Expression régulière

Victor Marsault
Aldric Degorre

CPOO 2015

Outline

Terminologie

- Un *alphabet* est un ensemble de lettre :
  ex : $A = \{a, b, c\}$.
- Un *mot* est une suite finie de lettre :
  ex : $aabc$ ou le mot vide ε .
- Un *langage* est un ensemble de mots :
  ex : l'ensemble des mots avec un nombre pair de a .

- La *concaténation* de deux mots est le mot résultat de la mise bout à bout de deux mots :
  ex : $(aabc \cdot bac) = aabcbac$.
- Le passage à l'étoile d'un mot : concaténation du mot avec lui-même un nombre arbitraire de fois :
  ex : $(aa)^* = \{aa, aaaa, aaaaaa, aaaaaaaa\}$

Langage régulier

A : un alphabet.
$Rat(A)$ est le *plus petit* ensemble de langages

- contenant $\emptyset$, $\{\varepsilon\}$;
- contenant $\{a\}$ pour toute lettre $a \in A$;
- stable par union ensembliste ;
- stable par concaténation ensembliste ;

$$L \cdot L' = \{u \cdot v \mid u \in L,\ v \in L'\}$$

- stable par passage à l'étoile ;

$$L^* = \{u_0 \cdot u_1 \cdot \ldots \cdot u_i \mid i \in \mathbb{N} \text{ et } u_0, u_1, \ldots, u_i \in L\}$$

Qu'est-ce que cela signifie

- $\{\varepsilon\}$ et $\emptyset$ sont des langages réguliers ;
- Pour toute lettre $a \in A$, $\{a\}$ est un langage régulier ;
- Si L et L' sont deux langages réguliers :
  - L^* est un langage régulier.
  - $L \cup L'$ est un langage régulier.
  - $L \cdot L'$ est un langage régulier.
  - ($L \cap L'$ est une langage régulier).

Qu'est-ce que cela signifie – inversement

Si le langage L est régulier, alors il vérifie l'une des conditions suivantes :

- L est vide ;
- L est égal à $\{\varepsilon\}$;
- L est égal à $\{a\}$ pour une certaine lettre $a \in A$;
- L est de la forme M^*, pour un certain langage régulier M ;
- L est de la forme $M \cup K$, pour certains langages réguliers M, K.
- L est de la forme $M \cdot K$, pour certains langages réguliers M, K.

Expression régulière

Les langages réguliers sont exactement ceux que l'on peut exprimer à l'aide d'*expressions régulières*

Exemple : Le langage des mots sur {a,b,c} ayant un nombre pair de a est régulier car il peut être exprimé par :

$$(\{b\} \cup \{c\})^* \cdot [a \cdot (\{b\} \cup \{c\})^* \cdot a \cdot (\{b\} \cup \{c\})^*]^* \cdot (\{b\} \cup \{c\})^*$$

Exemple : Le langage des mots avec un nombre premier de a n'est pas régulier car on ne peut pas l'exprimer avec un langage régulier.

Théorème (Kleene)

Un langage est régulier si et seulement si il est reconnu par un automate fini.

Théorème (Kleene)

Un langage est régulier si et seulement si il est reconnu par un automate fini.

Lemme de l'étoile (Kleene)

L : un langage régulier.
Il existe un entier n tel que tout mot $u \in L$ de longueur au moins n se factorise en

$$u = x \cdot v \cdot y$$

et $(x \cdot v^* \cdot y)$ est inclus dans L.

Théorème (Kleene)

Un langage est régulier si et seulement si il est reconnu par un automate fini.

Lemme de l'étoile (Kleene)

L : un langage régulier.
Il existe un entier n tel que tout mot $u \in L$ de longueur au moins n se factorise en

$$u = x \cdot v \cdot y$$

et $(x \cdot v^* \cdot y)$ est inclus dans L.

$\rightarrow$ Sert à montrer qu'un langage n'est pas régulier.

Outline

Recherche de motif : `Pattern` et `Matcher`

`Pattern`

- Construit à partir d'une chaine de caractère représentant une expression régulières (`static Pattern Pattern.compile(String)`).
- Représente un automate (car réutilisable).

`Matcher`

- Construit à partir d'un motif et d'une chaine (`Matcher Pattern.matcher(String)`)
- Confronte l'expression régulière à l'argument (plusieurs confrontations possibles : `matches`, `find`, etc).

Recherche de motif – Exemple 1


```
public static void main(String[] args) {
    String chaine= "expr Hello expr World";
    String regexp= "expr";
```

Recherche de motif – Exemple 1


```
public static void main(String[] args) {
    String chaine= "expr Hello expr World";
    String regexp= "expr";
    Pattern motif= Pattern.compile(regexp);
```

Recherche de motif – Exemple 1


```
public static void main(String[] args) {
    String chaine= "expr Hello expr World";
    String regexp= "expr";
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);
```

Recherche de motif – Exemple 1


```
public static void main(String[] args) {
    String chaine= "expr Hello expr World";
    String regexp= "expr";
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
    }
    System.out.println(nbOccurences); //2
}
```

Recherche de motif – Exemple 2


```
public static void main(String[] args) {
    String chaine= "nanane";
    String regexp= "nan";
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
    }
    System.out.println(nbOccurences);
```

Recherche de motif – Exemple 2


```
public static void main(String[] args) {
    String chaine= "nanane";
    String regexp= "nan";
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
    }
    System.out.println(nbOccurences); //1, on ne retourne
                                      //pas en arrière
```

Recherche de motif – Exemple 2


```
public static void main(String[] args) {
    String chaine= "nanane";
    String regexp= "nan";
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
    }
    System.out.println(nbOccurences); //1, on ne retourne
                                      //pas en arrière
    //Recherche à partir d'un certain indice
    System.out.println(recherche.find(1)); //true  "nanane"
```

Recherche de motif – Exemple 2


```
public static void main(String[] args) {
    String chaine= "nanane";
    String regexp= "nan";
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
    }
    System.out.println(nbOccurences); //1, on ne retourne
                                      //pas en arrière
    //Recherche à partir d'un certain indice
    System.out.println(recherche.find(1)); //true  "nanane"
    System.out.println(recherche.start()+"-"+recherche.end());
    //2-5
}
```

