

14 avril 2010

**Contrôle d'informatique no 3**

Durée : 1 heure 45'

Nom : .....

Groupe : 

Prénom : .....

No sujet	1	2
Nombre points	(49 + 34) points	27 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.

**Sujet no 1.****Remarques initiales :**

- Il est conseillé de lire ce sujet jusqu'à la fin, avant de commencer la rédaction de la solution.
- Le point **1.2** de ce sujet peut être résolu, éventuellement, avant la résolution du point **1.1**, mais après avoir lu et compris ce que le point **1.1** demande.
- En Java, la fonction trigonométrique sinus est implémentée sous la forme de la méthode statique *sin* de la classe *java.lang.Math* ; cette méthode a un seul argument de type *double* qui correspond à la mesure d'un angle exprimée en radians et retourne un résultat de type *double* qui correspond au sinus de cet angle.

- Afin de transformer la mesure d'un angle de degrés en radians, on peut utiliser la méthode statique *toRadians* de la classe *java.lang.Math* ; cette méthode a un seul argument de type *double* qui correspond à une mesure d'angle exprimée en degrés et retourne un résultat de type *double* qui correspond à la mesure du même angle exprimée en radians.
- D'une manière générale, si deux projectiles sont tirés d'un même endroit (caractérisé par une accélération gravitationnelle donnée) et sous un même angle initial, le projectile qui a la vitesse initiale la plus grande (en norme) arrive plus loin (c'est-à-dire a la portée la plus grande).
- De même, si deux projectiles sont tirés d'un même endroit (caractérisé par une accélération gravitationnelle donnée) et avec la même vitesse initiale, le projectile qui part avec un angle initial de  $45^\circ$  arrive plus loin que celui qui part avec un autre angle initial (c'est-à-dire, l'angle de  $45^\circ$  assure une portée maximale).

**1.1** Le but du premier point de ce sujet est d'écrire le code source de deux classes Java qui pourront être utilisées dans l'étude des problèmes de balistique, à savoir : une classe de base appelée *Tir* et une classe dérivée appelée *TirGalactique*.

D'abord, on suppose qu'un projectile est lancé à la surface de la Terre et que l'accélération gravitationnelle a la valeur constante de  $9.81 \text{ m/s}^2$ . Une instance de la classe de base *Tir* doit être alors un objet qui encapsule l'angle initial de tir (*alpha* mesuré en degrés) et la norme de la vitesse initiale du projectile (*v* mesurée en m/s).

Ensuite, on suppose qu'un projectile est lancé à la surface d'une autre planète et que l'accélération gravitationnelle a une valeur constante qui peut être différente de  $9.81 \text{ m/s}^2$ . Une instance de la classe dérivée *TirGalactique* doit être alors un objet qui encapsule aussi cette accélération gravitationnelle (*g* en  $\text{m/s}^2$ ) (en plus de l'angle initial de tir et de la norme de la vitesse initiale).

La classe de base *Tir* fait partie d'un package appelé *cms\_ctr3*, est publique et dérive directement de la classe racine *java.lang.Object*. Dans le corps de la classe *Tir*, on définit :

- deux champs privés *alpha* et *v*, sans valeurs initiales explicites, qui servent à stocker les valeurs réelles de l'angle initial de tir (exprimé en degrés) et, respectivement, de la norme de la vitesse initiale du projectile (exprimée en m/s) ;
- une méthode publique « getter » nommée *getAlpha* qui retourne (sans aucune vérification) la valeur du champ privé *alpha* ;

- une méthode publique « setter » nommée **setAlpha** qui permet la modification de la valeur du champ privé **alpha** ; cette méthode doit respecter les consignes suivantes :
  - si la valeur de son argument est comprise dans l'intervalle ouvert (**0 ; 90**), on stocke dans le champ **alpha** la valeur de l'argument de la méthode ;
  - autrement, on stocke dans le champ **alpha** la valeur **45** et on affiche le message « **Angle non valide !** » dans la fenêtre console ;
- une méthode publique « getter » nommée **getV** qui retourne (sans aucune vérification) la valeur du champ privé **v** ;
- une méthode publique « setter » nommée **setV** qui permet la modification de la valeur du champ privé **v** ; cette méthode doit respecter les consignes suivantes :
  - si la valeur de son argument est strictement positive, on stocke dans le champ **v** la valeur de l'argument de la méthode ;
  - autrement, on stocke dans le champ **v** la valeur **10** et on affiche le message « **Vitesse non valide !** » dans la fenêtre console ;
- un constructeur public avec deux arguments de type réel et qui doit respecter les consignes suivantes :
  - la valeur du premier argument « est stockée » dans le champ **alpha** par un appel à la méthode « setter » adéquate ;
  - la valeur du deuxième argument « est stockée » dans le champ **v** par un appel à la méthode « setter » adéquate ;
  - le message « **Un tir terrestre !** » est affiché dans la fenêtre console ;
- une méthode publique (d'instance) nommée **calculerPortee**, qui n'a pas d'argument et qui retourne la valeur réelle de la portée du tir correspondant à l'objet appelant la méthode ; cette portée est calculée (en mètres) avec la formule :

$$\frac{v^2 \cdot \sin(2 \cdot \alpha)}{9.81}$$

où l'angle **alpha** est exprimé en radians ;

