Travail pratique #1

Ce travail doit être fait individuellement.

Notions mises en pratique : Le respect d'un ensemble de spécifications, l'héritage, le polymorphisme, les classes abstraites, et la classe `ArrayList`.

1. Spécifications

Il s'agit de concevoir et d'implémenter les classes de base pour la création (éventuelle) d'un petit prototype d'un éditeur Markdown (capable de transformer le texte sélectionné dans un style Markdown donné).

Ce travail consiste en deux parties :

1. Implémentation de la hiérarchie de classes `StyleMD` qui implémente différents styles Markdown.
2. Implémentation des classes qui modélisent un document au format Markdown.

Classes fournies (À NE PAS MODIFIER) à importer dans votre projet

1) Classe `TestsStyleMD` : pour tester la hiérarchie de classes `StyleMD` de la partie 1.
2) Classe `ElementTextuelMDInvalideException` : exception utilisée dans la partie 2.
3) Classe `StyleMDInvalideException` : exception utilisée dans la partie 2.
4) Classe `RecetteCrepesMD` : pour tester PARTIELLEMENT les classes de la partie 2.

1.1 PARTIE 1 – IMPLÉMENTATION DE STYLES AU FORMAT MARKDOWN

Le format Markdown est un format texte qui peut être facilement converti en HTML. On peut donc écrire un document en texte utilisant la syntaxe Markdown (beaucoup plus simple que HTML), pour ensuite le transformer en HTML en utilisant un outil de conversion). Il existe plusieurs éditeurs Markdown qui permettent de formater du texte au format Markdown, selon le style désiré, par exemple : Titre 1, titre 2, titre 3, liste numérotée, liste à puces, hyperlien, etc. Ces éditeurs permettent aussi souvent de visualiser le Markdown en HTML stylisé.

Si ça vous intéresse :

Référence pour la syntaxe Markdown : https://daringfireball.net/projects/markdown/syntax

Dingus, pour visualiser du Markdown converti en HTML : https://daringfireball.net/projects/markdown/dingus

Pour ce travail, vous devez implémenter sept (7) styles Markdown qui transforment du texte non formaté en texte formaté selon la syntaxe Markdown. Plus précisément, vous devez implémenter une hiérarchie de classes où la classe de base (`StyleMD`) est abstraite, et où chaque sous-classe (concrète) implémente un style particulier (voir figure 1). Chaque style concret sera expliqué dans les sections suivantes.

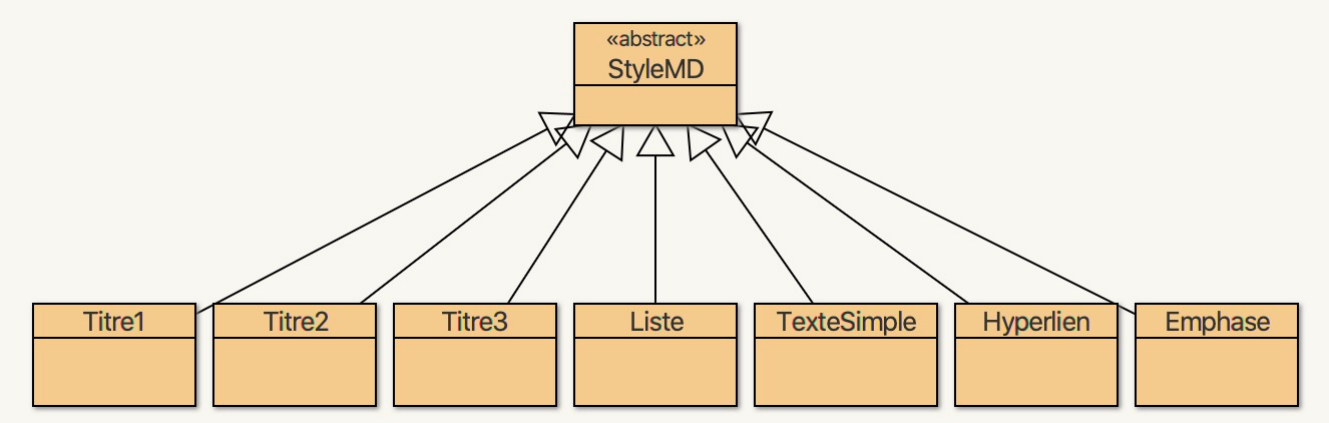

Figure 1

1.1.1 La classe abstraite `StyleMD`

Vous devez d'abord concevoir une classe abstraite nommée `StyleMD` contenant les membres suivants :

1) Deux constantes de classe publiques

Nom de la constante	Type	Valeur d'initialisation et description
`BLOC`	`int`	Initialisée à 0. Cette constante désigne une disposition de type `BLOC` pour un élément formaté en Markdown.
`LIGNE`	`int`	Initialisé à 1. Cette constante désigne une disposition de type `LIGNE` pour un élément formaté en Markdown.

