IFT1025 Programmation 2
Examen Intra, Automne 2015

Professeur : Pascal Vincent

Jeudi 29 octobre 2015, 13h30 - 15h30.

— Prénom :
— Nom :
— Code permanent :
— Programme d'études :

Directives pédagogiques : Seule documentation permise : deux feuilles recto-verso, format letter ($8\frac{1}{2}'' \times 11''$), comprenant votre résumé de cours. Aucun appareil électronique n'est permis (à l'exception d'une montre pour connaître l'heure). L'examen est sur 100 pts. Répondez directement sur la feuille de l'énoncé, avec des réponses brèves mais précises.

1 Classe CouvreChef (20 pts)

Dans les questions suivantes, quand on dit "méthode" sans rien préciser, on veut dire méthode non static. Pour les méthodes static, on précisera "méthode static". **Assurez-vous que les méthodes que vous définissez ou appelez ont exactement le bon nombre de paramètres (ni trop ni pas assez).**

Écrivez une classe **CouvreChef** comportant :

1. une propriété **private** *circonference* de type double, qui contiendra la circonférence du couvre chef en cm.
2. des méthodes accesseur **public getCirconference** et **setCirconference** qui permettront d'accéder à et de modifier la valeur de la propriété circonference
3. une méthode **toString** qui retournera la string "Je suis un couvre chef de ... cm de circonférence" (où les ... sont remplacés par la valeur de la circonférence).
4. un *constructeur* prenant en paramètre une valeur de circonférence initiale
5. un autre *constructeur* qui ne prend pas de paramètre (constructeur par défaut) et qui initialise la circonference à une valeur réelle au hazard entre 20 et 200 (faites appel à java.util.Random.nextDouble() qui retourne un nombre entre 0.0 et 1.0)
6. une méthode **multiplieCirconference** qui permettra de *changer* la circonférence d'un couvre-chef en la multipliant par un facteur (de type double) passé au moment de l'appel
7. une **méthode static multiplieCirconference_st** qui permettra de *changer* la circonférence d'un couvre-chef en la multipliant par un facteur (de type double) passé au moment de l'appel
8. une méthode **memeCirconference** qui permet de comparer les circonference de deux couvre chefs et retournera true si elles sont égales
9. une **méthode static memeCirconference_st** qui permet de comparer les circonference de deux couvre chefs et retournera true si elles sont égales

2 Programme EssaiCouvreChef (20 pts)

Écrivez une classe **EssaiCouvreChef** avec une méthode **main** qui effectuera les opérations suivantes

1. Assigne à une variable *a* un objet de type CouvreChef de circonférence 50cm
2. Assigne à une variable *b* un objet de type CouvreChef dont la circonférence sera la valeur passée comme argument au moment du lancement du programme EssaiCouvreChef (arguments de ligne de commande)
3. Assigne à une variable *c* un objet de type CouvreChef construit avec le constructeur par défaut (qui aura donc une circonference aléatoire)
4. Ajoute 5 à la circonférence du couvre chef *c*
5. Affiche le couvre chef *c* en utilisant la méthode **toString**.
6. Mutiplie la circonférence de *a* par 2 en appelant la méthode **multiplieCirconference**
7. Mutiplie la circonférence de *b* par 2 en appelant la méthode **multiplieCirconference_st**
8. Appelle la méthode **memeCirconference** pour comparer la circonférence de *a* et *b* (et affichera "a et b ont la même circonférence" ou bien "a et b n'ont pas la même circonférence" selon le cas)
9. Appelle la méthode **memeCirconference_st** pour comparer la circonférence de *a* et *b* (et affichera "a et b ont la même circonférence" ou bien "a et b n'ont pas la même circonférence" selon le cas)

3 Objets, références, passage par valeur et par référence (20 pts)

Programme EssaiCouvreChef2

On suppose la classe **CouvreChef** correctement implémentée (en fait seul son constructeur et ses méthodes accesseur **getCirconference** et **setCirconference** seront utilisées).