Classe de caractère – Exemple 1


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[bn][ae]"; //b ou n suivi de a ou e
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
```

Classe de caractère – Exemple 1


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[bn][ae]"; //b ou n suivi de a ou e
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Classe de caractère – Exemple 1


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[bn][ae]"; //b ou n suivi de a ou e
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"ba"  "na"  "ne"  (ba n na ne)
```

Classe de caractère – Exemple 1


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[bn][ae]"; //b ou n suivi de a ou e
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"ba"  "na"  "ne"  (ba n na ne)
    System.out.println(nbOccurences); //3
}
```

Plusieurs mots – Exemple


```
public static void main(String[] args) {
    String chaine= "foo barbar foo";
    String regexp= "(foo)|(bar)";//foo ou bar
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Plusieurs mots – Exemple


```
public static void main(String[] args) {
    String chaine= "foo barbar foo";
    String regexp= "(foo)|(bar)";//foo ou bar
    Pattern motif= Pattern.compile(regexp);
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"foo" "bar" "bar" "foo"
    System.out.println(nbOccurences); //4
}
```

Classe de caractère – Exemple 2


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[^aeiou][a-z]"; //non-voyelle puis lettre
    Pattern motif= Pattern.compile(regexp);      //quelconque
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Classe de caractère – Exemple 2


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[^aeiou][a-z]"; //non-voyelle puis lettre
    Pattern motif= Pattern.compile(regexp);      //quelconque
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"ba"  "nn"  "ne"  (ba nn a ne)
    System.out.println(nbOccurences); //3
}
```

Classe de caractère – Exemple 3


```
public static void main(String[] args) {
    String chaine= "BAnNanE";
    String regexp= "[^aeiou][a-z]"; //non-(voyelle minuscule)
    Pattern motif= Pattern.compile(regexp);    //puis lettre
    Matcher recherche= motif.matcher(chaine);  //minuscule
                                               //quelconque
    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Classe de caractère – Exemple 3


```
public static void main(String[] args) {
    String chaine= "BAnNanE";
    String regexp= "[^aeiou][a-z]"; //non-(voyelle minuscule)
    Pattern motif= Pattern.compile(regexp);    //puis lettre
    Matcher recherche= motif.matcher(chaine);  //minuscule
                                               //quelconque
    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"An"  "Na"  (B An Na nE)
    System.out.println(nbOccurences); //2
}
```

Classe de caractère – Exemple 4


```
public static void main(String[] args) {
    String chaine= ";An.anE";
    String regexp= "[^aeiou][a-zA-Z]"; //non-(voyelle minusc.)
    Pattern motif= Pattern.compile(regexp);    //puis lettre
    Matcher recherche= motif.matcher(chaine);  //quelconque

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Classe de caractère – Exemple 4


```
public static void main(String[] args) {
    String chaine= ";An.anE";
    String regexp= "[^aeiou][a-zA-Z]"; //non-(voyelle minusc.)
    Pattern motif= Pattern.compile(regexp);    //puis lettre
    Matcher recherche= motif.matcher(chaine);  //quelconque

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //";A"  ".a" "nE"  (;A n .a nE)
    System.out.println(nbOccurences); //3
}
```

Classe de caractère – Exemple 5


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[a-z&&[^b]][a-z&&[^a]]"; //une lettre sauf
    Pattern motif= Pattern.compile(regexp);   //b, puis une
    Matcher recherche= motif.matcher(chaine); //lettre sauf a

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Classe de caractère – Exemple 5


```
public static void main(String[] args) {
    String chaine= "bannane";
    String regexp= "[a-z&&[^b]][a-z&&[^a]]"; //une lettre sauf
    Pattern motif= Pattern.compile(regexp);   //b, puis une
    Matcher recherche= motif.matcher(chaine); //lettre sauf a

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"an"  "an" (b an n an e)
    System.out.println(nbOccurences); //2
}
```

Classe de caractère – Exemple 5


```
public static void main(String[] args) {
    String chaine= "bânännê et 8range ";
    String regexp= "\\p{L}\\p{L}\\p{L}\\p{L}";//quatre lettres
    Pattern motif= Pattern.compile(regexp);   //selon unicode
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    }
```

Classe de caractère – Exemple 5

```
public static void main(String[] args) {
    String chaine= "bânännê et 8range ";
    String regexp= "\\p{L}\\p{L}\\p{L}\\p{L}";//quatre lettres
    Pattern motif= Pattern.compile(regexp);   //selon unicode
    Matcher recherche= motif.matcher(chaine);

    int nbOccurences = 0;
    while (recherche.find()){ //tant que l'on trouve
                              //le motif dans la chaine
        nbOccurences++; //on incremente la variable
        System.out.println(recherche.group());
        //et on affiche le motif trouve
    } //"bânä" "rang"  (bânä nnê␣et␣8 rang e)
    System.out.println(nbOccurences); //2
}
```

Caractère spéciaux

Comment considérer un caractère spécial : ^.[]${}()?|+<>

- Echapper avec un \ (qu'il faut doubler...)
- Mettre entre crochet (sauf crochets et backslash)

```
String regexp1= "\\\\"; //un backslash
String regexp2= "\n"; //un retour à la ligne (unix)
String regexp3= "[.]"; //un point
String regexp4= "[\\[\\]^]"; //un crochet ou un ^
```

Récapitulatif (1)

- `^` et `$` : début et fin de chaîne.
- `\\` : échappe le caractère suivant.
- `.` : n'importe quel caractère (sauf fins de lignes).
- `[ ]` : un caractère appartenant à une certaine classe
  - `[a\\[\\]]` : `'a'`, `'['` ou `']'`
  - `[^a.^]` : n'importe quel caractère sauf `'a'`, `'.'` ou `'^'`.
  - `[a-z]` : n'importe quelle lettre en minuscule.
  - `[a-zA-Z^]` : n'importe quelle lettre ou `'^'`.
  - `[a-z&&[^vm]]` : n'importe quelle lettre sauf `'v'` ou `'m'`.
- `\\d` : un chiffre (comme `[0-9]`).
- `\\D` : un non-chiffre (comme `[^0-9]`).
- `\\s` : un caractère dit "blanc" (comme `[ \t\n\xOB\f\r]`).
- `\\S` : un caractère "non-blanc" (comme `[^\\s]`).
- `\\w` : caractère dit de "mot" (comme `[a-zA-Z_0-9]`).
- `\\W` : caractère "non-mot" (comme `[^\\w]`).
- `\\p{L}` : une 'lettre' selon unicode.

Outline

Optionnel (une fois ou pas du tout) : ?