NOTE : Une disposition de type `LIGNE` pour un élément de texte formaté en Markdown signifie qu'on veut enchaîner cet élément à l'élément précédent (sans commencer de nouveau paragraphe). Une disposition de type `BLOC` signifie qu'on veut commencer un nouveau paragraphe.

2) Un attribut d'instance

Nom de l'attribut	Type	Description
`disposition`	`int`	Indique la disposition de ce `StyleMD`. Doit être `LIGNE` ou `BLOC`.

Cet attribut est commun à tous les styles.

3) Un constructeur public avec le paramètre suivant

Nom du paramètre	Type	Description
`disposition`	`int`	La disposition pour ce `StyleMD`. Doit être `LIGNE` ou `BLOC`.

Ce constructeur initialise l'attribut d'instance `disposition` avec la valeur passée en paramètre. Si le paramètre `disposition` est invalide (n'est pas égal à `LIGNE` ou `BLOC`), on ignore le paramètre, et l'on initialise l'attribut `disposition` à `BLOC`.

4) Une méthode abstraite publique

Nom méthode : `formater`
Type retour : `String`

Paramètre	Type	Description
`texte`	`String`	Le texte à formater dans ce style Markdown.

La redéfinition de cette méthode (dans chaque sous-classe) retourne le format Markdown du texte donné en paramètre (dans le style représenté par la sous-classe).

5) Deux méthodes d'instance publiques

Nom méthode : `getDisposition`
Type retour : `int`
Paramètre : aucun

Ce *getter* retourne simplement la valeur de l'attribut `disposition`.

Nom méthode : `equals` (redéfinition)
Type retour : `boolean`

Paramètre	Type	Description
`autreStyle`	`Object`	L'autre style avec lequel comparer ce style.

Cette méthode est la **redéfinition** de la méthode `equals` (de la classe `Object`). Vous devez l'implémenter en respectant les règles d'implémentation spécifiées dans la Javadoc de la classe `Object`.

Deux objets de type `StyleMD` sont considérés comme égaux s'ils ont la même `disposition`.

---

6) De plus :

Dans la classe `StyleMD`, vous pouvez ajouter les éléments suivants, si vous le jugez pertinent, mais RIEN D'AUTRE :

- Constantes de classe
- Méthodes **protégées** - **méthodes communes** à deux ou plusieurs classes dérivées, pour éviter la répétition du code.

Dans les sections suivantes, on vous explique comment définir les sept (7) sous-classes concrètes dérivées de la classe `StyleMD`. Dans ces classes, qui implémentent différents styles (`Titre1`, `Titre2`, `Titre3`, `Liste`, `TexteSimple`, `Hyperlien`, et `Emphase`), vous devrez, entre autres, redéfinir la méthode `formater`.

ATTENTION :

Voici des spécifications à respecter pour la redéfinition de la méthode **`formater`** **dans toutes les sous-classes** :

1. Avant de faire le formatage en Markdown,
   - si le `texte` reçu en paramètre est `null`, vous devez le remplacer par la chaine de caractères `"null"`.
   - VOUS DEVEZ aussi éliminer les caractères blancs au début et à la fin du `texte` donné en paramètre, en utilisant la méthode `trim()` de la classe `String` : `texte = texte.trim();`
2. Lorsque la disposition d'un style est de type `BLOC`, il faut ajouter un saut de ligne (`'\n'`) **avant**, et un saut de ligne **après** le texte formaté en Markdown.
3. Lorsque la disposition d'un style est de type `LIGNE`, il faut ajouter un saut de ligne (`'\n'`) seulement **après** le texte formaté en Markdown.

1.1.2 La classe `Titre1`

Vous devez implémenter la classe `Titre1` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) Un constructeur public sans paramètre

Ce constructeur construit un `Titre1` dont la disposition est de type `BLOC`.

2) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `Titre1`. Au format Markdown, le style `Titre1` peut être spécifié en soulignant le `texte` avec une ligne formée de caractères `'='`.

Exemples :

Valeur du paramètre `texte`	**Texte formaté en Markdown**
`"Ceci est un titre 1"`	`"Ceci est un titre 1\n==================="`
`null`	`"null\n===="`

Note : dans le cadre de ce TP, la ligne formée de caractère `'='` doit être de même longueur que le texte qui est souligné.

N'oubliez pas que comme la disposition d'un `Titre1` est (toujours) de type `BLOC`, vous devez aussi ajouter un saut de ligne (`'\n'`) avant, et un saut de ligne après le texte formaté en Markdown. Dans le premier exemple donné ci-dessus, la chaine retournée par la méthode `formater` serait :

`"`**`\n`**`Ceci est un titre 1\n===================`**`\n`**`"`

N'oubliez pas non plus que lorsque, dans d'autres cas, la disposition est de type `LIGNE`, vous devez ajouter un saut de ligne seulement après le texte formaté en Markdown. Cette spécification ne sera pas répétée explicitement dans les sections suivantes, mais doit être respectée implicitement.

1.1.3 La classe `Titre2`

Vous devez implémenter la classe `Titre2` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) Un constructeur public sans paramètre

Ce constructeur construit un `Titre2` dont la disposition est de type `BLOC`.

2) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `Titre2`. Au format Markdown, le style `Titre2` peut être spécifié en soulignant le `texte` avec une ligne formée de caractères `'-'`.

Exemple :

Valeur du paramètre `texte`	Texte formaté en Markdown
`"Ceci est un titre 2"`	`"Ceci est un titre 2\n------------------"`

Note : dans le cadre de ce TP, la ligne formée de caractères `'-'` doit être de même longueur que le texte qui est souligné.

1.1.4 La classe `Titre3`

Vous devez implémenter la classe `Titre3` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) Un constructeur public sans paramètre

Ce constructeur construit un `Titre3` dont la disposition est de type `BLOC`.

2) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `Titre3`. Au format Markdown, le style `Titre3` peut être spécifié en ajoutant `"### "` avant le `texte`.

Exemple :

Valeur du paramètre `texte`	Texte formaté en Markdown
`"Ceci est un titre 3"`	`"### Ceci est un titre 3"`

1.1.5 La classe `Liste`

Vous devez implémenter la classe `Liste` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) Un attribut d'instance qui détermine si c'est une liste numérotée ou une liste à puces.

Nom de l'attribut	Type	Description
`listeNumerotee`	`boolean`	Cet attribut est `true` si c'est une liste numérotée, `false` si c'est une liste à puces.

2) Un constructeur public avec le paramètre suivant

Nom du paramètre	Type	Description
`listeNumerotee`	`boolean`	Le type de liste (numérotée (`true`) ou à puces (`false`)) pour cette `Liste`.

Ce constructeur construit une `Liste` dont la disposition est (toujours) de type `BLOC`, et l'attribut `listeNumerotee` de cette `Liste` est initialisé avec la valeur du paramètre.

3) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `Liste`.

- Si c'est une liste à puces :
  Au format Markdown, on peut indiquer une liste (à puces) d'éléments en insérant une étoile (`*`), suivi d'un espace, au début de chaque élément de la liste. Il s'agit donc, ici, d'ajouter la chaine `"* "` au début du `texte` donné, et de remplacer ensuite chaque caractère `'\n'` dans le `texte` donné par la chaine `"\n* "`.

- Si c'est une liste numérotée:
  Au format Markdown, on peut indiquer une liste (numérotée) d'éléments en insérant un nombre `nbr`, suivi d'un point, suivi d'un espace, au début de chaque élément de la liste. Dans le cadre de ce TP, `nbr` doit être égal à 1 devant le premier item de la liste, puis égal à 2, devant le deuxième item de la liste, et ainsi de suite jusqu'à `nbrElem` où `nbrElem` est le nombre d'éléments dans la liste. Il s'agit donc, ici, d'ajouter la chaine `"1. "` au début du `texte` donné, et de remplacer ensuite chaque caractère `'\n'` dans le `texte` donné par la chaine `"\nnbr. "`.

