COMP231

Advanced Programming

Chapter 4+10 Strings

Compiled By: Dr. Majdi Mafarja
Fall Semester 2017/2018

The `String` Class

- Constructing a String:

```
String message = "Welcome to Java";
String message = new String("Welcome to Java");
String s = new String();
```

- Obtaining String length and Retrieving Individual Characters in a string
- String Concatenation (concat)
- Substrings (substring(index), substring(start, end))
- Comparisons (equals, compareTo)
- String Conversions
- Finding a Character or a Substring in a String
- Conversions between Strings and Arrays
- Converting Characters and Numeric Values to Strings

Constructing Strings

String newString = new String(stringLiteral);

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

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

String message = "Welcome to Java";

Strings Are Immutable

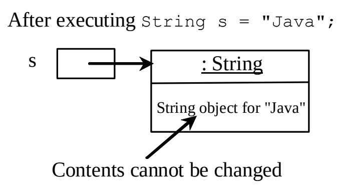
A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

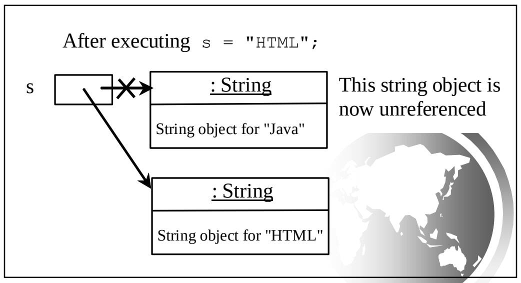
```
String s = "Java";
s = "HTML";
```

animation

Trace Code

```
String s = "Java";
s = "HTML";
```

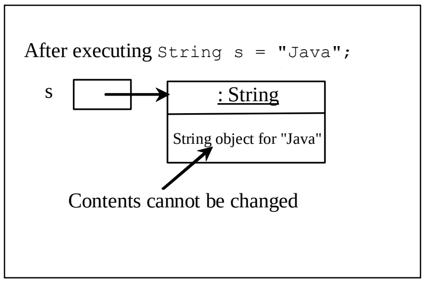

After executing String s = "Java";
s
: String
String object for "Java"
Contents cannot be changed

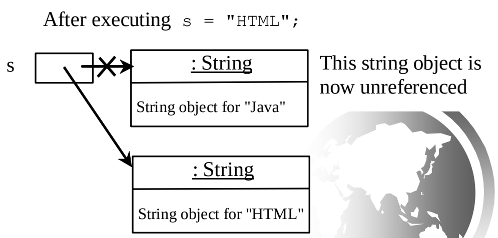

After executing s = "HTML";
s
: String
String object for "Java"
This string object is now unreferenced
: String
String object for "HTML"

animation

Trace Code

```
String s = "Java";
s = "HTML";
```


After executing String s = "Java";
s
: String
String object for "Java"
Contents cannot be changed


After executing s = "HTML";
s
: String
String object for "Java"
This string object is now unreferenced
: String
String object for "HTML"

Interned Strings

Since strings are immutable and are frequently used, to improve efficiency and save memory, the JVM uses a unique instance for string literals with the same character sequence. Such an instance is called *interned*. For example, the following statements:

