Strings & Wrapper Classes

CS102A Lecture 8

James YU

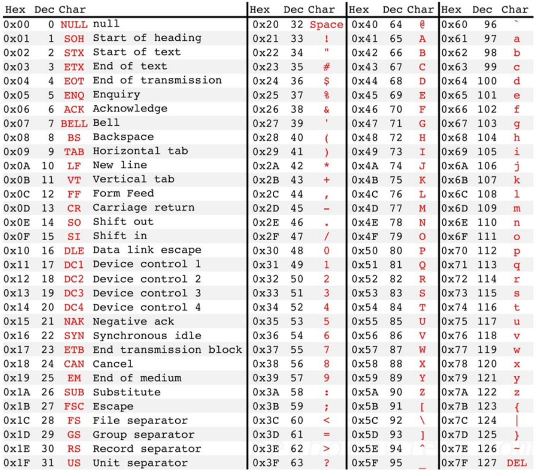
Nov. 2, 2020

南方科技大學
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Objective

- To create and manipulate strings.
  - Immutable character-string objects of class `String`.
  - Mutable character-string objects of class `StringBuilder`.
- To create and manipulate objects of class `Character`.
- Learn wrapper classes of primitive types.

Characters: Building blocks of Java programs



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

The primitive type char

- The char data type is a single 16-bit Unicode character.
  - \u0000 – \uffff: 65536 characters, covering characters for almost all modern languages, and a large number of symbols.
- Programs often contain character literals (in single quotes).

```java
char c1 = '\u0030';
char c2 = '\u0041';
char c3 = '\u4e2d';
char c4 = '\u56fd';
System.out.printf("%c %c %c %c", c1, c2, c3, c4);
```

```
0 A 中 国
```

The primitive type char

- The char data type is a single 16-bit Unicode character.
  - \u0000 – \uffff: 65536 characters, covering characters for almost all modern languages, and a large number of symbols.
- Programs often contain character literals (in single quotes).

```java
char c1 = 'a';
char c2 = 97;
char c3 = '\u0061';
char c4 = 'a' + 1;
System.out.printf("%c %c %c %c", c1, c2, c3, c4);
```

```
a a a b
```

String

- A string is a sequence of characters
  - `"I like Java programming"`
- A string may include letters, digits and various special characters, such as +, -, *, / and $.
  - `"I \u2665 Java programming"`

Creating String objects

- String objects can also be created by using the new keyword and various String constructors.

```java
String s1 = new String("hello world");
String s2 = new String(); // empty string (length is 0)
String s3 = new String(s1);
char[] charArray = {'h', 'e', 'l', 'l', 'o'};
String s4 = new String(charArray);
String s5 = new String(charArray, 1, 3); // string "ell"
```

String assignments

- A string may be assigned to a String reference.

```java
String s = "hello world";
```

  - The statement initializes String variable s to refer to a String object that contains the string "hello world".

```java
String s2 = s;
```

  - The statement makes s2 and s to refer to (sometimes we say “point to”, they mean the same thing) the same String object.

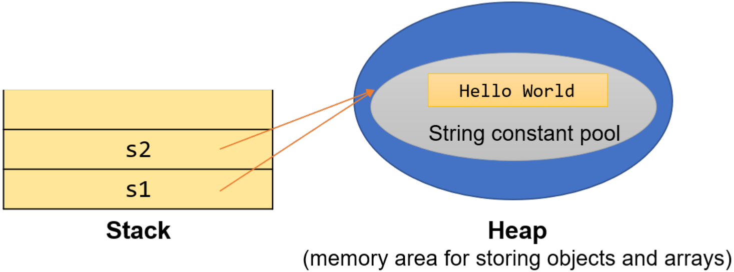
Comparing items

```java
char c1 = 'a';
char c2 = 'a';
if (c1 == c2)
  System.out.println("c1 and c2 are the same");
String s1 = new String("Hello");
String s2 = new String("Hello");
if (s1 == s2)
  System.out.println("s1 and s2 are the same");
String s3 = "Hello";
String s4 = "Hello";
if (s3 == s4)
  System.out.println("s3 and s4 are the same");
```

Creating String objects

- A string is an object of class String.
- String objects can be created by string literals (a sequence of characters in double quotes).

```
String s1 = "Hello World";
// no new objects will be created
String s2 = "Hello World";
```



Immutability

- In Java, `String` objects are immutable.
  - `String`s are constants; their values cannot be changed after they are created.
  - Because `String` objects are immutable, they can be shared safely.
- **Any modification creates a new `String` object.**

`String` methods

- `length` returns the length of a string (i.e., the number of characters).
- `charAt` helps obtain the character at a specific location in a string.
- `getChars` helps retrieve a set of characters from a string as a `char` array.
- These are instance methods that interact with the specific data of objects. Calling them requires an object reference.