```
public static void main(String[] args) {
    String regex= "(http://)?"
                + "(www.)?"
                + "victor.marsault.xyz"
                + "(/index.html)?";
    Pattern motif= Pattern.compile(regex);
```

Optionnel (une fois ou pas du tout) : ?

`Matcher.matches()` renvoie `true` si la chaîne *en entier* correspond.

```
public static void main(String[] args) {
    String regex= "(http://)?"
                + "(www.)?"
                + "victor.marsault.xyz"
                + "(/index.html)?";
    Pattern motif= Pattern.compile(regex);

    String addr1= "victor.marsault.xyz/index.html";
    boolean b1= motif.matcher(addr1).matches(); //true
```

Optionnel (une fois ou pas du tout) : ?

`Matcher.matches()` renvoie `true` si la chaîne *en entier* correspond.

```
public static void main(String[] args) {
    String regex= "(http://)?"
                + "(www.)?"
                + "victor.marsault.xyz"
                + "(/index.html)?";
    Pattern motif= Pattern.compile(regex);

    String addr1= "victor.marsault.xyz/index.html";
    boolean b1= motif.matcher(addr1).matches(); //true

    String addr2= "http://victor.marsault.xyz";
    boolean b2= motif.matcher(addr2).matches(); //true
```

Optionnel (une fois ou pas du tout) : ?

`Matcher.matches()` renvoie `true` si la chaîne *en entier* correspond.

```
public static void main(String[] args) {
    String regex= "(http://)?"
                + "(www.)?"
                + "victor.marsault.xyz"
                + "(/index.html)?";
    Pattern motif= Pattern.compile(regex);

    String addr1= "victor.marsault.xyz/index.html";
    boolean b1= motif.matcher(addr1).matches(); //true

    String addr2= "http://victor.marsault.xyz";
    boolean b2= motif.matcher(addr2).matches(); //true

    String addr3= "www.marsault.xyz";
    boolean b3= motif.matcher(addr3).matches(); //false
}
```

Etoile (n'importe quel nombre de fois) : *

`static bool Pattern.matches(String re, CharSequence chaine)` renvoie `true` si `chaine` correspond à `re`.

```
public class Main {
    public static void main(String[] args) {
      String re1= "a*"; //n'importe quel nombre de 'a'
      boolean b1= Pattern.matches(re1,"aaaa"); //true
```

Etoile (n'importe quel nombre de fois) : *

`static bool Pattern.matches(String re, CharSequence chaine)` renvoie `true` si `chaine` correspond à `re`.

```
public class Main {
    public static void main(String[] args) {
      String re1= "a*"; //n'importe quel nombre de 'a'
      boolean b1= Pattern.matches(re1,"aaaa"); //true
      boolean b2= Pattern.matches(re1,""); //true
```

Etoile (n'importe quel nombre de fois) : *

`static bool Pattern.matches(String re, CharSequence chaine)` renvoie `true` si `chaine` correspond à `re`.

```
public class Main {
    public static void main(String[] args) {
      String re1= "a*"; //n'importe quel nombre de 'a'
      boolean b1= Pattern.matches(re1,"aaaa"); //true
      boolean b2= Pattern.matches(re1,""); //true

      String re2= "(aa)*"; //un nombre pair de 'a';
      boolean b3= Pattern.matches(re2,"aaaa"); //true
```

Etoile (n'importe quel nombre de fois) : *

`static bool Pattern.matches(String re, CharSequence chaine)` renvoie `true` si `chaine` correspond à `re`.

```
public class Main {
    public static void main(String[] args) {
      String re1= "a*"; //n'importe quel nombre de 'a'
      boolean b1= Pattern.matches(re1,"aaaa"); //true
      boolean b2= Pattern.matches(re1,""); //true

      String re2= "(aa)*"; //un nombre pair de 'a';
      boolean b3= Pattern.matches(re2,"aaaa"); //true
      boolean b4= Pattern.matches(re2,"aaa"); //false

      String re3= "\"[a-zA-z]*\""; //des lettres entre " "
      boolean b5= Pattern.matches(re3,"\"bonjour\""); //true
}   }
```

N'importe quel nombre de fois (au moins une) : +


```
public class Main {
    public static void main(String[] args) {
        String reMot= "\\p{L}+";
        boolean b1= Pattern.matches(reMot,"Je"); //true
        boolean b2= Pattern.matches(reMot,""); //false
        boolean b3= Pattern.matches(reMot,"m'appelle");
                                                    //false
```

N'importe quel nombre de fois (au moins une) : +


```
public class Main {
    public static void main(String[] args) {
        String reMot= "\\p{L}+";
        boolean b1= Pattern.matches(reMot,"Je"); //true
        boolean b2= Pattern.matches(reMot,""); //false
        boolean b3= Pattern.matches(reMot,"m'appelle");
                                                    //false
        String rePresqueMot= "[\\p{L}-']+";
        boolean b4= Pattern.matches(rePresqueMot,"m'appelle");
                                                    //true
```

N'importe quel nombre de fois (au moins une) : +


```
public class Main {
    public static void main(String[] args) {
        String reMot= "\\p{L}+";
        boolean b1= Pattern.matches(reMot,"Je"); //true
        boolean b2= Pattern.matches(reMot,""); //false
        boolean b3= Pattern.matches(reMot,"m'appelle");
                                                    //false
        String rePresqueMot= "[\\p{L}-']+";
        boolean b4= Pattern.matches(rePresqueMot,"m'appelle");
                                                    //true
        String rePhrase= String.format
              ("(%s,? )*%s.", rePresqueMot, rePresqueMot);
```

N'importe quel nombre de fois (au moins une) : +