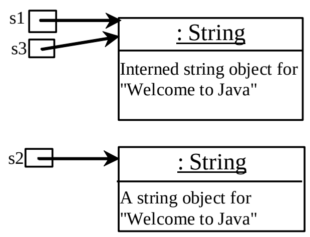
Examples

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

System.out.println("s1 == s2 is " + (s1 == s2));
System.out.println("s1 == s3 is " + (s1 == s3));
```


s1
s3
: String
Interned string object for "Welcome to Java"
s2
: String
A string object for "Welcome to Java"

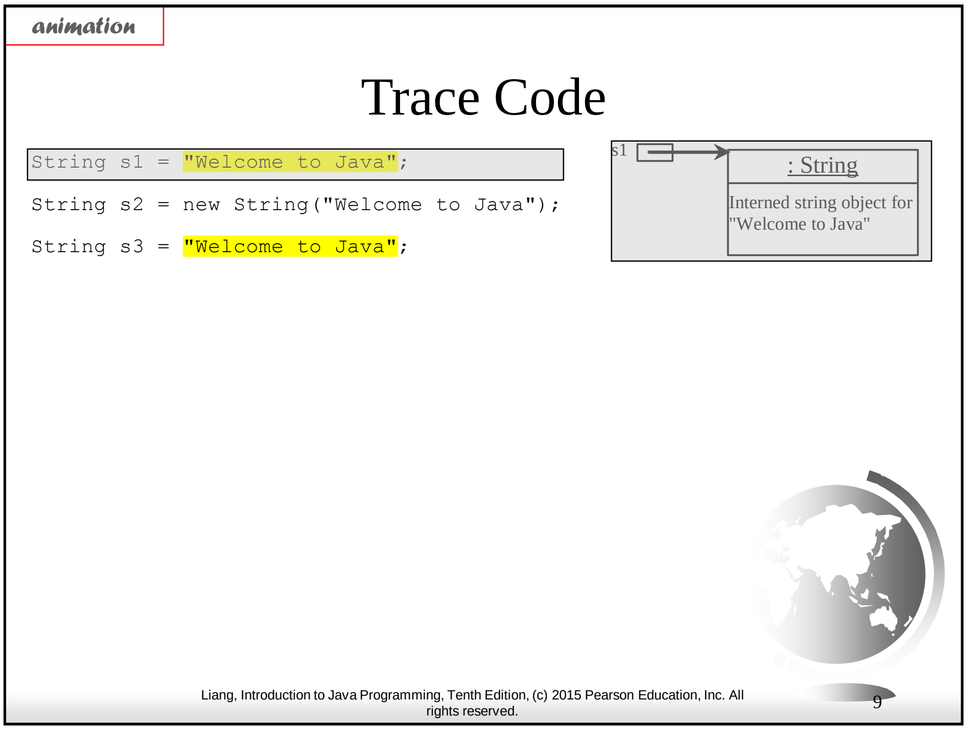
display

s1 == s is false

s1 == s3 is true

A new object is created if you use the new operator.

If you use the string initializer, no new object is created if the interned object is already created.


