Programmation & Algorithmique 2

TP - Séance 5

22 mars 2023

Matière visée : Interfaces et Héritage

1 Interfaces

1.1 MyBinaryInteger

Créez une classe `MyBinaryInteger` qui implémente l'interface `Comparable` de Java. Un objet de cette classe sert à représenter un entier. Le constructeur prend un entier en paramètre.

La comparaison de deux objets `MyBinaryInteger` est définie comme suit :

> Soient O_1 et O_2 deux objets `MyBinaryInteger`. Soient b_1 le nombre de bits à 1 de O_1 et b_2 le nombre de bits à 1 de O_2. Nous avons que :
> — $O_1 < O_2$ si et seulement si $b_1 < b_2$;
> — $O_1 > O_2$ si et seulement si $b_1 > b_2$;
> — $O_1 = O_2$ si et seulement si $b_1 = b_2$;

Implémentez également la méthode `String ToString()` qui retourne une représentation de l'entier en base 2.

1.1.1 Testeur

Créez un testeur pour votre classe `MyBinaryInteger`. Ce testeur doit :

1. demander à l'utilisateur d'entrer un entier i (utilisez la classe `Scanner`, `Scanner.nextInt`) ;
2. créer un tableau de `Comparable` de taille i ;
3. remplir le tableau d'objets `MyBinaryInteger` dont les entiers doivent être entrés par l'utilisateur ;
4. utiliser la méthode `sort` de la classe `Arrays` pour trier le tableau ;
5. imprimer les `MyBinaryInteger` dans l'ordre du tableau trié pour vérifier si le tri s'est bien réalisé.

1.2 Feu de signalisation

Dans cet exercice, nous allons représenter un feu de signalisation qui peut accueillir une file de véhicules.

1.2.1 Les classes et les interfaces

Nous allons considérer des objets `Car` et `Bicycle`. Une voiture (`Car`) a comme caractéristiques une plaque d'immatriculation (`licensePlate`, `String`) et une marque (`brand`, `String`). Un vélo (`Bicycle`) a comme caractéristiques une couleur (`color`, `String`) et une marque (`brand`, `String`). Veuillez redéfinir la méthode `toString()` pour qu'elle renseigne les caractéristiques des différents véhicules.

Veuillez créer une interface `Lineable` qui aura pour unique méthode `boolean canPass()`. Le rôle de cette méthode sera de retourner `true` si l'objet `Lineable` a l'autorisation de dépasser, `false` sinon.

Faites en sorte que `Car` et `Bicycle` implémentent l'interface `Lineable` en sachant qu'une voiture ne peut pas dépasser, mais qu'un vélo bien.

Veuillez construire une classe `TrafficLight` (feu de signalisation). Un objet de cette classe sera caractérisé par une file de véhicules qui attendent de pouvoir passer. Veuillez redéfinir la méthode `toString()` pour qu'elle retourne une représentation de la file d'attente. On doit pouvoir ajouter des véhicules à la file grâce à la méthode `add`. Cette méthode aura le comportement suivant :
lorsqu'un véhicule arrive, s'il ne peut pas dépasser, il se place à la fin de la file ; s'il peut dépasser, alors il dépasse tous les véhicules qui ne peuvent pas dépasser qui attendent déjà au feu. Un véhicule qui peut dépasser ne peut pas dépasser un véhicule qui peut également dépasser.

1.2.2 Le testeur

Veuillez créer une classe testeur qui va créer un objet `TrafficLight` et lui ajouter quelques véhicules (`Car` et `Bicycle`). Affichez ensuite l'état de la file pour vérifier que les véhicules ont respecté les règles de dépassement énoncées.

2 Réflexion sur l'héritage

Étant données les classes `A`, `B` et `C` dont le schéma d'héritage est représenté en figure 2, prédisez (donc sans exécuter le code) ce qui va être affiché en console si l'on exécute le fichier `Heritage.java` (disponible sur Moodle).