```
public class Main {
    public static void main(String[] args) {
        String reMot= "\\p{L}+";
        boolean b1= Pattern.matches(reMot,"Je"); //true
        boolean b2= Pattern.matches(reMot,""); //false
        boolean b3= Pattern.matches(reMot,"m'appelle");
                                                    //false
        String rePresqueMot= "[\\p{L}-']+";
        boolean b4= Pattern.matches(rePresqueMot,"m'appelle");
                                                    //true
        String rePhrase= String.format
              ("(%s,? )*%s.", rePresqueMot, rePresqueMot);
        boolean b5= Pattern.matches(rePhrase,
                "Je m'appelle Vïctôr, bonjour."); //true
    }
}
```

Glouton vs Réticent vs Possessif


```
public static void main(String[] args) {
    String chaine= "abc dfoo dc bafoo "

    Matcher glou= Pattern.compile(".*foo").matcher(chaine);
    while (glou.find())
        System.out.println(glou.group()); //abc dfoo dc bafoo
```

Glouton vs Réticent vs Possessif


```
public static void main(String[] args) {
    String chaine= "abc dfoo dc bafoo "

    Matcher glou= Pattern.compile(".*foo").matcher(chaine);
    while (glou.find())
        System.out.println(glou.group()); //abc dfoo dc bafoo

    Matcher reti= Pattern.compile(".*?foo").matcher(chaine);
    while (reti.find())
        System.out.println(reti.group()); //abc dfoo
                                          // dc bafoo
```

Glouton vs Réticent vs Possessif


```
public static void main(String[] args) {
    String chaine= "abc dfoo dc bafoo "

    Matcher glou= Pattern.compile(".*foo").matcher(chaine);
    while (glou.find())
        System.out.println(glou.group()); //abc dfoo dc bafoo

    Matcher reti= Pattern.compile(".*?foo").matcher(chaine);
    while (reti.find())
        System.out.println(reti.group()); //abc dfoo
                                          // dc bafoo

    Matcher poss= Pattern.compile(".*+foo").matcher(chaine);
    while (poss.find())
        System.out.println(poss.group());//rien

}
```

Récapitulatif (2)

Quantificateurs			Description
Glouton (greedy)	Réticent (reluctant)	Possessif (possessive)	
`(..)?`	`(..)??`	`(..)?+`	Une fois ou pas du tout.
`(..)*`	`(..)*?`	`(..)*+`	Pas du tout, une fois ou plus.
`(..)+`	`(..)+?`	`(..)++`	Une fois ou plus.
`(..){n}`	`(..){n}?`	`(..){n}+`	Exactement *n* fois.
`(..){n,}`	`(..){n,}?`	`(..){n,}+`	*n* fois ou plus.
`(..){n,m}`	`(..){n,m}?`	`(..){n,m}+`	Entre *n* et *m* fois.

Opérateur logique	Description
`XY`	Concaténation
`(..)|(..)`	Union (”ou” logique)

Outline

Principe

- Récupérer des sous-chaînes de l'expression matchée.
  $\rightarrow$ Extraction.
- Outil d'analyse syntaxique.

- Un groupe est une partie de l'expression régulière entre `( )` .
- Un groupe est extrait avec `Matcher.group(int)` (`Matcher.start(int)` et `Matcher.end(int)` donnent les indices de début et fin du groupe).
- Attention aux groupes sous quantificateurs.

Premier exemple


```
public static String main(String[] args) {
  Pattern motif= Pattern.compile("([^#=]*)#([^=]*)=(.*)");
  // <type sans '=' ni '#'> # <nom sans '='> = <valeur>;
```

Premier exemple


```
public static String main(String[] args) {
  Pattern motif= Pattern.compile("([^#=]*)#([^=]*)=(.*)");
  // <type sans '=' ni '#'> # <nom sans '='> = <valeur>;

  String flagString= "FlagOption#mode de debogage=false";
  Matcher m= motif.matcher(flagString);
  m.matches(); //true
  System.out.println(m.group(1)+","+m.group(2)+","+m.group(3));
  // "FlagOption", "mode de debogage", "false"
```

Premier exemple


```
public static String main(String[] args) {
  Pattern motif= Pattern.compile("([^#=]*)#([^=]*)=(.*)");
  // <type sans '=' ni '#'> # <nom sans '='> = <valeur>;

  String flagString= "FlagOption#mode de debogage=false";
  Matcher m= motif.matcher(flagString);
  m.matches(); //true
  System.out.println(m.group(1)+","+m.group(2)+","+m.group(3));
  // "FlagOption", "mode de debogage", "false"

  String flagString= "IntOption#profondeur de recherche=3";;
  Matcher m= motif.matcher(flagString);
  m.matches(); //true
  System.out.println(m.group(1)+","+m.group(2)+","+m.group(3));
  // "IntOption", "profondeur de recherche", "3"

}
```

Suppressions de blancs de début et de fin


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("\\s*(.*?)\\s*");
    //maximum de blanc, minimum de lettres, maximum de blanc
```

Suppressions de blancs de début et de fin


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("\\s*(.*?)\\s*");
    //maximum de blanc, minimum de lettres, maximum de blanc

    Matcher m= motif.matcher("   foo ");
    m.matches(); //true;
    System.out.println(m.group(1)); //foo;
```

Suppressions de blancs de début et de fin


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("\\s*(.*?)\\s*");
    //maximum de blanc, minimum de lettres, maximum de blanc

    Matcher m= motif.matcher("   foo ");
    m.matches(); //true;
    System.out.println(m.group(1)); //foo;
    System.out.println(m.start(1)+"-"+m.end(1)); //3-6
```

