

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/>

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 profils 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`.

Réponse

```
1 package blog;
2
3 public interface Publishable {
4     public long getPublicationDate();
5     public String getAuthor();
6 }
7
8
9
```

Fin réponse

✓ **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`.

Réponse

```
1 package blog;
2
3 public abstract class AbstractPublishableItem implements Publishable {
4     protected String author;
5     protected long publicationDate;
6
7     public AbstractPublishableItem(long publicationDate, String author) {
8         this.publicationDate = publicationDate;
9         this.author = author;
10    }
11
12    public String getAuthor() {
13        return this.author;
14    }
15 }
```

```

16     public long getPublicationDate() {
17         return this.publicationDate;
18     }
19 }
20

```

Fin réponse

✓ **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.

Réponse

```

1 package blog;
2
3 import java.util.List;
4
5 public interface Taggable {
6
7     public List<String> getTags();
8
9     public void addTag(String tag);
10
11     public void removeTag(String tag);
12
13     public int tagCount();
14
15 }

```

Fin réponse

✓ **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`.

Réponse

```

1 package blog;
2
3 import java.util.ArrayList;
4 import java.util.List;
5

```

```

6 public abstract class AbstractItem extends AbstractPublishableItem implements Taggable {
7
8     protected List<String> tagList;
9
10    public AbstractItem(long publicationDate, String author) {
11        super(publicationDate, author);
12        this.tagList = new ArrayList<String>();
13    }
14
15    public void addTag(String tag) {
16        if (tag != null && !this.tagList.contains(tag))
17            this.tagList.add(tag);
18    }
19
20    public void removeTag(String tag) {
21        this.tagList.remove(tag);
22    }
23
24    public int tagCount() {
25        return this.tagList.size();
26    }
27
28    public List<String> getTags() {
29        return this.tagList;
30    }
31
32
33 }

```

Fin réponse

✓ **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.

Réponse

```

1 package blog;
2
3 public class Message extends AbstractPublishableItem {
4
5     private String content;
6
7     public Message(long publicationDate, String author, String content) {
8         super(publicationDate, author);
9         this.content = content;
10    }
11
12    public String getContent() {
13        return this.content;
14    }
15
16 }

```

Fin réponse

✓ **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.

Réponse

```

1 package blog;
2
3 public class Picture extends AbstractItem {
4
5     private String url;
6
7     public Picture(long publicationDate, String author, String url) {
8         super(publicationDate, author);
9         this.url = url;
10    }
11
12    public String getURL() {
13        return this.url;
14    }
15
16 }
```

Fin réponse

- ✓ **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.

Réponse

```

1 package blog;
2
3 public class Video extends AbstractItem {
4
5     private String url;
6
7     public Video(long publicationDate, String author, String url) {
8         super(publicationDate, author);
9         this.url = url;
10    }
11
12    public String getURL() {
13        return this.url;
14    }
15
16 }
```

Fin réponse

- ✓ **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 profils 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 (puisque'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.

Réponse

```

1 package blog;
2
3 import java.util.List;
4
5 public interface BlogService {
6
7     public abstract String getTitle();
8
9     public abstract void post(Publishable item);
10
11     public abstract List<Publishable> getItems();
12
13     public abstract int getPublishableItemsCount();
14
15     public abstract int getTaggableItemsCount();
16
17     public abstract List<Publishable> findItemsByAuthor(String author);
18
19     public abstract Publishable getLatestItem();
20
21     public abstract List<Publishable> findItemsByTags(String[] tag);
22
23     public abstract List<Publishable> findItemsByContent(String[] str);
24
25     public abstract List<Publishable> findItemsByTagsOrContent(String[] tags);
26
27 }

```

Fin réponse

- ✓ **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`.

Réponse

```

1 package blog;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class BlogServiceImpl implements BlogService {
8
9     private String title;
10    private List<Publishable> items;
11
12    public BlogServiceImpl(String title) {
13        this.title = title;
14        this.items = new ArrayList<Publishable>();
15    }
16
17    public String getTitle() {
18        return this.title;
19    }
20
21    public void post(Publishable item) {
22        this.items.add(item);
23    }
24
25    public List<Publishable> getItems() {
26        return this.items;
27    }
28
29    public int getPublishableItemsCount() {
30        return this.items.size();
31    }
32
33    public int getTaggableItemsCount() {
34        int count = 0;
35        for (Publishable item : this.items) {
36            if (item instanceof Taggable) {
37                count++;
38            }
39        }
40        return count;
41    }
42
43    public List<Publishable> findItemsByAuthor(String author) {
44        List<Publishable> results = new ArrayList<Publishable>();
45        for (Publishable item : this.items) {
46            if (item.getAuthor().equals(author)) {
47                results.add(item);
48            }
49        }
50        return results;
51    }
52
53    public Publishable getLatestItem() {
54        if (this.items.size() == 0)
55            return null;
56
57        Publishable result = this.items.get(0);
58        for (Publishable item : this.items) {
59            if (item.getPublicationDate() >= result.getPublicationDate()) {
60                result = item;
61            }
62        }
63        return result;
64    }
65
66    public List<Publishable> findItemsByTags(String[] tag) {
67        List<Publishable> results = new ArrayList<Publishable>();
68        for (Publishable item : this.items) {
69            if (item instanceof Taggable) {
70                Taggable itemT = (Taggable) item;
71                List<String> tags = itemT.getTags();
72                if (tags.containsAll(Arrays.asList(tag))) {
73                    results.add(item);
74                }
75            }
76        }
77        return results;
78    }
79 }
80

```

```
81     public List<Publishable> findItemsByContent(String[] str) {  
82         List<Publishable> results = new ArrayList<Publishable>();  
83  
84         for (Publishable item : this.items) {  
85             if (item instanceof Message) {  
86                 Message msg = (Message) item;  
87  
88                 int index = 0;  
89                 while (index < str.length && msg.getContent().indexOf(str[index]) != -1)  
90                     index++;  
91                 if (index == str.length)  
92                     results.add(item);  
93             }  
94         }  
95         return results;  
96     }  
97  
98     public List<Publishable> findItemsByTagsOrContent(String[] tags) {  
99         List<Publishable> results = findItemsByTags(tags);  
100         results.addAll(findItemsByContent(tags));  
101         return results;  
102     }  
103 }  
104 }
```

Fin réponse

✓ **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.

