R1.01 – TP3(B)
Notions de classe - la classe String

L'objectif de ce TP est d'étendre vos connaissances relatives à la classe String et à son utilisation.

RAPPEL : le fichier UTILISATION_IJ accessible à partir de ce parcours, peut vous être utile…

1. Classe String - Rappels et compléments

LA CLASSE STRING REPRÉSENTE LES CHAÎNES DE CARACTÈRES.

Toutes les chaînes utilisées dans un programme java sont implémentées en tant qu'instances de cette classe.

✓ *DÉCLARATION D'UN OBJET DE TYPE* `String`

Déclarer une String équivaut à déclarer un tableau de caractères, accessibles individuellement par leur position, la 1ère position étant toujours 0.

EXEMPLE

```
String uneChaine = "Java";
// un nouvel objet de type String a été créé avec
// en position 0 : le caractère 'J', en position 3 : le caractère 'a'
```

✓ *LONGUEUR D'UNE CHAÎNE (son nombre de caractères) ET ACCÈS À UN DE SES CARACTÈRES :*

- *la méthode* `length()` *renvoie le nombre de caractères de la chaîne à laquelle elle est appliquée*
- *la méthode* `charAt(x)` *renvoie le caractère en position* `x` *dans la chaîne à laquelle elle est appliquée (ATTENTION :* `x` *doit être un entier compris entre* `0` *et* `length()-1`*)*

EXEMPLE

```
int nbCar = uneChaine.length();   // nbCar == 4
char unCar = uneChaine.charAt(2); // unCar == 'v'
```

✓ *COMPARAISON ENTRE DEUX CHAÎNES :*

Comparer deux chaînes revient à définir leur position respective dans l'ordre lexicographique, c'est-à-dire selon l'ordre d'encodage de leurs caractères (cf. Annexe en fin de ce document)

On ne peut pas utiliser les opérateurs de comparaison de valeurs numériques **<**, **<=**, **>** et **>=**.

Il faut utiliser la méthode `compareTo()` *ou la méthode* `compareToIgnoreCase()` *de la classe* `String`.

- *ces méthodes retournent un entier négatif, nul ou positif*
- *la comparaison est basée sur la valeur Unicode de chaque caractère* des chaînes comparées*
- *la méthode* `compareTo()` *prend en compte la casse (différenciation entre minuscules et majuscules), contrairement à la méthode* `compareToIgnoreCase()`

EXEMPLE

```
String uneAutreChaine = "ada";
int resComp = uneChaine.compareTo(uneAutreChaine); // resComp == -23
```

EXPLICATIONS

`ch1.compareTo(ch2)` retourne :

- un entier **< 0**, **si `ch1` précède `ch2`**, *dans l'ordre lexicographique*
- **0** , **si `ch1` et `ch2` sont identiques**
- un entier **> 0**, **si `ch1` suit `ch2`**, *dans l'ordre lexicographique*

Si `ch1` et `ch2` sont différentes et non vides :

Les caractères de `ch1` et ceux de `ch2` sont comparés un à un, à partir du premier.
La comparaison s'arrête dans l'un des cas (exclusifs) suivants :

- cas 1 *: égalité des caractères de même position dans* `ch1` *et* `ch2` *jusqu'au dernier de la chaîne la plus courte*
- cas 2 *: un caractère différent entre les deux chaînes a été rencontré*

Résultat de `compareTo` :

- cas 1 : (nombre de caractères de `ch1` – nombre de caractères de `ch2`)
- cas 2 : (code du dernier caractère comparé dans `ch1` - code du caractère de même position dans `ch2`)

Si `ch1` est vide, `compareTo` retourne l'opposé du nombre de caractères de `ch2`
Si `ch2` est vide, `compareTo` retourne le nombre de caractères de `ch2`

* *Un caractère est codé sur 2 octets – voir code décimal de certains caractères en annexe*

2. Classe Explore_Strings

- Placez-vous dans votre répertoire R1.01, ouvrez-y un terminal et lancez `IJ` avec la commande `idea`
- Sous `IJ`, créez un nouveau projet `TP3_B` (cf. fichier UTILISATION_IJ)