Suppressions de blancs de début et de fin

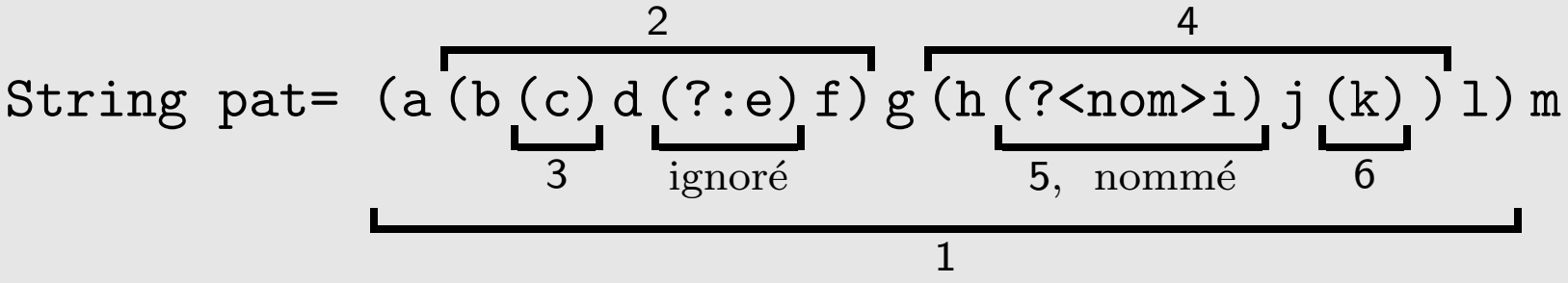

```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("\\s*(.*?)\\s*");
    //maximum de blanc, minimum de lettres, maximum de blanc

    Matcher m= motif.matcher("   foo ");
    m.matches(); //true;
    System.out.println(m.group(1)); //foo;
    System.out.println(m.start(1)+"-"+m.end(1)); //3-6

    Matcher m= motif.matcher("  foo bar\n\r");
    m.matches(); //true
    System.out.println(m.group(1)); //foo␣bar
```

Suppressions de blancs de début et de fin


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("\\s*(.*?)\\s*");
    //maximum de blanc, minimum de lettres, maximum de blanc

    Matcher m= motif.matcher("   foo ");
    m.matches(); //true;
    System.out.println(m.group(1)); //foo;
    System.out.println(m.start(1)+"-"+m.end(1)); //3-6

    Matcher m= motif.matcher("  foo bar\n\r");
    m.matches(); //true
    System.out.println(m.group(1)); //foo␣bar

    Matcher m= motif.matcher("\n \r  ");
    m.matches(); //true
    System.out.println(m.group(1)); //""
}
```

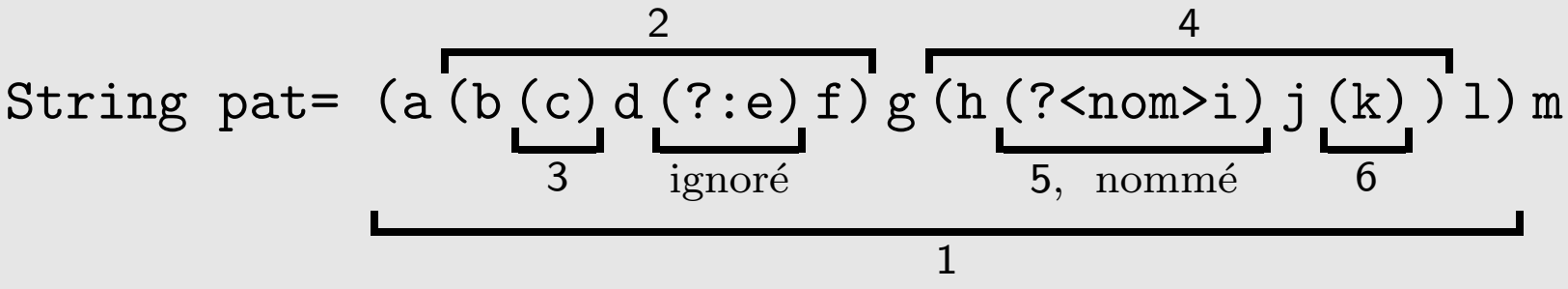
Numéro du groupe, groupe ignoré et groupe nommé

- Un groupe est numéroté par sa parenthèse ouvrante.
- Un groupe est ignoré s'il commence par `(?:` .
- Un groupe est nommé s'il commence par `(?<un nom>` .



Numéro du groupe, groupe ignoré et groupe nommé

- Un groupe est numéroté par sa parenthèse ouvrante.
- Un groupe est ignoré s'il commence par `(?:` .
- Un groupe est nommé s'il commence par `(?<un nom>` .




```
Matcher m= Pattern.compile(pat).matcher("abcdefghijklm");
m.matches();
for(int i=0; i<=6; i++)                    //0: abcdefghijklm
    System.out.println(m.group(i));        //1: abcdefghijkl
                                           //2:  bcdef
                                           //3:   c
                                           //4:       hijk
                                           //5:        i
                                           //6:          k
System.out.println(m.group("nom"));        //<nom>:    i
```

Groupe sous quantificateur


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("(f)?oo")
    Matcher m= motif.matcher("foo");
    m.matches(); //true
    System.out.println(m.group(1));} //"f"
```

Groupe sous quantificateur


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("(f)?oo")
    Matcher m= motif.matcher("foo");
    m.matches(); //true
    System.out.println(m.group(1));} //"f"

    Matcher m2= motif.matcher("oo");
    m2.matches(); //true
    System.out.println(m2.group(1)); //null
```

Groupe sous quantificateur


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("(f)?oo")
    Matcher m= motif.matcher("foo");
    m.matches(); //true
    System.out.println(m.group(1));} //"f"

    Matcher m2= motif.matcher("oo");
    m2.matches(); //true
    System.out.println(m2.group(1)); //null

    motif= Pattern.compile("(f?)oo");
    m1= motif.matcher("foo");
    m1.matches(); //true
    System.out.println(m3.group(1));} //"f"
```

Groupe sous quantificateur


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("(f)?oo")
    Matcher m= motif.matcher("foo");
    m.matches(); //true
    System.out.println(m.group(1));} //"f"

