

Contrôle d'informatique no 3

Durée : 1 heure 45'

Nom :

Groupe :

Prénom :

No	1	2
Total points	120 points (4+3+45+42+26)	77 points

Remarque générale : toutes les questions qui suivent se réfèrent au langage de programmation Java (à partir du JDK 5.0) et les réponses doivent être rédigées à l'encre et d'une manière propre sur ces feuilles agrafées.

Remarques initiales :

- Il est conseillé de lire les sujets jusqu'à la fin, avant de commencer la rédaction de la solution.
- Les sujets peuvent être résolus (éventuellement) séparément, mais après avoir lu et compris l'ensemble des énoncés.
- Si vous ne savez pas implémenter une méthode, écrivez son en-tête, réduisez son corps à un simple commentaire et passez à la suite.
- L'aire d'un rectangle de côtés a et b vaut $a*b$.
- L'aire (totale) d'un pavé (ou parallélépipède) de côtés a , b et c vaut $2*(a*b+b*c+c*a)$, tandis que son volume vaut $a*b*c$.
- La classe **Object** est la classe prédéfinie (ou standard) du package *java.lang* ; elle est la classe "racine", ancêtre de toute autre classe Java.
- La fonction "valeur absolue" est implémentée en Java par la méthode statique *abs* de la classe **Math** du package *java.lang*.
- Faites attention aux éventuels **casts** obligatoires ainsi qu'aux accès aux champs privés.

Sujet no 1.

On considère un projet Java qui contient deux interfaces **publicques** et trois classes **publicques** regroupées dans un package nommé *cms_ctr3*, à savoir :

- les interfaces *ICalculable* et *IFusionnable* ;
- la classe *Rectangle* qui implémente les deux interfaces mentionnées ci-dessus ;
- la classe *Pave* qui dérive de la classe de base *Rectangle* ;
- la classe principale *CP_Ctr3Exo1* qui utilise les classes *Rectangle* et *Pave*.

Le but de cet exercice est de réaliser une application autonome interactive qui aide l'utilisateur à travailler avec des rectangles et des pavés (c'est-à-dire des parallélépipèdes). Chaque interface et chaque classe sont **publicques** et définies dans un fichier à part. On vous demande d'écrire les codes complets des interfaces *ICalculable* et *IFusionnable* et des classes *Rectangles* et *Pave*, en respectant les consignes précisées. En outre, le code source de la classe principale *CP_Ctr3Exo1* est donné et vous devrez indiquer quels seront les résultats affichés dans la fenêtre console suite à son exécution

1.1 L'interface **publicque** *ICalculable* fait partie du package *cms_ctr3* et contient les en-têtes des deux méthodes suivantes :

- la méthode nommée *calculerAire*, sans argument et qui retourne un résultat de type numérique réel ;
- la méthode nommée *calculerVolume*, sans argument et qui retourne un résultat de type numérique réel.

Indiquer ci-dessous le code complet de l'interface *ICalculable*.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

1.2 L'interface **publique** *IFusionnable* fait partie du package *cms_ctr3* et contient l'en-tête d'une seule méthode nommée *fusionner*, avec un seul argument de type *Object* et qui retourne un résultat de type *Object*.

Indiquer ci-dessous le code complet de l'interface *IFusionnable*.

.....

.....

.....

.....

.....

.....

.....

.....

1.3 Une instance de la classe *Rectangle* est un objet de type *Rectangle* qui encapsule les valeurs des deux côtés *a* et *b* d'un rectangle.

La classe *Rectangle* est **publique**, fait partie du package *cms_ctr3* et implémente les interfaces *ICalculable* et *IFusionnable*.

De plus, la classe *Rectangle* définit :

- a) un champ (d'instance) nommé *a*, de type numérique réel, déclaré **privé**, avec *1* comme valeur initiale explicite et qui sert à stocker la valeur d'un côté du rectangle ;
- b) un champ (d'instance) nommé *b*, de type numérique réel, déclaré **privé**, avec *1* comme valeur initiale explicite et qui sert à stocker la valeur de l'autre côté du rectangle ;
- c) 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 *a* ;
- d) 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 *a* ; plus précisément, si la valeur de son argument noté lui aussi *a* est strictement positive, cette méthode copie la valeur de cet argument dans le champ *a* ;
- e) une méthode **publique** (d'instance) "**getter**" similaire à celle précisée au point c) mais pour le champ privé *b* ;

- f) une méthode **public** (d'instance) "**setter**" similaire à celle précisée au point d) mais pour le champ privé **b** ;
- g) un constructeur **public** avec deux arguments de type numérique réel et qui permet la création d'un objet correspondant à un nouveau rectangle ; par des appels aux méthodes "setters" adéquates, ce constructeur "stocke" la valeur de son premier argument dans le champ **a** et la valeur de son deuxième argument dans le champ **b** du nouvel objet créé.

