Chapter 9 Strings

Motivations

Often you encounter the problems that involve string processing and file input and output. Suppose you need to write a program to replace all occurrences of a word with a new word in a file. How do you solve this problem? This chapter introduces strings and text files, which will enable you to solve this problem.

Objectives

- To use the **String** class to process fixed strings (§9.2).
- To construct strings (§9.2.1).
- To understand that strings are immutable and to create an interned string (§9.2.2).
- To compare strings (§9.2.3).
- To get string length and characters, and combine strings (§9.2.4).
- To obtain substrings (§9.2.5).
- To convert, replace, and split strings (§9.2.6).
- To match, replace, and split strings by patterns (§9.2.7).
- To search for a character or substring in a string (§9.2.8).
- To convert between a string and an array (§9.2.9).
- To convert characters and numbers into a string (§9.2.10).
- To obtain a formatted string (§9.2.11).
- To check whether a string is a palindrome (§9.3).
- To convert hexadecimal numbers to decimal numbers (§9.4).
- To use the **Character** class to process a single character (§9.5).
- To use the **StringBuilder** and **StringBuffer** classes to process flexible strings (§9.6).
- To distinguish among the **String**, **StringBuilder**, and **StringBuffer** classes (§9.2–9.6).
- To learn how to pass arguments to the **main** method from the command line (§9.7).

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?

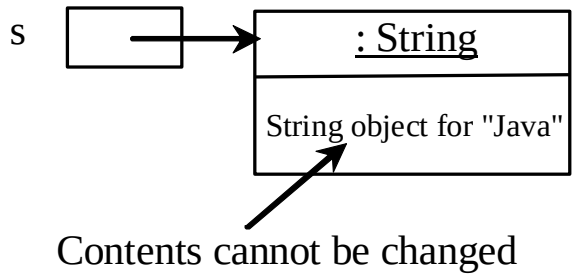
```
String s = "Java";
s = "HTML";
```

animation

Trace Code

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

After executing `String s = "Java";`

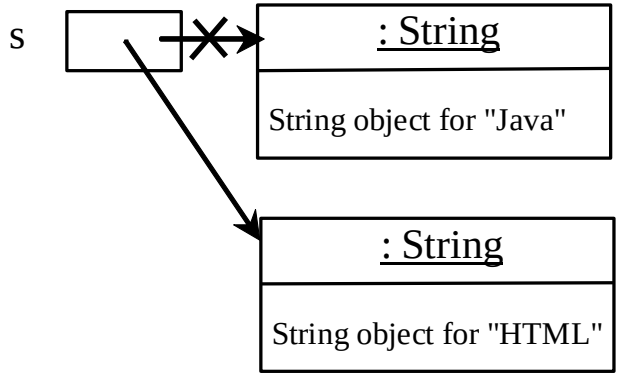


animation

Trace Code

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

After executing `s = "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:

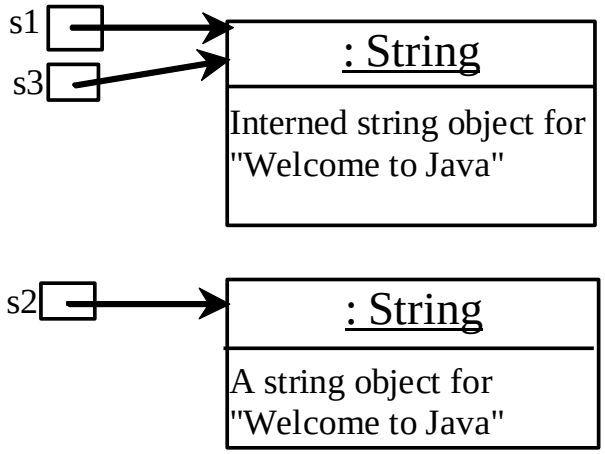
Examples

```
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));
```



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.

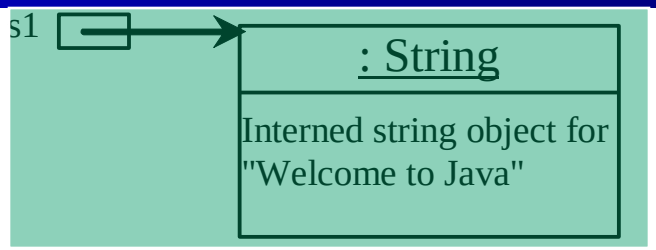
animation

Trace Code

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

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

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

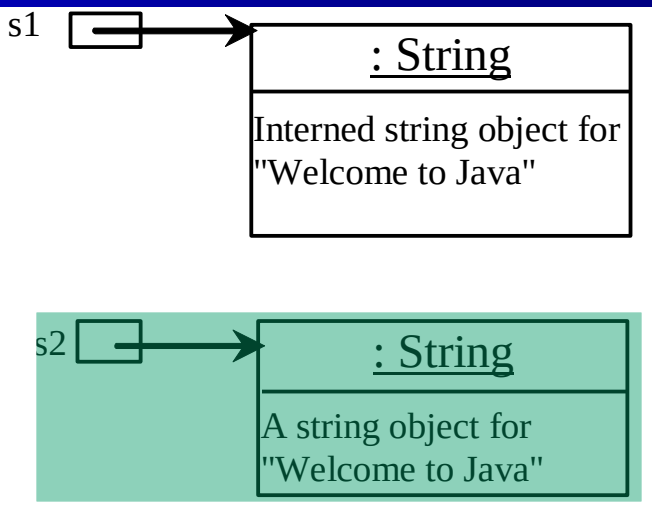


Trace Code

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

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

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

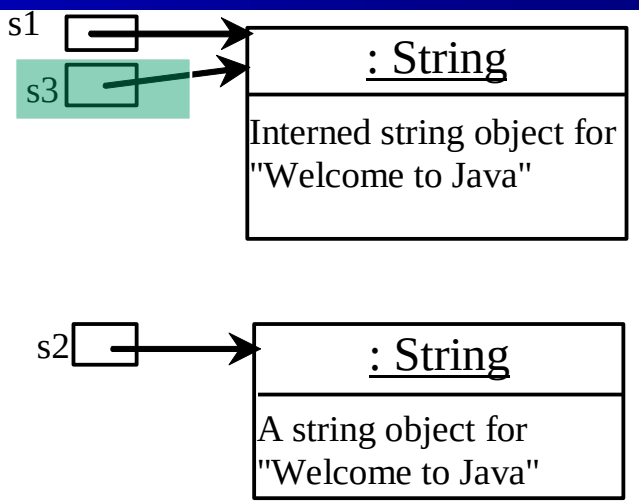


Trace Code

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

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

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



String Comparisons

java.lang.String	
+equals(s1: Object): boolean	Returns true if this string is equal to string s1.
+equalsIgnoreCase(s1: String): boolean	Returns true if this string is equal to string s1 case-insensitive.
+compareTo(s1: String): int	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: String): int	Same as compareTo except that the comparison is case-insensitive.
+regionMatches(toffset: int, s1: String, offset: int, len: int): boolean	Returns true if the specified subregion of this string exactly matches the specified subregion in string s1.
+regionMatches(ignoreCase: boolean, toffset: int, s1: String, offset: int, len: int): boolean	Same as the preceding method except that you can specify whether the match is case-sensitive.
+startsWith(prefix: String): boolean	Returns true if this string starts with the specified prefix.
+endsWith(suffix: String): boolean	Returns true if this string ends with the specified suffix.

String Comparisons

- equals

```
String s1 = new String("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
}
```

String Comparisons, cont.

- `compareTo(Object object)`

```
String s1 = new String("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 contents
}
else
   // s1 is less than s2
```

String Length, Characters, and Combining Strings

java.lang.String	
+length(): int	Returns the number of characters in this string.
+charAt(index: int): char	Returns the character at the specified index from this string.
+concat(s1: String): String	Returns a new string that concatenate this string with string s1.

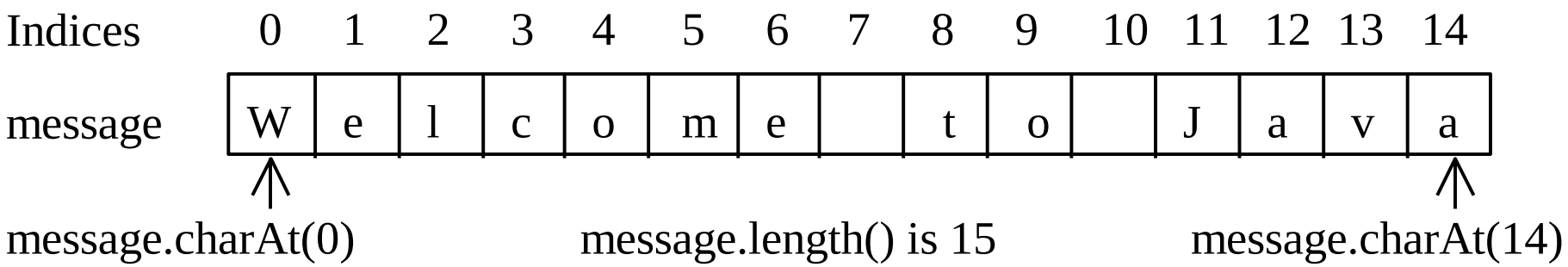
Finding String Length

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`



String Concatenation

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

String s3 = s1 + s2;

s1 + s2 + s3 + s4 + s5 same as

(((s1.concat(s2)).concat(s3)).concat(s4)).concat(s5);

Extracting Substrings

java.lang.String
+substring(beginIndex: int): String
+substring(beginIndex: int, endIndex: int): String

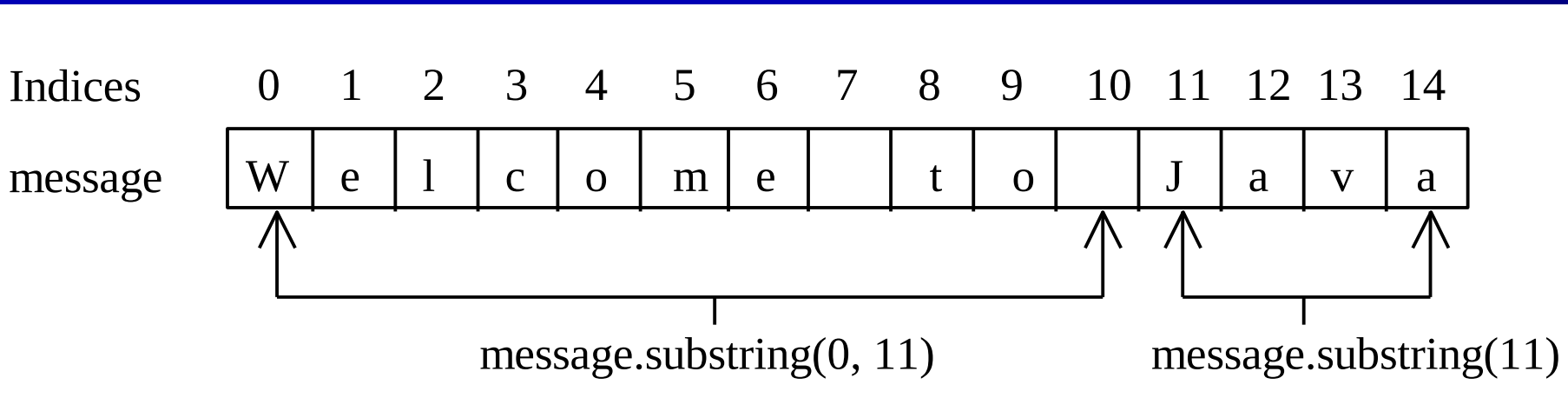
Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string, as shown in Figure 8.6.

Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1, as shown in Figure 8.6. Note that the character at endIndex is not part of the substring.

Extracting Substrings

You can extract a single character from a string using the charAt method. You can also extract a substring from a string using the substring method in the String class.

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



Converting, Replacing, and Splitting Strings

java.lang.String	
+toLowerCase(): String	Returns a new string with all characters converted to lowercase.
+toUpperCase(): String	Returns a new string with all characters converted to uppercase.
+trim(): String	Returns a new string with blank characters trimmed on both sides.
+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".toLowerCase() returns a new string, welcome.

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

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

"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 aNNNbNNNNNNNc.

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]);
```

Finding a Character or a Substring in a String

java.lang.String	
+indexOf(ch: char): int	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
+indexOf(ch: char, fromIndex: int): int	Returns the index of the first occurrence of ch after fromIndex in the string. Returns -1 if not matched.
+indexOf(s: String): int	Returns the index of the first occurrence of string s in this string. Returns -1 if not matched.
+indexOf(s: String, fromIndex: int): int	Returns the index of the first occurrence of string s in this string after fromIndex. Returns -1 if not matched.
+lastIndexOf(ch: int): int	Returns the index of the last occurrence of ch in the string. Returns -1 if not matched.
+lastIndexOf(ch: int, fromIndex: int): int	Returns the index of the last occurrence of ch before fromIndex in this string. Returns -1 if not matched.
+lastIndexOf(s: String): int	Returns the index of the last occurrence of string s. Returns -1 if not matched.
+lastIndexOf(s: String, fromIndex: int): int	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

`"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.

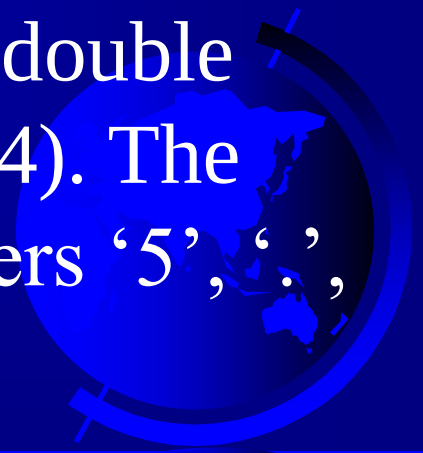
`"Welcome to Java".indexOf("Java", 5)` returns 11.

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

`"Welcome to Java".lastIndexOf('a')` returns 14.

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’.



Problem: Finding Palindromes

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

The `Character` Class

java.lang.Character	
+Character(value: char)	Constructs a character object with char value
+charValue(): char	Returns the char value from this object
+compareTo(anotherCharacter: Character): int	Compares this character with another
+equals(anotherCharacter: Character): boolean	Returns true if this character equals to another
+isDigit(ch: char): boolean	Returns true if the specified character is a digit
+isLetter(ch: char): boolean	Returns true if the specified character is a letter
+isLetterOrDigit(ch: char): boolean	Returns true if the character is a letter or a digit
+isLowerCase(ch: char): boolean	Returns true if the character is a lowercase letter
+isUpperCase(ch: char): boolean	Returns true if the character is an uppercase letter
+toLowerCase(ch: char): char	Returns the lowercase of the specified character
+toUpperCase(ch: char): char	Returns the uppercase of the specified character

Examples

Character charObject = new Character('b');

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

Problem: Counting Each Letter in a String

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

CountEachLetter

Run

`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

Main Method Is Just a Regular Method

You can call a regular method by passing actual parameters. Can you pass arguments to main? Of course, yes. For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

Command-Line Parameters

```
class TestMain {
  public static void main(String[] args) {
    ...
  }
}
```

```
java TestMain arg0 arg1 arg2 ... argn
```

Processing Command-Line Parameters

In the main method, get the arguments from `args[0]`, `args[1]`, ..., `args[n]`, which corresponds to `arg0`, `arg1`, ..., `argn` in the command line.

Problem: Calculator

- Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

Calculator

Run

java Calculator "2 + 3"

java Calculator "2 - 3"

java Calculator "2 / 3"

java Calculator "2 * 3"

Companion Website

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.

Companion Website

Matching Strings

```
"Java".matches("Java");
"Java".equals("Java");

"Java is fun".matches("Java.*")
"Java is cool".matches("Java.*")
"Java is powerful".matches("Java.*")
```

Companion Website

Regular Expression Syntax

Regular Expression	Matches	Example
x	a specified character x	Java matches Java
.	any single character	Java matches J..a
(ab|cd)	a, b, or c	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 [1-9]	Java2 matches "Java[\\d]"
\D	a non-digit	$Java matches "[\\D][\\D]ava"
\w	a word character	Java matches "[\\w]ava"
\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*	Java matches "[\\w]*"
p+	one or more occurrences of pattern *p*	Java matches "[\\w]+"
p?	zero or one occurrence of pattern *p*	Java matches "[\\w]?Java" Java matches "[\\w]?ava"
p{n}	exactly n occurrences of pattern *p*	Java matches "[\\w]{4}"
p{n,}	at least n occurrences of pattern *p*	Java matches "[\\w]{3,}"
p{n,m}	between n and m occurrences (inclusive)	Java matches "[\\w]{1,9}"

Companion Website

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.

Companion Website

Examples

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

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

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