- une méthode publique (d'instance) nommée **comparer** qui a un seul argument de type **Tir** et qui retourne un résultat de type valeur numérique entière ; plus précisément, après avoir affiché le message « **On est sur la Terre !** » dans la fenêtre console, cette méthode retourne la valeur :
  - **1** si la valeur de la portée correspondant à l'objet appelant la méthode est plus grande que la valeur de la portée correspondant à l'objet argument de la méthode ;

- **-1** si la valeur de la portée correspondant à l'objet appelant la méthode est plus petite que la valeur de la portée correspondant à l'objet argument de la méthode ;
- **0** si la valeur de la portée correspondant à l'objet appelant la méthode est égale à la valeur de la portée correspondant à l'objet argument de la méthode ;
- une méthode public (d'instance) nommé **toString** qui redéfinit la méthode homonyme de la classe racine **java.lang.Object** ; cette méthode sans argument et qui retourne un résultat de type **String** est précisée plus loin et n'a pas besoin d'être écrite par vous.

La classe dérivée **TirGalactique** fait partie du package **cms\_ctr3**, est publique et dérive de la classe de base **Tir**. Dans le corps de la classe **TirGalactique**, on définit :

- un champ (propre) privé **g**, sans valeur initiale explicite, qui sert à stocker la valeur réelle de l'accélération gravitationnelle (exprimée en m/s<sup>2</sup>) ;
- une méthode publique « getter » nommée **getG** qui retourne (sans aucune vérification) la valeur du champ privé **g** ;
- une méthode publique « setter » nommée **setG** qui permet la modification de la valeur du champ privé **g** ; cette méthode doit respecter les consignes suivantes :
  - si la valeur de son argument est strictement positive, on stocke dans le champ **g** la valeur de l'argument de la méthode ;
  - autrement, on stocke dans le champ **g** la valeur **9.81** et on affiche le message « **Accélération non valide !** » dans la fenêtre console ;
- un constructeur public avec trois arguments de type réels et qui doit respecter les consignes suivantes :
  - les valeurs du premier et du deuxième argument « sont stockées » dans les champs hérités **alpha** et, respectivement, **v** par un appel explicite au constructeur de la classe de base ;
  - la valeur du troisième argument « est stockée » dans le champ propre **g** par un appel à la méthode « setter » adéquate ;
  - le message « **Un tir galactique** » est affiché dans la fenêtre console ;
- une méthode publique (d'instance) nommée **calculerPortee**, qui n'a pas d'argument et qui retourne la valeur réelle de la portée du tir correspondant à l'objet appelant la méthode ; cette portée est calculée (en mètres) avec la formule :

$$\frac{v^2 \cdot \sin(2 \cdot \alpha)}{g}$$

où l'angle **alpha** est exprimé en radians ;

- une méthode publique (d'instance) nommée **comparer** qui a un seul argument de type **TirGalactique** et qui retourne un résultat de type valeur numérique entière ; plus précisément, après avoir affiché le message « **On est dans l'espace !** » dans la fenêtre console, cette méthode retourne la valeur :
  - **1** si la valeur de l'accélération gravitationnelle correspondant à l'objet appelant la méthode est plus grande que la valeur de l'accélération gravitationnelle correspondant à l'objet argument de la méthode ;
  - **-1** si la valeur de l'accélération gravitationnelle correspondant à l'objet appelant la méthode est plus petite que la valeur de l'accélération gravitationnelle correspondant à l'objet argument de la méthode ;
  - **0** si la valeur de l'accélération gravitationnelle correspondant à l'objet appelant la méthode est égale à la valeur de l'accélération gravitationnelle correspondant à l'objet argument de la méthode ;
- une méthode public (d'instance) nommé **toString** qui redéfinit la méthode homonyme de la classe de base **Tir** ; cette méthode sans argument et qui retourne un résultat de type **String** est précisée plus loin et n'a pas besoin d'être écrite par vous.

On présente ci-dessous le squelette de la classe **Tir** et il faut compléter ce canevas en fonction des indications données en commentaire.

```
//déclaration de package
.....

//en-tête de la classe Tir
.....

{
    //déclaration des champs
    .....

    //définition de la méthode getAlpha
    .....
    .....
    .....
    .....
```

//définition de la méthode **setAlpha**

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

//définition de la méthode **getV**

.....

.....

.....

.....

.....

.....