1.1. Dans le projet TP3_B, ajoutez une classe `Explore_Strings` et codez-y la fonction suivante :

```
private static int nbOccCar(String uneChaine, char unCar) {
    //{} => {résultat = nombre de fois où le caractère unCar
    //       apparaît dans la chaîne uneChaine}
```

1.2. Créez sous cette fonction, une procédure `main`, dans laquelle vous ajouterez :

- ✓ la déclaration d'une variable de type `String`, nommée `lipogramme` et initialisée avec :
  `"Un rond pas tout à fait clos, fini par un trait horizontal";`
- ✓ la déclaration d'une variable de type `String`, nommée `lesVoyellesSansAccent` et initialisée avec : "aeiouy"
- ✓ les instructions conduisant à l'affichage du nombre de fois où chaque voyelle non accentuée, *en casse minuscule*, apparaît dans la chaîne `lipogramme`

1.3. Exécutez et testez

1.4. Dans la classe `Explore_Strings` pour chacune des fonctions dont l'en-tête est donné ci-dessous :

- ✓ copiez et collez au-dessus de la procédure `main` l'en-tête de la fonction
- ✓ codez cette fonction
- ✓ complétez le code de la procédure `main` avec les instructions permettant de les tester sur les chaînes suivantes :

```
String ch1 = "il y a huit mots dans cette phrase.";
String ch2 = "Il était une fois un petit bonhomme de Foix.";
```

- ✓ compilez, exécutez et vérifiez si vous obtenez les résultats attendus (*corrigez votre code sinon*)

FONCTIONS À DÉVELOPPER ET TESTER AU FUR ET À MESURE DE CETTE QUESTION

(a) *Nombre de majuscules sans accent dans une chaîne donnée*

```
private static int nbLettresMajSansAccent(String uneChaine) {
    // {}  => {résultat = nombre de lettres majuscules
    //         non accentuées dans la chaîne uneChaine}
```

(b) *Nombre de mots dans une chaîne donnée*

```
private static int nbMots(String uneChaine) {
    // {uneChaine ne contient que :
    //  * des lettres de l'alphabet (accentuées ou non)
    //  * des espaces
    //  * un point final}
    //  => {résultat = nombre de mots dans uneChaine
          Note : un mot est constitué de lettres de l'alphabet}
```

(c) *Nombre de caractères des mots les plus longs dans une chaîne donnée*

```
private static int longPlusLongsMots(String uneChaine) {
    // {uneChaine ne contient que :
    //  * des lettres de l'alphabet (accentuées ou non)
    //  * des espaces
    //  * un point final}
    //  => {résultat = nombre de lettres maximal des mots de uneChaine
          Note : un mot est constitué de lettres de l'alphabet}
```

3. Classe Compare_Strings

3.1. Création de la classe Compare_Strings et premiers tests de comparaison entre des chaînes

- ✓ Depuis un terminal, exécutez la commande `cp -r /users/info/pub/1a/R1.01/TP3B_Files ~/R1.01`
- ✓ Ouvrez avec un éditeur de texte, le fichier `Compare_Strings.txt` qui a été copié dans votre répertoire `R1.01/TP3B_Files` et copiez son contenu (CTRL + A *puis* CTRL + C)
- ✓ Dans le projet `TP3_B` créez une classe java `Compare_Strings`
- ✓ Placez le curseur à droite de la 1ère accolade, passez à la ligne et collez le contenu du presse-papiers (CTRL + V)
- ✓ Étudiez attentivement ce code : *déclarations*, *commentaires* et *instructions*
- ✓ Dans la section `// 1 - Premiers tests`, remplacez la chaîne `toBeReplaced` par le code attendu

  Exemple :

  > *Dans la ligne suivant le commentaire* `//(a) Casse prise en compte`
  > *il faut remplacer* `toBeReplaced` *par* `unChat.compareTo(unChien)`

- ✓ Exécutez et vérifiez que vous obtenez la trace-ci-dessous (*sinon, corrigez votre code*)

```
-----------------------------------------------------
Premiers tests de comparaison de chaînes
-----------------------------------------------------
Chaînes comparées...
    unChat --> "chat"
    unChien --> "chien"
    unPetitChat --> "chaton"
Comparaisons entre ces chaînes (casse prise en compte)
    unChat comparé à unChien --> -8
    unChien comparé à unPetitChat --> 8
    unPetitChat comparé à unChat ---> 2
    unPetitChat comparé à unPetitChat ---> 0
-----------------------------------------------------
```

