Examen – Session Principale

Programmation orientée objet

Durée : 1h30 - Documents non autorisés

Nom : **Prénom :** **Classe :**

N.B : Aucune feuille, autre que ces feuilles d'examens, ne sera acceptée en réponse

Questions de cours: (5 pts)

1. Qu'est ce que l'encapsulation ?

L'encapsulation est un principe clé dans la POO qui consiste à regroupe des données (attributs) et un comportement (méthodes) dans une même classe et à réglementer l'accès aux données de l'extérieur.

2. Citer les deux autres concepts de la Programmation Orientée Objet.

Héritage – Abstraction - Polymorphisme

3. Citer au moins 2 différences entre une interface et une classe abstraite

Une interface est une classe dont toutes les méthodes sont abstraites.

Une classe qui implémente une interface doit donner un corps à toutes ses méthodes. Alors qu'une classe qui dérive d'une classe abstraite n'est pas obligée de redéfinir toutes ses méthodes et peut même n'en redéfinir aucune.

Une interface peut être instanciée alors qu'une classe abstraite ne peut qu'être dérivée.

Une classe peut implémenter plusieurs interfaces mais ne peut dériver que d'une seule classe abstraite

4. Un attribut statique est aussi appelé : (entourer la bonne réponse)
   - **a.** variable d'instance
   - **b.** variable de classe
   - **c.** variable d'interface
   - **d.** variable locale
5. Une classe qui implémente une interface... : (entourer la ou les bonnes réponses)
   - **a.** ...est obligatoirement une interface elle aussi
   - **b.** ...est obligatoirement une classe concrète
   - **c.** ...peut être une classe concrète à condition de définir toutes les méthodes de l'interface
   - **d.** ...est obligatoirement une classe concrète si elle définit toutes les méthodes de l'interface

NE RIEN ECRIRE ICI

6.

a. Si une classe B hérite d'une classe A, on dit que :

	Vrai	Faux
B spécialise A	X	
B généralise A		X
B possède au moins tous les champs et les méthodes de A	X	
A possède au moins tous les champs et les méthodes de B		X
Toute instance de B peut être considérée comme un A	X	
Toute instance de A peut être considérée comme un B		X

b. Si les classes Pomme et Orange héritent de la classe Fruit et la classe Golden hérite de la classe Pomme alors on peut écrire :

	Vrai	Faux
i. Fruit [] tab = new Orange[10] ;	X	
ii. Fruit [] tab = new Fruit [10] ;	X	
iii. Golden [] tab = new Pomme[10] ;		X
iv. Golden [] tab = new Orange[10] ;		X
v. Pomme [] tab = new Golden[10]	X	
Parmi les quatre propositions ci-dessus, laquelle permet de créer un tableau pouvant contenir oranges, des pommes et des golden (i, ii, iii, iv ou v)?	Ii	

NE RIEN ECRIRE ICI

Exercice 1 : (5 pts)

1. Donner le résultat d'exécution de ce programme : **(3,5 pts)**

```
class Exercice1 {

public static void
main(String[] args) {
   D d1 = new D();
   D d2 = new D(true);
   E e1 = new E();
   E e2 = new E(false);
   C c = new C();

 d1.m1();
 d2.m1();
 d2.m3();
 e1.m1();
 e1.m3();
 e2.m3();
 c.m2();

System.out.println(d1.n);
System.out.println(d1.b);
System.out.println(d1.i);
System.out.println(d2.n);
System.out.println(d2.b);
System.out.println(d2.i);
System.out.println(e1.n);
System.out.println(e1.b);
System.out.println(e1.o);
System.out.println(e2.n);
System.out.println(e2.b);
System.out.println(e2.o);
System.out.println(c.n);
System.out.println(c.d);

    }
}
```

```
abstract class A {
    static int n = 0;
    A() {  n++;     }
    A(boolean b) {
        if (b)    n++;
      }
}

abstract class B extends A {
    boolean b = false;
    B() { super();  }
    B(boolean b ) {
        super(b);
        n++;
    }
    abstract void m1();
}

class C extends A {
    int d = 0;
    C() { super(true); }
    void m1() {
      if (d == 1)
            d++;
          }
    void m2() {
        if (d == 2)
            d--;
       }
}

class D extends B {
    int i = 1;
    D() {
        super();
        b = false;
    }
```

```
D(boolean b) {
        super(b);
        n++;
        i++;
    }
    void m1() {

        if (b)    b = false;
        else      b = true;

    }
    void m3() {
        i++;
    }
}
class E extends B {
    int o;
    E() {
        super();
        o++;
    }
    E(boolean b) {
        super(b);
        o = 4;
    }
    void m1() {
     o--;

    }
    void m3() {
        o++;
    }
}
```

