

UE Programmation Orientée Objet
Examen première session – 2 juin 2010

3 heures - notes et photocopiés de cours/TD/TP autorisés
 livres, calculatrices et appareils de communication interdits
 dictionnaires de langue autorisés pour candidats étrangers

▷ Les 4 exercices sont totalement indépendants et l'ordre dans lequel ils apparaissent ne préjuge pas de leur difficulté réelle ou supposée.

Exercice 1 : Textes et formatages

(Les classes de cet exercice appartiendront au package `texte`).

Un même contenu de texte structuré peut être formaté de différentes manières, par exemple selon l'usage que l'on peut en avoir ou le média sur lequel on souhaite l'afficher. L'annexe 1 propose un même contenu de texte formaté de deux manières différentes : en ASCII simple et en HTML.

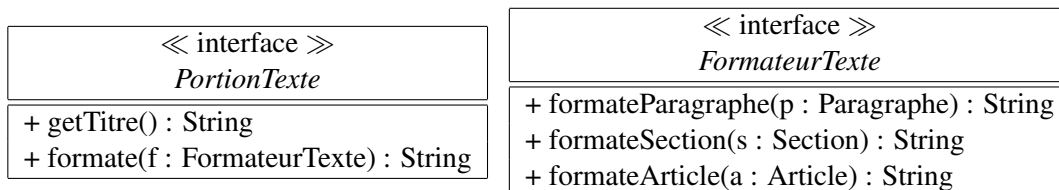
On appelle *texte structuré* un texte qui se décompose en plusieurs portions de texte. Une *portion de texte* comporte un titre et un contenu et peut être transformée selon un formatage donné. On distingue deux types de portions de texte : les *paragraphes* et les *sections*. Le contenu d'un paragraphe est une chaîne de caractères. Le contenu d'une section est une liste de paragraphes.

On s'intéressera par la suite à des textes structurés, que nous appellerons *article*, qui sont composés dans cet ordre :

- d'un titre,
- d'un auteur,
- d'un résumé, qui est un paragraphe dont le titre est nécessairement "Résumé",
- d'une suite de portions de texte (quelconques).

Le texte de l'annexe 1 correspond à cette structure.

Portions de texte et formateurs sont définies par les interfaces suivantes du paquetage `texte`.



La méthode `formate` a pour résultat la chaîne de caractères correspondant au formatage de la portion de texte selon le formateur fourni. Le titre d'une portion de texte est fourni à sa construction.

`Paragraphe`, `Section` et `Article` sont trois classes qui implémentent l'interface `PortionTexte` et qui permettent de représenter les paragraphes, les sections, et les articles.

Q 1. Donnez le code java de la classe `Paragraphe`. Le contenu est fourni à la construction d'un paragraphe et on doit disposer d'une méthode permettant d'accéder à ce contenu.

Q 2. Donnez un diagramme UML de classe **détaillé** pour la classe `Section`, sachant que

- à la construction la liste des paragraphes qui composent une section est vide, il doit donc être possible d'ajouter un paragraphe à une section.
- la classe implémente l'interface `Iterable<Paragraphe>` (voir Annexe 2) et fournit donc la méthode associée permettant d'accéder aux paragraphes qui composent la section.

Faites apparaître sur votre schéma UML, les dépendances vers les interfaces implémentées.

Il n'est pas demandé de code Java pour cette classe.

Q 3. Donnez le code java de la classe `Article` sachant que :

- l'auteur et le contenu du résumé sont des chaînes de caractères également fournies à la création, et, à la construction, la liste des portions de texte qui composent l'article est vide,
- la classe implémente l'interface `Iterable<PortionTexte>` qui permet d'accéder aux portions de texte qui composent l'article (voir Q 2).

Q 4. Donnez le code java de la classe `FormateurHTML` dont la méthode `formateArticle` produit le texte formaté présenté à l'annexe 1.2.

Vous exploiterez évidemment de manière pertinente la forme structurée du texte. Vous pourrez également tenir compte du fait que la syntaxe HTML est toujours de la forme `<balise>contenu</balise>`.

Les titre de l'article, auteur, résumé, sections, paragraphes et leurs titres apparaissent suffisamment explicitement dans l'annexe pour vous permettre de faire l'analyse du résultat à produire.

Exercice 2 : Démineur

(Les classes de cet exercice appartiendront au package `demineur`).