The method `length`

```java
public class StringExamples {
  public static void main(String[] args) {
    String s1 = "hello world";
    System.out.printf("s1: %s", s1);
    System.out.printf("\nLength of s1: %d", s1.length());
  }
}
```

The method charAt

```java
public class StringExamples {
  public static void main(String[] args) {
    String s1 = "hello world";
    System.out.printf("s1: %s", s1);
    System.out.print("\nThe string reversed is: ");
    for(int count = s1.length() - 1; count >=0; count--) {
      System.out.printf("%c", s1.charAt(count));
    }
  }
}
```

The method getChars

- getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

```java
public class StringExamples {
  public static void main(String[] args) {
    String s1 = "hello world";
    char[] charArray = new char[5];
    System.out.printf("s1: %s\n", s1);
    s1.getChars(0, 5, charArray, 0);
    for(char c : charArray) {
      System.out.print(c);
    }
  }
}
```

Comparing Strings

- When primitive-type values are compared with ==, the result is true if both values are identical.

```java
int a = 2, b = 2;
if (a == b) System.out.println("a = b"); // prints a = b
```

- When references (memory addresses) are compared with ==, the result is true if both references refer to the same object in memory.

```java
String s1 = "Hello World";
String s2 = "Hello World";
if(s1 == s2) System.out.println("s1 = s2"); // prints s1 = s2
```

Comparing Strings

```java
String s1 = "Hello World";
String s2 = s1 + "";
if(s1 == s2) System.out.println("s1 = s2"); // prints s1 = s2?
```

- No. The condition will evaluate to false because the String variables s1 and s2 refer to two different String objects, although the strings contain the same sequence of characters.
- To compare the actual contents (or state information) of objects (strings are objects) for equality, a method must be invoked.

The method equals

- Method equals tests any two objects for equality – the strings contained in the two String objects are identical.

```java
String s1 = "Hello World";
String s2 = s1 + "";
if(s1.equals(s2)) System.out.println("s1 = s2"); // true
```

- Uses lexicographical comparison – it compares the integer Unicode values that represent each character in each String.

```java
String s1 = "hello";
String s2 = "HELLO";
if(s1.equals(s2)) System.out.println("s1 = s2"); // false
```

The method equalsIgnoreCase

- Method equalsIgnoreCase ignores whether the letters in each String are uppercase or lowercase when performing a comparison.

```java
String s1 = "hello";
String s2 = "HELLO";
if(s1.equalsIgnoreCase(s2)) System.out.println("s1 = s2");
```

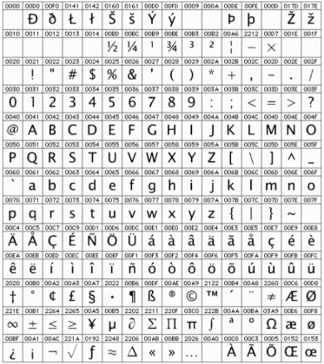
The method compareTo

- compareTo compares two strings (lexicographical comparison):
  - Returns 0 if the Strings are equal (identical contents).
  - Returns a negative number if the String that invokes compareTo (s1) is less than the String that is passed as an argument (s2).
  - Returns a positive number if the String that invokes compareTo (s1) is greater than the String that is passed as an argument (s2).

```java
String s1 = "hello";
String s2 = "HELLO";
int result = s1.compareTo(s2); // value of result?
```

Comparing strings

- What does it mean when we say a string `s1` is greater than another string `s2`?
  - When we sort last names, we naturally consider that “Jones” > “Smith”, because the letter ‘J’ comes before ‘S’ in the alphabet of 26 letters.
  - All characters in computers are represented as numeric codes. The characters form an ordered set (“a very large alphabet”).
  - When the computer compares `String`s, it actually compares the numeric codes of the characters in the `String`s.