    Matcher m2= motif.matcher("oo");
    m2.matches(); //true
    System.out.println(m2.group(1)); //null

    motif= Pattern.compile("(f?)oo");
    m1= motif.matcher("foo");
    m1.matches(); //true
    System.out.println(m3.group(1));} //"f"

    m2= motif.matcher("oo");
    m2.matches(); //true
    System.out.println(m4.group(1)); //""
}
```

Groupe sous quantificateur (2)


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("((foo)|(bar))*")
    Matcher m= motif.matcher("barfoofoo");
    m.matches(); //true
    System.out.println(m.group(1));} //"foo"
```

Groupe sous quantificateur (2)


```
public static String main(String[] args) {
    Pattern motif= Pattern.compile("((foo)|(bar))*")
    Matcher m= motif.matcher("barfoofoo");
    m.matches(); //true
    System.out.println(m.group(1));} //"foo"

    Matcher m2= motif.matcher("foobar");
    m2.matches(); //true
    System.out.println(m2.group(1));} //"bar"

    Matcher m3= motif.matcher("");
    m3.matches(); //true
    System.out.println(m2.group(1));} //null
}
```

Référence arrière

Dans une expression régulière, la chaîne spéciale `"\\2"` signifie *la sous-chaîne précédemment extraite par le groupe 2*
Même chose pour `"\\k<un nom>"` pour un groupe nommé.

Référence arrière

Dans une expression régulière, la chaîne spéciale `"\\2"` signifie *la sous-chaîne précédemment extraite par le groupe 2*
Même chose pour `"\\k<un nom>"` pour un groupe nommé.

```
public static void main(String[] args) {
    Pattern p= Pattern.compile("([a-z]*)\\1");
    boolean b1= p.matcher("foofoo").matches();//true
    boolean b2= p.matcher("foobar").matches();//false
    boolean b3= p.matcher("barbar").matches();//true
}
```

Référence arrière

Dans une expression régulière, la chaîne spéciale `"\\2"` signifie *la sous-chaîne précédemment extraite par le groupe 2*
Même chose pour `"\\k<un nom>"` pour un groupe nommé.

```
public static void main(String[] args) {
    Pattern p= Pattern.compile("([a-z]*)\\1");
    boolean b1= p.matcher("foofoo").matches();//true
    boolean b2= p.matcher("foobar").matches();//false
    boolean b3= p.matcher("barbar").matches();//true
}
//Cette expression p correspond au langage des carrés
//(uu), qui n'est pas régulier.
```

Au delà de la régularité (2)


```
public static main (String[] args) {
    String bal= "[<](?<balise>[a-z]*)[>].*[<]/\\k<balise>[>]"
    // <XXxxX> n'importe quoi </XXxxX>
    Pattern p= Pattern.compile(bal);
```

Au delà de la régularité (2)


```
public static main (String[] args) {
    String bal= "[<](?<balise>[a-z]*)[>].*[<]/\\k<balise>[>]"
    // <XXxxX> n'importe quoi </XXxxX>
    Pattern p= Pattern.compile(bal);

    String reu= "<reussite> texte quelconque </reussite>";
    Matcher m= p.matcher(reu);
    boolean b1= m.matches(); //true
    String s1= m.group("balise"); //"reussite"
```

Au delà de la régularité (2)

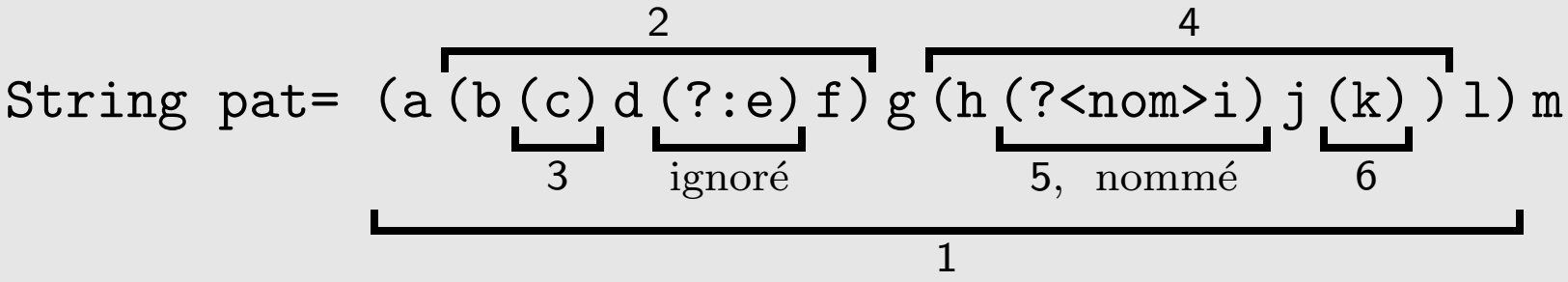

```
public static main (String[] args) {
    String bal= "[<](?<balise>[a-z]*)[>].*[<]/\\k<balise>[>]"
    // <XXxxX> n'importe quoi </XXxxX>
    Pattern p= Pattern.compile(bal);

    String reu= "<reussite> texte quelconque </reussite>";
    Matcher m= p.matcher(reu);
    boolean b1= m.matches(); //true
    String s1= m.group("balise"); //"reussite"

    String ech="<echec> texte tout aussi quelconque </ECHEC>";
    Matcher m2= p.matcher(ech);
    boolean b2= m2.matches(); //false
    String s2= m2.group("balise"); //IllegalStateException
```

Au delà de la régularité (2)


```
public static main (String[] args) {
    String bal= "[<](?<balise>[a-z]*)[>].*[<]/\\k<balise>[>]"
    // <XXxxX> n'importe quoi </XXxxX>
    Pattern p= Pattern.compile(bal);

    String reu= "<reussite> texte quelconque </reussite>";
    Matcher m= p.matcher(reu);
    boolean b1= m.matches(); //true
    String s1= m.group("balise"); //"reussite"

    String ech="<echec> texte tout aussi quelconque </ECHEC>";
    Matcher m2= p.matcher(ech);
    boolean b2= m2.matches(); //false
    String s2= m2.group("balise"); //IllegalStateException

    //Pourquoi nommer?
    String bals= String.format("\\s*(%s\\s*)*", bal);
}   //Successions de balises séparés par des espaces.
```

Récapitulatif (3)

- Un groupe est numéroté par sa parenthèse ouvrante.
- Un groupe est ignoré s'il commence par `(?:` .
- Un groupe est nommé s'il commence par `(?<un nom>` .