Voici un extrait de ce que l'on peut lire sur Wikipédia à propos du jeu *Démineur* ([http://fr.wikipedia.org/wiki/Demineur_\(jeu\)](http://fr.wikipedia.org/wiki/Demineur_(jeu))):

Le démineur est un jeu vidéo de réflexion dont le but est de localiser des mines cachées dans un champ. (...) Le champ de mines est représenté par une grille.

Chaque case de la grille peut soit cacher une mine, soit être vide. Le but du jeu est de découvrir toutes les cases libres sans faire exploser les mines, c'est-à-dire sans cliquer sur les cases qui les dissimulent.

Lorsque le joueur clique sur une case libre et que toutes les cases adjacentes le sont également, une case vide est affichée. Si en revanche au moins l'une des cases avoisinantes contient une mine, un chiffre apparaît, indiquant le nombre de cases adjacentes contenant une mine. En comparant les différentes informations récoltées, le joueur peut ainsi progresser dans le déminage du terrain. S'il se trompe et clique sur une mine, il a perdu.

Dans cet exercice on s'intéresse à la programmation de ce jeu, et plus particulièrement de son modèle sans s'occuper ni des aspects graphiques ni de la gestion de la souris.

On représente la notion de case du jeu à l'aide de la classe `demineur.Case` :



Case
- bombe : boolean
- etat : Etat
+ Case(bombe : boolean)
+ isBombe() : boolean
+ getEtat() : Etat
+ decouvre()

où `Etat` est un type énuméré contenant 2 valeurs : `cachee` et `decouverte`. Initialement une case est cachée. `isBombe` vaut `true` si et seulement si la case correspond à une bombe et une méthode `decouvre` est appelée lorsque la case est découverte.

On décide ici de modéliser le *champ* de mines du jeu par un tableau à 2 dimensions d'objets de type `Case`. A la création du jeu, on fixe la largeur et la hauteur du champ ainsi que le nombre exact de mines qu'il contient. Celles-ci sont alors positionnées aléatoirement.

A tout moment il est possible de connaître le nombre de cases découvertes dans le jeu et de savoir si le jeu est perdu ou non.

La classe `demineur.Position` est fournie :

Position
- x : int
- y : int
+Position(x: int, y : int)
+ getX() : int
+ getY() : int

La classe `Demineur` permet de modéliser le jeu.

Q 1. Donnez le code java de déclaration des attributs de la classe `Demineur` ainsi que de son constructeur (avec positionnement des mines).

Q 2. Donnez le code java de la méthode de la classe `Demineur` :

```
public Case getCaseAt(Position p) throws PositionInvalideException
```

elle fournit la case située à la position $p = (x, y)$ dans le tableau représentant le champ, l'exception est levée si les valeurs x et y ne correspondent pas à des indices valides pour ce tableau.

La classe `PositionInvalideException` est supposée définie dans le paquetage `demineur`. Elle possède un constructeur prenant en paramètre une chaîne représentant le message de l'exception.

Q 3. Donnez le code de la méthode de la classe `Demineur` :

```
public void decouvreCase(Position p)
```

qui est invoquée lorsque l'on découvre la case à la position p (typiquement lorsque l'on clique sur cette case). Il ne se passe rien si la case est déjà découverte ou si la position p n'est pas valide.

Si la case découverte est une bombe, le jeu est perdu et, dans ce cas, toutes les cases encore cachées sont révélées.

Q 4. Donnez le code de la méthode de la classe `Demineur` :

```
public int nbBombesAdjacentes(Position p)
```

qui renvoie le nombre de cases adjacentes de la position p qui sont occupées par une bombe. La position p est supposée valide.

Q 5. Donnez le code de la méthode de la classe `Demineur` :

```
public void affiche()
```

qui affiche ("à l'écran") le champ de mines ligne par ligne. Chaque case est représentée par un caractère :

- 'B' si la case est découverte et occupée par une bombe,
- 'X' pour une case non découverte,
- ' ' pour une case vide découverte sans mine adjacente,
- 'nb' pour une case vide découverte où nb est un chiffre compris entre 1 et 8 donnant le nombre de cases adjacentes occupées par une mine.

Exercice 3 : Abonnement téléphonique

(Les classes de cet exercice appartiendront au package `telephonie`).

Afin de disposer des données sur l'utilisation des abonnements téléphoniques, un opérateur de téléphonie a besoin de connaître pour chaque abonné le nombre d'appels émis, la durée cumulée (en secondes) de ces appels et le nombre de SMS envoyés.