- ✓ **Analyse de la trace…**
  - Quel est l'ordre lexicographique respectif des chaînes `unChat`, `unChien` et `unPetitChat` ?
  - Expliquez les résultats de chaque comparaison **(valeurs -8, 8, 2 et 0) -** cf. Annexe

3.2. Calcul du résultat de la comparaison entre deux chaînes saisies par l'utilisateur

OBJECTIF : Calculer le résultat de la comparaison entre deux chaînes sans utiliser la méthode compareTo

ALGORITHME :
- Saisie de deux chaînes
- Si au moins l'une d'elles est vide, calculer immédiatement le résultat de la comparaison (cf. page 1)
- Sinon :
  - monter une boucle **`while`** qui compare *tant que cela est possible*, un caractère de la 1ère chaîne avec celui de même position dans la 2ème chaîne, en partant des premiers caractères de ces chaînes
  - en sortie de boucle, déterminer le résultat de la comparaison selon le cas qui a provoqué l'arrêt de l'itération

NOTES (1) *La condition de maintien dans la boucle* **`while`** *devant porter sur 2 cas exclusifs (cf. page 1), il vous sera nécessaire d'employer l'opérateur* `&&` *entre ces deux cas*

Quand `(a && b)` *est évalué, si* `a` *est faux,* `b` *n'est pas évalué.*

(2) *Le code décimal d'un caractère* `unChar`*, s'obtient par conversion en entier :* `(int) unChar`

Dans la classe `Compare_Strings` – `// 2 - Comparaisons entre des chaînes ...`

- ✓ Complétez, ou modifiez les parties notées *`À COMPLÉTER`*, ou *`À MODIFIER`*
- ✓ Compilez, exécutez et vérifiez que le résultat de la comparaison par programme est bien égal à celui donné par le `compareTo` (cf. Dernières instructions de la classe `Compare_Strings`

ANNEXE : CODE DÉCIMAL DE CERTAINS CARACTÈRES EN JAVA

Caractères représentant les chiffres de 0 à 9

'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
48	49	50	51	52	53	54	55	56	57

Lettres minuscules non accentuées de l'alphabet français

'A'	'B'	'C'	'D'	'E'	'F'	'G'	'H'	'I'	'J'	'K'	'L'	'M'
65	66	67	68	69	70	71	72	73	74	75	76	77
'N'	'O'	'P'	'Q'	'R'	'S'	'T'	'U'	'V'	'W'	'X'	'Y'	'Z'
78	79	80	81	82	83	84	85	86	87	88	89	90

Lettres majuscules non accentuées de l'alphabet français

'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'j'	'k'	'l'	'm'
97	98	99	100	101	102	103	104	105	106	107	108	109
'n'	'o'	'p'	'q'	'r'	's'	't'	'u'	'v'	'w'	'x'	'y'	'z'
110	111	112	113	114	115	116	117	118	119	120	121	122

Voyelles accentuées (en minuscule ou majuscule)

'à'	'À'	'â'	'Â'	'ä'	'Ä'	'è'	'È'	'é'	'É'	'ê'	'Ê'	'ë'	'Ë'
224	192	226	194	228	196	232	200	233	201	234	202	235	203
'î'	'Ï'	'ï'	'Ï'	'ô'	'Ô'	'ö'	'Ö'	'ù'	'Ù'	'û'	'Û'	'ü'	'Ü'	'ÿ'
238	207	239	207	244	212	246	214	249	217	251	219	252	220	255

Lettres avec cédille, entrelacement ou tilde

'ç'	'Ç'	'æ'	'Æ'	'œ'	'Œ'	'ñ'	'Ñ'
231	199	230	198	339	338	241	209

Espace, ponctuation et autres symboles courants

''	'!'	'"'	'#'	'$'	'%'	'&'	'''	'('	')'	'*'	'+'	','
32	33	34	35	36	37	38	39	40	41	42	43	44
'-'	'.'	'/'	':'	';'	'<'	'='	'>'	'?'	'@'	'['	'\'	']'
45	46	47	58	59	60	61	62	63	64	91	92	93
'^'	'_'	'`'	'{'	'|'	'}'	'~'	'€'
94	95	96	123	124	125	126	8 364