Chapter 7 Strings

Prerequisites for Part II

Chapter 5 Arrays

Chapter 6 Objects and Classes

Chapter 7 Strings

Chapter 8 Inheritance and Polymorphism

Chapter 9 Abstract Classes and Interfaces

Chapter 10 Object-Oriented Modeling

Chapter 15 Exceptions and Assertions

Chapter 16 Simple Input and Output

You can cover GUI after Chapter 8

Chapter 11 Getting Started with GUI Programming

You can cover Exceptions and I/O after Chapter 8

Chapter 12 Event-Driven Programming

Liang, Introduction to Java Programming, Fifth Edition, (c) 2005 Pearson Education, Inc. All rights reserved. 0-13-148952-6
Objectives

- To use the `String` class to process fixed strings (§7.2).
- To use the `Character` class to process a single character (§7.3).
- To use the `StringBuffer` class to process flexible strings (§7.4).
- To use the `StringTokenizer` class to extract tokens from a string (§7.5).
- To know the differences among the `String`, `StringBuffer`, and `StringTokenizer` classes (§7.2-7.5).
- To use the JDK 1.5 `Scanner` class for console input and scan tokens using words as delimiters (§7.6).
- To input primitive values and strings from the keyboard using the `Scanner` class (§7.7).
- To learn how to pass strings to the `main` method from the command line (§7.8).
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
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+String()</code></td>
<td>Constructs an empty string</td>
</tr>
<tr>
<td><code>+String(value: String)</code></td>
<td>Constructs a string with the specified string literal value</td>
</tr>
<tr>
<td><code>+String(value: char[])</code></td>
<td>Constructs a string with the specified character array</td>
</tr>
<tr>
<td><code>charAt(index: int): char</code></td>
<td>Returns the character at the specified index from this string</td>
</tr>
<tr>
<td><code>compareTo(anotherString: String): int</code></td>
<td>Compares this string with another string</td>
</tr>
<tr>
<td><code>compareToIgnoreCase(anotherString: String): int</code></td>
<td>Compares this string with another string ignoring case</td>
</tr>
<tr>
<td><code>concat(anotherString: String): String</code></td>
<td>Concat this string with another string</td>
</tr>
<tr>
<td><code>endsWith(suffix: String): boolean</code></td>
<td>Returns true if this string ends with the specified suffix</td>
</tr>
<tr>
<td><code>equals(anotherString: String): boolean</code></td>
<td>Returns true if this string is equal to another string</td>
</tr>
<tr>
<td><code>equalsIgnoreCase(anotherString: String): boolean</code></td>
<td>Checks if this string equals another string ignoring case</td>
</tr>
<tr>
<td><code>getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin): void</code></td>
<td>Copies characters from this string into the destination character array</td>
</tr>
<tr>
<td><code>indexOf(ch: int): int</code></td>
<td>Returns the index of the first occurrence of ch</td>
</tr>
<tr>
<td><code>indexOf(ch: int, fromIndex: int): int</code></td>
<td>Returns the index of the first occurrence of ch after fromIndex</td>
</tr>
<tr>
<td><code>indexOf(str: String): int</code></td>
<td>Returns the index of the first occurrence of str</td>
</tr>
<tr>
<td><code>indexOf(str: String, fromIndex: int): int</code></td>
<td>Returns the index of the first occurrence of str after fromIndex</td>
</tr>
<tr>
<td><code>lastIndexOf(ch: int): int</code></td>
<td>Returns the index of the last occurrence of ch</td>
</tr>
<tr>
<td><code>lastIndexOf(ch: int, fromIndex: int): int</code></td>
<td>Returns the index of the last occurrence of ch before fromIndex</td>
</tr>
<tr>
<td><code>lastIndexOf(str: String): int</code></td>
<td>Returns the index of the last occurrence of str</td>
</tr>
<tr>
<td><code>lastIndexOf(str: String, fromIndex: int): int</code></td>
<td>Returns the index of the last occurrence of str before fromIndex</td>
</tr>
<tr>
<td><code>regionMatches(toffset: int, other: String, offset: int, len: int): boolean</code></td>
<td>Returns true if the specified subregion of this string exactly matches the specified subregion of the string argument</td>
</tr>
<tr>
<td><code>length(): int</code></td>
<td>Returns the number of characters in this string</td>
</tr>
<tr>
<td><code>replace(oldChar: char, newChar: char): String</code></td>
<td>Returns a new string with oldChar replaced by newChar</td>
</tr>
<tr>
<td><code>startsWith(prefix: String): boolean</code></td>
<td>Returns true if this string starts with the specified prefix</td>
</tr>
<tr>
<td><code>substring(beginIndex: int): String</code></td>
<td>Returns the substring from beginIndex</td>
</tr>
<tr>
<td><code>substring(beginIndex: int, endIndex: int): String</code></td>
<td>Returns the substring from beginIndex to endIndex</td>
</tr>
<tr>
<td><code>toCharArray(): char[]</code></td>
<td>Returns a char array consisting characters from this string</td>
</tr>
<tr>
<td><code>toLowerCase(): String</code></td>
<td>Returns a new string with all characters converted to lowercase</td>
</tr>
<tr>
<td><code>toUpperCase(): String</code></td>
<td>Returns a new string with all characters converted to uppercase</td>
</tr>
<tr>
<td><code>trim(): String</code></td>
<td>Returns a string with blank characters trimmed on both sides</td>
</tr>
<tr>
<td><code>copyValueOf(data: char[]): String</code></td>
<td>Returns a new string consisting of the char array data</td>
</tr>
<tr>
<td><code>valueOf(char): String</code></td>
<td>Returns a string consisting of the character c</td>
</tr>
<tr>
<td><code>valueOf(double): String</code></td>
<td>Same as copyValueOf(data: char[]): String</td>
</tr>
<tr>
<td><code>valueOf(float): String</code></td>
<td>Returns a string representing the double value</td>
</tr>
<tr>
<td><code>valueOf(int): String</code></td>
<td>Returns a string representing the float value</td>
</tr>
<tr>
<td><code>valueOf(long): String</code></td>
<td>Returns a string representing the int value</td>
</tr>
<tr>
<td><code>valueOf(c: char)</code></td>
<td>Returns a string representing the long value</td>
</tr>
</tbody>
</table>
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?