Exemples :

Liste à puces

Valeur du paramètre `texte`	Texte formaté en Markdown pour une liste à puces
`"item1\nitem2\nitem3"`	`"* item1\n* item2\n* item3"`

Liste numérotée

Valeur du paramètre `texte`	Texte formaté en Markdown pour une liste à puces
`"item1\nitem2\nitem3\nitem4"`	`"1. item1\n2. item2\n3. item3\n4. item4"`

4) La définition du *getter* suivant

Nom méthode : `isListeNumerotee`
Type retour : `boolean`
Paramètre : aucun

Cette méthode retourne simplement la valeur de l'attribut `listeNumerotee`.

5) La redéfinition de la méthode `equals`

Nom méthode : `equals`
Type retour : `boolean`

Paramètre	Type	Description
`autreListe`	`Object`	L'autre liste avec laquelle comparer cette liste.

Cette méthode est la **redéfinition** de la méthode `equals` (de la classe `StyleMD`). Vous devez l'implémenter en respectant les règles d'implémentation spécifiées dans la Javadoc de la classe `Object`.

Deux objets de type `Liste` sont considérés comme égaux si tous leurs attributs (`disposition` et `listeNumerotee`) sont égaux.

1.1.6 La classe `TexteSimple`

Vous devez implémenter la classe `TexteSimple` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) Un constructeur public avec le paramètre suivant

Nom du paramètre	Type	Description
`disposition`	`int`	La disposition pour ce `TexteSimple`. Doit être `LIGNE` ou `BLOC`.

Un `TexteSimple` peut être de disposition `BLOC` ou `LIGNE`. Ce constructeur initialise donc l'attribut d'instance `disposition` avec la valeur passée en paramètre. Si le paramètre `disposition` est invalide (n'est pas égal à `LIGNE` ou `BLOC`), on ignore le paramètre, et l'on initialise l'attribut `disposition` à `BLOC`.

2) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `TexteSimple`. Dans le cadre de ce TP, nous dirons qu'un texte simple est un texte où l'on a remplacé tous les sauts de lignes `'\n'` par un saut de ligne HTML (`<br/>`).

Exemple :

Valeur du paramètre `texte`	Texte formaté en Markdown
`"Ligne 1\nLigne 2\nLigne 3"`	`"Ligne 1 Ligne 2 Ligne 3"`

1.1.7 La classe `Hyperlien`

Vous devez implémenter la classe `Hyperlien` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) L'attribut d'instance suivant

Nom de l'attribut	Type à la déclaration	Description
`url`	`String`	L'URL de la ressource ciblée par cet hyperlien.

2) Un constructeur public avec les deux paramètres suivants

Nom du paramètre	Type	Description
`url`	`String`	L'URL de la ressource ciblée par cet `Hyperlien`.
`disposition`	`int`	La disposition pour cet `Hyperlien`. Doit être `LIGNE` ou `BLOC`.

Ce constructeur construit un `Hyperlien` en initialisant ses attributs `url` et `disposition` avec les valeurs données en paramètres. Cependant, si le paramètre `disposition` est invalide (n'est pas égal à `LIGNE` ou `BLOC`), on ignore le paramètre, et l'on initialise l'attribut `disposition` à `BLOC`. Si le paramètre `url` est `null` ou si `url.trim()` est vide, on ignore le paramètre, et l'on assigne la valeur `"http://localhost"` à l'attribut `url`.

3) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `Hyperlien`.

Au format Markdown, on peut construire un hyperlien en donnant le `texte` du lien entre crochets, suivi de l'URL entre parenthèses.

Exemple avec l'attribut `url` égal à `"https://www.google.com"` :

Valeur du paramètre `texte`	Texte formaté en Markdown
`"GOOGLE"`	`"[GOOGLE](https://www.google.com)"`

4) La définition du *getter* suivant

Nom méthode : `getUrl`
Type retour : `String`
Paramètre : aucun

Cette méthode retourne simplement la valeur de l'attribut `url`.

5) La redéfinition de la méthode `equals`

Nom méthode : `equals`
Type retour : `boolean`

Paramètre	Type	Description
`autreHyperlien`	`Object`	L'hyperlien avec lequel comparer cet hyperlien.

Cette méthode est la **redéfinition** de la méthode `equals` (de la classe `StyleMD`). Vous devez l'implémenter en respectant les règles d'implémentation spécifiées dans la Javadoc de la classe `Object`.

Deux objets de type `Hyperlien` sont considérés comme égaux si tous leurs attributs (`disposition` et `url`) sont égaux.