Exécutez (mentalement) au fur et à mesure le programme EssaiCouvreChef2 suivant, en dessinant et faisant évoluer l'état de la mémoire (pratiquez-vous sur un papier brouillon).

```
public class EssaiCouvreChef2
{
    public static void main(String[] args)
    {
        CouvreChef a = new CouvreChef(10);
        CouvreChef b = new CouvreChef(20);
        CouvreChef c = new CouvreChef(30);

        CouvreChef[] t = new CouvreChef[5];
        t[0] = a;
        t[1] = b;
        t[2] = c;

        a = b;
        b.setCirconference( 10*c.getCirconference() );

        t[3] = new CouvreChef(40);
        t[4] = a;

        System.out.println("Variables: ");
        System.out.println("a: "+a.getCirconference());
        System.out.println("b: "+b.getCirconference());
        System.out.println("c: "+c.getCirconference());

        System.out.println("Tableau:");
        System.out.println("t[0]: "+t[0].getCirconference());
        System.out.println("t[1]: "+t[1].getCirconference());
        System.out.println("t[2]: "+t[2].getCirconference());
        System.out.println("t[3]: "+t[3].getCirconference());
        System.out.println("t[4]: "+t[4].getCirconference());

        System.out.println("Appels de fonctions:");
        ajoute(t[3],2);
        retranche(c,2);
        System.out.println("t[3]: " + t[3].getCirconference());
        System.out.println("c: " + c.getCirconference());
    }

    public static void aj(double circonference, double val)
    {
        System.out.println("circonference avant:"+circonference);
        circonference = circonference+val;
        System.out.println("circonference apres:"+circonference);
    }

    public static void ret(CouvreChef couvre, double val)
    {
        couvre.setCirconference(couvre.getCirconference()-val);
```

```
    }

    public static void ajoute(CouvreChef couv, double val)
    {
        System.out.println("ajoute avant:"+couv.getCirconference());
        aj(couv.getCirconference(), val);
        System.out.println("ajoute apres:"+couv.getCirconference());
    }

    public static void retranche(CouvreChef couv, double val)
    {
        System.out.println("retranche avant:"+couv.getCirconference());
        ret(couv, val);
        System.out.println("retranche apres:"+couv.getCirconference());
    }
}
```

— Ci-dessous, dans la moitié gauche de la page, écrivez au fur et à mesure **tout** ce que le programme affichera (dans l'ordre où il l'affichera).

— Dans la moitié droite, dessinez clairement **l'état de la mémoire à la fin du programme** (juste avant de quitter la fonction main). Pour cela on demande de représenter toutes les cases mémoires utilisées par les variables de la fonction main ainsi que tous les objets CouvreChef créés (dans chacun écrirez la valeur de sa propriété circonference). Représentez les références par des flèches appropriées.

4 Héritage et Polymorphisme

4.1 Écriture de sous-classes (10 pts)

Écrivez les classes suivantes :

ATTENTION : souvenez-vous que circonference est un attribut private de CouvreChef : vous ne pouvez donc pas y accéder *directement* dans les sous-classes.

- Une classe Chapeau, sous-classe de CouvreChef
  - qui n'ajoute pas d'attribut supplémentaire à CouvreChef
  - avec un constructeur prenant en paramètre la circonférence (et avec la contrainte qu'il doit initialiser correctement la circonference, mais **sans appeler explicitement setCirconference**)
  - redéfinit la méthode **toString** pour qu'elle retourne plutôt "Je suis un **chapeau** de ... cm de circonférence" (où les ... sont remplacés par la valeur de la circonférence).
  - définit une méthode **retrecit()** qui réctrécit le chapeau de moitié en multipliant sa circonference par 0.5 (vous pouvez pour cela appeler la méthode multiplieCirconference).
- Une classe Casquette, sous-classe de CouvreChef
  - qui ajoute un attribut **public** *longueur_visiere*
  - avec un constructeur prenant en paramètre la circonférence *et* la longuer de la visière (et avec la contrainte qu'il doit initialiser correctement les attributs de l'objet, mais **sans appeler explicitement setCirconference**)
  - redéfinit la méthode **toString** pour qu'elle retourne plutôt "Je suis une **casquette** de ... cm de circonférence et avec une vivière de ... cm" (où les ... sont remplacés par les bones valeurs).
- Une classe ChapeauMelon, sous-classe de Chapeau
  - qui n'ajoute pas d'attribut supplémentaire
  - avec un constructeur prenant en paramètre la circonférence (et avec la contrainte qu'il doit initialiser correctement la circonference, mais **sans appeler explicitement setCirconference**)
  - redéfinit la méthode **toString** pour qu'elle retourne plutôt "Je suis un **chapeau melon** de ... cm de circonférence" (où les ... sont remplacés par la valeur de la circonférence).

