Développement informatique avancé orienté application - TP

L'héritage

Virginie Van den Schrieck

Ce TP se base sur les classes réalisées pendant les TPs précédents. Si vous n'avez pas fini ce dernier, mettez-vous en ordre au plus vite, et soyez sûrs de rattraper votre retard avant la prochaine séance !

Les notions illustrées par les exercices ci-dessous sont expliquées dans la section 2.5 du site INGInious du cours.

1 Classe Employé

Ecrivez une classe `Personne`. Une personne est caractérisée par son nom, son prénom et son numéro de registre national.

Documentez, testez et écrivez les méthodes `equals()` et `toString()` de la classe `Personne`, ainsi que les constructeurs ad-hoc.

Définissez à présent deux classes supplémentaires, `Employé` et `Indépendant`. Un `Employé` est une personne avec en plus un salaire (`int`) et un employeur (`String`). Un indépendant est une personne avec un numéro de TVA (`String`).

Documentez et testez les méthodes `equals()` et `toString()` de ces deux classes, ainsi que des constructeurs avec paramètres. Implémentez-les ensuite en exploitant l'héritage (mot-clé `super`).

Ensuite, analysez le code suivant. Observez le type des variables utilisées. D'après vous, qu'est ce qui s'affichera à la console ?

```
public static void main(String [] args){
    Personne pers = new Personne("Jules", "Dupont", 123234);
    Personne emp = new Employe("Jules", "Dupont", 123234, "EPHEC", 1500);
    Personne ind = new Independant("Jules", "Dupont", 123234, "BE0123456789");
    System.out.println(pers);
    System.out.println(emp);
    System.out.println(ind);
}
```

Pour finir, dans une classe de test, copiez, adaptez éventuellement puis exécutez le code suivant. Expliquez ce code et le résultat qui s'affiche.

2 Classe Calculatrice

Reprenez la classe Calculatrice déjà implémentée dans les TPs précédents. Comment pourriez-vous faire pour proposer une version améliorée de la calculatrice, de telle sorte qu'elle puisse stocker un résultat en mémoire en plus de la valeur courante ?

Dessinez le diagramme UML de votre solution, puis implémentez-la (+ tests et spécifications).

3 Gestion de bibliothèque

Un éditeur souhaite disposer d'un logiciel de gestion de ses ouvrages. Il veut pouvoir garder le titre, le nom de l'auteur, et l'année d'édition. Il doit pouvoir donner la première page du livre, ainsi qu'un extrait. Par défaut, l'extrait est la première page du livre. Parmi les livres vendus, un cas particulier existe : Les dictionnaires.

Un dictionnaire est caractérisé par sa langue, et contient 10 définitions par page. L'extrait d'un dictionnaire est sa première définition.

Vous devez implémenter ce logiciel. Pour cela, suivez les étapes suivantes :

1. Dessinez le diagramme UML de l'application.
2. Créez un package bibliotheque. Dans ce package, créez une classe Livre, qui aura comme attributs :
   — Un nom d'auteur, un titre et une année d'édition
   — Un tableau de String permettant de stocker les pages du livre.
3. Créez, et documentez les getters et setters de l'application. Pour l'auteur et le titre, si un string vide est donné, le logiciel doit stocker les valeurs « Auteur inconnu » et « Titre inconnu ».
4. Documentez, testez puis générez automatiquement les méthodes equals() et hashCode() pour la classe Livre. *Pour les curieux : Qu'est ce que cette méthode hashCode() ? Pourquoi est-elle associée à equals() ?*
5. Créez deux constructeurs avec paramètres : Un permettant de spécifier l'auteur, le titre et l'année d'édition, et l'autre spécifiant les quatre variables d'instance. Les constructeurs doivent utiliser les setters de la classe. Par défaut, un livre est prévu pour contenir 100 pages.
6. Créez, documentez, testez et implémentez une méthode `getPage(int i)`, qui renvoie la page à l'indice i. Si la page n'existe pas, la valeur null est renvoyée. *Utiliser des valeurs spéciales n'est pas la meilleure façon de faire en java. Nous verrons plus tard la méthode des Exceptions, qui est plus adaptée.*
7. Créez, documentez, testez et implémentez la méthode `setPage(int i, String p)`, qui insère la page donnée à l'indice i. Elle renvoie la

page qui se trouvait précédemment à cet emplacement. Si l'indice n'est pas conforme, un message d'erreur est renvoyé.

8. Ajoutez la méthode `toString()` à la classe. Cette méthode renvoie le titre, l'auteur et l'année d'édition.
9. Créez, documentez, testez et implémentez la méthode `getFirstPage()`, qui renvoie la première page du livre. Pensez à réutiliser le code que vous avez déjà écrit.
10. Créez, documentez, testez et implémentez la méthode `extrait()`. Pensez à réutiliser le code que vous avez déjà écrit.
11. Créez et documentez la classe `Dictionnaire`. Choisissez les bonnes options pour générer automatiquement le fichier Dictionnaire en fonction du diagramme UML.
12. Créez une nouvelle variable d'instance `dictPages`, qui est un tableau à deux dimensions permettant de stocker 100 pages de 10 définitions. Cette variable sera utilisée à la place de pages. Modifiez les constructeurs en fonction.
13. Spécialisez les méthodes `equals`, `hashCode` (utilisez Eclipse !) et `toString`
14. Dans Dictionnaire, réécrivez les méthodes `getPage` et `setPage` pour qu'elles travaillent sur base d'une chaîne de caractères reprenant les 10 définitions, une par ligne. Devez-vous également modifier `getExtrait()` et `getFirstPage` ? Faites les modifications nécessaires.
15. Ajoutez des méthodes spécialisées `getDefinition()` et `setDefinition`.
16. Testez toutes vos méthodes dans une classe JUnit `TestDictionnaire`.