ECE PARIS
ÉCOLE D'INGÉNIEURS

POO Java
ING3 2017-2018
CORRIGE Examen DS 1

Durée : 1h30 **AUCUN DOCUMENT - AUCUNE CALCULATRICE**

Lisez attentivement chaque partie du sujet avant de vous lancer dans vos réponses.

Partie 1 : 8 QCM ≈ 15 minutes (4 POINTS). Une seule réponse par question :
0,5 point par réponse juste, -0,25 point par réponse fausse, 0 point sans réponse

Q1) Soit la déclaration de 3 variables *locales* ci-dessous. Sélectionnez l'expression qui compile et exécute sans erreur.
String a = "aaa", b = "a", c;

a) c += a; **b) c = a + b;** **c) c = a - b;** **d) c = 'a';**

Q2) Un attribut d'une classe avec la *visibilité par défaut* est visible par …

a) … toutes les classes du programme
b) … toutes les classes de son package
c) … seulement par la classe elle-même
d) … seulement par la classe elle-même et ses sous-classes

Q3) Sélectionnez la seule phrase qui est fausse.

a) Une classe abstraite peut être instanciée
b) Une classe abstraite peut implémenter des méthodes
c) Une classe abstraite peut définir des prototypes de méthodes
d) Une classe abstraite peut définir des attributs

Q4) Qu'affiche le programme à l'exécution?

```
public class N
{
    private int i = 1;
    public static void go (N n, int i){
        i = 3;
        n.i = 4;
    }
    public static void main (String argv[]) {
        N n = new N();
        int i=2;
        N.go(n,i);
        System.out.print(n.i + " " + i);
    }
}
```

a) "1 2" **b) "1 3"** **c) "4 2"** **d) "4 3"**

Q5) Combien de références de type String sont créées par l'instruction ci-dessous?
String[] strings = new String[10];

a) 0 **b) 9** **c) 10** **d) 11**

Q6) Parmi les déclarations suivantes, sélectionnez l'instruction légale.

```
public interface T {
    void t();
}
public class M implements T {
    void t();
    void m();
}
public class N extends M {
    void n();
}
```

a) T t1 = new T();
b) T t2 = new N(); t2.n();
c) N t3 = new M();
d) M t4 = new N();

Q7) Qu'affiche le programme?

```
public class A {
    void prn () { System.out.println (this.getVal()); }
    int getVal () { return 10; }
}
public class B extends A {
    int getVal () { return 20; }
}
public class C extends B {
    public static void main (String args[]) {
        new B().prn();
        new C().prn();
    }
}
```

a) "10", "10" **b) "20", "10"** **c) "20", "20"** **d) Une erreur**

Q8) Identifiez la (les) ligne(s) numérotée(s) du programme ci-dessous qui génère(nt) une erreur.

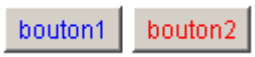
```
public class DD {
    private int a = 10;
    protected int b;
}
public class EE extends DD {
    int c;
    public void display() {
        System.out.println (a); // Line (1)
        System.out.println (b); // Line (2)
        System.out.println (c); // Line (3)
    }
    public static void main(String[] args) {
        new EE().display();
    }
}
```

a) Line (1) **b) Line (2)** **c) Line (3)** **d) Les 3 lignes génèrent une erreur**

Des extraits de la documentation des API Java sont fournis en ANNEXE dans les dernières pages.
N'oubliez pas d'importer les librairies nécessaires pour chacune de vos classes à coder ci-dessous

Partie 2 : code Java Swing et Listeners ≈ 30 minutes (6 POINTS)

Il s'agit de créer une fenêtre qui dispose des deux boutons suivants :



Chacun des 2 boutons interagit sur l'autre bouton en changeant sa couleur : par exemple, si l'on clique sur « bouton1 », la couleur de « bouton2 » est changée du rouge au bleu (ou du bleu au rouge), et vice-versa. On utilisera les méthodes suivantes de la classe **JButton** : *getForeground* pour retourner la couleur d'un bouton, et *setForeground* pour changer sa couleur passée en argument (voir ANNEXE). Exemples de couleurs en Java : **Color.blue** pour le bleu, **Color.red** pour le rouge etc.