animation

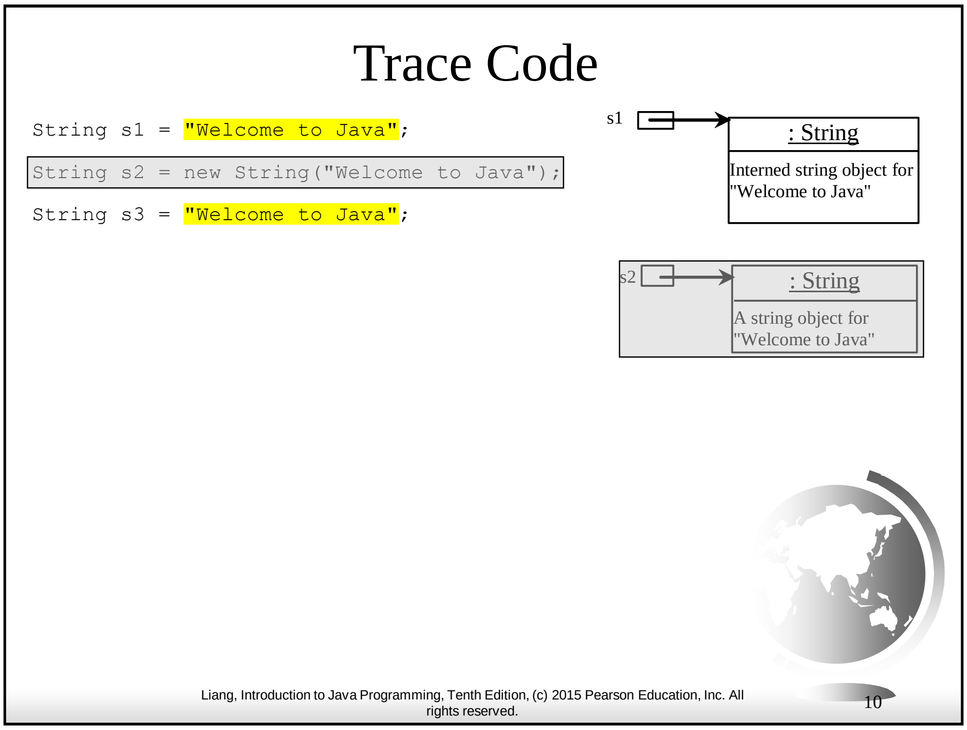
Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```

s1
: String
Interned string object for "Welcome to Java"

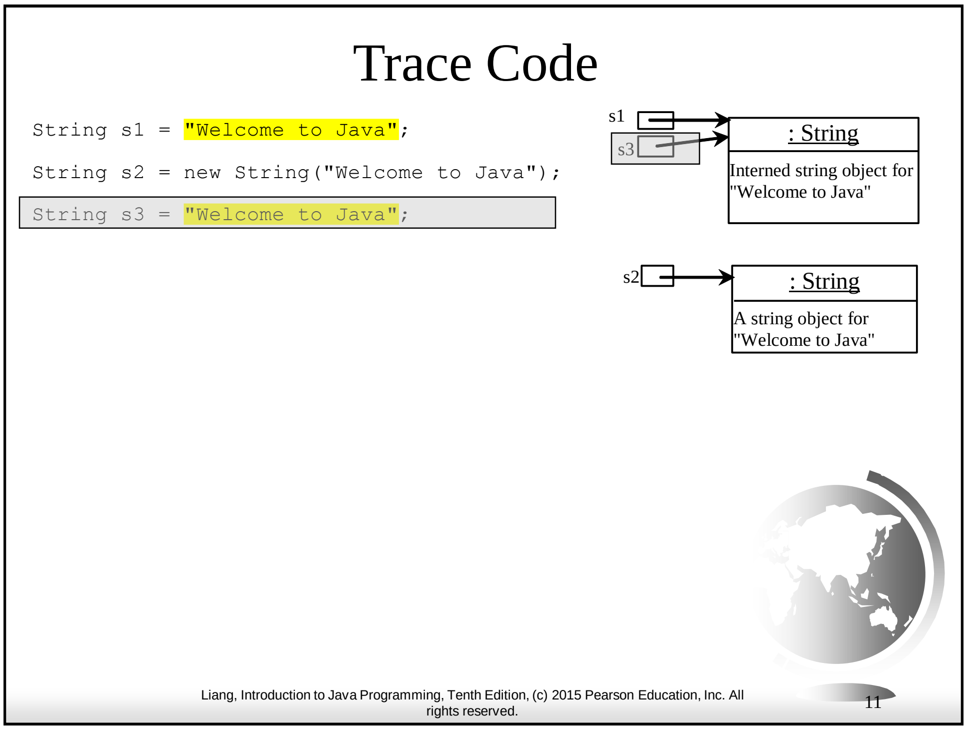

Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```

s1
: String
Interned string object for "Welcome to Java"
s2
: String
A string object for "Welcome to Java"

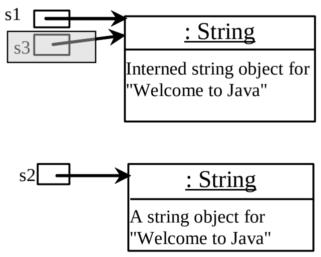


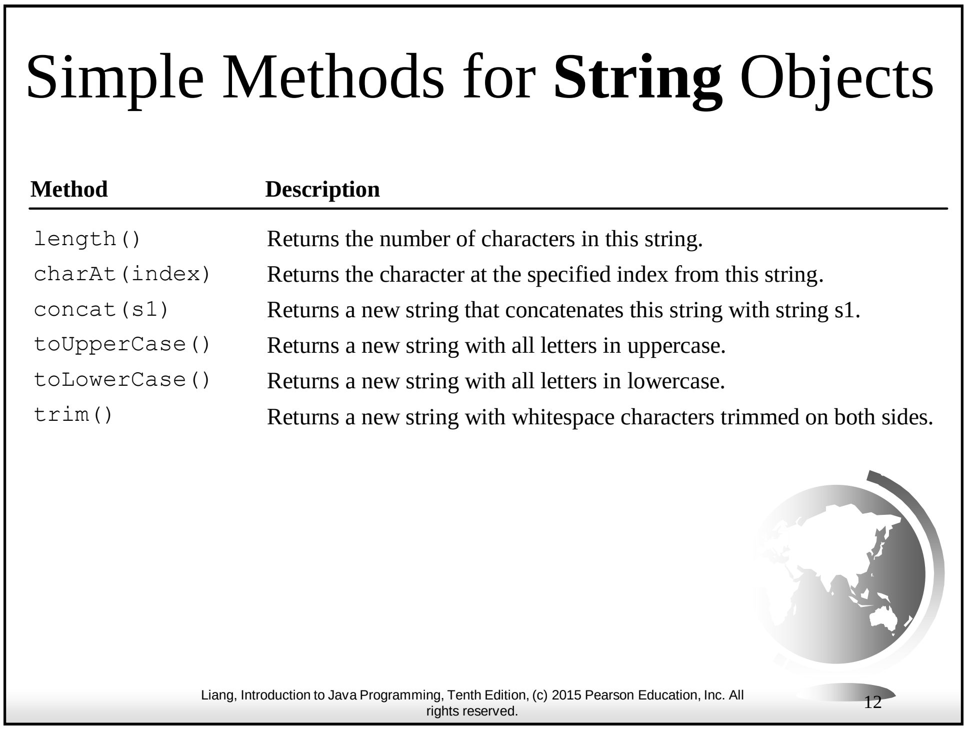
Trace Code

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";
```


