TP6 : Blog (ancien examen)

POO : Programmation Orientée Objet

Première année

L'objectif du TP est de développer un micro-système de blog, un site web constitué par la réunion de billets agglomérés au fil du temps. Ces billets peuvent être des paragraphes de texte (message), des images ou des vidéos.
Les billets de type image ou vidéo seront dits *taggables* dans le sens où il sera possible de leur associer des mots clés (*tag*). Il sera ainsi possible de rechercher tous les billets de ce type qui comporteront un certain ensemble de *tags* (par exemple, tous les billets comportant les *tags* `geek` et `esial`). Au contraire, les billets de type message ne pourront être *taggés*, la recherche se fera alors sur le contenu du message.

Éléments fournis. Un ensemble de classes de test vous sont fournies, elles sont disponibles :
- dans le répertoire `/home/depot/1A/POO/TP6`
- sur internet, `http://www.loria.fr/~oster/poo-tp6.tar.gz`

Tests et Compilation. Les classes de test fournies vous permettront de tester les différentes classes que vous implémenterez. Ces classes vous afficheront quels sont les tests qui ne s'exécutent pas correctement et vous donneront à titre indicatif un score pouvant s'apparenter à votre note de TP.

▶ Question 1: `blog.Publishable`.

Implémentez une interface publique `blog.Publishable` dont les profiles des méthodes sont les suivants :
- une méthode `getPublicationDate()` sans paramètre et dont le type de retour est `long`.
- une méthode `getAuthor()` sans paramètre et dont le type de retour est `String`.

✓ Validation 1: Testez votre implémentation.

Compilez et exécutez la classe de test fournie nommée `test.TestPublishable`. Si nécessaire, corriger les différentes erreurs dans votre code.

▶ Question 2: `blog.AbstractPublishableItem`.

Implémentez une classe abstraite `blog.AbstractPublishableItem` qui réalise l'interface `blog.Publishable`.
Cette classe doit :
- conserver la date de publication (type `long`) et l'auteur du billet (type `String`).
- définir un constructeur dont les paramètres sont la date de publication et l'auteur du billet (respecter cet ordre de définition).
- définir les méthodes de l'interface `blog.Publishable`.

✓ Validation 2: Testez votre implémentation.

Les classes de test sont `test.TestAbstractPublishableItem1` et `test.TestAbstractPublishableItem2`.

▶ Question 3: `blog.Taggable`.

Implémentez une interface publique `blog.Taggable` dont les profiles des méthodes sont les suivants :
- une méthode `addTag()` dont le paramètre est un tag (type `String`) et sans type de retour. Cette méthode servira à ajouter un tag.
- une méthode `removeTag()` dont le paramètre est un tag (type `String`) et sans type de retour. Cette méthode servira à supprimer un tag.
- une méthode `tagCount()` sans paramètre dont le type de retour est un entier (`int`). Cette méthode permettra de connaître le nombre de tag sur un billet.
- une méthode `getTags()` sans paramètre et dont le type de retour est une liste de chaîne de caractères (`List<String>`). Cette méthode permettra d'obtenir la liste des tags définis sur un billet.

✓ Validation 3: Testez votre implémentation.

La classe de test est `test.TestTaggable`.

▶ Question 4: `blog.AbstractItem`.

Implémentez une classe abstraite `blog.AbstractItem` qui hérite de la classe `blog.AbstractPublishableItem`

et qui réalise l'interface `blog.Taggable`.
Cette classe doit :
- conserver une liste de tags (liste de chaîne de caractères).
- définir un constructeur dont les paramètres sont la date de publication et l'auteur du billet (respecter cet ordre de définition).
- définir les méthodes de l'interface `blog.Taggable`.

Il faut noter que :
- la méthode `addTag()` n'autorisera pas à ajouter deux fois le même tag.
- la méthode `getTags()` retournera si nécessaire une liste vide et non pas une référence `null`.

✓ Validation 4: Testez votre implémentation.

Les classes de test sont `test.TestAbstractItem1` et `test.TestAbstractItem2`.

▶ Question 5: `blog.Message`.

Implémentez une classe `blog.Message` qui hérite de la classe `blog.AbstractPublishableItem`. Cette classe doit :
- conserver un contenu de type chaîne de caractères (le corps du message).
- définir un constructeur dont les paramètres sont la date de publication, l'auteur du billet et le corps du message (respecter cet ordre de définition).
- définir une méthode `getContent()` sans paramètre dont le type de retour est une chaîne de caractères qui correspond au contenu du message.

✓ Validation 5: Testez votre implémentation.

Les classes de test sont `test.TestMessage1` et `test.TestMessage2`.

▶ Question 6: `blog.Picture`.

Implémentez une classe `blog.Picture` qui hérite de la classe `blog.AbstractItem`. Cette classe doit :
- conserver un contenu de type chaîne de caractères, l'adresse HTTP (URL) de l'image, par exemple `http://www.monsite.com/logo.png`.
- définir un constructeur dont les paramètres sont la date de publication, l'auteur du billet et l'adresse de l'image (respecter cet ordre de définition).
- définir une méthode `getURL()` sans paramètre dont le type de retour est une chaîne de caractères qui correspond à l'adresse de l'image.

