

Introspection et annotations

Exercice 1 *L'ArrayList mystère*

Vous disposez d'une classe dans laquelle l'un des attributs est une `ArrayList` (brute, non paramétrée) qui est créée puis remplie, mais vous ne savez pas de quoi !

Question 1. Écrivez une méthode retournant pour un indice `i` les méthodes dont vous disposez sur l'objet de rang `i`.

Question 2. Écrivez une méthode retournant la classe la plus spécifique dont tous les objets de l'`ArrayList` sont instances. Notez que la méthode de la classe `Class` `public Class getSuperclass()` retourne la superclasse de la `Class` appelante.

Exercice 2 *Fabrique de personnages*

Question 3. Ecrivez 3 classes représentant des personnages d'un jeu fictif : une classe abstraite `Personnage` introduisant entre autres un nom, un nombre de points et un nombre de vies ; une sous-classe `PersonnageInvisible` (les personnages invisibles peuvent se rendre invisible un certain temps, puis ils redeviennent visibles pendant un temps minimum) introduisant un booléen spécifiant s'ils sont visibles ou pas, le temps pendant lequel ils peuvent rester invisibles et le temps d'attente avant de pouvoir redevenir invisible, ainsi qu'une méthode permettant de se rendre invisible ; et une autre sous classe représentant les personnages bonus (ces personnages gagnent des vies supplémentaires quand ils franchissent certains seuils de points) qui introduisent tous les combien de points le nombre de vies est augmenté (avec redéfinition de l'accesseur en écriture sur les points qui met à jour le cas échéant le nombre de vies). On introduira les accesseurs et les constructeurs paramétrés et par défaut.

Question 4. Créez une classe fabrique de personnages, dont une méthode permet de prendre en paramètre un nom de classe de personnage (`PersonnageInvisible` ou `PersonnageBonus`) et qui retourne une instance de cette classe en utilisant le constructeur par défaut. On utilisera bien sûr l'introspection.

Question 5. Créez une autre méthode qui prend comme paramètre une instance de personnage, puis demande interactivement à un utilisateur de la fabrique les valeurs des attributs de cette instance. Proposez une version avec introspection puis en utilisant la liaison dynamique (en définissant une méthode d'initialisation dans les classes personnage). Notez que la méthode de la classe `Class` `public Field[] getFields()` retourne tous les attributs publics de la `Class` appelante. Dans la classe `Field` les méthodes `public String getName()` et `public void set(Object objet, Object valeur)` permettent respectivement d'obtenir le nom de l'attribut et d'en changer la valeur pour un objet donné.)

Exercice 3 *Migration de personnages*

Question 6. Dans la fabrique de personnages, ajoutez une méthode qui prenne comme paramètre une instance de l'une des classes de personnages (classe origine), le nom d'une autre classe dans laquelle on veut transformer le personnage (classe cible) et retourne un objet de la classe cible dont tous les attributs ont été initialisés :

- pour les attributs communs (de même nom) entre les deux classes, la valeur est recopiée
- pour les attributs de la classe cible que n'a pas la classe origine, la valeur est demandée lors d'une interaction avec l'utilisateur de la fabrique.

Exercice 4 *Annotation Todo*

Reprenez l'annotation `Todo` portant sur les méthodes, utilisable à l'exécution, dont les éléments précisent :

- le type de la tâche à effectuer sur la méthode (écrire, améliorer la complexité, refactoriser, tester)
- le numéro de la version pour laquelle ce doit être effectué
- la durée estimée de la tâche

Question 7. Annotez vos classes personnage avec cette annotation.

Question 8. Écrivez une classe permettant de manipuler ces annotations avec des méthodes permettant, pour un ensemble de classes incluses dans un tableau :

- de lister toutes les annotations `Todo`,
- de récupérer toutes les annotations correspondant à un certain type de tâche passé en paramètre
- de retourner la durée cumulée des tâches à réaliser lors d'une certaine version.

Exercice 5 *Une méthode `toString()` générique (F. Mallet)*

Question 9. Utilisez l'introspection pour créer une méthode `toString()` générique. Cette méthode `toString` prend en paramètre un objet de type `Object` et affiche la valeur de chacun de ses champs. Attention, si les champs sont des références sur d'autres objets on descendra en profondeur pour afficher "récursivement" leurs valeurs également.

Question 10. Proposez une variante de la méthode `toString` admettant un deuxième paramètre qui est la profondeur à laquelle on souhaite descendre dans la structure. Notez la méthode de la classe `Class` `public boolean isPrimitive()` permet de vérifier si le type de l'élément étudié est un type primitif).

Exercice 6 *Implémentation partielle du lookup de méthodes*

Question 11. Écrivez un programme qui reçoit en entrée le nom d'une méthode et l'objet sur lequel elle doit être invoquée (objet receveur), et qui vérifie si dans la classe de l'objet cette méthode existe (vous vérifierez que l'objet n'est pas null). On utilisera la méthode `getDeclaredMethods()` de la classe `Class`. Si le programme ne la trouve pas dans la classe, il va la chercher dans la superclasse, et ainsi de suite. Si celle-ci n'existe pas, le programme devra lever une exception `MethodeInexistanteException`. Sinon, il invoquera la méthode sur l'objet.

Pour commencer, prenez en compte des méthodes sans paramètres. Il faudra vérifier ici que la méthode est accessible avant de l'invoquer (méthode publique pour simplifier), sinon il faudra lever une exception de type `MethodeNonAccessibleException`.

Exercice 7 *Injecteur de dépendances*

Soient les classes suivantes :

```
class A { B b;
    public void m() {System.out.println("Je suis m dans A."); b.n();}
}
class B { public void n() {System.out.println("Je suis n dans B");} }
```

Question 12. Si l'on écrit dans un main le code suivant, que se passe-t-il ? `A a = new A(); a.m();`

Afin de résoudre le problème souligné ci-dessus, et au lieu d'initialiser l'attribut `b` avec une instance de `B` dans le code, nous allons faire ça depuis un programme externe, que nous devons écrire et qu'on va appeler injecteur de dépendances. Ce genre d'outils existent réellement, comme dans l'édition Java EE (Java Enterprise Edition).

Question 13. Définissez une annotation `InjectMe` portant sur les attributs et utilisable à l'exécution. Celle-ci sera utilisée pour annoter les attributs qui doivent être initialisés depuis l'extérieur (par l'injecteur).

Question 14. Écrivez un programme permettant de prendre en entrée l'instance d'une classe ayant des attributs non initialisés et annotés `InjectMe`. Il va ensuite récupérer les types de ces attributs annotés, qui doivent être des classes, instancier ces classes et affecter les références des instances créées aux attributs. Nous allons simplifier le problème, en considérant que les types des attributs annotés sont des classes qui sont référencées dans le `CLASSPATH` lors de l'exécution de votre programme (elles sont donc accessibles depuis votre programme, il suffit de les instancier). Nous allons considérer également que ces classes comportent des constructeurs sans paramètres.