4.2 Polymorphisme : accessibilité des méthodes et exécution des méthodes redéfinies (17 pts)

On suppose que vous avez correctement défini les classes à la question précédente. Et on considère le programme ci-dessous.

Pour chaque ligne de sa fonction **main**, écrivez à droite de la ligne

- EC si cette ligne donnera lieu à une erreur au moment de la compilation (une erreur vous sera signalée par le compilateur, considérez alors qu'on éliminera cette ligne du programme)
- EX si cette ligne donnera lieu à une erreur au moment de l'exécution de votre programme (le programme plantera en exécutant cette ligne là)
- OK si la ligne est correcte et s'exécute mais n'affiche rien
- OK **suivi de ce qui est affiché** en exécutant cette ligne (si la ligne affiche quelquechose)

```
public class EssaiCouvreChef3
{

    public static void bonjour(Chapeau c)
    {
        System.out.println("BONJOUR " + c.toString());
    }

    public static void main(String[] args)
    {
        // ------------------------------------------------------------

        CouvreChef c1 = new CouvreChef(10);

        System.out.println(c1.getCirconference());

        System.out.println(c1.toString());

        bonjour(c1);

        System.out.println( ((Casquette)c1).longueur_visiere);

        c1.retrecit();

        ((Chapeau)c1).retrecit();

        // ------------------------------------------------------------

        Chapeau c2 = new Chapeau(20);

        System.out.println(c2.getCirconference());

        System.out.println(c2.toString());

        System.out.println( ((CouvreChef)c2).toString() );

        bonjour(c2);

        System.out.println( ((Casquette)c2).longueur_visiere);

        c2.retrecit();

        ((CouvreChef)c2).retrecit();
```

```
        // ------------------------------------------------------------
        CouvreChef c3 = new Casquette(30, 5);
        System.out.println(c3.getCirconference());
        System.out.println(c3.toString());
        System.out.println(c3.longueur_visiere);
        bonjour(c3);
        System.out.println(  ((Casquette)c3).longueur_visiere);
        // ------------------------------------------------------------
        Object c4 = new ChapeauMelon(40);
        System.out.println(c4.getCirconference());
        System.out.println(c4.toString());
        bonjour(c4);
        System.out.println(  ((Casquette)c4).longueur_visiere);
        Chapeau c6 = (Chapeau)c4;  System.out.println(c6.toString());
        System.out.println( ((CouvreChef)c4).toString() );
        ChapeauMelon c6prime = c4; System.out.println(c6prime.toString());
        c4.retrecit();
        ((Chapeau)c4).retrecit();
        ChapeauMelon c7 = (ChapeauMelon)c4; c7.retrecit();
        // ------------------------------------------------------------
        ChapeauMelon c4b = new ChapeauMelon(50);
        bonjour(c4b);

    }
}
```

4.3 Tableau de couvre chefs (13pts)

Dans un fichier EssaiCouvreChef4 écrivezles méthodes statis suivantes :

- **afficheCouvreChefs** qui :
  - reçoit en paramètre un tableau de CouvreChef
  - affiche chaque couvre chef du tableau (en faisant appel à son toString)
- **creeCouvreChefAleatoire()** qui :
  - choisit aléatoirement (en tirant aléatoirement un nombre entier avec *java.util.Random.nextInt()*) de créer soit un Chapeau, un ChapeauMelon ou une Casquette
  - choisit aléatoirement les attributs initiaux à donner à cet objet (nombres réels dans une plage de valeur de votre choix)
  - crée l'objet en question
  - le retourne
- **creeTableauDeCouvreChefsAleatoires** qui :
  - prend en paramètre uniquement la taille du tableau de CouvreChef à créer (un nombre entier)
  - crée un tableau de cette taille
  - le remplit de couvre chefs créés aléatoirements (vous pouvez pour cela appeler la méthode que vous avez précédemment définie)
  - retourne le tableau ainsi créé
- **main** qui appelle vos fonctions préalablement définies poue :
  - créer un tableau de couvre chefs aléatoire qui contiendra 10 couvre chefs
  - l'afficher