s1
s3
: String
Interned string object for "Welcome to Java"
s2
: String
A string object for "Welcome to Java"



Simple Methods for **String** Objects

Method	**Description**
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string s1.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

Simple Methods for **String** Objects

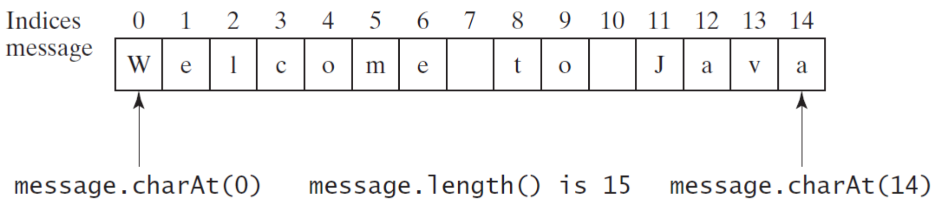
Strings are objects in Java. The methods in the preceding table can only be invoked from a ***specific string instance***. For this reason, these methods are called *instance methods*. A non-instance method is called a *static method*. A static method can be invoked without using an object. All the methods defined in the **Math** class are static methods. They are not tied to a specific object instance. The syntax to invoke an instance method is

referenceVariable.methodName(arguments).

Getting String Length

```
String message = "Welcome to Java";
System.out.println("The length of " + message + " is "
  + message.length());
```

Getting Characters from a String



String message = **"Welcome to Java"**;