1.1.8 La classe `Emphase`

Vous devez implémenter la classe `Emphase` qui hérite de `StyleMD`, et qui contient les membres suivants :

1) Un constructeur public sans paramètre

Ce constructeur construit un objet `Emphase` dont la disposition est (toujours) de type `LIGNE`.

2) La redéfinition de la méthode `formater`

Cette méthode retourne le format Markdown du `texte` donné en paramètre, dans le style `Emphase`. Au format Markdown, le style `Emphase` consiste à mettre l'emphase sur un texte donné (le mettre en gras), et peut être spécifié en ajoutant la chaine `"**"` avant et après le `texte` donné.

Exemple :

Valeur du paramètre `texte`	Texte formaté en Markdown
`"Ce texte est important"`	`"**Ce texte est important**"`

1.1.9 Précisions sur l'implémentation des sept (7) sous-classes concrètes précédentes

Dans les sous-classes de la classe `StyleMD`, outre ce qui est spécifié, vous pouvez ajouter des constantes de classe, mais rien d'autre.

Aussi, assurez-vous de mettre ce qui est commun à une ou plusieurs sous-classes dans la superclasse afin d'éviter la répétition de code inutile.

1.1.10 Tests de la classe `SytleMD` (et ses sous-classes)

La classe `TestsStyleMD` est une classe de tests qui vous est fournie pour tester la hiérarchie de classes `StyleMD`. Pour exécuter les tests, importez cette classe dans votre projet, et exécutez sa méthode `main`. La note sur 25 affichée à la console correspond à la note que vous obtiendrez pour les tests de ces huit (8) classes.

Assurez-vous d'avoir bien testé, et débogué vos classes de la partie 1 AVANT de faire la partie 2.

1.2 PARTIE 2 – DOCUMENT MARKDOWN

La partie 2 de ce travail consiste à implémenter deux classes qui serviront à modéliser un document au format Markdown.

1.2.1 La classe ElementTextuelMD

Cette classe modélise un élément de texte devant être formaté dans un style Markdown donné. Elle contient les membres suivants.

1) Les deux attributs d'instance suivants

Nom de l'attribut	Type à la déclaration	Description
`texte`	`String`	L'élément textuel à formater.
`style`	`StyleMD`	Le style Markdown dans lequel on veut formater le `texte`.

2) Un constructeur public avec les deux paramètres suivants

Nom du paramètre	Type	Description
`texte`	`String`	Le texte de l'objet `ElementTextuelMD` à créer.
`style`	`StyleMD`	Le style Markdown de l'objet `ElementTextuelMD` à créer.

Ce constructeur construit un `ElementTextuelMD` en initialisant ses attributs `texte` et `style` par les valeurs correspondantes données en paramètres.

Exceptions levées :

- Lève une `ElementTextuelMDInvalideException` si le `texte` donné en paramètre est invalide (`texte` est `null` ou `texte.trim()` est vide).
- Lève une `StyleMDInvalideException` si le `texte` est valide, mais que le `style` est `null`.

Lorsqu'une exception est levée, l'objet n'est pas créé.

3) Les cinq (5) méthodes d'instance suivantes

Nom méthode : `getTexte` (*getter*)
Type retour : `String`
Paramètre : aucun

Cette méthode retourne simplement la valeur de l'attribut `texte`.

Nom méthode : `getStyle` (*getter*)
Type retour : `StyleMD`
Paramètre : aucun

Cette méthode retourne simplement la valeur de l'attribut `style`.

Nom méthode : `setTexte` (*setter*)
Type retour : `void`

Paramètre	Type	Description
`texte`	`String`	Le nouveau `texte` à assigner à cet objet

Cette méthode permet de modifier l'attribut `texte` de cet `ElementTextuelMD` par le `texte` donné en paramètre.

Exception levée :

- Lève une `ElementTextuelMDInvalideException` si le `texte` donné en paramètre est invalide (`texte` est `null` ou `texte.trim()` est vide).

La modification n'est pas effectuée lorsqu'une exception est levée.

Nom méthode : `setStyle` (*setter*)
Type retour : `void`

Paramètre	Type	Description
`style`	`StyleMD`	Le nouveau `style` à assigner à cet objet

Cette méthode permet de modifier l'attribut `style` de cet `ElementTextuelMD` par le `style` donné en paramètre.