- Dans une expression régulière, la chaîne spéciale `"\\2"` signifie *la sous-chaîne précédemment extraite par le groupe 2* Même chose avec `"\\k<un nom>"` pour un groupe nommé.
- Lors d'un replacement, la sous-chaîne `$2` dans la chaîne de remplacement fait référence au deuxième groupe capturé dans la chaîne à remplacer.

Outline

Notion de région

La région est la portion de la chaîne dans laquelle on cherche :
$\rightarrow$ initialement égale à `[0, str.length()]`.

Changer la région

- Modifie toutes les fonctions.
- Permet de faire des recherche récursives.

- `Matcher.region(int,int)` change la *région* courante.
- `int Matcher.regionStart()` et `int Matcher.regionEnd()` donnent le début et la fin de la région courante.

Recherche précisément circonscrite


```
public static boolean rechercheePrecisemmentEntre
        (Matcher m, int debut, int fin) {
    m.region(debut, fin);
    return (m.matches());
```

Recherche précisément circonscrite


```
public static boolean rechercheePrecisemmentEntre
        (Matcher m, int debut, int fin) {
    m.region(debut, fin);
    return (m.matches()); //Peu propre: on ne remet pas le
                          //matcher dans son état d'origine
}
```

Recherche précisément circonscrite


```
public static boolean rechercheePrecisemmentEntre
        (Matcher m, int debut, int fin) {
    m.region(debut, fin);
    return (m.matches()); //Peu propre: on ne remet pas le
                          //matcher dans son état d'origine
}
public static boolean commenceA(Matcher m, int debut) {
    m.region(debut, m.regionEnd());
    return (m.lookingAt()); //Cherche le motif au début de
                            //la région seulement.
}
```

Recherche précisément circonscrite


```
public static boolean rechercheePrecisemmentEntre
        (Matcher m, int debut, int fin) {
    m.region(debut, fin);
    return (m.matches()); //Peu propre: on ne remet pas le
                          //matcher dans son état d'origine
}
public static boolean commenceA(Matcher m, int debut) {
    m.region(debut, m.regionEnd());
    return (m.lookingAt()); //Cherche le motif au début de
                            //la région seulement.
}
public static boolean main(String[] args) {
  Matcher m= Pattern.compile("a+").matcher("aaabbbaaa");
  boolean b1= commenceA(m, 1); //true
  String str= m.group(); //"aa"
```

Recherche précisément circonscrite


```
public static boolean rechercheePrecisemmentEntre
        (Matcher m, int debut, int fin) {
    m.region(debut, fin);
    return (m.matches()); //Peu propre: on ne remet pas le
                          //matcher dans son état d'origine
}
public static boolean commenceA(Matcher m, int debut) {
    m.region(debut, m.regionEnd());
    return (m.lookingAt()); //Cherche le motif au début de
                            //la région seulement.
}
public static boolean main(String[] args) {
  Matcher m= Pattern.compile("a+").matcher("aaabbbaaa");
  boolean b1= commenceA(m, 1); //true
  String str= m.group(); //"aa"
  boolean b2= commenceA(m, 3); //false
```

Recherche précisément circonscrite


```
public static boolean rechercheePrecisemmentEntre
        (Matcher m, int debut, int fin) {
    m.region(debut, fin);
    return (m.matches()); //Peu propre: on ne remet pas le
                          //matcher dans son état d'origine
}
public static boolean commenceA(Matcher m, int debut) {
    m.region(debut, m.regionEnd());
    return (m.lookingAt()); //Cherche le motif au début de
                            //la région seulement.
}
public static boolean main(String[] args) {
  Matcher m= Pattern.compile("a+").matcher("aaabbbaaa");
  boolean b1= commenceA(m, 1); //true
  String str= m.group(); //"aa"
  boolean b2= commenceA(m, 3); //false
  boolean b3= m.find(3); //true
}
```

Avenir (*look ahead*) et passé (*look behind*)

- `(?<=...)` vérifier une expression **avant** la position courante
- `(?=...)` vérifier une expression **après** la position courante

Avenir (*look ahead*) et passé (*look behind*)

- `(?<=...)` vérifier une expression **avant** la position courante
- `(?=...)` vérifier une expression **après** la position courante

```
public static void main(String[] args) {
    String str= "x1x2x3x45x";
    Pattern p= Pattern.compile("x([0-9])x");
    Matcher m= p.matcher(str);
    while (m.find())
      System.out.println(m.group(1)); //1 en argument!
    //1 3 (x1x 2 x3x 45x)
```

Avenir (*look ahead*) et passé (*look behind*)

- `(?<=...)` vérifier une expression **avant** la position courante
- `(?=...)` vérifier une expression **après** la position courante

```
public static void main(String[] args) {
    String str= "x1x2x3x45x";
    Pattern p= Pattern.compile("x([0-9])x");
    Matcher m= p.matcher(str);
    while (m.find())
      System.out.println(m.group(1)); //1 en argument!
    //1 3 (x1x 2 x3x 45x)

    Pattern p2= Pattern.compile("(?<=x)[0-9](?=x)");
    Matcher m2= p2.matcher(str);
    while (m2.find())
      System.out.println(m2.group());
    //1 2 3 (x <-1-> x <-2-> x <-3-> x45x)
}
```

Retour sur `split`

Le séparateur donné en argument à `split` est une expr. régulière.

```
public static String[] splitSansRedondance
        (String aSep, String separateur) {
    return (aSep.split(String.format("(%s)+",separateur)));
}
```