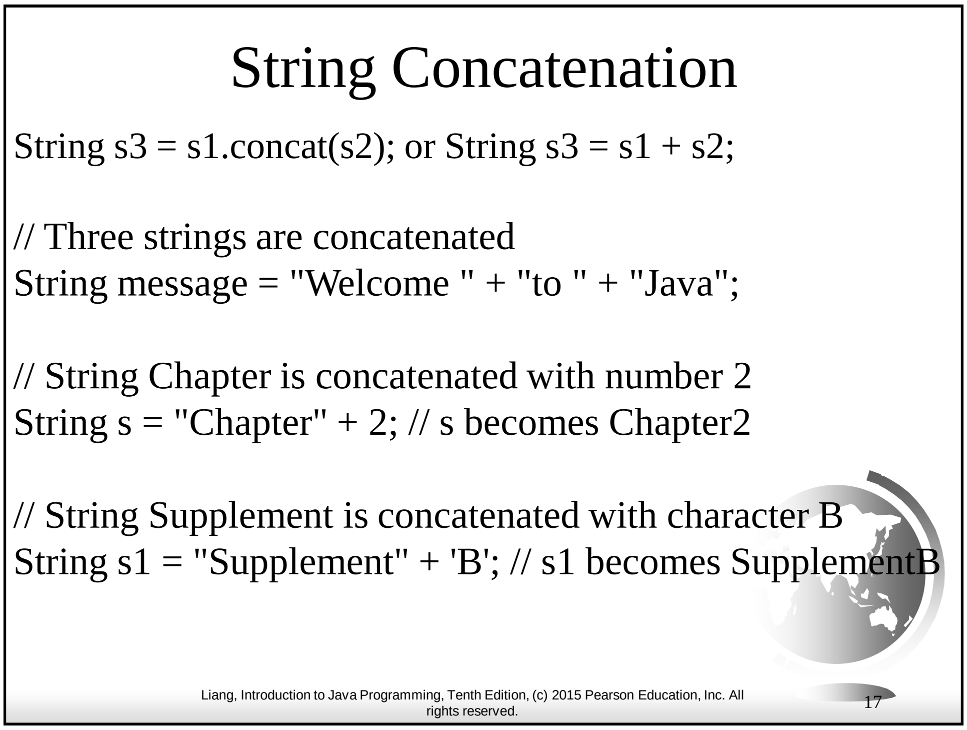
System.out.println(**"The first character in message is "**
  + message.charAt(0));

Converting Strings

"Welcome".toLowerCase() returns a new string, welcome.

"Welcome".toUpperCase() returns a new string, WELCOME.

" Welcome ".trim() returns a new string, Welcome.



