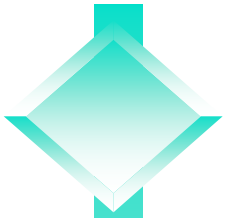
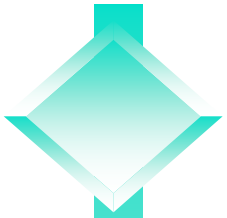


UNIVERSITÉ PARIS SORBONNE PARIS IV

Cours n° 7

Traitement de caractères



Sommaire

1. **Codage des caractères**
    1. **Codage ASCII et ASCII étendue**
    2. **Codage Unicode**
2. **Classe String**
    1. **Conversion et manipulation**
    2. **Comparaison et recherche**
    3. **Classe StringBuffer**
3. **Composants internationaux pour Unicode (ICU)**
    1. **Normalisation**
    2. **Ordre lexicographique**

Bibliographie

Livres

Bernard Desgraupes, «Passeport pour Unicode», Vuibert

Patrick Andries, «Unicode 5.0 en pratique », Dunod

Sites

www.icu-project.org/ (site officiel)

Tutoriel

ICU user guide (version 4.0)

Codage ASCII

Codage des caractères de la langue anglaise sur 7 bits

Norme American Standard Code for Information Interchange

96 caractères

- chiffres (48 à 57)
- majuscules (65 à 90)
- minuscules (97 à 122)

32 caractères « de contrôle » (CR, BEL, ESC, …)

Pas de lettres accentuées

Codage ASCII étendue (1/2)

Ajout de 128 caractères (codage sur 8 bits)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	□	□	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	□	□	□
9	□	‘	’	“	”	•	–	—	˜	™	š	›	œ	□	□	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Extension iso-latin1 (8859-1)

Codage ASCII étendue (2/2)

Langages et extensions

Danois, allemand, anglais, finlandais, français, galicien, irlandais, islandais, italien, catalan, néerlandais, norvégien, portugais, suédois et espagnol (iso-8859-1)

Croate, polonais, roumain, slovaque, slovène, tchèque et hongrois (iso-8859-2)

Espéranto (iso-8859-3)

Estonien, letton et lituanien (iso-8859-4)

Bulgare, macédonien, russe, serbe et ukrainien (iso-8859-5)

Arabe (iso-8859-6)

Grec moderne (iso-8859-7)

Hébreu (iso-8859-8)

Turc (iso-8859-9)

Groenlandais (Inuit) et lapon (Sami) (iso-8859-10)

Norme Unicode

Codage universel et non ambiguë des caractères

Définition par un consortium international (www.unicode.org),

Norme ouverte (ajout de nouveaux caractères)

Espace de codage (0x0 à 0x10FFFF) : 1 048 576 caractères différents

2F00 **Kangxi Radicals** **2FDF**

2F0	2F1	2F2	2F3	2F4	2F5	2F6	2F7	2F8	2F9	2FA	2FB	2FC	2FD
一 2F00	凵 2F10	士 2F20	己 2F30	支 2F40	比 2F50	瓜 2F60	示 2F70	聿 2F80	衣 2F90	辰 2FA0	革 2FB0	鬲 2FC0	鼻 2FD0
丨 2F01	刀 2F11	夕 2F21	巾 2F31	攴 2F41	毛 2F51	瓦 2F61	内 2F71	肉 2F81	襾 2F91	辵 2FA1	韋 2FB1	鬼 2FC1	齊 2FD1
丶 2F02	力 2F12	夂 2F22	干 2F32	文 2F42	氏 2F52	甘 2F62	禾 2F72	臣 2F82	見 2F92	邑 2FA2	韭 2FB2	魚 2FC2	齒 2FD2

5 normes de représentation

UTF-8 (8 bits), UTF-16 (16 bits), UCS-2 (16 bits), UTF-32 (32 bits), UCS-4 (32 bits)

Codage UTF-8

Représentation de longueur variable (compression)

