Jour 3 - POO - Poly

L'ART ABSTRAIT

Job 0

Le programme **AbstractFinal** ci-dessous implémente une petite hiérarchie de 4 classes (A, B, C et D). Il y a trois erreurs dans la méthode main et une erreur dans la classe D. Toutes les erreurs sont dues à une utilisation erronée des modificateurs abstract et final. Expliquez ces erreurs.

```
class AbstractFinal {
    public static void main(String[] args) {
        A[] tab = new A[3];
        tab[0] = new A();
        tab[1] = new B();
        tab[2] = new C();
        tab[1].b = 2;
        ((C)tab[2]).c = 3;
    }
}
```

```
abstract class A {
    int a;
}
class B extends A {
    int b;
}
class C extends A {
    final double c = 1;
}
abstract class D extends A {
    double d;
    int operation(int a) {
        return (a * 2);
    }
    abstract int calcul(int b) {
    }
```

Job 1

Programmer la hiérarchie de classes "**Rectangle coloré héritant de Rectangle**" en obéissant aux contraintes suivantes :

- La classe **Rectangle** possède les attributs double **largeur** et **hauteur**.
- La classe **RectangleColore** hérite de **Rectangle** et possède un attribut **couleur** de type String
- Le code résultant doit pouvoir être testé avec le programme principal suivant :

```
class ToStringEq
{

    public static void main(String[] args)
        {
            System.out.println("Test 1 :");
            Rectangle rect = new Rectangle(12.5, 4.0);
            System.out.println(rect);
            System.out.println();

            System.out.println("Test 2: ");
            // le type de rect1 est RectangleColore
            // l'objet contenu dans rect1 est de type RectangleColore
            RectangleColore rect1 = new RectangleColore(12.5, 4.0, "rouge");
            System.out.println(rect1);
            System.out.println();


            System.out.println("Test 3 :");


            // le type de rect2 est Rectangle
            // l'objet contenu dans rect2 est de type RectangleColore
            Rectangle  rect2 = new RectangleColore(25.0/2, 8.0/2, new String("rouge"));
            System.out.println(rect2);


            System.out.println (rect1.equals(rect2)); // 1.
            System.out.println (rect2.equals(rect1)); // 2.
            System.out.println(rect1.equals(null));   // 3.
            System.out.println (rect.equals(rect1));  // 4.
            System.out.println (rect1.equals(rect));  // 5.
        }
```

Et produire en sortie :

```
Test 1 :
Rectangle :
 largeur = 12.5
 hauteur = 4.0

Test 2:
Rectangle :
 largeur = 12.5
 hauteur = 4.0
 couleur = rouge

Test 3 :
Rectangle :
 largeur = 12.5
 hauteur = 4.0
 couleur = rouge
true
true
false
false
false
```

Les méthodes toString et equals nécessaires ne doivent pas comporter de duplication de code.

Job 2

Vous vous intéressez dans cet exercice à décrire les données d'un jeu simulant des combats de magiciens.

Dans ce jeu, il existe trois types de cartes : **les terrains**, **les créatures** et **les sortilèges**.

Les terrains possèdent **une couleur** (parmi 5 : blanc('B'), bleu ('b'), noir ('n'), rouge ('r') et vert ('v').)

Les créatures possèdent **un nom**, **un nombre de points de dégâts** et **un nombre de points de vie**.

Les sortilèges possèdent **un nom** et **une explication** sous forme de texte.

De plus, chaque carte, indépendamment de son type, possède un **coût**. Celui d'un terrain est 0.

Dans un programme **Magic.java**, proposez (et implémentez) une hiérarchie de classes permettant de représenter des cartes de différents types.

Chaque classe aura un constructeur permettant de spécifier la/les valeurs de ses attributs. De plus, chaque constructeur devra afficher le type de la carte.

Le programme doit utiliser la conception orientée objet et ne doit pas comporter de duplication de code.