Exception levée :

- Lève une `StyleMDInvalideException` si le `style` donné en paramètre est invalide (`style` est `null`).

La modification n'est pas effectuée lorsqu'une exception est levée.

Nom méthode : `toString` (redéfinition)
Type retour : `String`
Paramètre : aucun

Cette méthode retourne le `texte` de cet `ElementTextuelMD` formaté selon le `style` de cet `ElementTextuelMD`.

Cette méthode n'est en fait qu'un appel à la méthode `formater` du `style` de cet objet.

1.2.2 La classe DocumentMD

Cette classe modélise un document Markdown. Un tel document est composé d'une liste ordonnée d'`ElementTextuelMD`. Notez que le mot "ordonnée" ici ne veut pas dire "triée", mais plutôt que les éléments dans la liste ont une position (0 à nbr éléments - 1) bien particulière et significative. Nous utiliserons une `ArrayList` pour stocker les éléments textuels Markdown. Cette classe doit contenir les membres suivants.

1) L'attribut d'instance suivant

Nom de l'attribut	Type à la déclaration	Description
`elements`	`ArrayList<ElementTextuelMD>`	La liste ordonnée des éléments textuels Markdown qui constitue ce document.

2) Un constructeur public sans paramètre

Ce constructeur instancie la liste `elements` qui doit être vide. Ce peut être le constructeur par défaut (pseudoconstructeur).

3) Les méthodes d'instance publiques suivantes

Nom méthode : `taille`
Type retour : `int`
Paramètre : aucun

Cette méthode retourne le nombre d'éléments textuel Markdown dans ce document (c'est le nombre d'éléments dans la liste `elements`).

Nom méthode : `ajouterElementTextuel`
Type retour : `void`

Paramètre	Type	Description
`eltTexteMD`	`ElementTextuelMD`	L'élément textuel Markdown à ajouter à la fin de ce document.

Cette méthode permet d'ajouter l'élément textuel Markdown `eltTextMD` à la fin de ce document (c.-à-d. à la fin de la liste `elements`).

Nom méthode : `ajouterElementTextuel`
Type retour : `void`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, où l'on veut ajouter `eltTexteMD`.
`eltTexteMD`	`ElementTextuelMD`	L'élément textuel Markdown à ajouter dans ce document, à la `position` donnée.

Cette méthode permet d'ajouter `eltTexteMD` donné, à la `position` donnée, dans ce document (c.-à-d. dans la liste `elements`). Après l'ajout, la position des éléments qui étaient aux `positions [position..taille()-1[` (avant l'ajout) aura augmentée de 1 (décalage de ces éléments d'une position vers la droite).

Exceptions levées :

- Lève une `ElementTextuelMDInvalideException` si `eltTexteMD` est `null`.
- Lève une `IndexOutOfBoundsException` si `eltTexteMD` n'est pas `null`, mais que la `position` est invalide (`position < 0 || position > taille()`)

Lorsqu'une exception est levée, l'ajout n'est pas effectué.

Nom méthode : `supprimerElementTextuel`
Type retour : `ElementTextuelMD`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, de l'élément textuel Markdown à supprimer.

Cette méthode permet de supprimer l'élément textuel Markdown se trouvant à la `position` donnée, dans ce document (c.-à-d. se trouvant à cette `position` dans la liste `elements`), et retourne l'élément supprimé. Après la suppression, la position des éléments qui étaient aux `positions [position + 1..taille()-1[` (avant la suppression) aura diminuée de 1 (décalage de ces éléments d'une position vers la gauche).

Exception levée :

- Lève une `IndexOutOfBoundsException` si la `position` est invalide (`position < 0 || position >= taille()`)

Lorsqu'une exception est levée, la suppression n'est pas effectuée.

Nom méthode : `remplacerElementTextuel`
Type retour : `ElementTextuelMD`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, de l'élément textuel Markdown à remplacer par `eltTexteMD`.
`eltTexteMD`	`ElementTextuelMD`	L'élément textuel Markdown avec lequel on veut remplacer l'élément à la `position` donnée.

Cette méthode permet de remplacer un élément textuel Markdown existant à la `position` donnée, dans ce document (c.-à-d. se trouvant à cette `position` dans la liste `elements`), par celui passé en paramètre (`eltTexteMD`). De plus, la méthode retourne l'élément qui a été remplacé par `eltTexteMD`.