Codage sur un à quatre octets,

128 premiers caractères (ASCII) sur 1 octet,

0	bits 6-0

128 à 2047 caractères (Extension Europe et Moyen-Orient) sur deux octets,

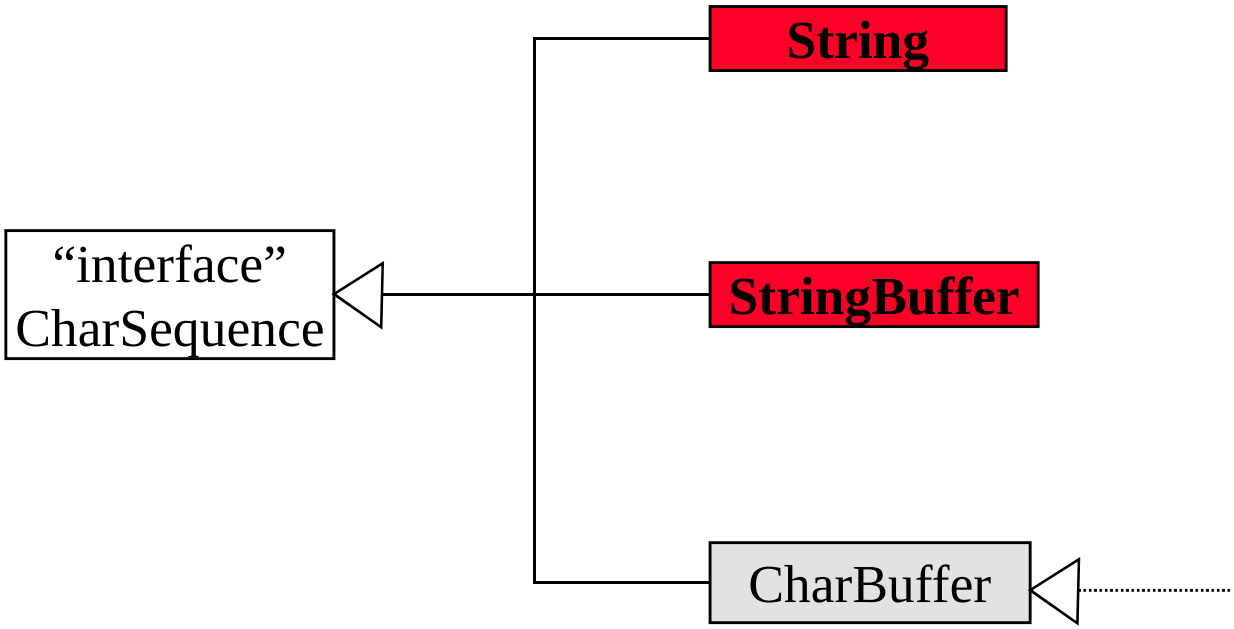
1	1	0	bits 10-6	1	0	bits 5-0

2048 à 65535 caractères sur trois octets (Asie)

1	1	1	0	bits 15-12	1	0	bits 11-6	1	0	bits 5-0

Arborescence de classe

Deux classes concrètes



Interface CharSequence

Spécification du contrat que toutes les types de chaînes de caractères doivent implémenter

Accès au caractère d'indice i de la chaîne de caractères

```
char  charAt (int i)
```

Longueur de la chaîne de caractères

```
int length ()
```

Sous-chaîne de caractères de l'indice début (inclus) à l'indice fin (exclus)

```
CharSequence subSequence (int début, int fin)
```

Constructeurs

Variables non modifiables après instanciation

Nombreux constructeurs

Construction à partir du tableau d'octets tb (jeu de caractères par défaut)
String (byte[] tb)

Construction à partir d'une partie du tableau d'octets tb (jeu de caractères par défaut)
String (byte[] tb , int début, int taille)

Construction à partir d'une partie du tableau d'octets tb avec le jeu de caractères jc)
String (byte[] tb , int début, int taille, String jc)

Construction à partir du tableau de caractère tc
String (char[] tc)

Construction à partir d'une partie du tableau de caractère tc
String (char[] tc , int début, int taille)

