Les expressions régulières

I. Définition :

Les regex permettent:

- Vérifier la syntaxe,
- Remplacer une partie d'une chaîne de caractères,
- Découper une chaîne.

Une regex se définit par une suite de motifs décrivant entièrement ou en partie le contenu. Le contenu peut être décrit en définissant:

- La position du motif
- Le type du motif: en utilisant une syntaxe explicite ou des raccourcis.
- Le nombre d'occurrence du motif en utilisant les quantificateurs.

II. Création d'un motifs :

Position du motif :

La position est définie directement par son emplacement dans la regex par rapport aux autres.
On peut s'aider des motifs:

Symbole	Signification	Exemple
^	Début de ligne	`"^2015"`: la chaine commence par "2015…"
$	Fin de ligne	`"2015$"`: la chaîne se termine par "…2015"

Utilisation des motifs de début et fin de ligne

Ne pas hésiter à utiliser le motif de début et de fin de ligne, par exemple:
"2015" et **"2015/12/04″** peuvent être validées par la même regex: **"[0-9]{4}" alors "^[0-9]{4}$"** est seulement valide pour **"2015"**.

Type de motif :

Echappement

Utiliser `"\"` pour "échapper" les caractères utilisés plus bas.

Par exemple:

Pour reconnaître la chaîne "Pourquoi?", on peut utiliser la regex `"Pourquoi\?"`.

Classe de caractères

Symbole	Signification	Exemple
()	Groupement ordonné de caractères	`"(aze)"` chaîne contenant "aze"
|	"ou" pour indiquer un groupement ou un autre	`"((aze)|(qsd))"` chaîne contenant "aze" ou "qsd"
-	Intervalle de caractères	`"[0-9]"`: un caractère de 0 à 9 `"[A-Z]"`: un caractère de A à Z en majuscule `"[A-Za-z]"`: un caractère de A à Z en majuscule ou en minuscule
[ ]	Ensemble de caractères possibles	`"[0-9]"`

`[^]`	Tout sauf l'ensemble de caractères	`"[^a-d]"`: tous les caractères sauf a, b, c ou d

Exemple de classes de caractères usuels:

Regex	Explication
`[a-z]`	Caractères minuscules
`[a-zA-Z]`	Caractères majuscules et minuscules
`[\w-[0-9_]]`	Caractères alphabétiques sans les caractères numériques

Raccourcis ou classes abrégées :

On peut utiliser des raccourcis pour simplifier la syntaxe des regex ou indiquer des motifs supplémentaires:

Raccourci	Signification	Equivalence
`.`	Désigne tous les caractères	
`\n`	Nouvelle ligne	
`\r`	Retour à la ligne	
`\t`	Caractère de tabulation	
`\s`	Caractère d'espacement (espace, tabulation, saut de page etc…)	`[\n\r\t]`
`\S`	Caractère n'étant pas un caractère d'espacement	`[^\n\r\t]`
`\d`	Un chiffre	`[0-9]`
`\D`	Caractère n'étant pas un chiffre	`[^0-9]`
`\w`	Caractère alphanumérique (les caractères accentués et spéciaux sont inclus)	`[a-zA-Z0-9_]`
`\W`	Caractère n'étant un caractère alphanumérique	`[^a-zA-Z0-9_]`

Quantificateurs :

On place le motif de quantification pour indiquer les répétitions, juste après le motif définissant la classe de caractères:

"h{6}" indique que le caractères "h" doit être répété 6 fois.

Symbole	Signification	Exemple
`+`	1 fois ou plus	`"(aze)+"`: la chaîne "aze" doit être utilisée 1 fois ou plus
`?`	0 ou 1 fois	`"(aze)?"`: la chaîne "aze" doit être utilisée 0 ou 1 fois
`*`	0 ou plusieurs fois	
`{x}`	Exactement x fois	`"(aze){5}"`: la chaîne "aze" doit être utilisée exactement 5 fois
`{x,}`	x fois ou plus	
`{x,y}`	Entre x et y fois	`"(aze){3,6}"`: la chaîne "aze" doit être utilisée entre 3 et 6

III. Utilisation des expressions régulières :

La classe **Regex** en .NET se trouve dans le namespace **System.Text.RegularExpressions**.
Une **regex** se définit sous forme de chaîne de caractères (avec des guillemets). L'utilisation du caractères d'échappement "\" doit être doublé ou il faut préfixer la chaîne avec @.

Exemple:

Pour définir la regex "^[\d]$" en .NET, il faut utiliser la chaîne "^[\\d]$" ou @"^[\d]$".

III.1 Validation d'une chaîne de caractères

La validation se fait avec "**Regex.IsMatch()**":

Exemple de Validation d'une adresse mail:

```
string mailAddress = "user@server.com";
Regex regex = new Regex(@"^([\w]+)@([\w]+)\.([\w]+)$");
bool isValid = regex.IsMatch(mailAddress);
```

III.2 Remplacement d'une chaîne de caractères

On peut remplacer une chaîne par une autre en utilisant "Regex.Replace".