2.1) [3 points] **(0,5 point pour l'import de librairies)** Écrire le code Java d'une classe *UnBouton* qui hérite de la classe **JButton**, implémente l'interface **ActionListener (0,5 point)** et définit les propriétés suivantes d'un bouton :

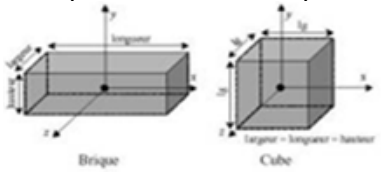
- Ses attributs privés : sa couleur de type **Color** et son texte de type String **(0,5 point)**
- Un constructeur qui initialise sa couleur et son texte avec les paramètres **(0,5 point)**
- La méthode *actionPerformed* : si l'on clique sur le « bouton1 », le « bouton2 » change de couleur (s'il est bleu, il passe au rouge et vice-versa) ; et si l'on clique sur le « bouton2 », le « bouton1 » change de couleur. **(1 point)**

2.2) [3 points] Écrire le code Java d'une classe *DeuxBoutons* **(0,5 point si hérite de JFrame ou définit un attribut de Jframe)** qui définit les propriétés suivantes :

- Ses attributs privés : les deux boutons instances de la classe *UnBouton* de la question **2.1) (0,5 point)**
- Le **main** qui crée la fenêtre avec ses deux boutons « bouton1 » et « bouton2 » **(0,5 point)** et la rend visible **(0,5 point)**. Le « bouton1 » est initialisé à bleu, le « bouton2 » à rouge **(0,5 point)**. Chaque bouton doit servir de *listener* pour l'autre bouton **(0,5 point)**

Partie 3 : code Java classe abstraite, héritage et polymorphisme ≈ 45 minutes (10 POINTS)

On souhaite disposer d'un ensemble de classes permettant de manipuler des formes tridimensionnelles :



Pour cela, on propose la hiérarchie de classes de la figure 1 suivante :

Figure 1 : hiérarchie des classes

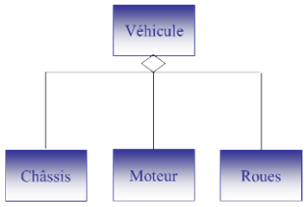
Les définitions des classes sont les suivantes :

- La classe *Point3D* définit 3 attributs public réels *x*, *y* et *z* d'un point à 3 dimensions. Sa méthode *bouger* prend comme paramètres 3 réels pour bouger (translater) les 3 attributs du point avec chacun de ses paramètres.
- La classe *Forme* est une classe abstraite. Cette classe définit un attribut protected de type *Point3D* qui représente son centre de gravité. Elle dispose des méthodes suivantes :
  - Un *getter* qui retourne le centre de gravité de la forme.
  - Deux méthodes abstraites *calculerSurface()* et *calculerVolume()* : elles devront calculer et retourner respectivement la surface et le volume réels des formes héritées ci-dessous.
- La classe *Brique* hérite de *Forme* et définit 3 attributs réels : *largeur*, *longueur* et *hauteur*. La méthode *calculerSurface()* retourne la surface d'une brique : 2 * (*largeur* longueur* + *largeur* hauteur* + *longueur* hauteur*). La méthode *calculerVolume()* retourne le volume d'une brique : *largeur* longueur* hauteur*.
- La classe *Cube* est (hérite de) une brique pour laquelle les 3 dimensions *largeur* = *longueur* = *hauteur*.

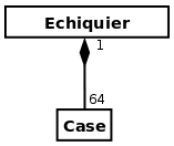
3.1) [2 points]
Complétez la hiérarchie des classes de la *figure 1* en y intégrant la classe *Point3D* et tous les attributs et méthodes, en respect de la définition de cette classe **(0,5 point)**. Expliquez brièvement la (ou les) relation(s) interclasses **(0,5 point)**. Pour chaque attribut, précisez leur niveau de visibilité (- pour **private**, + pour **public** ou # pour **protected**) et leur type **(0,5 point)**. Pour les méthodes toutes publiques, précisez le type (**void** ou autre) de la méthode et leur(s) paramètre(s) **(0,5 point)**.