Construction à partir d'une copie d'une chaîne de caractères cc
String (String cc)
String (StringBuffer cc)

testConstructeurs.java

Test des constructeurs

```
String s1, s2, s3 = null;
byte[] tb = {112, 114, 111, 112, 114, 105, -23, 116, -23, 115, 32, 100,
101, 115, 32, 99, 111, 110, 115, 116, 114, 117, 99, 116, 101, 117, 114,
115};
s1 = new String(tb); System.out.println(s1);
      // propriétés des constructeurs
s2 = new String(tb, 11, 3); System.out.println(s2);
      // des

try { s3 = new String(tb, "ISO-8859-5"); }
catch (IOException e) {} System.out.println(s3);
      // propri?t?s des constructeurs

char[] tc = {'p', 'r', 'o', 'p', 'r', 'i', 'é', 't', 'é', 's'};
String s4 = new String(tc); System.out.println(s4);
      //   propriétés
String s5 = new String(tc, 6, 3); System.out.println(s5);
      //   été
String s6 = new String(s4 + " " + s2); System.out.println(s6);
      //   propriétés des
```

Méthodes de conversion et de manipulation

Méthodes de conversion

Conversion en tableau d'octets (jeu de caractères par défaut ou jc)

byte[] **getBytes**() byte[] **getBytes**(String jc)

Conversion d'une partie de la chaîne de caractères en tableau de caractères

void **getChars**(int début, int fin, char[] tc, int i)

Conversion des majuscules en minuscules (règles par défaut ou rc)

String **toLowerCase**() String **toLowerCase**(Locale rc)

Conversion des minuscules en majuscules (règles par défaut ou rc)

String **toUpperCase**() String **toUpperCase**(Locale rc)

Méthodes de manipulation

Ajout en fin de la chaîne d'une chaîne de caractères cc

String **concat**(String cc)

Elimination des séparateurs en fin de la chaîne

void **trim**()

Recherche et remplacement du caractère c1 par le caractère c2

String **replace**(char c1, char c2)

testConversion.java

Test de conversion et de manipulation

```
String s1 = new String("Le Norvégien Sigurd Pettersen a remporté");
System.out.println(s1);
      // Le Norvégien Sigurd Pettersen a remporté

String s2 = new String(s1.toUpperCase());
System.out.println(s2);
      // LE NORVÉGIEN SIGURD PETTERSEN A REMPORTÉ

String s3 = new String(s2.toLowerCase());
System.out.println(s3);
      // le norvégien sigurd pettersen a remporté

String s4 = new String(s3.concat(" la course "));
System.out.println(s4 + ".");
      // le norvégien sigurd pettersen a remporté la course .

String s5 = new String(s4.trim());
System.out.println(s5 + ".");
      // le norvégien sigurd pettersen a remporté la course.
```

Méthodes de comparaison et de recherche

Méthodes de comparaison

Comparaison avec la chaîne de caractères cc (ordre lexicographique avec ou sans casse)

int **compareTo**(String cc) int **compareToIgnoreCase**(String cc)

Méthodes de recherche

Comparaison avec entre sous-chaînes de caractères

boolean **regionMatches**(boolean ic, int i, String cc, int j, int taille)

Vérification si la chaîne de caractères cc est une préfixe

boolean **startsWith**(String cc)

Vérification si la chaîne de caractères cc est une suffixe

boolean **endsWith**(String cc)

Recherche de la première occurrence d'un caractère ou d'une chaîne

int **indexOf**(int c) int **indexOf**(int c, int i)

int **indexOf**(String cc) int **indexOf**(String cc, int i)

Recherche de la dernière occurrence d'un caractère ou d'une chaîne

int **lastIndexOf**(int c) int **lastIndexOf**(int c, int i)

int **lastIndexOf**(String cc) int **lastIndexOf**(String cc, int i)

testCompare.java

Test de comparaison et de recherche

```
String s1 = new String("la fille du roi zoulou Goodwill Zwelithini") ;
String s2 = new String("comme conseiller du roi zoulou Cetawayo.") ;

System.out.println(s1.endsWith("."));
      // false

System.out.println(s2.endsWith("."));
      // true

System.out.println(s1.indexOf("roi zoulou"));
      // 12

System.out.println(s2.indexOf("roi zoulou"));
      // 20

System.out.println(s1.regionMatches(true, 12, s2, 20, 10));
      // true
```

Expressions régulières

Méthodes utilisant des expressions régulières

Vérification si comptabilité avec l'expression régulière er

boolean **matches**(String er)

Remplacement de chaque sous-chaîne compatible avec er par la chaîne cc

String **replaceAll**(String er, String cc)

Remplacement de la première sous-chaîne compatible avec er par la chaîne cc

String **replaceFirst**(String er, String cc)

Segmentation complète de la chaîne (délimiteur compatible avec er)

