

Strings & Characters

Learning Objectives

- To be able to create and manipulate String values
- To be able to compare String values

Aside: Literal Values

- Literal values are "Hard-coded" values that are written in the code exactly as how they should be evaluated.
- Used most often for initializing a variable or as part of an expression

```
int a = 3;           // 3 is an int literal value
double b = a * 3.14; // 3.14 is a double literal
String s = "3.14";  // "3.14" is a string literal
```

Strings

- Strings hold sequences of characters (a, b, c, \$, etc)
- Can perform operations on strings like concatenation and others
- Anything between `""` is a string literal

Strings are “objects” of the String class, although they behave in many ways like a primitive type, so we study it now.

String Variable Declaration & Initialization

`String` variables work just like `int`, `double`, and `boolean` variables: declare them by writing the type & name of the variable and then give them an initial value.

```
String variableName = "stringLiteral";  
String firstName = "Lisa";
```

String values

A String holds a sequence of characters

- Characters include things like 'a', 'b', '1', '\$', '%', '.', etc.

These characters are stored in a *sequence*, and are numbered from the front of the sequence starting with **0**. The last element is at index **length - 1**.

```
String example = "Hello!";
                012345
String birthday = "August 29";
                012345678
String empty    = "";
```



We usually start counting at 0 in programming. Will see this more with arrays :)

String values: null

A String, since it is technically an *object type*, can be initialized to a `null` reference

A `null` reference means that the variable does not refer to a space in memory

```
String variableName;    // default String variable value is null
String nulledVar = null; // this sets a variable to null explicitly
```

 *More on null in future lectures about objects. Just keep this in the back of your mind for now.* 

String operations: Concatenation

Use the `+` or `+=` operators to concatenate (combine) two Strings

```
String a = "Serena";  
String b = " Williams";  
String c = a + b;  
System.out.println(c); // prints Serena Williams
```


String operations: Concatenation

Use the `+` or `+=` operators to append a primitive type value to a String



- will automatically convert that value to String

```
String a = "Serena";  
String b = " Williams";  
String c = a + b + 100;  
System.out.println(c); // prints Serena Williams100
```

Aside: Object methods and `.`

The `+` and `+=` operator on strings is somewhat unique. Normally performing an operation on an object requires different syntax: the `.` operator.

```
String a = "Serena";  
String b = " Williams";  
String c = a.concat(b); // same as a + b  
System.out.println(c); // prints Serena Williams
```

 There is NO space around the `.` 

*String methods: **length()***

`length()` method returns the number (an `int`) of characters in the string, including spaces and special characters like punctuation.

```
String a = "Serena";  
int len = a.length(); // S, e, r, e, n, a is 6 characters  
System.out.println(len); // prints 6
```

String methods: `substring()`

`substring(int from, int to)`

- returns a new string with the characters in the current string starting with the character at the `from` index and ending at the character *before* the `to` index

```
String a = "Serena";  
String b = a.substring(0, 3);  
System.out.println(b); // prints "Ser"  
String c = a.substring(2, 4);  
System.out.println(c); // prints "re"
```

```
Serena  
012345
```

String methods: `substring()`

`substring(int from)`

- returns a new string with all the characters in the current string starting after the character at the `from` index.

```
String a = "Serena";  
String c = a.substring(3);  
System.out.println(c); // prints "ena"
```

```
Serena  
012345
```

String methods: `indexOf()`

`indexOf(String str)` searches for the string `str` in the current string and returns:

- the index of the beginning of `str` in the current string,
- or `-1` if it isn't found

```
String a = "Serena";  
int x = a.indexOf("er"); // x has value 1  
int y = a.indexOf("ena"); // y has value 3  
int z = a.indexOf("sa"); // z has value -1
```

String methods: `charAt()`

`str.charAt(int index)` returns the `char` at position `index` in the input `str`:

- `index` must be between `0` and `str.length() - 1`
- return type is `char`, which is a data type used for storing individual characters

```
String a = "Serena";  
char x = a.charAt(0); // x has value 'S'  
char y = a.charAt(2); // y has value 'r'  
char z = a.charAt(5); // z has value 'a'
```

Comparing Strings

Strings (and objects) **cannot** be compared using operators like `==`, `<`, `>`.

The method `first.compareTo(String second)` compares two strings character by character.