7

true

1

7

true

3

7

NE RIEN ECRIRE ICI

---

false

1

7

false

5

7

0

2. Dresser, en appliquant le formalisme UML, le diagramme de classes correspondant au programme ci-dessus. **(1,5 pts)**

NE RIEN ECRIRE ICI

Exercice 2 : (10 pts)

Noter bien que : *pour comparer 2 variables s1 et s2 de type String, on utilise if(s1.equals(s2)) … ;*

La classe Robot modélise l'état et le comportement de robots virtuels.
Chaque robot correspond à un objet qui est une instance de cette classe.
Chaque robot :

- a un nom (attribut **nom** : chaîne de caractères)
- a une position : donnée par les attributs entiers **x** et **y**, sachant que x augmente en allant vers l'Est et y augmente en allant vers le Nord,
- a une direction : donnée par l'attribut direction qui prend une des valeurs **"Nord", "Est", "Sud" ou "Ouest"**
- peut avancer d'un pas en avant dans la même direction où il se trouve déjà: avec la méthode sans paramètre **avancer()**
- peut tourner à droite de 90° avec la méthode sans paramètre **droite()** pour changer de direction (si sa direction était "Nord" elle devient "Est", si c'était "Est" elle devient "Sud", etc.). Avec la méthode **droite()**, les robots ne peuvent pas tourner à gauche.
- peut afficher, à travers la méthode **afficher()**, son état en détail

Le nom, la position et la direction d'un robot lui sont donnés au moment de sa création. Le nom est obligatoire mais on peut ne pas spécifier la position et la direction, qui sont définis par défaut à (0,0) et "Est".

1. Écrire les instructions Java qui permettent de définir la classe Robot, en respectant le principe de l'encapsulation des données. **(3 pts)**

```
class Robot{                                                    0, 5
    protected String nom;
    protected int x, y;
    protected String direction;
    public void avancer(){                                      0, 5
            switch(direction){
                    case "Nord": y++; break;
                    case "Est": x++; break;
                    case "Sud": y--; break;
                    case "Ouest": x--;
            }
    }
    public void droite(){                                       0, 5
            switch(direction){
                    case "Nord": direction = "Est"; break;
                    case "Est": direction = "Sud"; break;
                    case "Sud": direction = "Ouest"; break;
                    case "Ouest": direction = "Nord";
```

```
            }
        }
        public void afficher(){
                System.out.println("Le nom du robot est "+nom+" et sa
position est ("+x
                +","+y+") et sa direction est "+direction);
        }

public Robot(String nom){
                this.nom = nom;
                this.x = 0;
                this.y = 0;
                direction = "Est";
        }
public Robot(String nom, int x, int y, String direction)    {
this(nom);
this.x = x;
this.y = y;
this.direction = direction;
}

}
```

0,5

0, 5

0, 5

NE RIEN ECRIRE ICI

---

2. On veut améliorer ces robots en en créant une Nouvelle Génération, les **RobotNG** qui ne remplacent pas les anciens robots mais peuvent cohabiter avec eux.
Les RobotNG savent faire la même chose mais aussi :

- avancer de plusieurs pas en une seule fois grâce à une méthode **avancer()** qui prend en paramètre le nombre de pas
- tourner à gauche de 90° grâce à la méthode **gauche()**
- faire demi-tour grâce à la méthode **demiTour()**

a. Écrire cette nouvelle classe **RobotNG** en la dérivant celle de la première question, sans modifier la classe **Robot** : **(2 pts)**

i. dans un 1er temps, les nouvelles méthodes appellent les anciennes méthodes pour implémenter le nouveau comportement :