String[] **split**(String er)

Segmentation incomplète de la chaîne (délimiteur compatible avec er)

String[] **split**(String er, int l)

testExpReg.java

Test d'expressions régulières

```
String s1, s2, s3[], er1, er2, er3, er4;
s1 = new String("la fille du roi zoulou Goodwill Zwelithini.");

er1 = new String("[a-zA-Z\\s\\.]*");
System.out.println(er1 + " " + s1.matches(er1));
      // [a-zA-Z\s\.]* true

er2 = new String("((([a-z]*)|([A-Z][a-z]*))(\\s|\\.))*");
System.out.println(er2 + " " + s1.matches(er2));
      // ((([a-z]*)|([A-Z][a-z]*))(\s|\.))* true

er3 = new String("[A-Z][a-z]*");
s2 = new String(s1.replaceAll(er3,"NomPropre "));
System.out.println(er3 + " " + s2);
// [A-Z][a-z]* la fille du roi zoulou NomPropre  NomPropre .

er4 = new String("\\s"); s3 = s1.split(er4);
System.out.println(er4 + " " + s3.length + " " );
for (int i=0;i < s3.length;i++) System.out.print(s3[i] + " ,");
// \s 7 la ,fille ,du ,roi ,zoulou ,Goodwill ,Zwelithini. ,
```

Classe StringBuffer - Principes

Variables modifiables

Accesseurs en écriture (modification),
Manipulation en place de chaînes de caractères,
Minimisation de l'espace mémoire utilisée

Constructeurs

Construction à partir d'une chaîne de caractères cc de type String

StringBuffer(String cc)

Construction d'une chaîne de caractères vide de capacité taille

StringBuffer(int taille)

Méthodes de modification

Remplacement du ième caractère de la chaîne par le caractère c

void **setCharAt** (int i, char c)

Remplacement de la chaîne de caractères pour son inverse

void **reverse** ()

Remplacement de la sous-chaîne [début fin[par la chaîne cc

CharSequence **replace** (int début, int fin, String cc)

Classe StringBuffer - Méthodes

Méthodes de manipulation

Ajout, en fin de la chaîne, d'une chaîne de caractères (conversion d'un type primitif, d'un tableau de caractères, ou chaîne de caractères cc)

StringBuffer **append**(float x)

…

StringBuffer **append**(char[] tc)

StringBuffer **append**(String cc) StringBuffer **append**(StringBuffer cc)

Ajout, à la position i de la chaîne, d'une chaîne de caractères (conversion d'un type primitif, d'un tableau de caractères, ou chaîne de caractères cc)

StringBuffer **insert**(int i, double x)

…

StringBuffer **insert**(int i, String cc)

Suppression du ième caractère de la chaîne

StringBuffer **deleteCharAt**(int i)

Suppression de la sous-chaîne [début fin[

StringBuffer **deleteCharAt**(int début, int fin)

testStrBuffer.java

Test de la classe StringBuffer

```
String s1= new String("la fille du roi zoulou Goodwill Zwelithini   ");
StringBuffer sb1 = new StringBuffer(s1);
int i;

for (i=0;i < sb1.length();i++) {
      char car = sb1.charAt(i);
      if ((car >= 'A') && (car <= 'Z')) {
            car += 'a' - 'A'; sb1.setCharAt(i, car); }
      } System.out.println(sb1 + ".");
      // la fille du roi zoulou goodwill zwelithini   .

i = sb1.length() - 1;
while ((i >= 0) && (sb1.charAt(i)==' ')) sb1.deleteCharAt(i--);
sb1.append("."); System.out.println(sb1);
      // la fille du roi zoulou goodwill zwelithini.

sb1.replace(0, 8,  "le fils"); System.out.println(sb1);
      // le fils du roi zoulou goodwill zwelithini.
```

Principes

Projet OpenSource supporté par IBM

1997 Internationalization API (JDK 1.1)
1999 IBM classes for Unicode
2004 Création du projet sous SourceForge

Bibliothèque d'API sur la gestion de textes Unicode

Normalisation et translitérations
Recherche et comparaison (expression régulières, ordre lexicographique)
Gestion des calendriers et des dates

Disponible en C++ et Java (jar icu4j-4_8)