Ces données sont regroupées pour un abonné dans la classe `DonneesAbonnement` :

DonneesAbonnement
- nbAppels : int - dureeAppels : int - nbSMS : int
+ DonneesAbonnement() + nouvelAppelEmis(dureeEnSecondes : int) + nouveauSMSEnvoye() + getNbAppelsEmis() + getDureeAppels() : int + getNbSMSEnvoyes() : int

Cet opérateur de téléphonie décide de rassembler les données de l'ensemble de ses abonnés dans la classe `DonneesGlobales` à l'aide d'une table de hachage dont :

- les clefs sont des instances de la classe `NumeroTelephone`, dont le seul attribut est une chaîne de caractères représentant le numéro de téléphone à 10 chiffres de l'abonnement, numéro évidemment impossible à changer,
- les valeurs sont des objets de la classe `DonneesAbonnement`

Q 1. Donnez le code java de la classe `NumeroTelephone`.

Q 2. Recopiez le code suivant (sauf les commentaires) en complétant aux endroits indiqués.

Certaines méthodes que l'on pourrait naturellement envisager pour cette classe ne sont pas mentionnées ni demandées (elles sont évoquées par les "... " en fin de classe).

COMPLETER

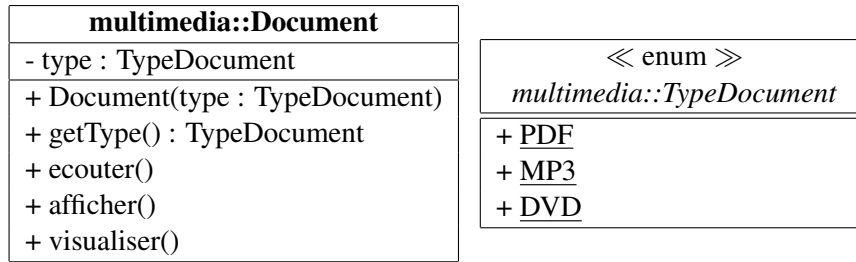
```
public class DonneesGlobales {
    COMPLETER attribut(s)
    public DonneesGlobales() {
        // initialement aucun numéro n'est connu
        COMPLETER
    }
    /** ajoute un nouveau numéro dont il faut gérer les données,
     * si le numéro existe déjà, il ne se passe rien (les données sont conservées)
     * @param numero le nouveau numéro à prendre en compte
     */
    public void nouveauNumero(NumeroTelephone numero) {
        COMPLETER
    }
    /** indique que le numéro a émis un appel de la durée indiquée
     * il ne se passe rien si le numéro est inconnu
     * @param numero le numéro qui a émis l'appel
     * @param dureeEnSecondes la durée de l'appel émis
     */
    public void nouvelAppel(NumeroTelephone numero, int dureeEnSecondes) {
        COMPLETER
    }
    /** retourne le nombre de SMS envoyés en moyenne par les numéros gérés
     * @return le nombre de SMS envoyés en moyenne par les numéros gérés
     */
    public float nombreMoyenDeSMS() {
        COMPLETER
    }
    ...
}
```

Exercice 4 : Un peu de conception

On considère un lecteur multi-média capable de lire des documents de différents types : fichiers MP3, PDF et DVD. D'autres types de documents sont envisageables.

Il est ainsi possible d'*écouter* un MP3, d'*afficher* un PDF et de *visualiser* un DVD.

Un étudiant propose de modéliser ces différents documents par la classe :



et donne alors de le code (simplifié ici) de la classe LecteurMultimedia suivant :

```
package multimedia;
public class LecteurMultimedia {
    ...
    public void lire(Document doc) {
        if (doc.getType() == TypeDocument.PDF) {
            doc.afficher();
        } else if (doc.getType() == TypeDocument.MP3) {
            doc.ecouter();
        } else {
            doc.visualiser();
        }
    }
}
```

Q 1. Expliquez **clairement** à cet étudiant en quoi la conception de la solution qu'il propose n'est pas une bonne conception orientée objet.

Q 2. Proposez une correction de "bonne solution" pour les documents et LecteurMultimedia.

Votre solution fera apparaître les notions Document, LecteurMultimedia, lire(), écouter(), visualiser() et afficher() (sans rentrer plus dans dans les détails du code que ce qui est donné dans la version proposée).

