CURS JAVA SE

@IBM - Eugen Barbu

Working with Strings (1)

Concatenating strings

The + operator concatenates Strings:

- String a = "This" + " is a " + "String";

Primitive types used in a call to println are automatically converted to Strings

- System.out.println("answer = " + 1 + 2 + 3);
- System.out.println("answer = " + (1+2+3));
- Do you get the same output from the above examples?

Other examples:

"hello" + 42 //"hello42"	1 + 2 + "abc" //"3abc"
1 + "abc" + 2 //"1abc2"	"abc" + 9 * 3 //"abc27"
"abc" + 1 + 2 //"abc12"	"1" + 1 //"11"
"abc" + (1+2) //"abc3"	4 - 1 + "abc" //"3abc"
1 + “2” + "abc" //"12abc"	"abc" + 4 - 1 //error, why?

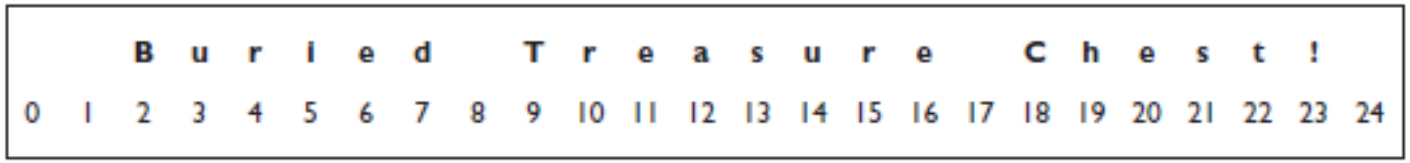
Working with Strings (2)

Commonly used methods of the String class:

+ charAt(int) : char - returns a char value from a int index value in relationship to the referenced string
+ endsWith(String) : boolean
+ indexOf(int) : int
+ indexOf(int, int) : int
+ indexOf(String) : int - returns the position of a String
+ indexOf(String, int) : int
+ length() : int - returns the length of a string
+ replace(char, char) : String - replaces a char with another
+ replace(CharSequence, CharSequence) : String
+ startsWith(String, int) : boolean
+ startsWith(String) : boolean - checks if String starts with a specific pattern
+ substring(int) : String
+ substring(int, int) : String - from position x to position y
+ trim() : String - removes tailing whitespaces

Exercises (1)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest! ";**



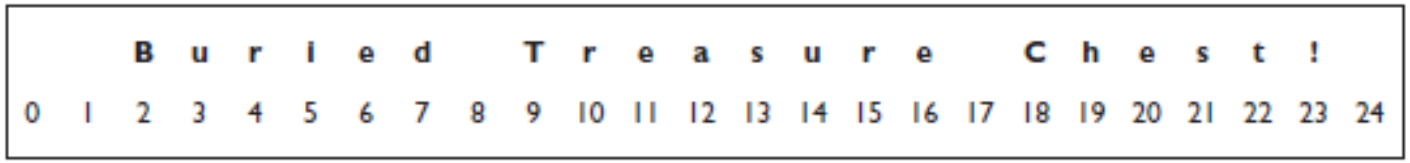
+ charAt(int) : char

The String class's charAt method returns a primitive char value from a specified int index value in relationship to the referenced string object.

```
/* Returns the 'blank space' character from location 0 */
char c1 = pirateMessage.charAt(0);
/* Returns the character 'B' from location 2 */
char c2 = pirateMessage.charAt(2);
/* Returns the character '!' from location 23 */
char c3 = pirateMessage.charAt(23);
/* Returns the 'blank space' character from location 24 */
char c4 = pirateMessage.charAt(24);
```

Exercises (2)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest! ";**



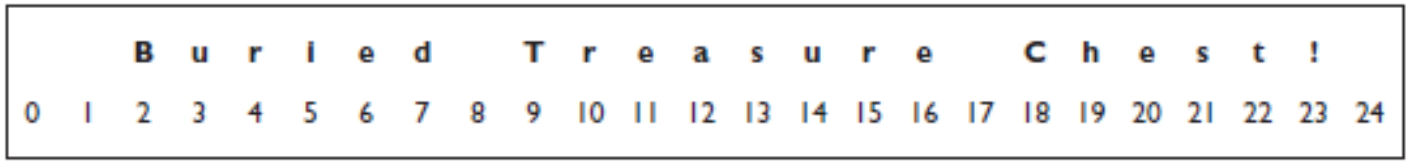
+ indexOf(int) : int
+ indexOf(int, int) : int
+ indexOf(String) : int
+ indexOf(String, int) : int

The String class's indexOf methods return primitive int values representing the index of a character or string in relationship to the referenced string object.

```
/* Returns the integer 3 as it is the first 'u' in the string. */
int i1 = pirateMessage.indexOf('u'); // 3
/* Returns the integer 14 as it is the first 'u' in the string past location 9. */
int i2 = pirateMessage.indexOf('u', 9); // 14
/* Returns the integer 13 as it starts at location 13 in the string. */
int i3 = pirateMessage.indexOf("sure"); // 13
/* Returns the integer -1 as there is no Treasure string on or past location 10 */
int i4 = pirateMessage.indexOf("Treasure", 10); // -1!
/* Returns the integer -1 as there is no character u on or past location 100 */
int i5 = pirateMessage.indexOf("u", 100); // -1!
```

Exercises (3)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest! ";**



+ length() : int

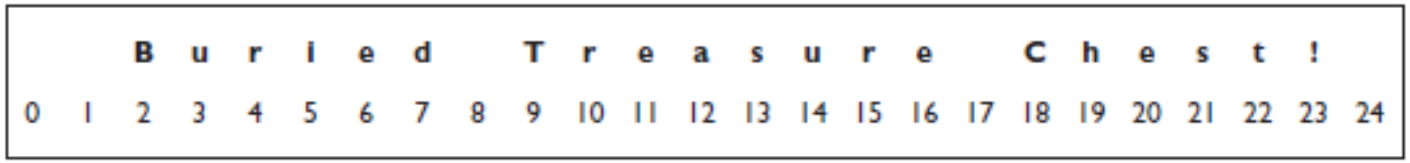
The String class's length method returns a primitive int value representing the length of the referenced string object.