- avancer de n pas se fait en avançant de 1 pas n fois,
- « tourner à gauche » se fait en tournant 3 fois à droite,
- faire demi-tour se fait en tournant 2 fois à droite

class RobotNG extends Robot {	0,25
public RobotNG(String nom) { super(nom); }	**0, 5**
public RobotNG(String nom, int x, int y, String direction) {	**0, 5**
super(nom, x, y, direction); }	
public void avancer(int n){ for(int i=0; i<n; i++) avancer(); }	0,25
public void gauche() { for(int i=0; i<3; i++) droite(); }	0,25
public void demiTour(){ droite(); droite(); }	0,25

NE RIEN ECRIRE ICI

ii. Donner une 2ème solution plus efficace qui change directement l'état de l'objet sans faire appel aux anciennes méthodes (ne pas oublier de tenir compte des droits d'accès utilisés !) (**(1,5 pts)**

public void avancer(int pas) {
if (getDirection().equals("Nord"))
setY(getY() + pas);
else if (getDirection().equals("Est"))
setX(getX() + pas);
else if (getDirection().equals("Sud"))
setY(getY() - pas);
else setX(getX() - pas); }
	// **version 1** Avec getters et setters si private

public void avancer(int pas) {
switch(direction){
case "Nord": y+=pas; break;
case "Est": x+=pas; break;
case "Sud": y-=pas; break;
case "Ouest": x-=pas;;
}
}
	// **Version 2** sans getters et setters si protected
	0,5 pt

public void gauche() { switch(direction){
case "Nord": direction = "Ouest"; break;
case "Ouest": direction = "Sud"; break;
case "Sud": direction = "Est"; break;
case "Est": direction = "Nord";
}
}
	0, 5 pt

public void demiTour() { switch(direction){
case "Nord": direction = "Sud"; break;
case "Ouest": direction = "Est"; break;
case "Est": direction = "Ouest"; break;
case "Sud": direction = "Nord";
}
	// usage de setters et getters obligatoire si private
	0, 5 pt

}	

3. On veut mettre ensemble dans un tableau 2 objets de type **Robot** et 2 de type **RobotNG**. **(1,5 pts)**

   a. Comment déclarer le tableau ?

..... Robot[] tab= new Robot[4] ; **0, 5 pt**

.....

   b. Comment remplir le tableau ?

```
..... tab[0] = new Robot("Robot 1"); // ou Robot("Robot 1", 3, 5, "Sud");
      tab[1] = new Robot("Robot 2");
      tab[2] = new RobotNG("RobotNG 1");
      tab[3] = new RobotNG("RobotNG 2");
```

0,5 pt

   c. Comment afficher l'état de tous les robots contenus dans le tableau ?

Version 1 `for (Robot r : tableau)  r.afficher();`

Version 2 `for(int i=0; i<3; i++)  tab[i].afficher();` **0, 5 pt**

   d. Modifier la classe RobotNG pour pouvoir **activer()** un **mode « Turbo »** et le **desactiver()**. Dans ce mode, chaque pas est multiplié par 3 ; redéfinir alors la méthode **avancer()**. L'appel à la méthode **afficher()** devra indiquer à la fin si le robot est en mode Turbo ou pas. Ne pas oublier de modifier le constructeur de **RobotNG** pour intégrer ce nouvel attribut. **(2 pts)**

```
class RobotNG extends Robot {  private boolean modeTurbo; //… }
public void activerTurbo() {modeTurbo = true;}
    public void desactiverTurbo() {modeTurbo = false;}
    public RobotNG(String nom, boolean modeTurbo){ super(nom);
this.modeTurbo = modeTurbo; }
    public void afficher() {
          super.afficher();
          if(modeTurbo) System.out.println("le Mode Turbo est activé");
          else System.out.println("le Mode Turbo est desactivé");
    }
public void avancer(){
          if(modeTurbo)
```

0,5 pt

0,5 pt

0, 5 pt

0, 5 pt

```
                {avancer(3);}
            else
                {super.avancer();}  // ou advancer();
        }
```