialise) les variables (locales) dont on a besoin pour réaliser le dialogue avec l'utilisateur ;
- b) on déclare deux variables locales de type numérique réel nommées **k** et **d** ;
- c) on affiche à l'écran un message qui demande à l'utilisateur d'introduire la valeur de la constante d'élasticité du ressort ;
- d) on récupère la valeur introduite par l'utilisateur et on la stocke dans la variable **k** ;
- e) on affiche à l'écran un message qui demande à l'utilisateur d'introduire la valeur de la déformation du ressort ;
- f) on récupère la valeur introduite par l'utilisateur et on la stocke dans la variable **d** ;
- g) on déclare 4 variables locales de type **Ressort** nommées **r1**, **r2**, **r3** et **r4** ;
- h) on crée un nouvel objet correspondant à un ressort de constante d'élasticité **k** et de déformation **d** et on stocke son adresse dans la variable **r1** ;
- i) (à l'aide du constructeur avec un argument de la classe **Ressort**) on crée un nouvel objet correspondant à un ressort qui a la même constante d'élasticité et la même déformation que le ressort **r1**, et on stocke l'adresse du nouvel objet dans la variable **r2** ;
- j) dans la variable **r3**, on stocke l'adresse d'un (nouvel) objet correspondant à un ressort équivalent au montage en série des ressorts **r1** et **r2** ; pour cela, il convient d'appeler la méthode *serialiser (...)* de la classe **Ressort** ;
- k) on affiche à l'écran le résultat retourné par un appel à la méthode *toString( )* pour le ressort **r3** ;
- l) dans la variable **r4**, on stocke l'adresse d'un (nouvel) objet correspondant à un ressort équivalent au montage en parallèle des ressorts **r1** et **r2** ; pour cela, il convient d'appeler la méthode *paralleliser (...)* de la classe **Ressort** ;
- m) on déclare une variable (locale) de type tableau de valeurs numériques réelles et on y stocke le résultat retourné par un appel à la méthode *calculer( )* (de la classe **Ressort**) pour le ressort correspondant à la variable **r4** ;
- n) on affiche le message *La force vaut XXX et l'énergie vaut YYY !* où **XXX** est la valeur du premier élément et **YYY** est la valeur du deuxième élément du tableau ci-dessus.



[illegible]

## Sujet no 2.

Préciser les messages qui seront affichés à l'écran suite à l'exécution du projet contenant les deux classes suivantes (qui appartiennent à un même package).

```
package cms_ctr2;
class Rectangle
{
    double x,y;

    Rectangle( ){
        this(10, 20);
        System.out.println("Je viens au monde");
    }

    Rectangle(double a, double b){
        System.out.println("J'arrive au monde !");
        x=a;
        y=b;
    }

    Rectangle(double a, double b, boolean flag){
        if(flag)
            System.out.println("Salut tout le monde !");
        x=a;
        y=b;
    }

    Rectangle composer(Rectangle r){
        System.out.println("On compose !");
        return new Rectangle(x+r.y, y+r.x);
    }

    static void rectifier(Rectangle r){
        if(r.x < r.y)
            r.x = r.y;
        else
            r.y = r.x;
        System.out.println("On rectifie !");
    }

    int comparer(Rectangle r){
        System.out.println("On compare " + toString()
                           + " avec " + r.toString());
        if(x*y < r.x*r.y)
            return -1;
        if(x*y > r.x*r.y)
            return 1;
        return 0;
    }
}
```

```

static boolean verifier(Rectangle r1, Rectangle r2){
    System.out.println("On vérifie " + r1.toString()
                       + " avec " + r2.toString());
    if(r1.comparer(r2) >= 0)
        return true;
    else
        return false;
}

public String toString(){
    return "(" + x + " x " + y + ")";
}

static void afficher(Rectangle tab[], int i, int j){
    for(int k=i; k<=j; k++)
        System.out.println(tab[k]);
}
}
} //fin de la classe Rectangle

```

```

public class CP_Ctr2Exo2 {
    public static void main(String[] args)
    {
        System.out.println("***Naissances***");
        Rectangle r1 = new Rectangle(5,2);
        System.out.println("-----");
        Rectangle r2 = new Rectangle();
        System.out.println("-----");
        Rectangle r3 = new Rectangle(-1,2);
        System.out.println("-----");
        System.out.println(r1.toString());
        System.out.println(r2.toString());
        System.out.println(r3.toString());

        System.out.println("***Affectations***");
        r3 = r2;
        r2 = r1;
        r1 = r3;
        System.out.println(r1.toString());
        System.out.println(r2.toString());
        System.out.println(r3.toString());
    }
}