Ajoutez ensuite aux cartes une méthode **afficher()** qui, pour toute carte, affiche son coût et la valeur de ses arguments spécifiques.

Créez de plus une classe **Jeu** pour représenter un jeu de cartes, c'est-à-dire une collection de telles cartes.
Cette classe devra avoir une méthode **piocher()** permettant d'ajouter une carte au jeu. On supposera qu'un jeu comporte au plus 10 cartes. Le jeu comportera également une méthode **jouer()** permettant de jouer une carte. Pour simplifier, on jouera les cartes dans l'ordre où elles sont stockées dans le jeu, et on mettra la carte jouée à **null** dans le jeu de cartes.

Pour finir, dans la méthode main, constituez un jeu contenant divers types de cartes et faites afficher le jeu grâce à une méthode afficher propre à cette classe.

Job 3

Un programmeur amateur produit le code suivant pour tester ses connaissances en POO :

```
class Polymorph
{ public static void main(String[] args)

        {
            Forme[] tabFormes =
                {
                    new Cercle("rouge"),
                    new Triangle("jaune")
                };

            Collect formes = new Collect(10);

            // Une collection de formes
            // contenant une copie des objets definis
            // dans le tableau tabFormes
            for (int i = 0; i < tabFormes.length; ++i)
                formes.add(new Forme(tabFormes[i]));
            formes.dessine();
        }
}

class Forme
{
    private String couleur;


    public Forme(String uneCouleur)
        {
            couleur = uneCouleur;
        }
    public Forme(Forme other)
        {
        this.couleur = other.couleur;
        }
    public  void dessine(){
        System.out.println("Une forme " + couleur);
    }
}
```

```
class Triangle extends Forme
{
    public Triangle(String uneCouleur)
        {
            super(uneCouleur);
        }

    public Triangle(Triangle autreTriangle)
        {
            super(autreTriangle);
        }

    public void dessine()
        {
            super.dessine();
            System.out.println("toute pointue");
        }
}


class Cercle extends Forme
{


    public Cercle(String uneCouleur)
        {
            super(uneCouleur);
        }
    public Cercle(Cercle autreCercle)
        {
            super(autreCercle);
        }
    public void dessine()
        {
            super.dessine();
            System.out.println("toute ronde");
        }
}


class Collect
{
    private Forme collect[];

    private int index;


    public Collect(int indexMax)
        {
            collect = new Forme[indexMax];
            index = -1;
        }

    public void add(Forme a)
        {
            if (index < collect.length - 1)
            {
                ++ index;
                collect[index] = a;
            }

        }
    public void dessine()
        {
            for (int i = 0; i <= index; ++i)
                collect[i].dessine();
        }
}
```

Il s'attend à ce que son programme produise l'affichage suivant :

```
Une forme rouge
toute ronde
Une forme jaune
toute pointue
```

Questions:

1. Expliquez pourquoi ce programme principal ne fait pas réellement ce qu'il veut.
2. Qu'affiche-t-il ?
3. Corrigez ce programme de sorte qu'il fasse ce que le programmeur voulait à l'origine. Vous donnerez le code Java de tout élément ajouté en précisant à quelle classe il appartient et indiquerez les éventuelles modifications à apporter à la méthode main.

Les contraintes à respecter sont les suivantes :

- la déclaration de **tabFormes** doit rester inchangée ;
- il ne doit pas y avoir de tests de types dans la boucle qui copie les données de **tabFormes** dans la collection.

Job 4 – Mini-projet

On souhaite créer une application java d'une gestion commerciale.
On vous demande de créer les classes suivantes :

1. **Personne** : classe abstraite qui contient les :
    - attributs protégés : identite, nomSocial et adresse
    - méthodes protégées :
        - personne (int ident, string nomsocial, string adresse) : constructeur paramétré qui permet d'initialiser l'objet Personne
        - les assesseurs (get et set) de chacun des attributs protégés
        - void affiche() : qui renvoie toutes les informations concernant la personne.