Exceptions levées :

- Lève une `ElementTextuelMDInvalideException` si `eltTexteMD` est `null`.
- Lève une `IndexOutOfBoundsException` si `eltTexteMD` n'est pas `null`, mais que la `position` est invalide (`position < 0 || position >= taille()`)

Lorsqu'une exception est levée, le remplacement n'est pas effectué.

Nom méthode : `obtenirElementTextuel`
Type retour : `ElementTextuelMD`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, de l'élément textuel Markdown qu'on veut obtenir.

Cette méthode retourne l'élément textuel Markdown existant à la `position` donnée, dans ce document (c.-à-d. se trouvant à cette `position` dans la liste `elements`).

Exception levée :

- Lève une `IndexOutOfBoundsException` si la `position` est invalide (`position < 0 || position >= taille()`)

Lorsqu'une exception est levée, l'élément n'est pas retourné.

Nom méthode : `modifierStyle`
Type retour : `void`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, de l'élément textuel Markdown dont on veut modifier le style par le `style` donné.
`style`	`StyleMD`	Le nouveau style qu'on veut assigner à l'élément textuel Markdown se trouvant à la `position` donnée, dans ce document.

Cette méthode permet de modifier le style de l'élément textuel Markdown existant à la `position` donnée, dans ce document (c.-à-d. se trouvant à cette `position` dans la liste `elements`), par le `style` donné.

Exceptions levées :

- Lève une `IndexOutOfBoundsException` si la `position` est invalide (`position < 0 || position >= taille()`)
- Lève une `StyleMDInvalideException` si la position est valide, mais que le `style` donné est `null`.

Lorsqu'une exception est levée, la modification du style n'est pas effectuée.

Nom méthode : `modifierTexte`
Type retour : `void`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, de l'élément textuel Markdown dont on veut modifier le texte par le `texte` donné.
`texte`	`String`	Le nouveau texte qu'on veut assigner à l'élément textuel Markdown se trouvant à la `position` donnée, dans ce document.

Cette méthode permet de modifier le `texte` de l'élément textuel Markdown existant à la `position` donnée, dans ce document (c.-à-d. se trouvant à cette `position` dans la liste `elements`), par le `texte` donné.

Exceptions levées :

- Lève une `IndexOutOfBoundsException` si la `position` est invalide (`position < 0 || position >= taille()`)
- Lève une `ElementTextuelMDInvalideException` si la `position est valide`, mais que le `texte` donné est invalide (`texte` est `null` ou `texte.trim()` est vide).

Lorsqu'une exception est levée, la modification du texte n'est pas effectuée.

Nom méthode : `obtenirMarkdown`
Type retour : `String`

Paramètre	Type	Description
`position`	`int`	La position, dans ce document, de l'élément textuel Markdown dont on veut obtenir le texte au format Markdown.

Cette méthode retourne le texte formaté selon le style Markdown de l'élément textuel Markdown existant à la `position` donnée, dans ce document (c.-à-d. se trouvant à cette `position` dans la liste `elements`).

Utilisez la méthode `toSring` de l'élément textuel Markdown en position `position`.

Exception levée :

- Lève une `IndexOutOfBoundsException` si la `position` est invalide (`position < 0 || position >= taille()`)

Lorsqu'une exception est levée, la modification du texte n'est pas effectuée.

Nom méthode : `toString` (redéfinition)
Type retour : `String`
Paramètre : aucun

Cette méthode retourne une chaine formée de la concaténation de tous les éléments textuels Markdown de ce document (formaté selon leur style Markdown), dans l'ordre (en commençant par l'élément en position 0 dans la liste `elements` jusqu'à l'élément à la fin de cette liste). Un document vide (dont la liste `elements` est vide) retourne une chaine vide.

1.2.3 Tests des classes `ElementTextuelMD` et `DocumentMD`

La casse `RecetteCrepesMD` est une classe de tests PARTIELS qui vous est fournie. Exécutez sa méthode `main`, et vérifiez que ce qui est affiché à la console (une recette de crêpe, entre autres) est identique à ce qui est montré en commentaire au bas de cette classe. Cette classe ne teste pas toutes les méthodes ni tous les cas de tests dans les méthodes. **Vous devez faire en plus vos propres tests pour tester correctement ces deux classes**.

