CSE 1710

Lecture 16

Text, Strings

Goals/To do:

Given a string and a character, derive the **frequency of the character within the string**

Given a string, a target character and a replacement character, **implement character substitution.**

Given a numeric value in string format, **parse into numeric type**

Goals/To understand:

- difference between `char`, `String`, and `StringBuffer`
- The non-primitive String masquerades as a primitive type
- Pattern-matching abstractions (regular expressions)
- The difference between raw and formatted text; how to separate content from presentation

Unicode

- Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.
- Java and other languages use Unicode

- a *unicode character*:
  - is a non-negative numeric value
  - has a corresponding character according to the Unicode character tables (as defined by the Unicode Consortium)

The letter J is found in
column '004' and
row 'A',
which makes '004A'

This is a hexadecimal number, denoted
\u004A

To convert a hexadecimal number to decimal:

$d_3d_2d_1d_0 = d_3\times16^3 + d_2\times16^2 + d_1\times16^1 + d_0\times16^0$

where d_i takes on values [0, 15]
the value 10 is denoted by A (or a)
the value 11 is denoted by B
the value 12 is denoted by C
the value 13 is denoted by D
the value 14 is denoted by E
the value 15 is denoted by F

so to convert \u004A to decimal:
$= 0\times16^3 + 0\times16^2 + 4\times16^1 + 10\times16^0$
$= 4\times16 + 10\times1$
$= 64 + 10$
$= 74$

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019	) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D	] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

http://unicode.org/charts/PDF/U0000.pdf

Unicode

- The Unicode Standard consists of a repertoire of more than 109,000 characters covering 93 scripts
  - Cyrillic, Latin, Bengali, Thai, Greek, …
  - the basic set is “Controls and Basic Latin”
  - U000.pdf, also see Appendix A of JBA
- Unicode value denoted \uXXXX, where XXXX is a hexadecimal value
  - the decimal value 15 is represented as \u000F
- unicode makes is possible to talk about the *distance* between two character

How Java uses Unicode

- `String` and `StringBuffer` objects encapsulate a string as a *sequence* of *unicode characters*
- the `char` primitive data type make use of unicode as well

L16App1

About the character sequence…

- the `String` and `StringBuffer` classes encapsulate a string as a *sequence* of characters
- the sequence is indexed L16App2
  - the *first* position is index "0"
  - the *final* position is index "the length of the sequence minus 1"
- `String` services to tell us about the sequence
  - `int : length()`
  - `char : charAt(int)`
  - `String : substring(int, int)`
    - first value is start index, inclusive; second value is end index, exclusive
- what if index is out of bounds?
- what if end index is smaller than start index?

What if the sequence has no characters at all?

- this is the *empty string*
- the string has length **zero**
- what if you hear the term "null string"? what does this mean?
  - not really a correct-formed phrase, there is no such thing
  - often used to mean a string reference that is set to null.

How can we modify the sequence?

- Once a string object is created, it cannot be changed.
  - This is called *immutability*
  - Strings are *immutable*
- Instead of modifying the sequence, we just create new strings.
- It is fast and easy, thanks to the + operator

L16App2b

- Given this, is it correct to say that String has mutators?
  - not technically; they are actually *generators of new modified objects*

Iterating over a String

- many different ways to iterate
- several different services to use…

L16App3

- `theString.toCharArray()`
  - provides array that we can iterate over
  - `for (char c : theString.toCharArray()) { }`
- `theString.charAt(index)`
  - we can iterate over index values
- `theString.substring(startIndex,finishIndex)`
  - we can iterate over the starting index values

String matching/comparison (basic)

- does s1 *match* s2?
- does c1 *match* c2?

- what does the equality boolean operator == tell us?
  - `boolean isMatch = c1==c2;`
  - `boolean isMatch = s1==s2;`

- what does `.equals(String)` tell us?
  - `boolean isMatch = s1.equals(s2);` L16App4

- what does `.compareTo(String)` tell us?
  - `int differingIndexPos = s1.compareTo(s2);`

Difference between creation of `String` objects

- can construct String objects two different ways
  - implicit construction
    - use the pretend "literal" format
    - e.g.,
      `String s1 = "Hello";`
  - explicit construction
    - use keyword `new`
    - e.g.,
      `String s3 = new String("Hello");`
  - explicit construction always creates new object
  - if object with the same state already exists, implicit construction will re-use previously created object

Elaboration of "`compareTo(String)`"

(sort of) *"tell me whether the passed string comes before this string in the dictionary"*

"aardvark".compareTo("anvil")

- *anvil* does not come before *aardvark* in the dictionary, so the result is no (negative value)

"anvil".compareTo("aardvark")

- *aardvark* **does** come before *anvil* in the dictionary, so the result is yes (positive value)

(better) *"tell me whether the passed string comes before this string in the dictionary and, for the first character that is the determining factor, what is the distance"*

- the second character is the determining factor ('a' vs 'n', there is a distance of 13 between them)

Counting Character Frequency

- Given a string and a character, derive the **frequency of the character within the string**
- need to put together *iteration* and *comparsion*
  1. iterate over the string to examine each character
  2. for each character, compare to target
  3. conditionally update counter

L16App5

L16App6