Retour sur split

Le séparateur donné en argument à split est une expr. régulière.

```
public static String[] splitSansRedondance
        (String aSep, String separateur) {
    return (aSep.split(String.format("(%s)+",separateur)));
}
public static void main(String[] args){
  String res= splitSansRedondance("  foo  bar   bar foo  ",
          " ");
  //["", "foo", "bar", "bar", "foo"]
```

Retour sur split

Le séparateur donné en argument à split est une expr. régulière.

```
public static String[] splitSansRedondance
        (String aSep, String separateur) {
    return (aSep.split(String.format("(%s)+",separateur)));
}
public static void main(String[] args){
  String res= splitSansRedondance("  foo  bar   bar foo  ",
          " ");
  //["", "foo", "bar", "bar", "foo"]

  String res= splitSansRedondance("  foo  bar   bar foo  ",
          "[a-z]");
  //["␣␣", "␣␣", "␣␣␣", "␣", "␣␣"]
}
```

Retour sur `replaceAll` (1)

Le séparateur donné en argument à `replaceAll` est une expr. rég.

```
public static void main (String[] args){
    String str= "Je m'appelle Vïctör, bonjour.";
    String resultat1= str.replaceAll("\\p{L}", "x");
    //xx x'xxxxxxx xxxxxx, xxxxxxx.
```

Retour sur `replaceAll` (1)

Le séparateur donné en argument à `replaceAll` est une expr. rég.

```
public static void main (String[] args){
    String str= "Je m'appelle Vïctör, bonjour.";
    String resultat1= str.replaceAll("\\p{L}", "x");
    //xx x'xxxxxxx xxxxxx, xxxxxxx.
```

Retour sur `replaceAll` (1)

Le séparateur donné en argument à `replaceAll` est une expr. rég.

```
public static void main (String[] args){
    String str= "Je m'appelle Vïctör, bonjour.";
    String resultat1= str.replaceAll("\\p{L}", "x");
    //xx x'xxxxxxx xxxxxx, xxxxxxx.

    String resultat2= str.replaceAll("\\p{L}+", "x");
    //x x'x x, x.
```

Retour sur `replaceAll` (1)

Le séparateur donné en argument à `replaceAll` est une expr. rég.

```
public static void main (String[] args){
    String str= "Je m'appelle Vïctör, bonjour.";
    String resultat1= str.replaceAll("\\p{L}", "x");
    //xx x'xxxxxxx xxxxxx, xxxxxxx.

    String resultat2= str.replaceAll("\\p{L}+", "x");
    //x x'x x, x.

    String resultat3= str.replaceAll(
        "(?<=[^aeiou])[aeiou](?=[^aeiou])",  "@");
    //J@ m'@pp@ll@ Vïctör, b@njour
}
```

Retour sur `replaceAll` (2)

Les groupes capturées peuvent être utilisés avec `$i`

```
public static void main (String[] args){
    String str= "Je m'appelle Vïctör, bonjour.";
    String resultat1= str.replaceAll("\\p{L}+", "($0)");
    //(Je) (m)'(appelle) (Vïctör), (bonjour).
```

Retour sur `replaceAll` (2)

Les groupes capturées peuvent être utilisés avec `$i`

```
public static void main (String[] args){
    String str= "Je m'appelle Vïctör, bonjour.";
    String resultat1= str.replaceAll("\\p{L}+", "($0)");
    //(Je) (m)'(appelle) (Vïctör), (bonjour).

    String resultat2= str.replaceAll("(\\p{L}+) (\\p{L}+)",
                                     "$2 $1");
    //Je m'appelle Vïctör, bonjour.
    // \/ '      \/      , bonjour.
    // /\ '      /\      , bonjour.
    //m Je'Vïctör appelle, bonjour.
}
```

Options de Matcher

`Pattern Pattern.compile(String regex, int flags)`

- `Pattern.CASE_INSENSITIVE` : le matcher ignore la casse pour les caractères ASCII.
- `Pattern.UNICODE_CASE` : rend l'option précédente fonctionnelle pour tous les caractères unicodes.
- `Pattern.COMMENTS` : dans les expressions rationnelles les espaces sont ignorées (il faut donc les échapper si on veut qu'ils soient matchés).
- `Pattern.MULTILINE` : les caractères `^` et `$` signifient début et fin de ligne (au lieu de début et fin de la chaîne).
- `Pattern.DOTALL` : Le point (`.`) correspond à n'importe quel caractère, fin de lignes y compris (en temps normal, un point ne correspond pas aux fins de lignes).
- quelques autres.

Exemples


```
public static void main(String[] args){
    Pattern p= Pattern.compile("föœ",
          Pattern.CASE_INSENSITIVE | Pattern.UNICODE_CASE);
    boolean b1= p.matcher("Föœ").matches(); //true
    boolean b2= p.matcher("fÖŒ").matches(); //true
```

Exemples


```
public static void main(String[] args){
    Pattern p= Pattern.compile("föœ",
          Pattern.CASE_INSENSITIVE | Pattern.UNICODE_CASE);
    boolean b1= p.matcher("Föœ").matches(); //true
    boolean b2= p.matcher("fÖŒ").matches(); //true

    Pattern p2= Pattern.compile("a b\\ c d",
          Pattern.COMMENTS);
    boolean b3= p2.matcher("abcd").matches(); //false
    boolean b4= p2.matcher("ab cd").matches();//true
}
```

Récapitulatif (4)


```
class Matcher {
    //Façon de rechercher les correspondances
    boolean find(); //Cherche la prochaine sous-chaine corr.
    boolean find(int i); //Idem à partir de l'indice i.
    boolean matches(); //Vérifie si la région entière corr.
    boolean lookingAt(); //Cherche le début de la région

    //Résultat
    boolean group(); //correspondance trouvée
    boolean group(int i); //Sous-chaîne capturée par groupe i
    boolean group(String str); //idem par groupe nommé str
    int start(); int end(); //début et fin de la corresp.
    int start(int i); int end(int i); //idem groupe i

    //Région
    Matcher region(int deb, int fin); //Change la région.
    int regionStart(); int regionEnd(); //Debut et fin de
}                                       //la région courante.
```