Exemple de relations interclasses :

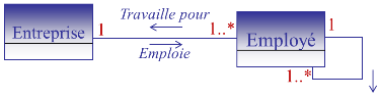
Agrégation (Dépendance faible)



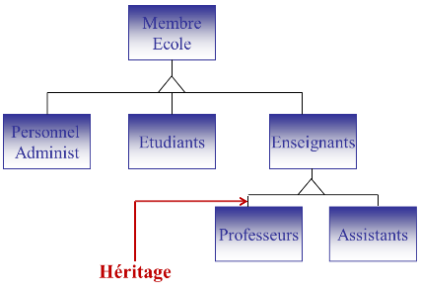
Composition (Dépendance forte)



Lien et Associations



Héritage



3.2) [5 points] **(0,5 point pour l'import de librairies)**
Implémentez et commentez le code Java des classes *Point3D*, *Forme*, *Brique* et *Cube*, **(0,5 point)** en respect des contraintes spécifiées sur ces classes dans la page précédente :

- Définir tous les attributs **(0,5 point)**

- Implémenter les constructeurs :
  - Le constructeur par défaut de la classe *Point3D* génère aléatoirement les 3 attributs entre 2.5 et 5.3. **(0,5 point)**
  - Le constructeur de la classe *Brique* avec en paramètres le centre de gravité, la *largeur*, la *longueur* et la *hauteur* qui initialisent les attributs **(0,5 point)**
  - Le constructeur de la classe *Cube* avec en paramètres le centre de gravité et une tri-dimension (la même pour la *largeur*, la *longueur* et la *hauteur*), par héritage du constructeur de sa classe mère *Brique* **(0,5 point)**

- Implémenter les méthodes de ces classes :
  - La méthode *bouger* dans les classes *Point3D* **(0,5 point)**
  - Le *getter* du centre de gravité dans la classe *Forme* **(0,5 point)**
  - Les méthodes polymorphes *calculerSurface()* **(0,5 point)** et *calculerVolume()* **(0,5 point)** dans les classes *Forme* et *Brique*

3.3) [3 points]
Implémentez une classe avec le **main**, qui effectue les traitements suivants :

- Déclarer et instancier un point de la classe *Point3D*. **(0,5 point)**
- Bouger les 3 dimensions du point de +1 en appelant la méthode *bouger* **(0,5 point)**
- Déclarer et saisir un nombre réel tant que ce nombre est négatif ou nul. **(0,5 point)**
- Déclarer et instancier un objet de la classe Cube, avec en paramètres le point comme centre de gravité, et le nombre réel saisi comme tri-dimension. **(0,5 point)**
- Afficher les 3 dimensions *x*, *y* et *z* du point. **(0,5 point)**
- Afficher la surface et le volume de l'objet du *Cube*, en appelant les méthodes *calculerSurface()* et *calculerVolume()*. **(0,5 point)**

ANNEXE : extraits de la documentation des API Java

javax.swing

Class JFrame

Constructor Summary	
JFrame() Constructs a new frame that is initially invisible.	
JFrame(String title) Creates a new, initially invisible Frame with the specified title.	

Method Summary	
Container	**getContentPane**() Returns the contentPane object for this frame.
void	**setContentPane**(Container contentPane) Sets the contentPane property.
void	**setLayout**(LayoutManager manager) Sets the LayoutManager.
void	**setSize**(int width, int height) Resizes this component so that it has width width and height height.
void	**setVisible**(boolean b) Shows or hides this Window depending on the value of parameter b.

Class JButton

Constructor Summary	
JButton() Creates a button with no set text or icon.	
JButton(String text) Creates a button with text.	

Method Summary	
String	**getText**() Returns the button's text.
void	**setText**(String text) Sets the button's text
void	**addActionListener**(ActionListener l) Adds an ActionListener to the button.
Color	getForeground() Gets the foreground color of this component.
void	setForeground(Color c) Sets the foreground color of this component.

java.awt

Class Color

Field Summary	
static Color	**blue** The color blue.
static Color	**red** The color red.

Class Container

Method Summary	
Component	**add**(Component comp) Appends the specified component to the end of this container.
void	**setLayout**(LayoutManager mgr) Sets the layout manager for this container.

java.awt.event

Interface ActionListener

Method Summary	
void	**actionPerformed**(ActionEvent e) Invoked when an action occurs.

Class ActionEvent

Method Summary	
Object	**getSource**() The object on which the Event initially occurred.
String	**toString**() Returns a String representation of this EventObject.

java.util

Class Scanner

Method Summary	
float	**nextFloat**() Scans the next token of the input as a float.

java.lang

Class Random

Method Summary	
float	**nextFloat**() Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.

Class Math

Method Summary	
static double	**random()** Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.