0000	00D0	00F0	0141	0142	0160	0161	00DD	00FD	0009	000A	00DE	00FE	000D	017D	017E
	Ð	ð	Ł	ł	Š	š	Ý	ý			Þ	þ		Ž	ž
0010	0011	0012	0013	0014	00BD	00BC	00B9	00BE	00B3	00B2	00A6	2212	00D7	001E	001F
					½	¼	¹	¾	³	²	¦	−	×		
0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	006A	006B	006C	006D	006E	006F
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	007F
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
00C4	00C5	00C7	00C9	00D1	00D6	00DC	00E1	00E0	00E2	00E4	00E3	00E5	00E7	00E9	00E8
Ä	Å	Ç	É	Ñ	Ö	Ü	á	à	â	ä	ã	å	ç	é	è
00EA	00EB	00ED	00EC	00EE	00EF	00F1	00F3	00F2	00F4	00F6	00F5	00FA	00F9	00FB	00FC
ê	ë	í	ì	î	ï	ñ	ó	ò	ô	ö	õ	ú	ù	û	ü
2020	00B0	00A2	00A3	00A7	2022	00B6	00DF	00AE	00A9	2122	00B4	00A8	2260	00C6	00D8
†	°	¢	£	§	•	¶	ß	®	©	™	´	¨	≠	Æ	Ø
221E	00B1	2264	2265	00A5	00B5	2202	2211	220F	03C0	222B	00AA	00BA	03A9	00E6	00F8
∞	±	≤	≥	¥	µ	∂	∑	∏	π	∫	ª	º	Ω	æ	ø
00BF	00A1	00AC	221A	0192	2248	2206	00AB	00BB	2026	00A0	00C0	00C3	00D5	0152	0153
¿	¡	¬	√	ƒ	≈	∆	«	»	…		À	Ã	Õ	Œ	œ

The method regionMatches

- regionMatches compare portions of two Strings for equality:
  - The first argument is the starting index in the String that invokes the method (s1).
  - The second argument is a comparison String.
  - The third argument is the starting index in the comparison String.
  - The last argument is the number of characters to compare between the two Strings.
  - Returns true only if the specified number of characters are lexicographically equal.

```java
String s1 = "Hello World";
String s2 = "hello world";
boolean result = s1.regionMatches(0, s2, 0, 5); // true or false?
```

The method regionMatches

- regionMatches is overloaded (it has a five-argument version):
  - When the first argument is true, the method ignores the case of the characters being compared.
  - The remaining arguments are identical to those described for the four-argument regionMatches method.

```java
String s1 = "Hello World";
String s2 = "hello world";
boolean result = s1.regionMatches(true, 0, s2, 0, 5); // true
```

The method startsWith & endsWith

- The methods startsWith and endsWith determine whether a string starts or ends with the method argument, respectively.

```
String s1 = "Hello World";
if(s1.startsWith("He")) System.out.print("true"); // true

String s1 = "Hello World";
if(s1.startsWith("llo", 2)) System.out.print("true"); // true

String s1 = "Hello World";
if(s1.endsWith("ld")) System.out.print("true"); // true
```

Locating characters in Strings

- indexOf locates the first occurrence of a character in a String.
  - If the method finds the character, it returns the character’s index in the String; otherwise, it returns -1.
- Two-argument version of indexOf:
  - Take one more argument: the starting index at which the search should begin.

```java
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.indexOf('c')); // 2
System.out.println(s.indexOf('$')); // -1
System.out.println(s.indexOf('a', 1)); // 13
```

Locating characters in `String`s

- `lastIndexOf` locates the last occurrence of a character in a `String`.
    - The method searches from the end of the `String` toward the beginning.
    - If it finds the character, it returns the character's index in the String; otherwise, it returns `-1`.
- Two-argument version of `lastIndexOf`:
    - The character and the index from which to begin searching backward.

```java
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.lastIndexOf('c')); // 15
System.out.println(s.lastIndexOf('$')); // -1
System.out.println(s.lastIndexOf('a', 8)); // 0
```

Locating substrings in Strings

- The versions of methods indexOf and lastIndexOf that take a String as the first argument perform identically to those described earlier except that they search for sequences of characters (or substrings) that are specified by their String arguments.

```java
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.indexOf("def")); // 3
System.out.println(s.indexOf("def", 7)); // 16
System.out.println(s.indexOf("hello")); // -1
System.out.println(s.lastIndexOf("def")); // 16
System.out.println(s.lastIndexOf("def", 7)); // 3
System.out.println(s.lastIndexOf("hello")); // -1
```

Extracting substrings from Strings

- substring methods create a new String object by copying part of an existing String object.
- The one-integer-argument version specifies the starting index (inclusive) in the original String from which characters are to be copied.
- Two-integer-argument version specifies the starting index (inclusive) and ending index (exclusive) to copy characters in the original String.

```java
String s = "abcdefghijklmabcdefghijklm";
System.out.println(s.substring(20)); // hijklm
System.out.println(s.substring(3, 6)); // def
```

Concatenating Strings

- String method concat concatenates two String objects and returns a new String object containing the characters from both original Strings. The original Strings to which s1 and s2 refer are not modified (recall that Strings are immutable).

```java
String s1 = "Happy ";
String s2 = "Birthday";
System.out.println(s1.concat(s2));
System.out.println(s1);
```