Annexe 1 : 2 formatages d'un même texte

Annexe 1.1 : Version ASCII "simple"

Exemple de texte (ceci est le titre du texte) par Timoléon (qui est l'auteur)

Résumé

Contenu du résumé d'un texte formaté en ASCII et HTML.

Titre de la première section

Titre du paragraphe 1.1

Je suis le contenu du paragraphe 1.1 de cet exemple de texte formaté en ASCII et HTML. Je suis le contenu du paragraphe 1.1 de cet exemple de texte formaté en ASCII et HTML.

Titre du paragraphe 1.2

Et moi je suis le contenu du paragraphe 1.2 de cet exemple de texte formaté en ASCII et HTML et qui suit le texte du paragraphe 1.1 de cet exemple.

Titre de la seconde section

Titre du paragraphe 2.1

Je suis le contenu du paragraphe 2.1 de cet exemple de texte formaté en ASCII et HTML. Je suis le contenu du paragraphe 2.1 de cet exemple de texte formaté en ASCII et HTML.

Titre du paragraphe 2.2

Et moi je suis le contenu du paragraphe 2.2 de cet exemple de texte formaté en ASCII et HTML et qui suit le texte du paragraphe 2.1 de cet exemple.

Titre de la troisième section

Titre du paragraphe 3.1

Contenu du paragraphe 3.1 : vous avez compris.

Titre du paragraphe 3.2

Contenu du paragraphe 3.2 : vous avez compris.

Titre du paragraphe 3.3

Contenu du paragraphe 3.3 : vous avez compris.

Conclusion (Titre d'un paragraphe "seul" - càd pas dans une section)

Contenu de la conclusion qui est un paragraphe non situé dans une Section.

Annexe 1.2 : Version HTML

```
<HTML>
```

```
<BODY>
```

```
<H1>Exemple de texte(ceci est le titre du texte)</H1>
```

```
<P>par Timoléon (qui est l'auteur)</P>
```

```
<H3>Résumé</H3>
```

```
<P>
```

```
Contenu du résumé d'un texte formaté en ASCII et HTML.
```

```
</P>
```

```
<H2>Titre de la première section</H2>
```

```
<H3>Titre du paragraphe 1.1</H3>
```

```
<P>Je suis le contenu du paragraphe 1.1 de cet exemple de texte formaté en ASCII et HTML. Je suis le contenu du paragraphe 1.1 de cet exemple de texte formaté en ASCII et HTML.</P>
```

```
<H3>Titre du paragraphe 1.2</H3>
```

```
<P>Et moi je suis le contenu du paragraphe 1.2 de cet exemple de texte
formaté en ASCII et HTML et qui suit le texte du paragraphe 1.1 de
cet exemple.</P>
```

```
<H2>Titre de la seconde section</H2>
```

```
<H3>Titre du paragraphe 2.1</H3>
```

```
<P>Je suis le contenu du paragraphe 2.1 de cet exemple de texte formaté
en ASCII et HTML. Je suis le contenu du paragraphe 2.1 de cet exemple
de texte formaté en ASCII et HTML.</P>
```

```
<H2>Titre du paragraphe 2.2</H2>
```

```
<P>Et moi je suis le contenu du paragraphe 2.2 de cet exemple de texte
formaté en ASCII et HTML et qui suit le texte du paragraphe 2.1 de
cet exemple.</P>
```

```
<H2>Titre de la troisième section</H2>
```

```
<H3>Titre du paragraphe 3.1</H3>
```

```
<P>Contenu du paragraphe 3.1 : vous avez compris.</P>
```

```
<H3>Titre du paragraphe 3.2</H3>
```

```
<P>Contenu du paragraphe 3.2 : vous avez compris.</P>
```

```
<H3>Titre du paragraphe 3.3</H3>
```

```
<P>Contenu du paragraphe 3.3 : vous avez compris.</P>
```

```
<H3> Conclusion (Titre d'un paragraphe "seul" - càd pas dans une section)</H3>
```

```
<P>Contenu de la conclusion qui est un paragraphe non situé dans une
Section.</P>
```

```
</BODY>
```

```
</HTML>
```

Annexe 2 : javadoc de l'interface `java.lang.Iterable<T>`

java.lang

Interface `Iterable<T>`

```
public interface Iterable<T>
```

Implementing this interface allows an object to be the target of the "foreach" statement.

Method Summary

```
Iterator<T> iterator()
```

Returns an iterator over a set of elements of type T.

Returns:

an Iterator.