TD n°7

TD sur machine, héritage

Exercice 1 La commande **javap** permet de désassembler le code produit par le compilateur **javac**. Cela peut vous permettre d'observer concrêtement ce que produit implicitement la compilation.

Par exemple, si vous définissez une classe qui n'a aucun constructeur, nous avons vu en cours que la compilation avec **javac** générait un constructeur par défaut.

De même, si dans le constructeur d'une sous-classe vous n'effectuez aucun appel explicite au constructeur de la super-classe, le code produit par la compilation de la sous-classe pour le constructeur comprendra comme première instruction l'appel du constructeur sans paramètre de la super-classe.

1. Ecrivez le code suivant, issu du TD 6. Sauvegardez dans un fichier, et compilez-le avec **javac**.

```
class Ex1b{
    int a ;
}

class Ex1b2 extends Ex1b{
    int b ;
    Ex1b2(int a , int b){
        this.a = a ;
        this.b = b ;
    }
}
```

2. Désassemblez le fichier Ex1b.class produit par la compilation en tapant **javap Ex1b**. Vous obtenez la description de la classe Ex1b telle qu'elle a été compilée par **javac**. De quelle classe Ex1b est-elle implicitement une sous-classe ? Repérez le prototype du constructeur par défaut.
3. De même, désassemblez le fichier Ex1b2.class. Utilisez ensuite la commande **javap -c Ex1b2**, ce qui vous permettra d'obtenir le détail du code exécuté par le constructeur, sous forme d'instructions élémentaires de la machine virtuelle. Il n'est pas nécessaire que vous compreniez le détail de toutes ces instructions. Quelle ligne vous semble être l'invocation du constructeur par défaut de Ex1b dans le constructeur de Ex1b2 ?
4. Que pouvez vous en déduire sur une des toutes premières instructions du constructeur de Ex1b ? Vérifiez votre hypothèse à l'aide de la commande **javap -c Ex1b**.

Exercice 2 Cet exercice est une reformulation du deuxième exercice du TD 6.

1. Ecrivez une classe `Figure`. Cette classe a des attributs privés abscisse et ordonnee, ainsi qu'une couleur. Les couleurs seront représentées par des objets de la classe `Color` que vous pouvez utiliser à condition de mettre en tête de votre fichier : `import java.awt.Color;`. Le constructeur de la classe `Figure` instanciera chaque attribut privé. La méthode `toString()` permettra de donner une représentation de l'objet sous forme de chaîne de caractères. Vous regarderez la documentation de la classe `java.awt.Color` pour choisir la constante correspondant à votre couleur préférée pour faire un test de votre classe.
2. Ecrivez deux classes `Carre` et `Rectangle` qui héritent de `Figure`. `Carre` a un attribut côté et `Rectangle` a des attributs hauteur et longueur. Comme d'habitude, testez vos classes.
3. Ajoutez à `Figure` une variable de classe privée `Vector` qui devra référencer **toutes les instances** de sa classe et de ses sous classes. Comment allez vous remplir ce vecteur ? Vous définirez également une méthode publique `getInstances()` qui permettra d'accéder à ce vecteur.
4. Dans `Carre` et `Rectangle`, redéfinissez cette méthode `getInstances()` de manière à ne récupérer que les instances qui correspondent à leur type. Il n'est pas nécessaire d'ajouter un attribut à ces classes ou de modifier leur constructeur.

Exercice 3 [Evaluateur d'expressions booléennes] **Pour ceux qui ont encore du temps** L'objectif de cet exercice est d'écrire un programme prenant une expression booléennes en notation postfixe en argument, affichant l'expression en ordre infixe (avec parenthèses) puis affichant le résultat de l'évaluation de cette expression.
Par exemple :

```
java Eval 1 0 + 1 .
```

affichera :

```
(true.(false+true)) = true
```

On utilise des booléens comme valeurs et les opérateurs sont : +(ou), .(et) et -(non). On définira une interface Expr ainsi que les classes suivantes : `Eval`, `OpBin`, `OpUn`, `Ou`, `Et`, `Non` et `MyBoolean`. Pour la pile, on pourra utiliser la classe prédéfinie `Stack`.

1. Ecrire le graphe d'héritage des classes demandées.
2. Ecrire l'interface `Expr` qui spécifie deux méthodes : `public boolean eval()` et `public String toString()`.
3. Ecrire la classe `MyBooleen` qui encapsule un booléen et implémente l'interface `Expr`. Pour cette classe comme pour les suivantes, il faudra écrire le(s) constructeur(s) dont vous avez besoin.
4. Ecrire les classes abstraites `OpBin` et `OpUn` qui contiennent les champs, constructeurs et méthodes communes respectivement aux opérateurs binaires et unaires.
5. Ecrire les classes `Ou`, `Et` et `Non`.
6. Ecrire la classe `Eval` qui contiendra la méthode `main` et tout ce qui est nécessaire pour construire un arbre à partir des arguments de la ligne de commande.