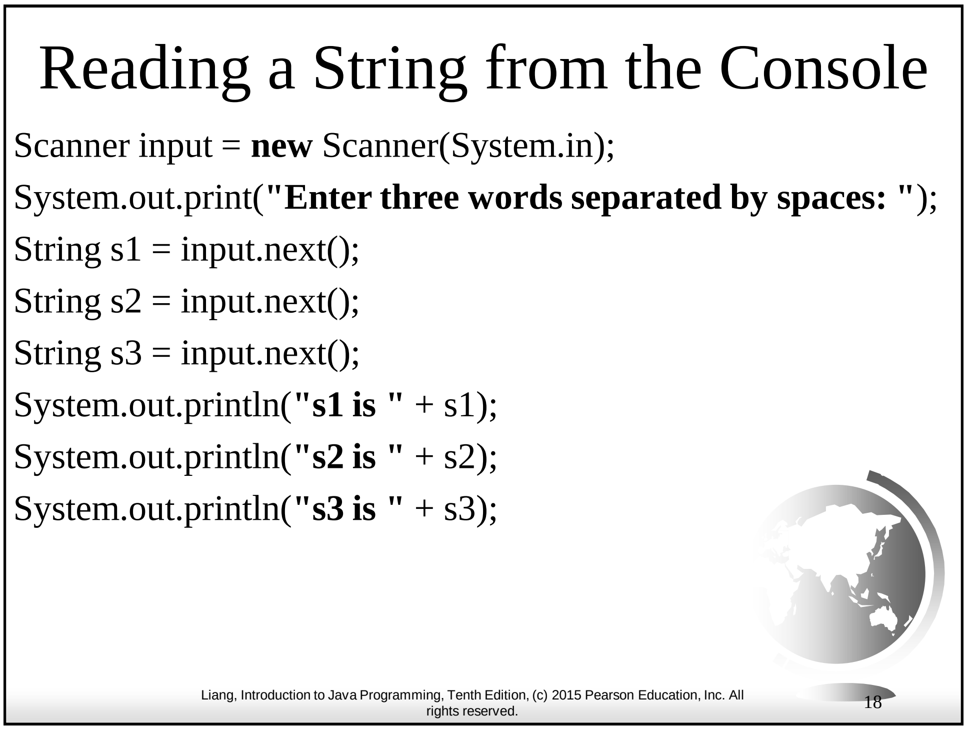
String Concatenation

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

// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```



Reading a String from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Reading a Character from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");
String s = input.nextLine();
char ch = s.charAt(0);
System.out.println("The character entered is " + ch);
```

Comparing Strings

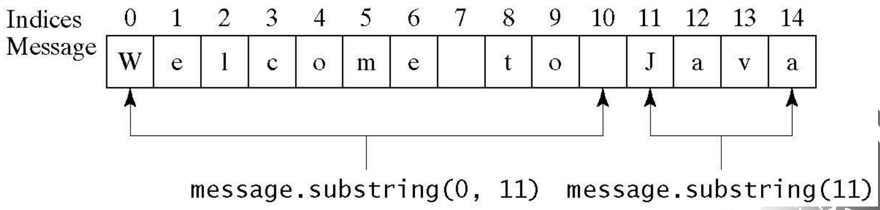
Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than `0`, equal to `0`, or less than `0` to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

OrderTwoCities

Run

Obtaining Substrings

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex – 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.

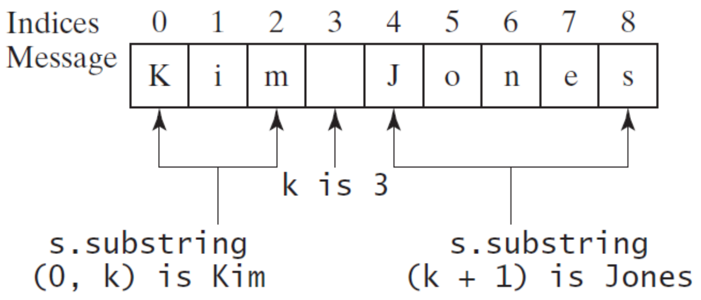


Finding a Character or a Substring in a String

Method	Description
`indexOf(ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

Finding a Character or a Substring in a String

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



Conversion between Strings and Numbers

```
int intValue = Integer.parseInt(intString);
double doubleValue = Double.parseDouble(doubleString);

String s = number + "";
```

Formatting Output

Use the printf statement.

System.out.printf(format, items);

Where format is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier	Output	Example
`%b`	`a boolean value`	`true or false`
`%c`	`a character`	`'a'`
`%d`	`a decimal integer`	`200`
`%f`	`a floating-point number`	`45.460000`
`%e`	`a number in standard scientific notation`	`4.556000e+01`
`%s`	`a string`	`"Java is cool"`

items

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

```
display           count is 5 and amount is 45.560000
```

String Format

- `System.out.printf ("%s = %d", “Ahmad", 19);`
- `String output = String.format("%s = %d", “Ahmad", 19);`
- `String.format("|%20d|", 93);`

```
// prints: |                  93|
```

- `String.format("|%-20d|", 93);`

```
// prints: |93                  |
```

FormatDemo

The example gives a program that uses **printf** to display a table.

FormatDemo Run

Replacing and Splitting Strings

java.lang.String	
+replace(oldChar: char, newChar: char): String	Returns a new string that replaces all matching character in this string with the new character.
+replaceFirst(oldString: String, newString: String): String	Returns a new string that replaces the first matching substring in this string with the new substring.
+replaceAll(oldString: String, newString: String): String	Returns a new string that replace all matching substrings in this string with the new substring.
+split(delimiter: String): String[]	Returns an array of strings consisting of the substrings split by the delimiter.

Examples

"Welcome".replace('e', 'A') returns a new string, WAlcomA.

"Welcome".replaceFirst("e", "AB") returns a new string, WABlcome.

"Welcome".replace("e", "AB") returns a new string, WABlcomAB.

"Welcome".replace("el", "AB") returns a new string, WABcome.

Splitting a String