2. **Client** : une sous-classe de la classe Personne et qui possède :
    - attribut privé : chiffreAffaire
    - méthodes publiques :
        - client (inti dent, string nomsocial, string adresse, double chiffreAffaire) : constructeur paramétré qui permet d'initialiser l'objet Client
        - les accesseurs publics de l'attribut chiffreAffaire
        - void affiche() : une méthode qui fait appel à la méthode affiche() de la super-classe et qui affiche toutes les informations qui concerne le client

3. **Article** : la classe contient
    - les attributs privés : référence, désignation, prixUnitaire, quantiteStock
    - les méthodes publiques :
        - article(...) : constructeur paramétré qui permet d'initialiser l'objet
        - article(Article a) : constructeur par copie qui permet d'initialiser l'objet Article à partir d'un objet a passé en paramètre
        - les accesseurs publics
        - void affiche() : qui affiche toutes les informations de l'article.

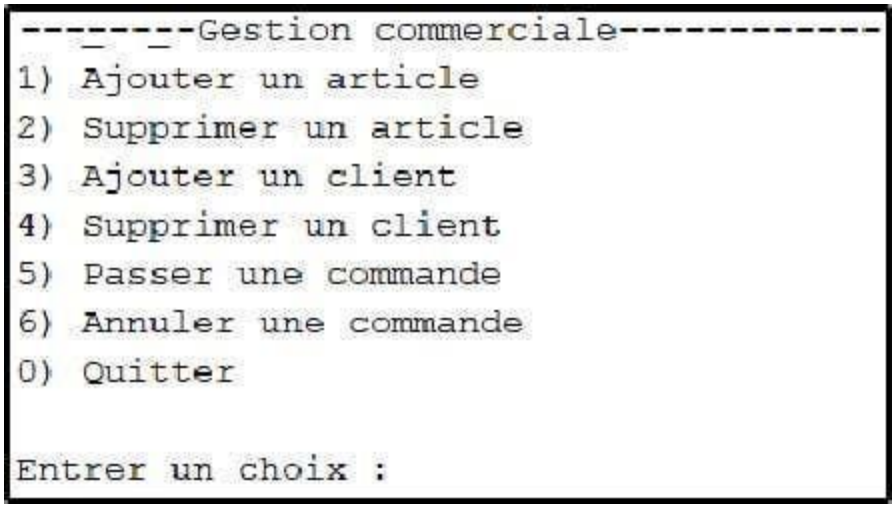
4. **Commande** : une classe qui permet de gérer les commandes des clients :
    - les attributs privés : numeroCommande, dateCommande, client
    - les méthodes publiques :
        - commande(...) : ...
        - les accesseurs publics...

5. **Ligne** : une classe-association qui permet de gérer les lignes de commandes :
    - les attributs privés : commande, article, quantiteCommande
    - les méthodes publiques :
        - ligne(...) : ...
        - les accesseurs publics...

6. **Commerciale** :
    - les attributs de type Vector : articles, clients, commandes, lignes
    La classe java.util.Vector doit être utilisée dans la classe Commerciale
    - les méthodes publiques :
        - void passerCommande(Commande c) : méthode qui permet d'ajouter une commande au vecteur des commandes

- void annulerCommande(Commande c) : méthode qui permet de supprimer une commande du vecteur commandes
- void ajouterArticle(Article a) : add article dans vecteur Articles
- void supprimerArticle(Article a)…
- void ajouterClient(Client c) : add client dans vecteur client
- void supprimerCLient(Client c)…
- static void main(String args[]) : programme principal qui permet d'afficher un menu ci-dessous et qui servira à manipuler la gestion commerciale

NB : la saisie des données nécessite l'utilisation de la classe Scanner



Compétences visées

- → Programmation Orientée Objet (POO)
- → JAVA

Rendu

Le projet est à rendre sur **https://github.com/prenom-nom/runtrackJava**.

Base de connaissances

- Apprendre le JAVA
- La syntaxe de base
- Aide-mémoire JAVA