TD 5 : "*Polymorphisation obscure*" dans une médiathèque

Modèles de conception (Visiteur)

Le site de l'organisation du cours, où vous allez trouver le projet à forker (TD5) :
https://github.com/IUTInfoMontp-M3105

Consignes : respecter les divers principes et bonnes pratiques de programmation

Date limite de rendu sur votre dépôt GitHub privé : **Dimanche 20 Octobre, 23h00**

On souhaite proposer à des médiathèques un module qui permet de calculer la date de retour d'un document emprunté (livres, CD, films, etc.). Chaque médiathèque peut choisir sa politique de durée de prêt et elle peut varier fortement entre les différents types de documents. Ainsi une médiathèque peut choisir de prêter tous ses documents pour un nombre de jours fixé par l'utilisateur ; une autre médiathèque peut choisir de prêter chaque type de document pour une durée différente (par exemple 10 jours pour un livre, 8 pour un CD, 5 pour un film, ...) ; et on peut aussi envisager qu'une médiathèque propose des cartes à ses abonnés, qui donneront des jours d'emprunt supplémentaires (par exemple, un abonné "*Bronze*" a droit à 2 jours en plus, un "*Argent*" a 5 jours en plus, un "*Or*" a 8 jours en plus, un abonné "*Lecture*" a 8 jours en plus pour les livres, un abonné "*Musique*" a 8 jours en plus pour les CD, ...). **On peut ajouter d'autres possibilités et toutes peuvent se combiner...**

Si on essaie de programmer la combinaison de toutes ces possibilités, on risque vite d'arriver à une "explosion de conditions" ; et si on souhaite intégrer une nouvelle politique d'emprunt pour une nouvelle médiathèque intéressée par ce module, il faudrait actualiser le code existant, situation qu'on évitera autant que possible (non-respect du principe ouvert/fermé).

Exercice 1 - création des documents

En premier lieu, on va mettre en place le code qui calcule les dates de retour de documents.

1. La classe `Document` vous est donnée dans le package `fr.umontpellier.iut.documents`. Elle possède des attributs privés `titre` et `dateRetour`. Sauf indication contraire, vous ne devriez pas **modifier** les méthodes et les attributs de cette classe.
   A partir de la classe `Document`, créez les classes correspondant aux livres, CD et films. Redéfinissez dans chacune de ces 3 classes la méthode `toString()` en évitant la duplication de code.

2. Écrivez une classe `CalculDate` qui propose la méthode `static LocalDate ajouter(int nbJours)` à laquelle on transmet un entier correspondant à un nombre de jours, et qui retourne une date future : la date du jour augmentée de ce nombre de jours. Voici la syntaxe pour obtenir cette date : `LocalDate.now().plusDays(nbJours);`

   La méthode `ajouter(int nbJours)` sera souvent utilisée dans le programme.

3. La classe `AppEmprunts` est la classe principale de ce module. Elle gère le fonds documentaire sous la forme d'une `Map` qui associe la cote d'un document (une chaîne de caractères) au document lui-même. Il faudra la compléter :

- Tout d'abord implémentez la méthode `constituerFonds()` pour initialiser le fonds documentaire à partir des informations suivantes :
  `LI_ORW_1` ⇒ `1984`, `LI_TOL_1` ⇒ `Le seigneur des anneaux`,
  `CD_STO_1` ⇒ `Satisfaction`, `CD_BEA_1` ⇒ `Abbey Road`,
  `FI_KUB_1` ⇒ `2001 Odyssée de l'espace`, `FI_SCO_1` ⇒ `Taxi Driver`.
- Dans la méthode `main(String args[])`, après avoir constitué le fonds de la médiathèque, créez un panier de documents à emprunter sous la forme d'une `ArrayList` de documents.
- Ajoutez une méthode qui gère un emprunt pour un nombre fixe de jours (15 par exemple) : on lui transmet une `ArrayList` (le panier des documents à emprunter), cette méthode affecte la date de retour de chacun des documents. Cette date de retour sera calculée en utilisant la méthode statique de la classe `CalculDate`.

Testez dans la méthode principale, en constituant un panier de 4 documents de types différents (on les retrouve grâce à la cote du document) et en affichant les informations sur chaque document du panier.

