Lecture 17
CSE11 – Fall 2013
Strings

Strings

- Strings are critical data types in almost every language
  - Convert input typed at keyboard to:
    - Commands
    - Variable input
  - Convert internal data to human-readable form
- Java has a large number of functions for working with Strings
- A String is an object-type in java, not a primitive type, However ....
  - It uses something called “interning” to store statically defined Strings
  -

String Interning

- string *interning* is a method of storing only one copy of each distinct string value, which must be *immutable* (constant, not changing)
- Java uses interning and there is a potential pitfall
  - Unmodified interns of the same string are equal (==)
  - Modified interns of the same string are NOT equal
- One can test equality of two strings (are they the same sequence of characters)
  - boolean “equals” method
  - string1.equals(string2)
- Example: StringFoo.java
-

StringFoo.java (main method)

```
public static void main (String [] args)
{
        String s1 = "Hello, Computer.";
        String s2 = new String("Hello, Computer.");
        String s3 = "Hello, Computer.";

        System.out.format ("Strings s1 and s2 are different objects.\n");
        System.out.format ("Strings s1 and s3 are string 'interns'.\n");

        System.out.format("s1: '%s'\n", s1);
        System.out.format("s2: '%s'\n", s2);
        System.out.format("s3: '%s'\n", s3);

        System.out.format("s1 == s2? %s\n", s1 == s2 );
        System.out.format("s1 == s3? %s\n", s1 == s3 );


        System.out.format ("Concatenate s1=s1+s2, s3=s3+s2 and check.\n");
        s1 = s1 + s2;
        s3 = s3 + s2;
        System.out.format("s1: '%s'\n", s1);
        System.out.format("s2: '%s'\n", s2);
        System.out.format("s3: '%s'\n", s3);
        System.out.format("s1 == s2? %s\n", s1 == s2 );
        System.out.format("s1 == s3? %s\n", s1 == s3 );
        System.out.format("s1.equals(s3) %s\n", s1.equals(s3));
}
```

false

true

false

true

String Length

- A String is *almost* an array of characters
  - length of string s1
    - s1.length()

    NOT
    - s1.length
- Retrieving the character (java type char) at location n
  - s1.charAt(n)
    - n = 0...(s1.length – 1);
  - Note: there is no method to change a single character in a String
  - Just like arrays,accessing in improper index generates an exception (StringIndexOutOfBoundsException)

Converting to/from a String and character array

- Create a char array from a String
  - char [] c1 = s1.toCharArray()
- Create a String from a char array
  - String s2 = new String(c1)
  - note: length of s2 is the c1.length
    - even when c1 has null ('\u0000') characters
    -
- You CANNOT modify a String, all methods that appear to modify a String, actually return a new String Object

Converting to/from a String and character array

- Create a char array from a String
  - char [] c1 = s1.toCharArray()
- Create a String from a char array
  - String s2 = new String(c1)
  - note: length of s2 is the c1.length
    - even when c1 has null ('\u0000') characters
    -
- You CANNOT modify a String, all methods that appear to modify a String, actually return a new String Object

Various String Functions

- http://docs.oracle.com/javase/6/docs/api/java/lang/String.html
- -15- different constructors
- Look at some common “groups”/”classes” of operations
  - Comparisons
  - Substrings
  - Converting/Splitting Strings
  - Finding “things” in Strings

Various String Functions

- http://docs.oracle.com/javase/6/docs/api/java/lang/String.html
- -15- different constructors
- Look at some common “groups”/”classes” of operations
  - Comparisons
  - Substrings
  - Converting/Splitting Strings
  - Finding “things” in Strings

Comparison

- equals(s1 : String) : boolean
- equalsIgnoreCase(s1 : String) : boolean
- compareTo(s1 : String) : int
- compareToIgnoreCase(s1 : String) : int
- regionMatches(toffset: int, s1: String, offset: int, len: int): boolean
- startsWith(prefix: String): boolean
- endsWith(suffix: String): boolean

equals(), equalsIgnoreCase()

- Returns boolean if Strings have the identical sequence of characters
  - s1.equals(s2)
- equalsIgnoreCase()
  - comparison without regard to case
  - a==A, b==B, c==C, ... and so on
  - s1.equalsIgnoreCase(s2)

compareTo(), compareToIgnoreCase()

- Compares strings in Lexicographic order
  - This is “Dictionary” order
    - e.g., apple < apples
    - s1.compareTo(s2)
- Computers defined lexicographic (or lexical) order by comparing the ASCII codes of each character in sequence.
- returns: 0 if Strings are EQUAL
- returns < 0 if s1 < s
- returns > 0 if s2 > s1

Lexicographic Ordering

- Characters are stored as ASCII codes.
- Lexicographic (or lexical) ordering is an element by element comparison of the ASCII codes

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

RegionMatches

- boolean regionMatches(int toffset, String other, int ooffset, int len)
- boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
  - Example
  - s1 = “This is home of the UCSD Tritons”
  - s2 =”The UCSD Triton Bookstore is in Price”
  - s1.regionMatches(20, s2, 4, 11) is true
  - s1.regionMatches(20, s2, 4, 12) is false

startsWith(), endsWidth()

- Determine if a string starts or ends with a substring
- startsWith(String prefix)
  - Tests if this string starts with the specified prefix.
- startsWith(String prefix, int toffset)
  - Tests if the substring of this string beginning at the specified index starts with the specified prefix
- endsWith(String suffix)
  - Tests if this string ends with the specified suffix.

Finding Indexes of Characters/Strings

- indexOf(ch: char): int
- indexOf(ch: char, fromIndex: int): int
- indexOf(s: String): int
- indexOf(s: String, fromIndex: int): int
- lastIndexOf(ch: char): int
- lastIndexOf(ch: char, fromIndex: int): int
- lastIndexOf(s: String): int
- lastIndexOf(s: String, fromIndex: int): int

Substrings

- Extract smaller strings from larger strings
- substring(int beginIndex)
  - Returns a new string that is a substring of this string.
  - “this is a longish string”.substring(9)
    - --> “longish string”
- substring(int beginIndex, int endIndex)
  - Returns a new string that is a substring of this string.
  - “this is a longish string”.substring(9,12)
    - --> “long”