- If they are **equal**, it returns **0**
- If the first string is alphabetically ordered **before** the second string, it returns a **negative number**
- If the first string is alphabetically ordered **after** the second string, it returns a **positive number**

Comparing Strings

```
/// S comes before W in the alphabet
String a = "Serena";
String b = "Williams";
System.out.println(a.compareTo(b)); // prints -4;
System.out.println(b.compareTo(a)); // prints 4;
```

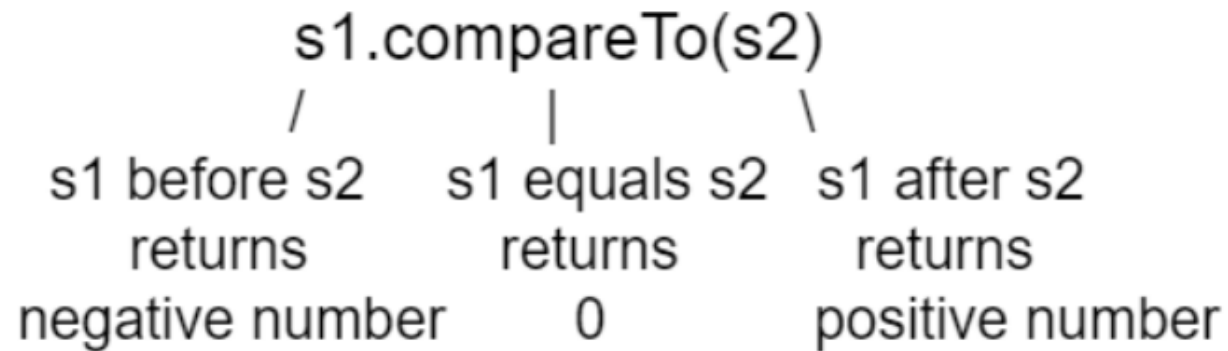


Figure 2: compareTo returns a negative or positive value or 0 based on alphabetical order

String Equality

DO NOT USE `==` TO CHECK IF TWO STRINGS ARE EQUAL!

String equality

The `equals(String other)` method compares the two strings character by character and returns a `boolean`.

```
String a = "Serena";  
String b = "Williams";  
System.out.println(a.equals(b)); // prints false  
System.out.println(a.equals(a)); // prints true
```

`compareTo`, `equals` and most string methods are case-sensitive!

```
"HI".equals("hi"); // returns false
```

Live Demo: StringManips.java

Write a program `StringManips.java` that does two things

Problem 1:

- Given a string, we will print a new string made of 3 copies of the last 2 characters of the original string.

Problem 2:

- Given a string, the program will print a version without both the first and last characters


Both assume the input strings have length ≥ 2 .

The `char` Data Type

Strings

Recall that Strings are sequences of characters:

"Harry Smith"  string of 11 characters including space (' ')

"215-898-3500"  string of 12 characters including digits and '-'

"a"  string of one character, 'a'

""  empty string (string of 0 characters)

String Iteration Toolkit

Given a `String s...`

- determine its length using `s.length()`
- get a character at a given position `i` using `s.charAt(i)`.
 - `i` must be between `0` and `s.length() - 1`

```
for (int i = 0; i < s.length(); i++) {  
    System.out.println(s.charAt(i));  
}
```


char

`char` is a primitive data type used to store a single character.

- `char` literals are expressed using single quotes (`' '`)
 - `'a'`, `'8'`, `' '` are all `char` values
 - `'aa'` is not a valid `char` because it expresses two characters!
- `char` values are represented using letters/digits/punctuation, but they are represented internally as *small integers*!
 - Computers only understand numbers (just 0 and 1, really), so we have to be clever about how we get them to think about symbols like letters.

ASCII, Unicode, & char

ASCII (American Standard Code for Information Interchange) is a system of assigning numbers to characters in order to store them in computers.

- System allowed for representing 128 different symbols using numbers 0–127.
- Fine back in the day, but quite limited, especially outside of American English alphabet

The diagram shows three horizontal arrows pointing right, representing bit patterns for columns 0 through 5. Below this is a grid where the top row contains bit patterns for each column (0-5) and the left column contains bit patterns for each row (0-15). The grid cells contain the corresponding ASCII character.

b ₇ ↓	Column →	0	1	2	3	4	5
0	0	NUL	DLE	SP	0	@	P
1	1	SOH	DC1	!	1	A	Q
0	2	STX	DC2	"	2	B	R
1	3	ETX	DC3	#	3	C	S
0	4	EOT	DC4	\$	4	D	T
1	5	ENQ	NAK	%	5	E	U
0	6	ACK	SYN	&	6	F	V
1	7	BEL	ETB	'	7	G	W
0	8	BS	CAN	(8	H	X
1	9	HT	EM)	9	I	Y
0	10	LF	SUB	*	:	J	Z
1	11	VT	ESC	+	;	K	[
0	12	FF	FS	,	<	L	\
1	13	CR	GS	-	=	M]
0	14	SO	RS	.	>	N	^
1	15	SI	US	/	?	O	_

ASCII, Unicode, & `char`

Unicode is **also** a system of assigning numbers to characters in order to store them in computers.