Exemple:

```
string stringToCorrect = "yo monsieur...";
Regex regex = new Regex("(lut|salut|yo)");
string replacedString = regex.Replace(stringToCorrect, "bonjour");
```

Il est possible de remplacer seulement les 'N' premières occurrences d'une chaîne en utilisant la surcharge de la classe Regex: "Regex.Replace(stringToCorrect, "bonjour", N)".

III.2.1 Remplacement d'une chaîne par capture

Il est possible de définir des parties d'une chaîne, de nommer ces parties puis de les utiliser pour reconstruire la chaîne de remplacement. Pour définir une partie d'une chaine, on utilise des parenthèses de capture:

Définition des parenthèses de capture:

Les parties d'une chaîne ou "groupe" sont capturées à l'aide de parenthèses de capture: "(...)". Une succession de plusieurs parenthèses de capture définit plusieurs groupes.

Par exemple:

```
"(\w+)(\s)(\w+)"
```

La chaîne précédente comporte 3 groupes:

- **Groupe 1** définit par "(\w+)": capturant une suite de caractères,
- **Groupe 2** définit par "(\s)": capturant un caractère d'espacement,
- **Groupe 3** définit par "(\w+)": capturant une autre suite de caractères.

Ainsi on peut utiliser ces groupes en les désignant par:

- "$1" pour le groupe 1,
- "$2" pour le groupe 2 et
- "$3" pour le groupe 3.

On peut donc utiliser les groupes dans une même chaine. Si on utilise la chaîne de départ "un deux", la regex de capture "(\w+)(\s)(\w+)" et la chaîne de remplacement "$3$2$1", le résultat de la substitution sera: "deux un".

Autre exemple:

Si on utilise la regex de capture "([\w\-.]+)@([\w\-.]+)" qui possède 2 groupes et la chaine d'entrée: "yop@cdiese.fr", les 2 groupes seront:

- Groupe 1 définit par "([\w\-.]+)": "yop"
- Groupe 2 définit par "([\w\-.]+)": "cdiese.fr".

On peut donc remplacer une adresse par un lien en exécutant:

```
string mailAddress = "yop@cdiese.fr";
Regex regex = new Regex(@"([\w\-.]+)@([\w\-.]+)");
string addressLink = regex.Replace(mailAddress, "$1@$2);
```

Autre possibilité pour nommer une sous-chaîne capturée

Dans les exemples précédents, les sous-chaînes capturées sont désignées par des index: $1, $2, $3, etc…

Il est possible de définir et de désigner les sous-chaînes plus explicitement.

Par exemple:

Dans l'exemple précédent, on avait utilisé la regex **"(\w+)(\s)(\w+)"**. On peut redéfinir cette regex en nommant les groupes: **"(?<word1>\w+)(\s)(?<word2>\w+)".**

Dans cet exemple, les groupes sont donc:

- Sous-chaîne "word1" capturée par "(?<word1>\w+)",
- Sous-chaîne "word2" capturée par "(?<word2>\w+)".

On peut utiliser les sous-chaînes avec:

- Sous-chaîne "word1" par "${word1}"
- Sous-chaîne "word2" par "${word2}"

III.3 Découpage d'une chaîne de caractères

III.3.1 Découpage par séparateur

Le découpage se fait en utilisant "Regex.Split".

Par exemple:

Pour séparer les mots d'une phrase contenant des caractères d'espacement:

```
string sentence = "un deux trois quatre";
Regex regex = new Regex(@"\s+");
string[] result = regex.Split(sentence);
```

Le résultat sous forme de tableau contiendra: "un", "deux", "trois" et "quatre".

III.3.2 Découpage par regroupement

Dans le cas du découpage par regroupement, on peut utiliser "Regex.Match" ou la forme statique "Regex.Matches".

Il faut définir une regex qui utilise des "parenthèses de capture" (**définies plus haut**).

Par exemple:

Si on définit la regex: "(\d{3})-(\d{3}-\d{4})" permet de capturer 2 groupes:

– Groupe 1: "(\d{3})" et
– Groupe 2: "(\d{3}-\d{4})".

Ainsi en exécutant:

```
Regex regex = new Regex(@"(\d{3})-(\d{3}-\d{4})");
Match m = regex.Match("212-435-6534");
string areaCode = m.Groups[1].ToString();
string phoneNumber = m.Groups[2].ToString();
```

Le groupe 1 s'obtient par **m.Groups[1]** et le groupe 2 par **m.Groups[2]**.
En utilisant la forme statique "Regex.Matches", on peut itérer sur plusieurs résultats trouvés dans une chaine.

Par exemple:

```
string input = "212-555-6666 906-932-1111 415-222-3333 425-888-9999";
MatchCollection matches = Regex.Matches(input, @"(\d{3})-(\d{3}-\d{4})");
foreach (Match match in matches)
{
    string areaCode = match.Groups[1].ToString();
    string phoneNumber = match.Groups[2].ToString();
}
```

Ainsi on pourra capturer successivement les 4 numéros: "212-555-6666", "906-932-1111", "415-222-3333" et "425-888-9999".