The method `replace`

- `replace` returns a new `String` object in which every occurrence of the first character argument is replaced with the second character argument. An overloaded version of method `replace` enables you to replace substrings rather than individual characters.

```java
String s1 = "Hello";
System.out.println(s1.replace('l', 'L')); // HeLLo
System.out.println(s1.replace("ll", "LL")); // HeLLo
```

String case conversion methods

- String method toUpperCase returns a new String with uppercase letters where corresponding lowercase letters exist in the original.
- String method toLowerCase returns a new String object with lowercase letters where corresponding uppercase letters exist in the original.

```java
String s1 = "Hello";
System.out.println(s1.toUpperCase()); // HELLO
System.out.println(s1.toLowerCase()); // hello
```

The method trim

- trim returns a new String object that removes all white-space characters at the beginning or end of the String on which trim operates.

```java
String s1 = " spaces ";
System.out.println(s1.trim()); // prints "spaces"
```

The method toCharArray

- toCharArray creates a new character array containing a copy of the characters in the string.

```java
String s1 = "hello";
char[] charArray = s1.toCharArray();
for(char c : charArray) System.out.print(c);
```

Tokenizing `String`s

- When you read a sentence, your mind breaks it into tokens — individual words and punctuation marks that convey meaning to you.
- `String` method `split` breaks a `String` into its component tokens, separated from each other by *delimiters*, typically white-space characters such as space, tab, new line, carriage return (`\r`).

Tokenizing Strings

```
Scanner input = new Scanner(System.in);
System.out.println("
    Enter a sentence and press Enter");
String sentence = input.nextLine();
String[] tokens = sentence.split(" ");
System.out.printf("Number of tokens: %d\n"
    , tokens.length);
for(String token : tokens) System.out.
    println(token);
input.close();
```

```
Enter a sentence and press
Enter
This is a sentence with
seven tokens
Number of tokens: 7
This
is
a
sentence
with
seven
tokens
```

The method `valueOf`

- Every object in Java has a `toString` method that enables a program to obtain the object’s String representation.
- Unfortunately, this technique cannot be used with primitive types because they do not have methods.
- Class `String` provides static methods that take an argument of any type and convert it to a `String` object.

Tokenizing Strings

```java
boolean booleanValue = true;
char charValue = 'Z';
int intValue = 7;
long longValue = 10000000000L;
float floatValue = 2.5f;
double doubleValue = 33.3333; // no f suffix, double is default
char[] charArray = {'a', 'b', 'c', 'd', 'e', 'f'};
System.out.println(String.valueOf(booleanValue));
System.out.println(String.valueOf(charValue));
System.out.println(String.valueOf(intValue));
System.out.println(String.valueOf(longValue));
System.out.println(String.valueOf(floatValue));
System.out.println(String.valueOf(doubleValue));
System.out.println(String.valueOf(charArray));
```

```
true
Z
7
10000000000
2.5
33.3333
abcdef
```

Wrapper classes

- Java has 8 primitive types: `boolean`, `char`, `double`, `float`, `byte`, `short`, `int` and `long`.
- Java also provides 8 type-wrapper classes: `Boolean`, `Character`, `Double`, `Float`, `Byte`, `Short`, `Integer` and `Long` that enable primitive-type values to be treated as objects.
  - Be careful: not `Int` or `Char`.

Character class

- The class Character is the type-wrapper class for the primitive type char.
- Character provides methods (mostly static ones) for convenience in processing individual char values.
  - isDigit(char c)
  - isLetter(char c)
  - isLowerCase(char c)

Character class

```java
Scanner sc = new Scanner(System.in);
System.out.println("Enter a character and press Enter:");
String input = sc.next();
char c = input.charAt(0);

System.out.printf("is digit: %b\n", Character.isDigit(c));
System.out.printf("is identifier start: %b\n", Character.
    isJavaIdentifierStart(c));
System.out.printf("is letter: %b\n", Character.isLetter(c));
System.out.printf("is lower case: %b\n:", Character.isLowerCase(c));
System.out.printf("is upper case: %b\n", Character.isUpperCase(c));
System.out.printf("to upper case: %c\n", Character.toUpperCase(c));
System.out.printf("to lower case: %c\n", Character.toLowerCase(c));

sc.close();
```

Character class

```
Enter a character and press Enter:
A
is digit: false
is identifier start: true
is letter: true
is lower case: false
is upper case: true
to upper case: A
to lower case: a
```

```
Enter a character and press Enter:
8
is digit: true
is identifier start: false
is letter: false
is lower case: false
is upper case: false
to upper case: 8
to lower case: 8
```

Character object

```java
Character c1 = 'A';
Character c2 = new Character('A');

if (c1 == c2)
  System.out.println("cc1 and cc2 are the same");

if (c1.equals(c2))
  System.out.println("cc1 and cc2 are the same");
```