

TP de PPO en JAVA

Polytech Lille GIS4 2018-2019

Objectifs : Collections (`java.util`) et interfaces abstraites. Packages et encapsulation.

1 Collections et interfaces

1.1 Bibliotheque V1

Nous partons de l'exemple de la bibliothèque d'ouvrages vu en cours. Le squelette des classes (`Ouvrage`, `Bibliotheque`, exceptions et `Application`) est fourni dans :

`~bcarre/public/tpBib`

copiez ce répertoire de travail sur votre compte.

Compléter ces classes selon les indications fournies, notamment la prise en compte des exceptions `OuvrageInconnuException` et `NonDisponibleException`.

Dans la classe principale `Application` (`main`), donner la possibilité d'emprunter et de retourner un ouvrage en demandant son code à l'utilisateur.

Tester.

1.2 Revues

La bibliothèque gère également des revues hebdomadaires. Les revues sont des ouvrages particuliers :

- ils ont en plus un champ `date` d'édition (pour simplifier : `int` de la forme AAAAMMJJ) et `numero(String)`
- l'opération `emprunter` est contrainte : une revue ne peut être empruntée qu'une semaine après sa date d'édition (pour simplifier : `date + 7`) sinon elle est considérée comme non disponible. Pour simplifier, ranger la date courante "en dur" comme `static` dans la classe `Bibliotheque`.

Créer la classe `Revue`. Modifier `Application` pour créer quelques revues et tester.

1.3 Bibliotheque V2

Ajouter les 2 méthodes suivantes à la classe `Bibliotheque` :

- `getOuvrages()` : renvoie la liste de tous les ouvrages (dont les revues) triés par nombre d'emprunts,
- `getReuves()` : renvoie la liste des revues triées par titre et date d'édition.

Afficher ces listes dans `Application`.

2 Packages et encapsulation

2.1 Packaging de base

Créer un package `lib` (et donc un répertoire de même nom) et y ranger toutes les classes, sauf `Application`.

1. Encapsulation :
 - rendre `public` les classes de `lib`
 - ajuster les modifieurs de visibilité des variables d’instance et des méthodes des classes de `lib` pour n’exporter (`public`) que les fonctionnalités offertes à l’utilisation de ce package.
2. Compiler.

Rappel : les packages doivent être accessibles soit dans le `CLASSPATH` (ou le paramètre `-cp` des commandes `javac` et `java`), soit plus simplement à partir du répertoire de travail `'.'`.

Dans ce dernier cas, il suffit de compiler en vous plaçant dans le répertoire père de `lib`, par exemple :

```
tpBib> ls -l
drwxr-xr-x 2 dupond gis4 216 Nov 2 2016 lib
tpBib> ls lib
Bibliotheque.java
NonDisponibleException.java
...
tpBib> javac lib/*.java
tpBib> ls lib
Bibliotheque.java
Bibliotheque.class
NonDisponibleException.java
NonDisponibleException.class
...
```
3. Créer un package `applications` (et donc un répertoire de même nom, au même niveau que `lib`) et y ranger la classe `Application`.
4. Compiler et tester.

Rappel : même remarque que précédemment pour le package `Application`, pour l’exécution faites :

```
tpBib> ls -l
drwxr-xr-x 2 dupond gis4 216 Nov 2 2016 lib
drwxr-xr-x 2 dupond gis4 80 Nov 2 2016 applications
tpBib> java applications.Application
```

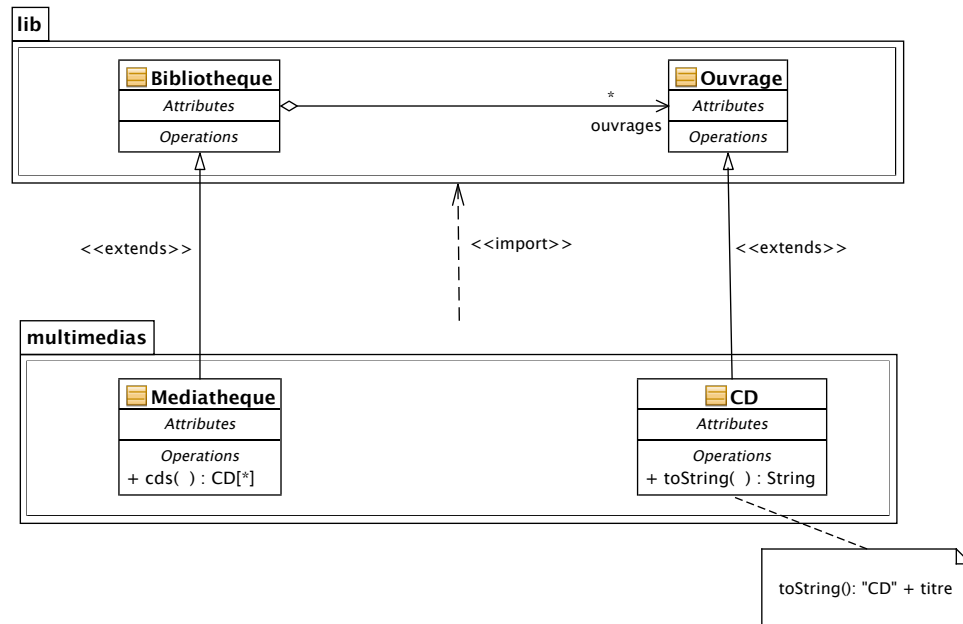
2.2 Jouer avec l’encapsulation

Expérimenter les règles d’encapsulation en “jouant” avec les modifieurs de visibilité des variables d’instances et des méthodes des classes de `lib`. Vérifier notamment que :

- seules les ressources `public` du package `lib` sont accessibles dans `Application` (les autres : `private`, `protected` ou “par défaut” ne le sont pas).
- au sein du package `lib` toutes les ressources sont visibles par toutes les classes sauf celles déclarées `private`. Essayer par exemple de rendre `private` les variables d’instance de la classe `Ouvrage` et d’y accéder dans `Bibliotheque` ou dans `Revue`.

2.3 Protected

Pour rappel, l’intérêt de `protected` (contrairement à `private` ou “par défaut”) est de laisser une classe ouverte à l’extension dans d’autres packages. Pour expérimenter cela, programmer la situation résumée dans le schéma UML suivant :



1. Rendre **protected** la Map `ouvrages` dans la classe `Bibliotheque` ainsi que les variables d'instance de la classe `Ouvrage` du package `lib`.
2. Créer les classes suivantes dans un package `multimedias` :
 - `CD` sous-classe de `lib.Ouvrage` qui redéfinit la méthode `toString()` comme spécifié sur le schéma.
 - `Mediatheque` sous-classe de `Bibliotheque` qui gère en plus des `CDs` (et pourquoi pas des `DVDs` ...) par polymorphisme de contenu de la Map d'`Ouvrage`. Elle ajoute la méthode `cds()` qui renvoie la liste des `CDs` en filtrant les ouvrages de la Map.
3. Vérifier les règles de visibilité des ressources du package `lib` dans les classes des packages `multimedias` et `applications` :
 - les ressources **public** sont accessibles dans toutes ces classes
 - les ressources **private** ou “par défaut” ne le sont pas
 - les ressources **protected** ne sont accessibles que dans les sous-classes.