1 Documentation de la classe `String`

Voici un extrait de la documentation de la classe `String` :

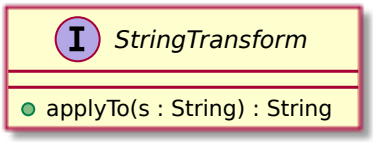
```java
/**
 * Returns a string that is a substring of this string. The
 * substring begins with the character at the specified index and
 * extends to the end of this string.
 * Examples:
 * "unhappy".substring(2) returns "happy"
 * "Harbison".substring(3) returns "bison"
 * "emptiness".substring(9) returns "" (an empty string)
 *
 * @param      beginIndex   the beginning index, inclusive.
 * @return     the specified substring.
 */
public String substring(int index){/* code */}
/**
 * Returns a string that is a substring of this string. The
 * substring begins at the specified {@code beginIndex} and
 * extends to the character at index {@code endIndex - 1}.
 * Thus the length of the substring is {@code endIndex-beginIndex}.
 * Examples:
 * "hamburger".substring(4, 8) returns "urge"
 * "smiles".substring(1, 5) returns "mile"
 *
 * @param      beginIndex   the beginning index, inclusive.
 * @param      endIndex     the ending index, exclusive.
 * @return     the specified substring.
 */
public String substring(int beginIndex, int endIndex){/* code */}
/**
 * Converts all of the characters in this {@code String} to lower
 * case using the rules of the default locale.
 * @return  the {@code String}, converted to lowercase.
 */
public String toLowerCase(){/* code */}
/**
 * Converts all of the characters in this {@code String} to upper
 * case using the rules of the default locale.
 * @return  the {@code String}, converted to uppercase.
 */
public String toUpperCase(){/* code */}
```

Question 1 : D'après la documentation, quelle est la valeur de l'expression `"Goodenough".substring(4).substring(0,2).toUpperCase()` ?

Question 2 : La classe `String` contient deux méthodes nommées `substring`. À quelle condition a-t-on le droit de définir deux méthodes ayant le même nom ? Comment appelle-t-on cette notion ?

2 Interface `StringTransform`

Dans cet exercice on va considérer une interface `StringTransform` décrite par le diagramme suivant :

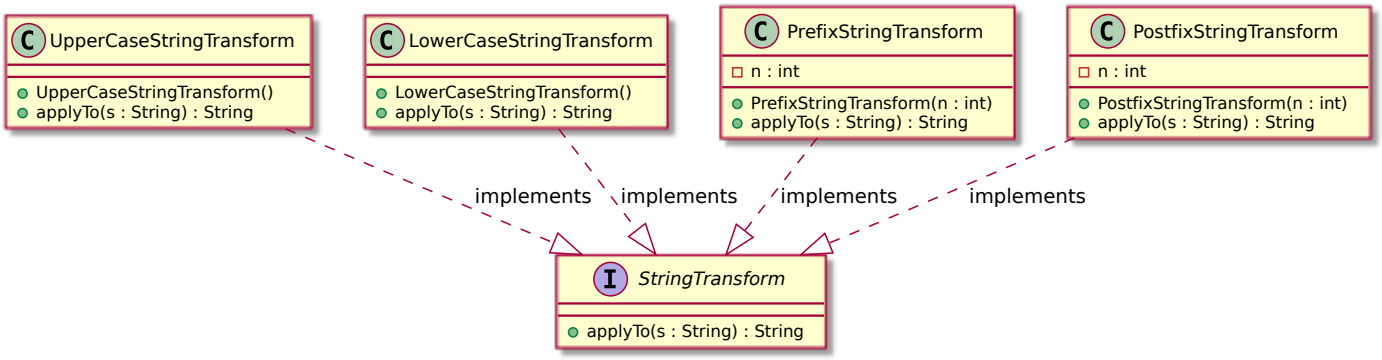


Les classes implémentant cette interface devront fournir une implémentation de la méthode `applyTo` qui produit une nouvelle chaîne de caractères issue de la transformation de la chaîne passée en argument.

Question 3 : Écrivez le code de l'interface `StringTransform`.

On considère les classes suivantes :

— `UpperCaseStringTransform` convertit les caractères de `s` en majuscules.
— `LowerCaseStringTransform` convertit les caractères de `s` en minuscules.
— `PrefixStringTransform` conserve les `n` premiers caractères de `s`.
— `PostfixStringTransform` conserve les `n` derniers caractères de `s`.