Principes

Forme canonique d'un texte Unicode

Unicode Normalization Form [Mark Davis &Martin Dürst 2008]
Indispensable pour toutes les méthodes de recherche

Différences structurelles

Combining sequence	Ç	⟷	C ◌̧
Ordering of combining marks	q+̇ +.	⟷	q +. +̇
Hangul	가	⟷	ㄱ + ㅏ
Singleton	Ω	⟷	Ω

Différences visuelles

Width, size, rotated	ｶ	カ	⁓	｛
Superscripts/subscripts	9		9	
Squared characters	アパート			

Classe Normalizer

Bibliothèques de méthodes statiques

Comparaison des formes canoniques

int **compare**(String s1, String s2, int options)
FOLD_CASE_DEFAULT, COMPARE_IGNORE_CASE, ..

4 types de normalisation (NFD, NFC, NFKC, NFKD)

Normalisation d'une chaîne de caractères

String **normalize**(String str, Normalizer.Mode mode)

Test de normalisation

boolean **isNormalized**(String str, Normalizer.Mode mode, UNICODE_3_2)

Concaténation normalisée de chaînes normalisées

String **concatenate**(String left, String right, Normalizer.Mode mode, UNICODE_3_2)

Décomposition et composition de caractères

String **compose**(String str, boolean compat)

String **decompose**(String str, boolean compat)

Classe Normalizer

testNormalizer.java

```
File fin = new File("src/cours07/caractères.txt");
BufferedReader br = new BufferedReader(new FileReader(fin));
s = br.readLine();
br.close();

File fout = new File("src/cours07/caractèresN.txt");
PrintWriter pw = new PrintWriter(fout);

System.out.println(s.length());

pw.print(s);
s1 = new StringBuffer(Normalizer.normalize(s, Normalizer.NFD));
char c = s1.charAt(8);
s1.setCharAt(5, c);
s1.deleteCharAt(8);

pw.print(Normalizer.compose(new String(s1), true));
pw.close();

// àñ
// ãn
```

Principes

Relation d'ordre

Unicode Collation Algorithm [Mark Davis &Ken Whistler 2008]
Indispensable pour toutes les méthodes rapides de recherche (tris)

Dépendance à la langue

Language	Swedish:	z < ö
	German:	ö < z
Usage	German Dictionary:	öf < of
	German Telephone:	of < öf
Customizations	Upper-first	A < a
	Lower-First	a < A

Cinq critères de comparaison

Level	Description*	Examples
L1	Base characters	role < roles < rule
L2	Accents	role < rôle < roles
L3	Case	role < Role < rôle
L4	Punctuation	role < "role" < Role
Ln	Tie-Breaker	role < ro□le < "role"

Classe RuleBasedCollator

Constructeur

RuleBasedCollator(String rules)
Ensemble de règles d'inégalité

UCA	Tailoring: & C < č <<< Č < ć <<< Ć
CUKIĆ RADOJICA	CUKIĆ RADOJICA
ČUKIĆ SLOBODAN	CUKIĆ SVETOZAR
CUKIĆ SVETOZAR	CURIĆ MILOŠ
ČUKIĆ ZORAN	CVRKALJ ĐURO
CURIĆ MILOŠ	ČUKIĆ SLOBODAN
ĆURIĆ MILOŠ	ČUKIĆ ZORAN
CVRKALJ ĐURO	ĆURIĆ MILOŠ

Règles d'inégalité d'un ordre lexcographique
String **getRules**()
Vérification de l'ordre lexicographique
boolean **isFrenchCollation**()
Propriétés de l'ordre lexicographique
boolean **isUpperCaseFirst**()

Ordre lexicographique
int **compare**(String source, String target)

testCollator.java

Classe RuleBasedCollator

```
RuleBasedCollator rbc1 = null, rbc2 = null;
try {
String simple = "&9 < a, A < b, B < c, C; ch, cH, Ch, CH < ca; d , D <
e, E << é, ê, è";
rbc1 = new RuleBasedCollator(simple);

// ordre lexicographique par défaut
rbc2 = (RuleBasedCollator)Collator.getInstance();
}
catch (Exception e) {}


String s1 = "charge", s2 = "casse";

System.out.println(rbc1.compare(s1, s2));
// -1

System.out.println(rbc2.compare(s1, s2));
//  1
```