/* Returns the string's length of 25 */

int i = pirateMessage.length(); // 25

Exercises (4)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest!** ";



+ startsWith(String, int) : boolean
+ startsWith(String) : boolean
The String class's startsWith method returns a primitive boolean value representing the results of a test to see if the supplied prefix starts the referenced String object.

```
/* Returns true as the referenced string starts with the compared string. */
boolean b1 = pirateMessage.startsWith(" Buried Treasure"); // true
/* Returns false as the referenced string does not start with the compared string. */
boolean b2 = pirateMessage.startsWith(" Discovered"); // false
/* Returns false as the referenced string does not start with the compared string at location 8. */
boolean b3 = pirateMessage.startsWith("Treasure", 8); // false
/* Returns true as the referenced string does start with the compared string at location 9. */
boolean b4 = pirateMessage.startsWith("Treasure", 9); // true
```

Exercises (5)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest! ";**

		B	u	r	i	e	d		T	r	e	a	s	u	r	e		C	h	e	s	t	!	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

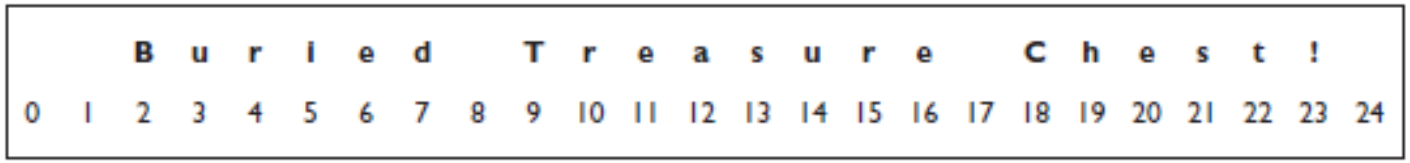
+ endsWith(String) : boolean

The String class's endsWith method returns a primitive boolean value representing the results of a test to see if the supplied suffix ends the referenced string object.

```
/* Returns true as the referenced string ends with the compared string. */
boolean b1 = pirateMessage.endsWith("Treasure Chest! "); // true
/* Returns false as the referenced string does not end with the compared string. */
boolean b2 = pirateMessage.endsWith("Treasure Chest "); // false
```

Exercises (6)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest! ";**

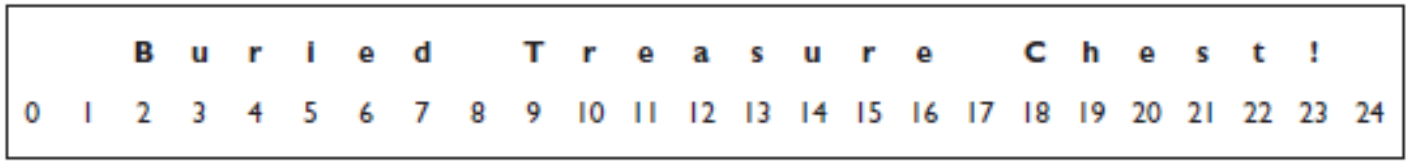


+ substring(int) : String
+ substring(int, int) : String
The String class's substring methods return new strings that are substrings of the referenced string object.

```
/* Returns the entire string starting at index 9. */
String s1 = pirateMessage.substring(9); // Treasure Chest!
/* Returns the string at index 9. */
String s2 = pirateMessage.substring(9, 10); // T
/* Returns the string at index 9 and ending at index 23. */
String s3 = pirateMessage.substring(9, 23); // Treasure Chest
/* Produces runtime error. */
String s4 = pirateMessage.substring(9, 8); // String index out of range: -1
/* Returns a blank */
String s5 = pirateMessage.substring(9, 9); // Blank
```

Exercises (7)

Let's use following Text: String pirateMessage = " **Buried Treasure Chest!** ";



+ trim() : String

The String class's trim method returns the entire string minus leading and trailing whitespace characters in relationship to the referenced string object.

```
/* "Buried Treasure Chest!" with no leading or trailing white spaces */
String s = pirateMessage.trim();
```

Exercises (8)

Exercitiu:

Java allows for methods to be chained together. Consider the following message:

```
String msg = " The night is dark and full of terrors! ";
```

We wish to change the message to read: “The cheeseburger is juicy and stuffed with bacon.”

Three changes need to be made to adjust the string as desired:
1. Trim the leading and trailing whitespace.
2. Replace the substring “night” with “cheeseburger”.
3. Replace the substring “full of terrors” with “stuffed with bacon”.
4. Add a period at the end of the sentence.

Exercitiu:

```
String tenCharString = "AAAAAAAAAA";
System.out.println(tenCharString.replace("AAA", "LLL"));
```

What is printed to the standard out?

StringBuffer

- The StringBuffer class provides a more efficient mechanism for building strings
- String concatenation can get very expensive
- String concatenation is converted by most compilers —including Eclipse —into a StringBuffer implementation
- If building a simple String, just concatenate; if building a String through a loop, use a StringBuffer

```
StringBuffer buffer = new StringBuffer(15);
 buffer.append("This is ");
 buffer.append("String");
 buffer.insert(7, " a");
 buffer.append('.');
 System.out.println(buffer.length());
 String output = buffer.toString();
 System.out.println(output);
```