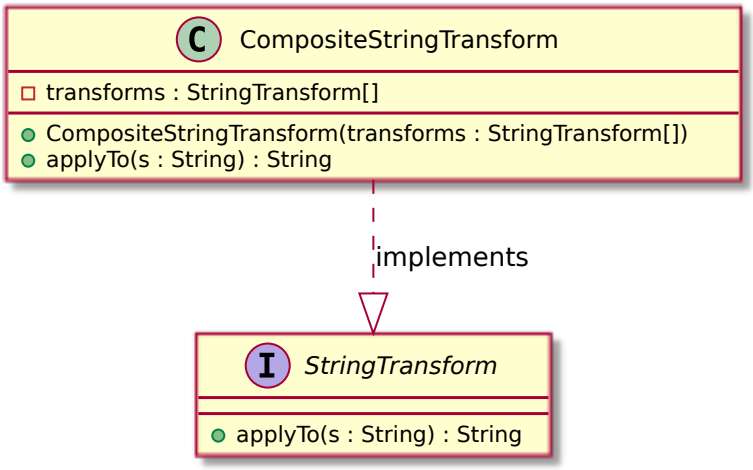


Question 4 : Écrivez le code des classes `UpperCaseStringTransform`, `LowerCaseStringTransform`, `PrefixStringTransform` et `PostfixStringTransform`.

On considère la classe `StringTransforms` avec la méthode de classe `String[] applyTransformToStrings(String[] strings, StringTransform transform)` qui applique la transformation `transform` aux chaînes du tableau `strings` et qui retourne un tableau contenant les chaînes de caractères issues de la transformation.

Question 5 : Écrivez le code de la classe `StringTransforms`.

On considère la classe `CompositeStringTransform` qui implémente l'interface `StringTransform` et qui applique successivement sur la chaîne `s` les transformations d'un tableau `StringTransform[] transforms` passé en argument du constructeur de la classe. Elle respecte le diagramme suivant :



Question 6 : Écrivez le code de la classe `CompositeStringTransform`.

3 Formes géométriques

On souhaite utiliser avoir les méthodes suivantes pour des formes géométriques (*shapes*) :

— `void draw(Painter painter, Color color)` : dessine la forme géométrique en utilisant le `painter` (la classe `Painter` utilisée pour dessiner) et la couleur donnés en argument.

— `void translate(double dx, double dy)` : applique la translation de vecteur (dx, dy) sur la forme géométrique.

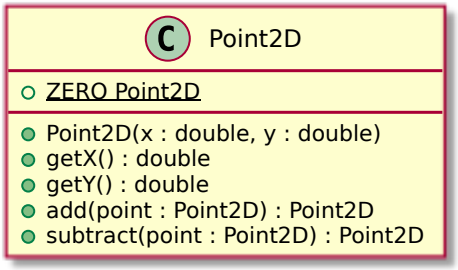
— `void rotate(double angle)` : applique la rotation centrée à l'origine de l'`angle` spécifié sur la forme géométrique.

On considère deux classes correspondant à des formes géométriques :

— `Polygon` : polygone construit à partir d'un tableau de points qui correspondront à ses sommets.

— `Disc` : construit à partir d'un centre et d'un rayon.

On supposera qu'on a à disposition une classe `Point2D` pour représenter des points dans le plan qui a le diagramme suivant :



Question 7 : Dessinez le diagramme des classes `Polygon` et `Disc` en ajoutant une interface (regroupant leurs méthodes communes) qu'elles devront implémenter.

On souhaite définir des classes correspondant à des transformations de forme géométriques :

— `Translation` : créée à partir de deux double dx et dy et correspond à la translation de vecteur (dx, dy).

— `OriginRotation` : créée à partir d'un `double angle`, et correspondant à la rotation de cet `angle` autour de l'origine.

— `Rotation` : créée à partir d'un `double angle` et d'un `Point2D center` et correspondant à la rotation de cet `angle` autour de `center`.

Contrairement aux transformations de `String` de l'exercice précédent, les transformations de formes géométriques devront modifier directement la forme.

Question 8 : Dessinez le diagramme des classes `Translation`, `OriginRotation` et `Rotation` en ajoutant une interface (regroupant leurs méthodes communes) qu'elles devront implémenter.

Pour réaliser une rotation d'`angle` de centre `center` (x, y), il faut appliquer au point les trois transformations suivantes dans cet ordre :

— une translation de vecteur $(-x, -y)$,

— une rotation d'`angle` ayant comme centre l'origine,

— une translation de vecteur (x, y).

Question 9 : Donner le code de la classe `Rotation` en supposant que les classes `Point2D`, `Translation` et `OriginRotation` sont déjà codées.