Java Lec 3 String

Monday, August 23, 2021 8:48 PM

Java_Strin...

The `String` Class

Strings

- A string is a sequence of characters
- Strings are objects of the `String` class
- String constants:

```
"Hello, World!"
```

- String variables:

```
String message = "Hello, World!";
```

- String length:

```
int n = message.length();
```

- Empty string:

```
""
```

new

Constructing a String:

String message = "Welcome to Java!”

String is an Object in Java.

String
+String() +String(value: String) +String(value: char[]) +charAt(index: int): char +compareTo(anotherString: String): int +compareToIgnoreCase(anotherString: String): int +concat(anotherString: String): String +endsWithSuffixe(suffix: String): boolean +equals(anotherString: String): boolean +equalsIgnoreCase(anotherString: String): boolean +indexOf(ch: int): int +indexOf(ch: int, fromIndex: int): int +indexOf(str: String): int +indexOf(str: String, fromIndex: int): int +intern(): String +regionMatches(toffset: int, other: String, offset: int, len: int): boolean +length(): int +replace(oldChar: char, newChar: char): String +startsWith(prefix: String): boolean +subString(beginIndex: int): String +subString(beginIndex: int, endIndex: int): String +toCharArray(): char[] +toLowerCase(): String +toString(): String

G8

Constructing Strings

Strings newString = new String(stringLiteral);

String message = new String("Welcome to Java!");

Since strings are used frequently, Java provides a shorthand notation for creating a string:

String message = "Welcome to Java!";

Concatenation

- Use the operator:

```
String name = "Dave";
String message = "Hello, " + name;
   // message is "Hello, Dave"
```

- If one of the arguments of the operator is a string, the other is converted to a string

```
String a = "Agent";
int n = 7;
String bond = a + n;
// bond is "Agent7"
```

Concatenation in Print Statements

- Useful to reduce the number of instructions

```
System.out.print("The total is ");
System.out.println(total);
```

versus

```
System.out.println("The total is " + total);
```

String

Converting between Strings and Numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(str);
```

- Convert to string:

```
String str = "" + n;
str = Integer.toString(n);
```

main (String[])

(cast) int, double

Class

→ MyClass(string)

Integer int

Double double

```
Integer.substring(...);
```

Rectangle

System.out.println(rect)

output

1st argument

args[0]

cast

primitive

int i = (int) args[0];

args[1]

String

public static void main(String args[])

5.0

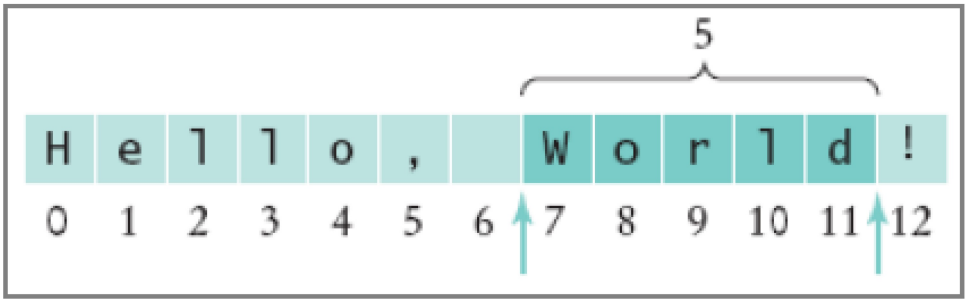
Substrings

```
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5);
```

- Supply start and "past the end" position
- First position is at 0, last position is (5-1) -> Hello

H	e	l	l	o	,		W	o	r	l	d	!
0	1	2	3	4	5	6	7	8	9	10	11	12

Substrings



Self Check

- Assuming the `String` variable `s` holds the value `"Agent"`, what is the effect of the assignment `s = s + s.length()`?
- Assuming the `String` variable `river` holds the value `"Mississippi"`, what is the value of `river.substring(1, 2)`? Of `river.substring(2, river.length() - 3)`?

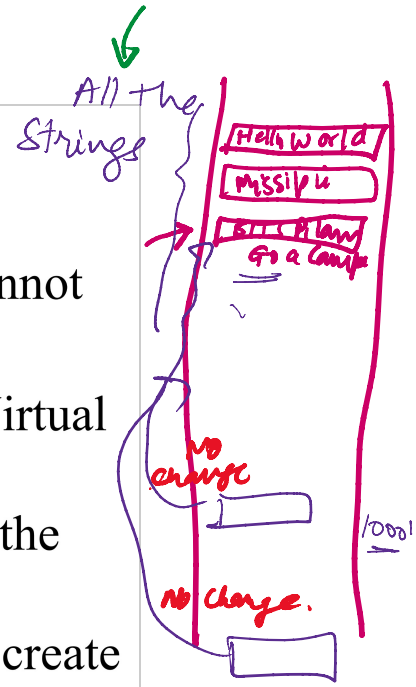
Answers

- `s` is set to the string `Agent5`
- The strings `"i"` and `"ssissi"`

Strings Are Immutable

A String object is immutable, whose contents cannot be changed.

To improve efficiency and save memory, Java Virtual Machine stores two String objects into the same object, if the two String objects are created with the same string literal using the shorthand notation. Therefore, the shorthand notation is preferred to create strings.



Strings Are Immutable, cont.

A string that is created using the shorthand notation is known as a *canonical string*.

You can use the String's intern method to return a canonical string, which is the same string that is created using the shorthand notation.

String s = "Hello";

s1 = new String("Hello");

↳ creates a new String.

→ search in the list

Hello.

Hello

Finding String Length

Finding string length using the `length()` method:

```
message = "Welcome";
message.length() (returns 7)
```

Retrieving Individual Characters in a String

- Do not use `message[0]`
- Use `message.charAt(index)`
- Index starts from `0`

Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
message	W	e	l	c	o	m	e		t	o		J	a	v	a

message.charAt(0) message.length() is 15 message.charAt(14)

String Concatenation

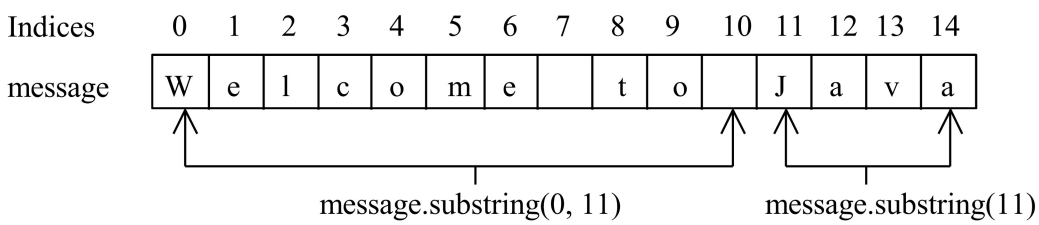
```
String s3 = s1.concat(s2);
```

String s3 = s1 + s2;

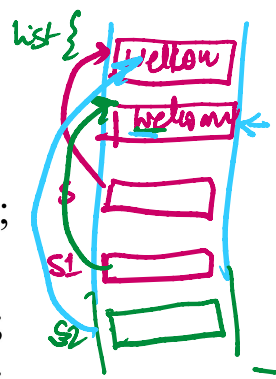
Extracting Substrings

`String` is an immutable class; its values cannot be changed individually.

```
String s1 = "Welcome to Java";
String s2 = s1.substring(0, 11) + "HTML";
```



String ==




```
String s = "Welcome to Java!";
String s1 = new String("Welcome to Java!");
String s2 = s1.intern();
System.out.println("s1 == s is " + (s1 == s));
System.out.println("s2 == s is " + (s2 == s));
System.out.println("s1 == s2 is " + (s1 == s2));
```

displays

s1 == s is false

s2 == s is true

s1 == s2 false

① shorthand

welcome to Java

s1.intern()

② new

i == j

Primitives

Objects.

shallow comparison

→ addresses

s == s1;

G9

String Comparisons: equals

```
String s1 = "Welcome";
String s2 = "welcome";

if (s1.equals(s2)){
  // s1 and s2 have the same contents
}
 if (s1 == s2) {
  // s1 and s2 have the same reference
}
```

compareTo(Object object)