Ensuite, la classe **Rectangle** (re)définit les méthodes mentionnées dans les deux interfaces implémentées, à savoir :

- h) la méthode **calculerAire** qui doit retourner la valeur de l'aire du rectangle correspondant à l'objet appelant ;
- i) la méthode **calculerVolume** qui doit retourner (toujours) la valeur zéro ;
- j) la méthode **fusionner** qui doit respecter les consignes suivantes :
 - i. la méthode affiche à l'écran le message **Fusionner de la classe Rectangle !** ;
 - ii. si la valeur de son argument est la valeur spéciale **null**, la méthode retourne la valeur spéciale **null** ;
 - iii. (autrement) si son argument est de type **Rectangle** (ou compatible par polymorphisme avec le type **Rectangle**), la méthode :
 - i. crée un nouvel objet de type **Rectangle** dont la valeur du champ **a** est la somme des valeurs des champs **a** de l'objet appelant et de l'objet argument et la valeur du champ **b** est la somme des valeurs des champs **b** de l'objet appelant et de l'objet argument ;
 - ii. retourne l'adresse de ce nouvel objet ;
 - iv. (autrement) la méthode retourne la valeur spéciale **null** ;

Finalement, la classe **Rectangle** :

- k) redéfinit la méthode publique **toString** héritée de la classe racine **Object** de sorte qu'elle retourne la chaîne de caractères :

Rectangle : longueur=XXX, largeur=YYY

où **XXX** est la valeur du champ **a** et **YYY** est la valeur du champ **b** de l'objet appelant.

Indiquer à la page suivante le code source complet de la classe **Rectangle**.

[illegible]

This image shows a full page of a handwriting practice worksheet. It consists of numerous horizontal rows, each defined by two parallel dotted lines. The rows are evenly spaced and extend across the entire width of the page, providing a guide for letter height and placement. There is no text or other markings on the page.

This image shows a full page of primary-ruled paper. It features approximately 28 horizontal dotted lines spaced evenly down the page, providing a guide for handwriting practice. The paper is otherwise blank, with no margins, text, or other markings.

1.4 Une instance de la classe *Pave* est un objet de type *Pave* qui encapsule les valeurs des trois côtés *a*, *b* et *c* d'un pavé, c'est-à-dire d'un parallélepède.

La classe *Pave* est **publique**, fait partie du package *cms_ctr3* et dérive de la classe *Rectangle*. Autrement dit, la classe *Pave* est la fille (ou la classe dérivée) de la classe mère (ou classe de base) *Rectangle*.

De plus, la classe *Pave* définit :

- a) un champ (d'instance propre) nommé *c*, de type numérique réel, déclaré **privé**, avec *1* comme valeur initiale explicite et qui sert à stocker la hauteur du pavé ;
- b) 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 (propre) privé *c* ;
- 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 (propre) privé *c* ; plus précisément, cette méthode copie la valeur absolue de son argument noté lui aussi *c* dans le champ *c* ;
- d) un constructeur **public** avec trois arguments de type numérique réel et qui permet la création d'un objet correspondant à un nouveau pavé ; ce constructeur doit respecter les consignes suivantes :
 - i. par un appel au constructeur de la classe mère, il "stocke" la valeur de son premier argument dans le champ (hérité) *a* et la valeur de son deuxième argument dans le champ (hérité) *b* du nouvel objet créé ;
 - ii. par un appel à la méthode "setter" adéquate, il "stocke" la valeur de troisième argument dans le champ (propre) *c* du nouvel objet.

Ensuite, la classe *Pave* redéfinit les méthodes héritées suivantes :

- e) la méthode *calculerAire* qui doit retourner la valeur de l'aire (totale) du pavé correspondant à l'objet appelant ;
- f) la méthode *calculerVolume* qui doit retourner le volume du pavé correspondant à l'objet appelant ;
- g) la méthode *fusionner* qui doit respecter les consignes suivantes :
 - i. la méthode affiche à l'écran le message *Fusionner de la classe Pave !* ;
 - ii. si la valeur de son argument est la valeur spéciale *null*, la méthode retourne la valeur spéciale *null* ;

- iii. (autrement) si la classe correspondant au type de l'objet argument est (exactement) la même que la classe correspondant au type de l'objet appelant, la méthode :

- i. crée un nouvel objet de type *Pave* dont les champs *a* et *b* ont, respectivement, les mêmes valeurs que les champs *a* et *b* de l'objet appelant et dont le champ *c* a comme valeur la somme des valeurs des champs *c* de l'objet appelant et de l'objet argument ;
- ii. retourne l'adresse de ce nouvel objet ;

iv. (autrement) la méthode retourne la valeur obtenue par un appel de la méthode d'origine (*fusionner* de la classe mère) avec le même objet argument ;

h) la méthode *toString* qui doit retourner le texte :

Rectangle : longueur=XXX, largeur=YYY; Pavé : hauteur=ZZZ

où **XXX** est la valeur du champ (hérité) **a**, **YYY** est la valeur du champ (hérité) **b** et **ZZZ** est la valeur du champ (propre) **c** de l'objet appelant.

Indiquer ci-dessous le code source complet de la classe *Pave*.

[illegible]

This image shows a full page of white paper with horizontal dashed 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.

[illegible]

1.5 On vous donne le code source de la classe principale *CP_Ctr3Exo1* qui fait partie du package *cms_ctr3* et qui utilise les classes présentées auparavant.

```
package cms_ctr3;

public class CP_Ctr3Exo1
{
    public static void main(String[ ] args)
    {
        Rectangle refMixte1 = new Pave(1, 2, 3);
        Rectangle ref2 = new Rectangle(10, 20);

        System.out.println(refMixte1.fusionner("gigi"));
        System.out.println("-----");

        System.out.println(refMixte1.fusionner(ref2));
        System.out.println("-----");

        System.out.println(ref2.fusionner(refMixte1));
        System.out.println("-----");

        System.out.println(refMixte1.fusionner(refMixte1));
        System.out.println("-----");

        System.out.println(ref2.fusionner(ref2));
    } //fin de la méthode main
} //fin de la classe principale
```

Remarque: si l'argument de la méthode *System.out.println()* est la référence vers un objet, le texte affiché à l'écran correspond en fait à la chaîne de caractères retournée par un appel implicite de la méthode *toString()* pour cet objet.

Préciser à la page suivante quels sont les résultats affichés dans la fenêtre console suite à l'exécution de la classe principale *CP_Ctr3Exo1*.

[illegible]

Sujet no 2.

Préciser les messages qui seront affichés à l'écran suite à l'exécution du projet contenant le code source suivant (qui est contenu dans un seul fichier).

```
package cms_ctr3;

class Vehicule{
    String marque = "Inconnue";
    double vitesse = 25;

    Vehicule(String m, double v){
        marque = m;
        if(v>0)        vitesse = v;
        System.out.println("Vehicule avec deux args !");}

    Vehicule() {
        this("NoName", 50);
        System.out.println("Vehicule sans arg !");}

    void demarrer() {
        System.out.println(toString() + " démarre !");}

    double accelerer() {
        vitesse = vitesse + 5;
        System.out.println("Un véhicule accélère !");
        return vitesse;}

    int depasser(Vehicule veh){
        System.out.println("Deux véhicules se dépassent ");
        if(vitesse < veh.vitesse)    return -1;
        if(vitesse == veh.vitesse)  return 0;
        return 1;}

    public String toString() {
        return "Un véhicule " + marque;}
} //fin de la classe Vehicule

class Voiture extends Vehicule{
    int nbPlaces;

    Voiture(int n){
        nbPlaces = n;
        System.out.println("Voiture avec un arg !");}

    Voiture(String m, double v, int n){
        super(m, v);
        nbPlaces = n;
        System.out.println("Voiture avec trois args !");}

    Voiture() {
        this("Renault", 70, 5);
        System.out.println("Voiture sans arg !");}
```

```

void accelerer(double deltaV){
    if(deltaV > 0)        vitesse = vitesse + deltaV;
    System.out.println("Une voiture accélère !");}

double accelerer( ){
    accelerer(10);
    System.out.println("Une voiture roule plus vite !");
    return vitesse;}

int depasser(Voiture auto){
    System.out.println("Deux voitures se dépassent !");
    return super.depasser(auto);}

public String toString( ){
    return "Une voiture " + marque + " à vitesse " + vitesse;}
} //fin de la classe Voiture

public class CP_Ctr3Exo2 {
    public static void main(String args[ ]){
        Vehicule t[ ] = new Vehicule[5];

        t[0] = new Voiture( );
        System.out.println("-----");
        t[1] = new Vehicule("Saab", 80);
        System.out.println("-----");
        t[2] = new Vehicule( );
        System.out.println("-----");
        t[3] = new Voiture(6);
        System.out.println("-----");
        t[4] = new Voiture("Fiat", 45, 8);
        System.out.println("-----");

        System.out.println("*****Première partie*****");
        for( int i=0; i<t.length; i++ ){
            t[i].demarrer( );
            System.out.println( t[i].accelerer( ) );
            System.out.println("-----");
        }

        System.out.println("*****Deuxième partie*****");
        for( int i=0; i<t.length-1; i++ ){
            System.out.println( t[i].depasser(t[i+1]) );
            if( t[i] instanceof Voiture ){
                System.out.println( ((Voiture)t[i]).depasser(t[i+1]) );
                if(t[i+1] instanceof Voiture)
                    System.out.println( ((Voiture)t[i]).depasser(((Voiture)t[i+1])) );
            } else if( t[i+1] instanceof Voiture )
                System.out.println( t[i].depasser(((Voiture)t[i+1])) );
            System.out.println("-----");
        }
    } //fin de la méthode main
} //fin de la classe principale

```

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.

This image shows a full page of a handwriting practice worksheet. It consists of approximately 28 horizontal rows. Each row is defined by two parallel dotted lines, creating a series of uniform gaps for letter height. The lines are evenly spaced across the entire page, providing a guide for consistent letter formation. There is no text or other markings on the page.