```
String[] tokens = "Java#HTML#Perl".split("#", 0);
for (int i = 0; i < tokens.length; i++)
  System.out.print(tokens[i] + " ");
```

displays

Java HTML Perl

Matching, Replacing and Splitting by Patterns

You can match, replace, or split a string by specifying a pattern. This is an extremely useful and powerful feature, commonly known as *regular expression*. Regular expression is complex to beginning students. For this reason, two simple patterns are used in this section. Please refer to Supplement III.F, “Regular Expressions,” for further studies.

"Java".matches("Java");

"Java".equals("Java");

"Java is fun".matches("Java.*");

"Java is cool".matches("Java.*");

Matching, Replacing and Splitting by Patterns

The replaceAll, replaceFirst, and split methods can be used with a regular expression. For example, the following statement returns a new string that replaces $, +, or # in "a+b$#c" by the string NNN.

```
String s = "a+b$#c".replaceAll("[$+#]", "NNN");
System.out.println(s);
```

Here the regular expression [$+#] specifies a pattern that matches $, +, or #. So, the output is aNNNbNNNNNNc.

Matching, Replacing and Splitting by Patterns

The following statement splits the string into an array of strings delimited by some punctuation marks.

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]");

for (int i = 0; i < tokens.length; i++)
  System.out.println(tokens[i]);
```

Convert Character and Numbers to Strings

The String class provides several static 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. For example, to convert a double value to a string, use String.valueOf(5.44). The return value is string consists of characters ‘5’, ‘.’, ‘4’, and ‘4’.

`StringBuilder` and `StringBuffer`

The `StringBuilder/StringBuffer` class is an alternative to the `String` class. In general, a StringBuilder/StringBuffer can be used wherever a string is used. StringBuilder/StringBuffer is more flexible than String. You can add, insert, or append new contents into a string buffer, whereas the value of a String object is fixed once the string is created.

`StringBuilder` Constructors

java.lang.StringBuilder	
+StringBuilder()	Constructs an empty string builder with capacity 16.
+StringBuilder(capacity: int)	Constructs a string builder with the specified capacity.
+StringBuilder(s: String)	Constructs a string builder with the specified string.

Modifying Strings in the Builder

java.lang.StringBuilder	
+append(data: char[]): StringBuilder	Appends a char array into this string builder.
+append(data: char[], offset: int, len: int): StringBuilder	Appends a subarray in data into this string builder.
+append(v: *aPrimitiveType*): StringBuilder	Appends a primitive type value as a string to this builder.
+append(s: String): StringBuilder	Appends a string to this string builder.
+delete(startIndex: int, endIndex: int): StringBuilder	Deletes characters from startIndex to endIndex.
+deleteCharAt(index: int): StringBuilder	Deletes a character at the specified index.
+insert(index: int, data: char[], offset: int, len: int): StringBuilder	Inserts a subarray of the data in the array to the builder at the specified index.
+insert(offset: int, data: char[]): StringBuilder	Inserts data into this builder at the position offset.
+insert(offset: int, b: *aPrimitiveType*): StringBuilder	Inserts a value converted to a string into this builder.
+insert(offset: int, s: String): StringBuilder	Inserts a string into this builder at the position offset.
+replace(startIndex: int, endIndex: int, s: String): StringBuilder	Replaces the characters in this builder from startIndex to endIndex with the specified string.
+reverse(): StringBuilder	Reverses the characters in the builder.
+setCharAt(index: int, ch: char): void	Sets a new character at the specified index in this builder.