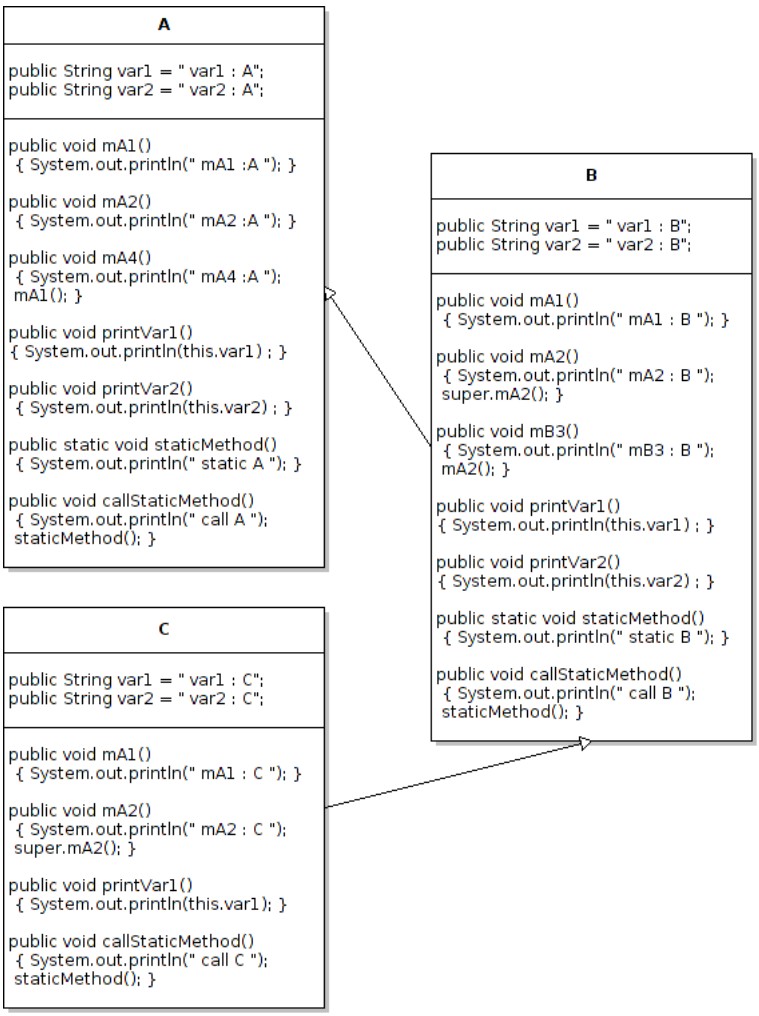

FIGURE 1 – Schéma d'héritage des classes A, B et C.

3 Héritage

3.1 Salles d'une école

Nous voulons représenter en Java deux sortes de salles : des salles de cours et des salles d'informatique. Toutes les salles sont décrites par les informations suivantes :

— un numéro.
— une capacité, c'est à dire un nombre de places.
— un type de tableau (décrit par une chaîne de caractères, "blanc" ou "noir").

Les salles de cours sont caractérisées par :

— la présence ou non d'un écran.
— la présence ou non d'un rétroprojecteur.

Les salles d'informatique sont caractérisées par :

— un nombre de postes de travail.
— le type des postes de travail de cette salle (décrit par une chaîne de caractères, par exemple "terminaux X", "pc sous linux" ou "macintosh").

Par exemple, pour une salle de cours, on aura :

Type de salle : salle de cours
Numéro : 12
Capacité : 60
Type de tableau : blanc
Ecran fixe : oui
Rétroprojecteur fixe : non

et pour une salle informatique, on aura :

Type de salle : salle d'informatique
Numéro : 14
Capacité : 20
Type de tableau : blanc
Nombre de postes : 12
Type de postes : pc sous linux

Veuillez maintenant résoudre les problèmes suivants :

1. Faites un schéma indiquant la (les) classe(s) que vous allez créer avec leur nom et leurs attributs. Précisez bien les relations d'héritage qu'il existe entre ces différentes classes.
2. Ecrivez l'implémentation de la ou des classes proposée(s) à la question 1 en y faisant figurer seulement les attributs.
3. Ecrivez un constructeur par défaut et un constructeur prenant des paramètres pour chaque classe.
4. Ecrivez une méthode retournant le nom de la classe, sous forme de chaîne de caractères.
5. Ecrivez les méthodes permettant de modifier ou de renvoyer les valeurs des attributs pour la ou les classes que vous avez écrites.
6. Ecrivez une méthode `public String toString()` pour chaque classe permettant de retourner une chaîne de caractères contenant les valeurs des attributs d'une instance de classe.
7. Tester dans un main toutes les méthodes que vous venez d'écrire.