//définition de la méthode **setV**

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
//définition du constructeur
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

```
//définition de la méthode calculerPortee
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

```
//définition de la méthode comparer
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

```
//redéfinition de la méthode toString  
//rien à ajouter ci-dessous  
public String toString()  
{  
    return "Tir [alpha=" + alpha + ", v=" + v + "];"  
}  
} //fin de la classe Tir
```

On présente ci-dessous le squelette de la classe ***TirGalactique*** et il faut compléter ce canevas en fonction des indications données en commentaire.

```
//déclaration de package
```

```
.....  
.....
```

```
//en-tête de la classe TirGalactique
```

```
.....  
.....
```

```
{
```

```
    //déclaration du champ propre
```

```
.....  
.....  
.....
```

```
    //définition de la méthode getG
```

```
.....  
.....  
.....  
.....  
.....
```

```
    //définition de la méthode setG
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```



```
//définition du constructeur
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

```
//définition de la méthode calculerPortee
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

```
//définition de la méthode comparer
```

```
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

```
//redéfinition de la méthode toString  
//rien à ajouter ci-dessous  
public String toString() {  
    return super.toString() + "    " + "TirGalactique [g=" + g  
    + " ]";  
}  
} //fin de la classe TirGalactique
```

1.2 Dans ce deuxième point du premier sujet, on vous donne le code source de la classe principale *CP\_Ctr3Exo1* qui fait partie du package *cms\_ctr3* et qui utilise les classes *Tir* et *TirGalactique*.

```
package cms_ctr3;

public class CP_Ctr3Exo1
{
    public static void main(String[] args)
    {
        System.out.println("*****");
        Tir tab[] = new Tir[4];
        tab[0] = new TirGalactique(-30, 15, 7);
        System.out.println("-----");
        tab[1] = new Tir(120, 10);
        System.out.println("-----");
        tab[2] = new TirGalactique(30, -15, 12);
        System.out.println("-----");
        tab[3] = new Tir(45, -5);

        System.out.println("*****");
        for(int i = 0; i<tab.length; i++)
        {
            System.out.print("Tab[" + i + "] : ");
            System.out.println(tab[i]);
        }

        System.out.println("*****");
        System.out.println(tab[1].comparer(tab[3]));
        System.out.println(tab[0].comparer(tab[1]));
        System.out.println(
            ((TirGalactique)tab[0]).comparer(tab[1]));
        System.out.println(tab[1].comparer(tab[0]));
        System.out.println(
            tab[1].comparer((TirGalactique)tab[0]));

        System.out.println("*****");
        System.out.println(tab[0].comparer(tab[2]));
        System.out.println(
            ((TirGalactique)tab[0]).comparer(tab[2]));
        System.out.println(
            tab[0].comparer((TirGalactique)tab[2]));
        System.out.println(
            ((TirGalactique)tab[0]).comparer((TirGalactique)tab[2]));
        System.out.println("*****");

        } //fin de la méthode main
    } //fin de la classe principale
}
```

Préciser ci-dessous quels sont les résultats affichés dans la fenêtre console suite à l'exécution de la classe *CP\_Ctr3Exo1*.

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 practice. There are no margins, text, or other markings on the page.

[illegible]

**Sujet no 2.** Choisir et encercler les lettres correspondant aux réponses correctes.

**2.1** Soit le programme suivant :

```
interface IVerifiable
{
    boolean verifier(double arg1, int arg2);
}

class MaClasse implements IVerifiable
{
    //ici on introduit la définition proposée ci-dessous
}
```

Quelles définitions de la méthode verifier, introduites **séparément** à la place du commentaire ci-dessus, produisent une erreur à la compilation ?

- a. `boolean verifier(Object arg1, Object arg2){return true;}`
- b. `int verifier(double arg1, int arg2){return -1;}`
- c. `boolean verifier(double arg1, int arg2){return true;}`
- d. `public boolean verifier(double arg1, int arg2){return false;}`

**2.2** Soit le programme suivant :

```
class A
{
    A faire(boolean arg1, char arg2){return new A();}
}

class B extends A
{
    //ici on introduit la définition proposée ci-dessous
}
```

Quelles définitions de la méthode faire, introduites **séparément** à la place du commentaire ci-dessus, produisent une erreur à la compilation ?

- a. `B faire(boolean arg1, char arg2){return new B();}`
- b. `A faire(boolean arg1, char arg2){return new B();}`
- c. `Object faire(boolean arg1, char arg2){return new A();}`
- d. `Object faire(boolean arg1, char arg2){return new B();}`
- e. `Object faire(boolean arg1, char arg2){return new Object();}`
- f. `Object faire(char arg2, boolean arg1){return new A();}`

**2.3** Soit le programme suivant :

```
class Mere{
    void methode(){ }
}

class Fille extends Mere{
    void methode() { }
}

public class CP{
    static void methode(Mere m, Fille f){
        m.methode();
        f.methode();
    }

    public static void main(String[] args)
    {
        Mere uneMere = new Mere();
        Fille uneFille = new Fille();
        Mere uneMereFille = new Fille();

        //ici on introduit l'appel proposé ci-dessous
    }
}
```

Quels appels de la méthode `methode`, introduits **séparément** à la place du commentaire ci-dessus, produisent une erreur à la compilation ou une exception à l'exécution ?

- a. `methode(uneMere, uneFille);`
- b. `methode(uneMere, uneMere);`
- c. `methode(uneFille, uneFille);`
- d. `methode(uneMereFille, uneMereFille);`
- e. `methode(uneMere, uneMereFille);`
- f. `methode(uneMereFille, (Fille)uneMereFille);`
- g. `methode(uneMereFille, uneFille);`