```
String s1 = "Welcome";
String s2 = "welcome";
  if (s1.compareTo(s2) > 0) {
 // s1 is greater than s2
}
else if (s1.compareTo(s2) == 0) {
  // s1 and s2 have the same reference
}
else
   // s1 is less than s2
```

String Conversions

The contents of a string cannot be changed once the string is created. But you can convert a string to a new string using the following methods:

length

- toLowerCase
- toUpperCase
- trim
- replace(oldChar, newChar)

HW

s1 = s1.toLowerCase

Finding a Character or a Substring in a String

"Welcome to Java!".indexOf('W')) returns 0.

"Welcome to Java!".indexOf('x')) returns -1.

"Welcome to Java!".indexOf('o', 5)) returns 9.

"Welcome to Java!".indexOf("come")) returns 3.

"Welcome to Java!".indexOf("Java", 5)) returns 11.

"Welcome to Java!".indexOf("java", 5)) returns -1.

Convert Character and Numbers to Strings

valueOf methods for converting a character, an array of characters, and numeric values to strings. These methods have the same name valueOf with different argument types char, char[], double, long, int, and float.

Convert a double value to a string
String.valueOf(5.44).

int [ ]

Rectangle

x, y, w, h
↓
x1, y1, w, h
toString()

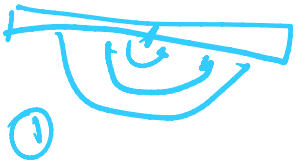
" ␣, ␣, ␣, ␣ "

Finding Palindromes

String

Checking whether a string is a palindrome: a string that reads the same forward and backward.

how many ways to write this code



1

The `Character` Class

Char

Character
+Character(value: char) +charValue(): char +compareTo(anotherCharacter: Character): int +equals(anotherCharacter: Character): boolean +<u>isDigit(ch: char): boolean</u> +<u>isLetter(ch: char): boolean</u> +<u>isLetterOrDigit(ch: char): boolean</u> +<u>isLowerCase(ch: char): boolean</u> +<u>isUpperCase(ch: char): boolean</u> +<u>toLowerCase(ch: char): char</u> +<u>toUpperCase(ch: char): char</u>

constructor

method

Examples

b

compareToIgnoreCase

charObject.compareTo(new Character('a')) returns 1
charObject.compareTo(new Character('b')) returns 0
charObject.compareTo(new Character('c')) returns -1
charObject.compareTo(new Character('d') returns –2
charObject.equals(new Character('b')) returns true
charObject.equals(new Character('d')) returns false

shallow ==

Deeper

Counting Each Letter in a String

HW

to Uppercase

Write a program that counts the number of occurrence of each letter in a string. Assume the letters are not case-sensitive.

① Over allocate

② Move if size > allocated size

space req. fixed

Array String

Immutable

① ② ③ ④

The `StringBuffer` Class

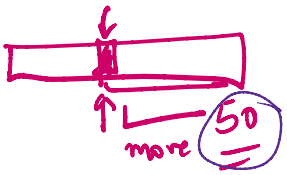
The StringBuffer class is an alternative to the String class. In general, a string buffer can be used wherever a string is used.

StringBuffer is more flexible than String. You can add, insert, or append new contents into a string buffer. However, the value of a string is fixed once the string is created.

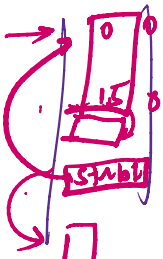
StringBuffer
+append(data: char[]): StringBuffer +append(data: char[], offset: int, len: int): StringBuffer +append(v: *aPrimitiveType*): StringBuffer +append(str: String): StringBuffer +capacity(): int +charAt(index: int): char +delete(startIndex: int, endIndex: int): StringBuffer +deleteCharAt(int index): StringBuffer +insert(index: int, data: char[], offset: int, len: int): StringBuffer +insert(offset: int, data: char[]): StringBuffer +insert(offset: int, b: *aPrimitiveType*): StringBuffer +insert(offset: int, str: String): StringBuffer +length(): int +replace(int startIndex, int endIndex, String str): StringBuffer +reverse(): StringBuffer +setCharAt(index: int, ch: char): void +setLength(newLength: int): void +substring(start: int): StringBuffer +substring(start: int, end: int): StringBuffer

all String

→ O(n)



String methods only



strb1 = new StringBuffer()

each input → move → delay

StringBuffer Constructors

- public StringBuffer()
  No characters, initial capacity 16 characters.
- public StringBuffer(int length) 50
  No characters, initial capacity specified by the length argument.
- public StringBuffer(String str)
  Represents the same sequence of characters as the string argument. Initial capacity 16 plus the length of the string argument.

Appending New Contents into a String Buffer

```
StringBuffer strBuf = new StringBuffer();
strBuf.append("Welcome");
strBuf.append(' ');
strBuf.append("to");
strBuf.append(' ');
strBuf.append("Java");
```

move

The StringTokenizer Class Constructors

- StringTokenizer(String s, String delim, boolean returnTokens)
- StringTokenizer(String s, String delim)
- StringTokenizer(String s)

space

① Obtain → Retrieve ②

Welcome to Java

The `StringTokenizer` Class Methods

- boolean hasMoreTokens()
- String nextToken()
- String nextToken(String delim)

StringTokenizer
+countTokens(): int +hasMoreTokens():boolean +nextToken(): String +nextToken(delim: String): String

Environment

String

delim

space

s1
s2
s3

,

GO → String Builder