Examples

stringBuilder.append("Java");

stringBuilder.insert(11, "HTML and ");

stringBuilder.delete(8, 11) changes the builder to Welcome Java.

stringBuilder.deleteCharAt(8) changes the builder to Welcome o Java.

stringBuilder.reverse() changes the builder to avaJ ot emocleW.

stringBuilder.replace(11, 15, "HTML")
  changes the builder to Welcome to HTML.

stringBuilder.setCharAt(0, 'w') sets the builder to welcome to Java.

The toString, capacity, length, setLength, and charAt Methods

java.lang.StringBuilder	
+toString(): String	Returns a string object from the string builder.
+capacity(): int	Returns the capacity of this string builder.
+charAt(index: int): char	Returns the character at the specified index.
+length(): int	Returns the number of characters in this builder.
+setLength(newLength: int): void	Sets a new length in this builder.
+substring(startIndex: int): String	Returns a substring starting at startIndex.
+substring(startIndex: int, endIndex: int): String	Returns a substring from startIndex to endIndex-1.
+trimToSize(): void	Reduces the storage size used for the string builder.

Problem: Checking Palindromes Ignoring Non-alphanumeric Characters

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

PalindromeIgnoreNonAlphanumeric

Run

Appendix H

Regular Expressions

A *regular expression* (abbreviated *regex*) is a string that describes a pattern for matching a set of strings. Regular expression is a powerful tool for string manipulations. You can use regular expressions for matching, replacing, and splitting strings.

Appendix H

Matching Strings

"Java".matches("Java");

"Java".equals("Java");

"Java is fun".matches("Java.*")

"Java is cool".matches("Java.*")

"Java is powerful".matches("Java.*")

Appendix H

Regular Expression Syntax

Regular Expression	*Matches*	*Example*
x	a specified character x	Java matches Java
.	any single character	Java matches J..a
(ab|cd)	ab or cd	ten matches t(en|im)
[abc]	a, b, or c	Java matches Ja[uvwx]a
[^abc]	any character except a, b, or c	Java matches Ja[^ars]a
[a-z]	a through z	Java matches [A-M]av[a-d]
[^a-z]	any character except a through z	Java matches Jav[^b-d]
[a-e[m-p]]	a through e or m through p	Java matches [A-G[I-M]]av[a-d]
[a-e&&[c-p]]	intersection of a-e with c-p	Java matches [A-P&&[I-M]]av[a-d]
\d	a digit, same as [0-9]	Java2 matches "Java[\\d]"
\D	a non-digit	$Java matches "[\\D][\\D]ava"
\w	a word character	Java1 matches "[\\w]ava[\\w]"
\W	a non-word character	$Java matches "[\\W][\\w]ava"
\s	a whitespace character	"Java 2" matches "Java\\s2"
\S	a non-whitespace char	Java matches "[\\S]ava"
*p**	zero or more occurrences of pattern *p*	aaaabb matches "a*bb" ababab matches "(ab)*"
p+	one or more occurrences of pattern *p*	a matches "a+b*" able matches "(ab)+.*"
p?	zero or one occurrence of pattern *p*	Java matches "J?Java" Java matches "J?ava"
p{n}	exactly n occurrences of pattern *p*	Java matches "Ja{1}.*" Java does not match ".{2}"
p{n,}	at least n occurrences of pattern *p*	aaaa matches "a{1,}" a does not match "a{2,}"
p{n,m}	between n and m occurrences (inclusive)	aaaa matches "a{1,9}" abb does not match "a{2,9}bb"

Appendix H

Replacing and Splitting Strings

java.lang.String	
+matches(regex: String): boolean	Returns true if this string matches the pattern.
+replaceAll(regex: String, replacement: String): String	Returns a new string that replaces all matching substrings with the replacement.
+replaceFirst(regex: String, replacement: String): String	Returns a new string that replaces the first matching substring with the replacement.
+split(regex: String): String[]	Returns an array of strings consisting of the substrings split by the matches.

Appendix H

Examples

```
String s = "Java Java Java".replaceAll("v\\w", "wi") ;

String s = "Java Java Java".replaceFirst("v\\w", "wi") ;


String[] s = "Java1HTML2Perl".split("\\d");
```