Programmation et modélisation orientées objet
(USAL5A & USAL2H)

Généricité

Dans ce TP, nous souhaitons modéliser des classes qui ont la même structure, modulo le type[1] de leurs composants. Depuis la version 5 de Java, cette modélisation est facilitée par l'introduction de la généricité dans la langage : une classe (ou un interface) peut maintenant être paramétrée par un type. Le types effectifs sont seulement précisés lors de la création d'une instance (mot-clé `new`).

La généricité permet de factoriser du code. Les collections, par exemple, sont naturellement génériques : une liste d'éléments se comporte en effet de la même manière, indépendamment du type de ces éléments.

Nous allons voir ci-dessous les avantages et inconvénients liés à l'usage des génériques, en partant de deux exemples : une classe `Person` et une classe `Point`.

1 Classe `Person`

La classe `Person` est définie ainsi :

```
class Person {
    String name;
    Integer age;
    ...
}
```

1. Utiliser la génération de code intégrée à l'IDE pour obtenir :
   - les *getters/setters* pour les attributs ci-dessus,
   - la méthode `toString()`.
2. Compléter la méthode `main` de la classe `Main` de manière à créer, puis initialiser (avec les *setters*), observer (avec les *getters*) et enfin afficher une `Person`.

2 Classe `Point`

La classe `Point` est définie ainsi :

```
class Point {
    Integer x;
    Integer y;
    ...
}
```

1. Utiliser la génération de code intégrée à l'IDE pour obtenir :
   - les *getters/setters* pour les attributs ci-dessus,
   - la méthode `toString()`.
2. Compléter la méthode `main` de la classe `Main` de manière à créer, puis initialiser (avec les *setters*), observer (avec les *getters*) et enfin afficher une `Point`.

1. le mot « type » fait référence ici indifféremment à la fois aux types primitifs, aux classes et aux interfaces.

3 Classe `Pair<A,B>`

Dans un nouveau sous-package, ajouter la classe générique `Pair` est définie ainsi :

```
class Pair<A,B> {
    A fst;
    B snd;
    ...
}
```

1. Utiliser la génération de code intégrée à l'IDE pour obtenir :
   - les *getters/setters* pour les attributs ci-dessus,
   - la méthode `toString()`.
2. Redéfinir la classe `Person` ainsi :

   ```
   class Person extends Pair<String, Integer> {}
   ```

   Redéfinir la classe `Point` ainsi :

   ```
   class Point extends Pair<Integer, Integer> {}
   ```

   Remarque : ces redéfinitions ne sont pas nécessaires, on peut utiliser directement `Pair<String, Integer>` et `Pair<Integer, Integer>`, mais elles sont parfois préférables car le nommage est explicite. Elles sont toutefois indispensables si on souhaite les étendre avec de nouvelles méthodes.
3. Vérifier que la méthode `main` compile et fonctionne toujours.

4 Compilation de la généricité

La généricité peut être compilée par remplacement (comme pour les *templates* de C++), ou par effacement (comme en Java). Nous allons maintenant simuler « à la main » les deux types de compilation.

4.1 Compilation par *inlining*

Dans un nouveau sous-package :

1. Dupliquer la classe `Pair<A,B>`, puis en utilisant les fonctions de l'éditeur, substituer `A` par `String` et `B` par `Integer`. Renommer ensuite `Pair<String, Integer>` en `Person`.
2. Dupliquer à nouveau la classe `Pair<A,B>`, puis en utilisant les fonctions de l'éditeur, substituer `A` par `Integer` et `B` par `Integer`. Renommer ensuite `Pair<Integer, Integer>` en `Point`.
3. Vérifier que la méthode `main` compile et fonctionne toujours.

4.2 Compilation par effacement (*erasure*)

Dans un nouveau sous-package :

1. Dupliquer la classe `Pair<A,B>`, puis en utilisant les fonctions de l'éditeur, substituer `A` par `Object` et `B` par `Object`. Renommer ensuite `Pair<Object, Object>` en `Pair`.
2. Redéfinir la classe `Person` ainsi :

   ```
   class Person extends Pair {}
   ```

   Redéfinir la classe `Point` ainsi :

   ```
   class Point extends Pair {}
   ```
3. Modifier la méthode `main` en ajoutant les *casts* nécessaires (réclamés par le compilateur et l'IDE). Vérifier que la méthode ainsi modifiée compile et fonctionne à nouveau.