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 ?

.....

.....

.....

.....

.....

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

.....

3. Citer au moins 2 différences entre une interface et une 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		
B généralise A		
B possède au moins tous les champs et les méthodes de A		
A possède au moins tous les champs et les méthodes de B		
Toute instance de B peut être considérée comme un A		
Toute instance de A peut être considérée comme un B		

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] ;		
ii. Fruit [] tab = new Fruit [10] ;		
iii. Golden [] tab = new Pomme[10] ;		
iv. Golden [] tab = new Orange[10] ;		
v. Pomme [] tab = new Golden[10]		
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)?		

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++;
    }
}
```

.....

.....

.....

.....

.....

.....

.....

NE RIEN ECRIRE ICI

_____

……

……

……

……

……

……

……

……

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.

……

……

……

……

……

……

……

……

……

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** :

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

……

……

……

……

……

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 !)

3. On veut mettre ensemble dans un tableau 2 objets de type **Robot** et 2 de type **RobotNG**.

a. Comment déclarer le tableau ?

NE RIEN ECRIRE ICI

---

b. Comment remplir le tableau ?

……

……

……

……

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

……

……

……

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.

……	……
……	……
……	……
……	……
……	……
……	……
……	……
……	……
……	……
……	……
……	……
……	……