```

```

System.out.println("***Manipulations***");
Rectangle r4 = new Rectangle(5,2,false);
Rectangle r5 = new Rectangle(7,3, false);
Rectangle r6 = r4.composer(r5);
Rectangle.rectifier(r4);
System.out.println(r4.toString());
System.out.println(r6.toString());

System.out.println("***Vérifications***");
Rectangle r7 = new Rectangle(5,2,false);
Rectangle r8 = new Rectangle(5,2,false);
if(r7==r8)
    System.out.println(r7.toString() + " le même que "
                        + r8.toString());
else
    System.out.println(r7.toString() + " différent de "
                        + r8.toString());
if(r7.equals(r8))
    System.out.println(r7.toString() + " égal au " +
                        r8.toString());
else
    System.out.println(r7.toString() + " pas égal à "
                        + r8.toString());
System.out.println("-----");
System.out.println(r7.comparer(r8));
System.out.println(Rectangle.verifier(r7, r8));
System.out.println("-----");
Rectangle r9 = new Rectangle(7,3,false);
System.out.println(r9.comparer(r8));
System.out.println(Rectangle.verifier(r8, r9));

System.out.println("***La cerise***");
Rectangle.afficher(new Rectangle [] {r4, r5, r7, r8, r9},
                    3, 1);
} //fin de la méthode main
} //fin de la classe principale

```

This image shows a full page of white paper with horizontal dotted lines. The lines are evenly spaced and run across the width of the page, providing a guide for handwriting or typing. There are no margins, text, or other markings on the page.

[illegible]



[illegible]

Remarque : Cette page doit rester normalement blanche (mais vous pouvez l'utiliser pour la rédaction de vos réponses si la place prévue à cet effet s'est avérée insuffisante).

```

package cms_ctr2;

public class Ressort {
    private double k;
    double d = 1;

    public void setK(double k){
        if(k >= 0)
            this.k = k;
        else
            this.k = 10;
    }

    public double getK( ){
        return k;
    }

    public Ressort(double k, double d){
        setK(k);
        if(d > 0)
            this.d = d;
    }

    public Ressort(Ressort r){
        if(r == null){
            k = 0;
            d = 0;
        }else{
            k = r.k ;    //setK(r.k);
            d = r.d;
        }
    }

    public void corriger(){
        if(d<0)
            d = -d;
    }
}

```

```

public double[] calculer( ){
    double [] tab = new double[2];
    tab[0] = k*d;
    tab[1] = k*d*d/2;
    return tab;
}

public static Ressort serialiser(Ressort r1,
                                Ressort r2){
    double ke = r1.k*r2.k / (r1.k+r2.k);
    double de = r1.d + r2.d;
    Ressort r = new Ressort(ke,de);
    return r;
}

public static Ressort paralleliser(Ressort r1,
                                    Ressort r2){
    double ke = 0;
    double de = 0;
    if(r1.d == r2.d)
    {
        ke = r1.k + r2.k;
        de = r1.d;
    }
    Ressort r = new Ressort(ke,de);
    return r;
}

public String toString( ){
    return "Un ressort de constante d'élasticité k="
        + k + " et de déformation d=" + d + " !";
}
}
} //fin de la classe Ressort

```

\*\*\*Naissances\*\*\*

J'arrive au monde !

-----

J'arrive au monde !

Je viens au monde

-----

J'arrive au monde !

-----

(5.0 x 2.0)

(10.0 x 20.0)

(-1.0 x 2.0)

\*\*\*Affectations\*\*\*

(10.0 x 20.0)

(5.0 x 2.0)

(10.0 x 20.0)

\*\*\*Manipulations\*\*\*

On compose !

J'arrive au monde !

On rectifie !

(5.0 x 5.0)

(8.0 x 9.0)

\*\*\*Vérifications\*\*\*

(5.0 x 2.0) différent de (5.0 x 2.0)

(5.0 x 2.0) pas égal à (5.0 x 2.0)

-----

On compare (5.0 x 2.0) avec (5.0 x 2.0)

0

On vérifie (5.0 x 2.0) avec (5.0 x 2.0)

On compare (5.0 x 2.0) avec (5.0 x 2.0)

true

-----

On compare (7.0 x 3.0) avec (5.0 x 2.0)

1

On vérifie (5.0 x 2.0) avec (7.0 x 3.0)

On compare (5.0 x 2.0) avec (7.0 x 3.0)

false

\*\*\*La cerise\*\*\*

(5.0 x 2.0)

(5.0 x 2.0)

(7.0 x 3.0)

.....