TP 5 - Interfaces, abstraction

Objectif pédagogique

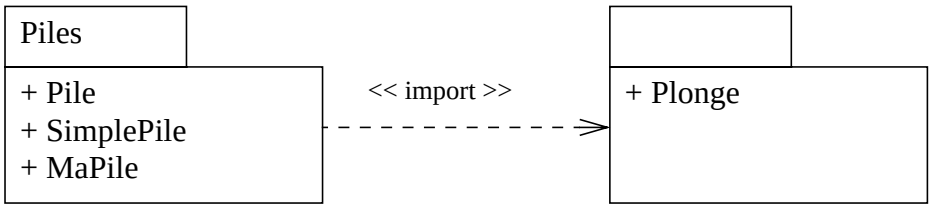
Utilisation d'une interface pour abstraire et masquer la mise en œuvre, regroupement en *package*.

Prérequis

TP 1 à 4 et notes du cours d'initiation à la POO.

Sujet

Nous allons créer un **package** `Piles` dans lequel nous mettrons l'**interface** `Pile` et **deux mises en œuvre** de `Pile` : `SimplePile` (fournie en .class) et `MaPile` que vous devrez écrire. Ce package sera ensuite utilisé par une classe de test fournie, `Plonge` :



Note sur la mise en œuvre des packages et les notions de visibilité :

Un package est un ensemble de classes et d'interfaces qui doivent être regroupées dans un même répertoire. Le répertoire doit posséder le même nom que le package. Toute classe ou interface du package doit stipuler `package <nom du package>;` comme première instruction.

Si on veut utiliser une classe qui se trouve dans le package `Piles` à l'extérieur du package, il faut :

- soit utiliser `import` en préambule du fichier qui contient le code qui utilise ce package :
```
import Piles.*;
…
SimplePile wonder = new SimplePile();
```
- soit référencer explicitement le package pour toute utilisation d'un de ses membres :
```
Piles.SimplePile wonder = new Piles.SimplePile();
```

L'intérêt de regrouper des classes dans un package est de pouvoir rendre visibles des méthodes (et attributs) à l'intérieur du package, mais pas d'un usage extérieur. Ainsi, par défaut toute méthode ni privée, ni protégée, ni publique n'est visible que dans le package courant. De plus, cela permet d'utiliser correctement javadoc qui structure la documentation en packages.

Pour compiler les classes d'un package il suffit de se mettre dans le répertoire contenant le package et de lancer `javac` sur chaque classe du package. Pour générer la documentation du package, il suffit de se mettre au niveau du dessus et taper `javadoc -d doc <nom du package>.`

ON RECOPIERA D'ABORD TOUT LE RÉPERTOIRE du TP5 dans son répertoire personnel afin de recupérer une structuration satisfaisante des fichiers à modifier.

Q0) Définir l'interface Pile correspondant à la documentation Javadoc disponible

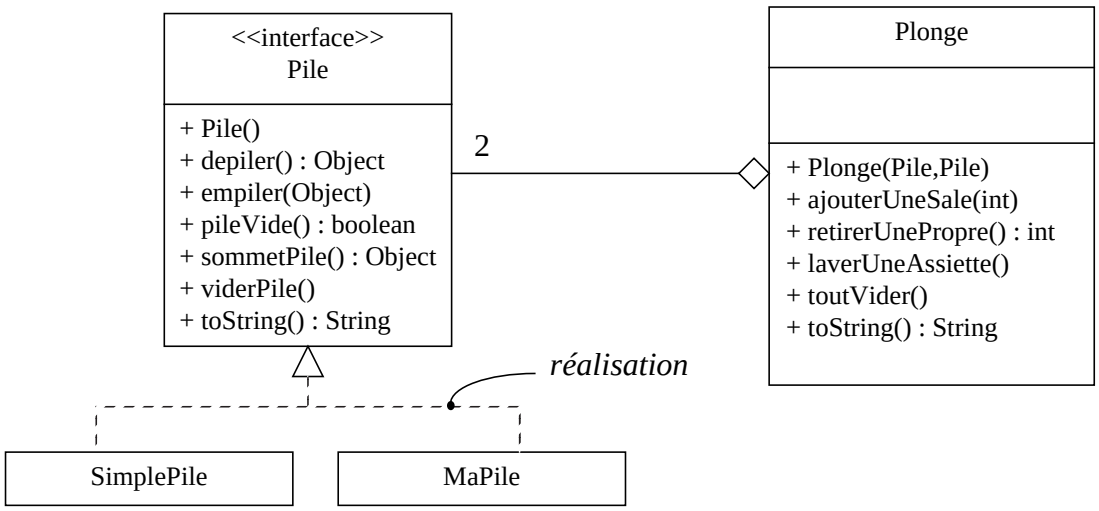
Le squelette de l'interface est donné (`Pile.java`).

Q1) Tester le programme Plonge.java avec la mise en œuvre SimplePile fournie

Votre fichier `Plonge.java` doit se trouver dans un répertoire différent de celui du package `Piles`.

Q2) Définir votre mise en œuvre de la Pile en utilisant un simple tableau

La mise en œuvre `SimplePile` ne fait aucun contrôle de débordement avant d'empiler, ni de vérification que la pile n'est pas vide avant de dépiler.
Écrivez une version de `Pile` dans `MaPile.java` (du package `Piles`). Cette mise en œuvre devra tester les erreurs.
Modifier et utiliser le programme `Plonge` pour vérifier votre mise en œuvre (1 seule ligne à changer !)



Notes sur la notion de pile

Une pile est une structure de données dont nous verrons une définition formelle en cours de SDD. Pour ce TP il suffit de savoir que les données sont sorties de la pile dans l'ordre inverse de leur entrée (LIFO : « *Last In First Out* »). Les opérations sur cette structure sont les suivantes :

`viderPile()` qui enlève tous les objets de la pile
`empiler(élément)` qui met l'élément en sommet de pile.
`depiler()` qui enlève l'élément du sommet de pile et le rend en résultat.
`sommetPile()` qui rend l'item du sommet de pile sans l'enlever.
`pileVide()` qui dit si la pile est vide ou non.

Les piles sont faciles à représenter par un tableau et un « pointeur de sommet » qui repère l'indice de la dernière valeur entrée.

Remarques

La mise en œuvre de `SimplePile` (`SimplePile.class`) ainsi que le squelette de l'interface `Pile` se trouvent sous `/home/clyde/d1/insa3/commun/java-sdd/tp5`