- System is very complicated—not exactly as simple as `'A' == 65`
- Can express characters from multiple alphabets and also emoji (🌮 🌴 😎)

Technically, Java uses the 16 bit Unicode standard to map `char` values to integers, but we'll often say "ASCII" (as-kee) for shorthand

STRINGS

ASCII, Unicode, & char

1	1	001	SOH	(start of heading)	33	21	041	0x21	!	65	41	101	0x65	A	97	61	141	0x97	a
			STX	(start of text)	34	22	042	0x22	"	66	42	102	0x66	B	98	62	142	0x98	b
			ETX	(end of text)	35	23	043	0x23	#	67	43	103	0x67	C	99	63	143	0x99	c
4	4	004	EOT	(end of transmission)	36	24	044	0x24	\$	68	44	104	0x68	D	100	64	144	0x100	d
5	5	005	ENQ	(enquiry)	37	25	045	0x25	%	69	45	105	0x69	E	101	65	145	0x101	e
6	6	006	ACK	(acknowledge)	38	26	046	0x26	&	70	46	106	0x70	F	102	66	146	0x102	f
7	7	007	BEL	(bell)	39	27	047	0x27	'	71	47	107	0x71	G	103	67	147	0x103	g
8	8	010	BS	(backspace)	40	28	050	0x28	(72	48	110	0x72	H	104	68	150	0x104	h
9	9	011	TAB	(horizontal tab)	41	29	051	0x29)	73	49	111	0x73	I	105	69	151	0x105	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	0x2A	*	74	4A	112	0x74	J	106	6A	152	0x106	j
11	B	013	VT	(vertical tab)	43	2B	053	0x2B	+	75	4B	113	0x75	K	107	6B	153	0x107	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	0x2C	,	76	4C	114	0x76	L	108	6C	154	0x108	l
13	D	015	CR	(carriage return)	45	2D	055	0x2D	-	77	4D	115	0x77	M	109	6D	155	0x109	m
14	E	016	SO	(shift out)	46	2E	056	0x2E	.	78	4E	116	0x78	N	110	6E	156	0x110	n
15	F	017	SI	(shift in)	47	2F	057	0x2F	/	79	4F	117	0x79	O	111	6F	157	0x111	o
16	10	020	DLE	(data link escape)	48	30	060	0x30	0	80	50	120	0x80	P	112	70	160	0x112	p
17	11	021	DC1	(device control 1)	49	31	061	0x31	1	81	51	121	0x81	Q	113	71	161	0x113	q
18	12	022	DC2	(device control 2)	50	32	062	0x32	2	82	52	122	0x82	R	114	72	162	0x114	r
19	13	023	DC3	(device control 3)	51	33	063	0x33	3	83	53	123	0x83	S	115	73	163	0x115	s
20	14	024	DC4	(device control 4)	52	34	064	0x34	4	84	54	124	0x84	T	116	74	164	0x116	t
21	15	025	NAK	(negative acknowledge)	53	35	065	0x35	5	85	55	125	0x85	U	117	75	165	0x117	u
22	16	026	SYN	(synchronous idle)	54	36	066	0x36	6	86	56	126	0x86	V	118	76	166	0x118	v
23	17	027	ETB	(end of trans. block)	55	37	067	0x37	7	87	57	127	0x87	W	119	77	167	0x119	w
24	18	030	CAN	(cancel)	56	38	070	0x38	8	88	58	130	0x88	X	120	78	170	0x120	x
25	19	031	EM	(end of medium)	57	39	071	0x39	9	89	59	131	0x89	Y	121	79	171	0x121	y
26	1A	032	SUB	(substitute)	58	3A	072	0x3A	:	90	5A	132	0x90	Z	122	7A	172	0x122	z
27	1B	033	ESC	(escape)	59	3B	073	0x3B	;	91	5B	133	0x91	[123	7B	173	0x123	{
28	1C	034	FS	(file separator)	60	3C	074	0x3C	<	92	5C	134	0x92	\	124	7C	174	0x124	
29	1D	035	GS	(group separator)	61	3D	075	0x3D	=	93	5D	135	0x93]	125	7D	175	0x125	}

char Values as Numbers

"Given a `char c`, how can I ask if it's a lowercase letter from `'a'–'z'`?"

- Since `char` values have a number representation, it means that we can straightforwardly order them using `>` and `<`

```
if ('a' <= c && c <= 'z') {  
    System.out.println(c + " is a lowercase letter");  
}
```

char Values as Numbers

"Given a `char c`, how can I turn it from an uppercase letter to a lowercase letter?"

- Since `char` values have a number representation, it means that we can modify them using simple arithmetic
- `'a' - 'z'` are `97-122` respectively
- `'A' - 'Z'` are `65-90`
- The difference between a lowercase and uppercase letter is `32`

```
char lowercase = c + 32;
```

Converting a `char` to a `String`

Use concatenation to append a `char` on to the end of an empty string (`""`) in order to get a `String` that contains just the `char` value.

```
char c = 'a';  
String s = "" + c;  
System.out.println(s); // prints "a"
```

⚠ ⚠ ⚠ Use this for Letter Viewer in HW1! ⚠ ⚠ ⚠