✓ Validation 6: Testez votre implémentation.

Les classes de test sont `test.TestPicture1` et `test.TestPicture2`.

▶ Question 7: `blog.Video`.

Implémentez une classe `blog.Video` qui hérite de la classe `blog.AbstractItem`. Cette classe doit :
- conserver un contenu de type chaîne de caractères, l'adresse HTTP (URL) de la vidéo, par exemple `http://www.monsite.com/trailer.avi`.
- définir un constructeur dont les paramètres sont la date de publication, l'auteur du billet et l'adresse de la vidéo (respecter cet ordre de définition).
- définir une méthode `getURL()` sans paramètre dont le type de retour est une chaîne de caractères qui correspond à l'adresse de la vidéo.

✓ Validation 7: Testez votre implémentation.

Les classes de test sont `test.TestVideo1` et `test.TestVideo2`.

▶ Question 8: `blog.BlogService`.

Implémentez une interface publique `blog.BlogService` dont les profiles des méthodes sont les suivants :
- une méthode `getTitle()` sans paramètre et dont le type de retour est une chaîne de caractères. Cette méthode permettra de connaître le titre du blog.
- une méthode `post()` dont le paramètre est un billet (de type `blog.Publishable`) et sans type de retour. Cette méthode permettra de publier un nouveau billet sur le blog.
- une méthode `getItems()` sans paramètre et dont le type de retour est une liste de billets (type `List<Publishable>`). Cette méthode permettra d'obtenir la liste de tous les billets publiés sur le blog.

- une méthode `getPublishableItemsCount()` sans paramètre et dont le type de retour est une valeur entière. Cette méthode permettra de connaître le nombre de billets publiés sur le blog.
- une méthode `getTaggableItemsCount()` sans paramètre et dont le type de retour est une valeur entière. Cette méthode permettra de connaître le nombre de billet publiés sur lesquels on peut ajouter des tags (et donc qui possède une méthode `addTag()`).
- une méthode `findItemsByAuthor()` dont le paramètre est le nom d'un auteur (une chaîne de caractères) et dont le type de retour est une liste de billets (de type `List<Publishable>`). Cette méthode permettra de consulter tous les billets rédigés par un auteur donné.
- une méthode `getLatestItem()` sans paramètre et dont le type de retour est un billet (type `blog.Publishable`). Cette méthode permettra d'obtenir le billet le plus récent. On se basera sur la date de publication du billet et non pas sur l'ordre dans lequel les billets ont été ajoutés sur le blog.
- une méthode `findItemsByTags()` dont le paramètre est un tableau de tag (tableau à taille fixe de chaîne de caractères) et dont le type de retour est une liste de billets (de type `List<Publishable>`). Cette méthode permettra de consulter tous les billets qui comportent *tous* les tags passés en paramètre. Cette méthode ne retournera donc pas de message (puisqu'il n'est pas possible d'ajouter des tags sur un message).
- une méthode `findItemsByContent()` dont le paramètre est un tableau à taille fixe de chaînes de caractères et dont le type de retour est une liste de billets (de type `List<Publishable>`). Cette méthode permettra de consulter tous les messages dont le corps comporte *tous* les mots donnés en paramètre.
- une méthode `findItemsByTagsOrContent()` dont le paramètre est un tableau de tag (tableau à taille fixe de chaîne de caractères) et dont le type de retour est une liste de billets (de type `List<Publishable>`). Cette méthode permettra de consulter tous les billets qui comportent *tous* les tags passés en paramètre. Elle retournera également tous les messages dont le corps comporte *tous* ces mots.

✓ Validation 8: Testez votre implémentation.

La classe de test est `test.TestBlogService`.

▶ Question 9: `blog.BlogServiceImpl`.

Implémentez une classe `blog.BlogServiceImpl` qui réalise l'interface `blog.BlogService`.
Cette classe doit :

- conserver le titre du blog (type `String`) et la liste de tous les billets publiés sur le blog (type `List<Publishable>`).
- définir un constructeur dont le paramètre est le nom du blog.
- définir les méthodes de l'interface `blog.BlogService`.

Il faut noter que :

- les méthodes devant retourner une liste de billets ne retourneront jamais une référence `null`. Si il n'y a pas d'éléments à retourner alors une liste vide est donnée comme résultat.
- le billet le plus récent est celui dont la date de publication (type `long`) est la plus élevée.
- l'appel de méthode `myStr.indexOf(lookingForStr)` définit dans la classe `String` retourne -1 si la chaîne `lookingForStr` n'est pas contenu dans la chaîne `myStr`.

✓ Validation 9: Testez votre implémentation.

Les classes de test sont `test.TestBlogServiceImpl1` et `test.TestBlogServiceImpl2`.

✓ Validation 10: Testez votre implémentation complète.

Afin de relancer l'ensemble des tests fournis, vous pouvez compiler et exécuter la classe de test nommée `blog.TestAll`.

Le diagramme ci-dessous illustre l'ensemble des classes et interfaces que vous avez à implémenter.