1.3 PRÉCISIONS

- Dans toutes les classes à faire, les noms et types de tous les attributs ou constantes, les noms de méthodes, l'ordre et le type des paramètres spécifiés doivent être respectés à la lettre.
- Vous devez respecter le **principe d'encapsulation** des données.
- N'oubliez pas d'écrire la **Javadoc**.
- Vous ne DEVEZ PAS modifier les classes d'exception fournies, car vos classes seront testées avec ces classes, telles quelles.
- Utilisez des constantes autant que possible.
- Il ne doit y avoir aucun affichage dans vos méthodes (pas de System.out)
- Toutes les classes doivent se trouver dans le **paquetage par défaut**.
- L'utilisation des expressions lambda et des streams (qu'on n'a pas vu(e)s encore) est interdite.
- Vous DEVEZ respecter le style Java.
- Vos fichiers à remettre doivent être encodés en UTF8.
- Votre code doit compiler et s'exécuter avec le JDK 8.

- Notez que ce travail ne comporte pas d'application Java, mais vous devrez composer des tests pour vous assurer du bon fonctionnement de vos méthodes.
- N'oubliez pas d'écrire (entre autres) votre nom complet et votre code permanent dans l'entête de vos classes.

Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé de ce TP est susceptible d'engendrer une perte de points.

Note : Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.

2. Détails sur la correction

2.1 LA QUALITÉ DU CODE (40 POINTS)

Concernant les critères de correction du code, lisez attentivement le document "Critères généraux de correction du code Java". De plus, votre code sera noté sur le respect des conventions de style Java vues en classe (dont un résumé se trouve dans le document "Conventions de style Java"). **Ces deux documents peuvent être téléchargés dans la section TRAVAUX PRATIQUES (ET BOITES DE REMISE) de la page Moodle du cours**.

Note : Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, ainsi que sur le respect des spécifications / consignes mentionnées dans ce document.

2.2 L'EXÉCUTION (60 POINTS)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

partie 1 (25 points)

Classe abstraite `StyleMD` et ses sous-classes.

La classe de tests vous est fournie pour la partie 1. Servez-vous-en !

partie 2 (35 points)

Classes `ElementTextuelMD` et `DocumentMD`

Une classe de tests PARTIELS vous est fournie (servez-vous-en), mais vous devrez aussi composer vos propres tests pour tester correctement ces deux classes. Les tests qui seront exécutés pour la correction de la partie 2 seront différents de ceux fournis.

Note : Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possibles.

3. Date et modalités de remise

3.1 REMISE

Date de remise : Au plus tard le **18 février 2022** à 23h59.

Les 10 fichiers à remettre :

1. StyleMD.java
2. Emphase.java
3. Hyperlien.java
4. Liste.java
5. TexteSimple.java
6. Titre1.java
7. Titre2.java
8. Titre3.java
9. ElementTextuelMD.java
10. DocumentMD.java

Remettre les fichiers séparément, NE PAS LES ARCHIVER (PAS DE .zip, .rar, etc.).
NE PAS REMETTRE tous les fichiers du projet, seulement les classes énumérées ci-dessus.

VÉRIFIEZ BIEN QUE VOUS AVEZ REMIS LES BONS FICHIERS SUR MOODLE (les .java et non les .class, par exemple).

Remise via Moodle uniquement.

Vous devez remettre (téléverser) vos fichiers sur le site du cours (Moodle). Vous trouverez la boite de remise dans la section **TRAVAUX PRATIQUES (ET BOITES DE REMISE).**

3.2 POLITIQUE CONCERNANT LES RETARDS

Une pénalité de 10% de la note finale, par jour de retard, sera appliquée aux travaux remis après la date limite. La formule suivante sera utilisée pour calculer la pénalité pour les retards : `Nbr points de pénalité = m / 144`, où `m` est le nombre de minutes de retard par rapport à l'heure de remise. Ceci donne 10 points de pénalité pour 24 heures de retard, 1.25 point de pénalité pour 3 heures, etc.

Aucun travail ne sera accepté après 1 jour (24 h) de retard, et la note attribuée sera 0.

3.3 REMARQUES GÉNÉRALES

- Aucun fichier reçu par courriel ne sera accepté. **En d'autres termes, un travail reçu par courriel sera considéré comme non remis.**
- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Moodle, Dropbox, Google Drive, One Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire (entre autres) votre nom complet, et votre code permanent dans l'entête de la classe à remettre.**
- **N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus!**

Ce travail est strictement individuel. Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL !