L3 SRI
Programmation Orientée Objet (session 1, 2019-20)

Durée = 1h30

Documents autorisés. Les réponses non justifiées ne seront pas comptabilisées.

Exercice 1 : Polymorphisme (7 points)

On dispose des 4 classes suivantes :

```
public class A {
    protected char att ;
    public A () {
        att = 'a' ;
    }
    public void affiche() {
        System.out.println("Dans A : "+att) ;
    }
}

public class B extends A {
    public B () {
        att = 'b' ;
    }
    public void affiche() {
        System.out.println("Dans B : "+att+
                " et au dessus "+super.att) ;
    }
}
```

```
public class C extends B {
    public C () {
        att = 'c' ;
    }
    public void affiche() {
        System.out.println("Dans C : "+att+
                " et au dessus "+super.att) ;
    }
}

public class Main {

    public static void main (String arg[]) {
        A tab[] = new A[3] ;
        tab[0] = new A() ;
        tab[1] = new B() ;
        tab[2] = new C() ;
        for (int i = 0 ; i <3 ; i++)
            tab[i].affiche() ;
    }
}
```

1. Donner le résultat obtenu lors de l'exécution de la classe `Main`.
2. Expliquer le(s) différent(s) cas de polymorphisme rencontré(s).
3. Donner un exemple des autres cas de polymorphisme que vous connaissez.

Exercice 2 : Problème (13 points)

On veut faire l'implémentation en Java de l'algorithme glouton de coloration d'un graphe pour un graphe non-orienté (voir algorithme 1).

Algorithm 1: Algorithme glouton pour coloration d'un graphe

Input: G : graphe non orienté
Data: ECD : ensemble des couleurs disponibles
ECV : ensemble des couleurs du voisinage
Δ : ensemble de couleurs
nbC : nombre de couleurs disponibles
s : sommet courant

```
begin
    ECD initialisé à vide;
    ajouter la couleur 1 à ECD;
    nbC initialisé à 1 ;
    for chaque sommet s de G do
        if s n'est pas coloré then
            mettre à jour ECV avec les couleurs des adjacents à s;
            Δ initialisé avec la différence ensembliste ECD \ ECV;
            if Δ est vide then
                incrémenter nbC de 1 ;
                ajouter nbC à ECD ;
                colorer s avec nbC;
            else
                colorer s avec le min de Δ;
```

On considérera qu'un graphe est composé d'une collection de sommets et que chaque sommet est composé :

— de son nom *(hypothèse : les noms sont des entiers numérotés de 1 à n, n étant l'ordre du graphe)*,
— de la collection des noms des sommets adjacents.

Remarquons que la structure de données décrivant le graphe étant dynamique, on ne connaît l'ordre [1] du graphe que lorsqu'il a été construit en intégralité.

1. Donnez la liste des classes qui permettraient de coder un objet `Graphe` et expliquez comment elles sont liées entre elles. Attention, on ne vous demande pas de coder chaque classe !
2. Donnez les attributs des classes que vous avez identifiez, en particulier ceux de la classe `Graphe`.
3. Donnez le constructeur de la classe `Graphe` produisant un graphe vide.
4. Donnez le code Java des méthodes suivantes de la classe `Graphe` (vous respecterez *scrupuleusement* les prototypes indiqués) :
   (a) `public void ajoutSommet(int x)` : fonction permettant d'ajouter le sommet de nom `x` au graphe courant. *Contrainte : le sommet de nom* `x` *ne devrait pas apparaître pas déjà dans le graphe ; s'il y est déjà alors il vous faudra lever une exception.*
   (b) `public void ajoutArete(int x, int y)` : fonction permettant d'ajouter l'arête `(x,y)` dans le graphe courant (donc le sommet de nom `y` en tant que sommet adjacent au sommet de nom `x` et vice-versa). *Hypothèse simplificatrice : vous considérerez que les sommets de nom* `x` *et* `y` *sont déjà dans le graphe. Contrainte : l'arête* `(x,y)` *ne devrait pas apparaître pas déjà dans le graphe ; si elle y est déjà alors il vous faudra lever une exception.*
   (c) `int getNbSom()` : récupération de l'ordre du graphe courant.
   (d) `ArrayList getNomsAdjacents(int x)` : récupération de la liste des noms des sommets adjacents à `x` dans le graphe courant sous la forme d'une `ArrayList`.

1. L'ordre d'un graphe correspond au nombre de sommets de ce graphe.

(e) `public String toString()` : fonction renvoyant la chaîne de caractères donnant le contenu du graphe courant sous la forme de la liste de ses sommets, en donnant entre parenthèses pour chaque sommet la liste de ses adjacents.

5. Proposez une structure de données permettant de mémoriser et d'utiliser de manière efficace les couleurs attribuées. *Hypothèse : les couleurs seront codées sous la forme d'entiers numérotés à partir de 1 et, à ce moment-là, l'ordre du graphe sera connu.*
6. Donnez le code Java de l'algorithme en considérant que vous disposez d'une classe `SetOfInt` avec le constructeur et les méthodes suivants (attention, la classe `SetOfInt` n'est donc pas à coder) :
   — `public SetOfInt()` : création d'un ensemble d'entiers vide,
   — `public void addIntToSetOfInt(int i)` : ajout de l'entier `i` à l'ensemble courant,
   — `public static SetOfInt difference(SetOfInt e1, SetOfInt e2)` : construction de l'ensemble correspondant à la différence ensembliste `e1` $\setminus$ `e2` (les entiers de `e1` qui ne sont pas dans `e2`),
   — `public int min()` : récupération du minimum des entiers stockés dans l'ensemble courant,
   — `public boolean isEmpty()` : renvoie `true` si l'ensemble courant est vide et `false` sinon.

Annexe

Pour la classe `Exception`	
`Exception(String s)`	crée une exception (s étant le message associé)
`String getMessage()`	récupère le message associé à l'exception
Pour la classe `ArrayList<T>`	
`int size()`	retourne la taille de la collection
`boolean add(T x)`	ajoute `x` en fin de la collection
`void set(int i, T x)`	modifie le ième élément avec la valeur `x`
`T get(int i)`	retourne le ième élément