Exercice 2 - les premières politiques d'emprunt

On ajoute maintenant la possibilité de changer la politique de durée d'emprunt. Pour le moment vous gérerez deux politiques : une durée d'emprunt pour un nombre de jours fixe (la politique vue dans l'Exercice 1), et une durée qui varie suivant le type du document. Mais comme évoqué dans l'introduction, d'autres politiques d'emprunts pourront s'ajouter.

1. Dans les TDs précédents (de cette année et de l'an dernier) vous avez déjà utilisé le modèle de conception *Stratégie*. Est-ce qu'il serait applicable ici et quels seraient les limites ? Justifiez en discutant avec votre enseignant.

2. Créez un package `fr.umontpellier.iut.politiques` et déclarez une interface `PolitiqueEmprunt` qui propose 3 méthodes `LocalDate calculerDateRetour(...)`, chacune avec un type de document différent en paramètre.

3. Implémentez l'interface `PolitiqueEmprunt` en 2 classes `EmpruntDateFixe` et `EmpruntSelonTypeDoc`. Les différentes méthodes `LocalDate calculerDateRetour(...)` devraient calculer et retourner des dates de retour en fonction du type de document passé en paramètre. Pour la classe `EmpruntFixeSelonTypeDoc` vous supposerez que les dates fixes sont : 10 jours pour les CDs, 20 jours pour les livres et 5 jours pour les films.

4. Ajoutez une méthode abstraite `void emprunter(PolitiqueEmprunt p)` à la classe `Document` et implémentez-la dans chacune des 3 classes `Livre`, `CD`, `Film`. En utilisant l'objet `p`, cette méthode doit mettre à jour la date de retour du document.

   Question : *le code des différentes implémentations de* `void emprunter(PolitiqueEmprunt p)` *est assez "répétitif", ne pourrait-on pas le factoriser dans la classe-mère* `Document` *? Justifiez.*

   Vous venez de mettre en place le modèle de conception *Visiteur* vu en cours : l'interface `PolitiqueEmprunt` est le visiteur et les sous-classes de `Document` sont des éléments visitables.

   Vous pouvez maintenant renforcer l'encapsulation des documents en changeant la visibilité de la méthode `setDateRetour(...)` en `protected`.

5. Vérifiez dans la classe principale (toujours en affichant les informations sur chaque document). Faites le diagramme de classe. Quels sont les avantages et les inconvénients de l'utilisation du modèle dans votre exercice ?

Exercice 3 - les profils des emprunteurs

On va maintenant utiliser l'architecture précédente afin de gérer d'autres facteurs n'ayant aucun rapport avec les documents : les profils d'emprunteur. Ainsi des nouvelles politiques d'emprunt pourront être définies en tenant compte de plusieurs avantages qu'un emprunteur pourrait posséder.

1. Déclarez un package `fr.umontpellier.iut.emprunteurs` avec une interface `Emprunteur` et implémentez-la en 6 classes différentes : `EmprunteurOr`, `EmprunteurArgent`, `EmprunteurBronze`, `EmprunteurStandard`, `EmprunteurCarteLecture`, `EmprunteurCarteMusique`.

2. Dans le package `fr.umontpellier.iut.politiques`, ajoutez une interface `AvantageEmprunteur` proposant des méthodes `int getBonusEmprunteur(...)` ayant en paramètre un sous-type d'`Emprunteur` (dans notre exemple il y aura donc 6 méthodes en tout). Cette méthode calculera le nombre de jours supplémentaires à accorder pour l'emprunt en fonction de la situation. Implémentez l'interface en trois classes : `AvantageStatut`, `AvantageCarteLecture`, `AvantageCarteMusique`. Afin de voir les différences lors des tests, vous choisirez des nombres de jours différents pour les différentes implémentations des méthodes `getBonusEmprunteur(...)`.

3. Ajoutez à l'interface `Emprunteur` une méthode `int getBonus(AvantageEmprunteur av)`, qui retourne le nombre de jours supplémentaires de prêt. Implémentez cette méthode dans chaque sous-type d'`Emprunteur` de façon à ce qu'elle appelle la méthode `getBonusEmprunteur(...)` de `AvantageEmprunteur`.

4. Vérifiez dans le programme principal en créant de nouvelles politiques d'emprunts qui combinent les différents avantages en fonction du type d'emprunteur.