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

After executing

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

Contents cannot be changed

After executing

```
s = "HTML";
```

This string object is now unreferenced
Canonical Strings

Since strings are immutable, to improve efficiency and save memory, the JVM stores two String objects in the same object if they were created with the same string literal using the shorthand initializer. Such a string is referred to as a *canonical string*. You can also use a String object’s intern method to return a canonical string, which is the same string that is created using the shorthand initializer.
Examples

String s = "Welcome to Java";

String s1 = new String("Welcome to Java");

String s2 = s1.intern();

String s3 = "Welcome to Java";

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

display

s1 == s is false
s2 == s is true
s == s3 is true
Finding String Length

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

```java
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: Welcome to Java

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

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

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

String comparisons

```java
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)`

```java
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 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:

- toLowerCase
- toUpperCase
- trim
- replace(oldChar, newChar)
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’.
Example 7.1
Finding Palindromes

Objective: Checking whether a string is a palindrome: a string that reads the same forward and backward.
# The Character Class

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Character(value: char)</td>
<td>Constructs a character object with char value</td>
</tr>
<tr>
<td>+charValue(): char</td>
<td>Returns the char value from this object</td>
</tr>
<tr>
<td>+compareTo(anotherCharacter: Character): int</td>
<td>Compares this character with another</td>
</tr>
<tr>
<td>+equals(anotherCharacter: Character): boolean</td>
<td>Returns true if this character equals to another</td>
</tr>
<tr>
<td>+isDigit(ch: char): boolean</td>
<td>Returns true if the specified character is a digit</td>
</tr>
<tr>
<td>+isLetter(ch: char): boolean</td>
<td>Returns true if the specified character is a letter</td>
</tr>
<tr>
<td>+isLetterOrDigit(ch: char): boolean</td>
<td>Returns true if the character is a letter or a digit</td>
</tr>
<tr>
<td>+isLowerCase(ch: char): boolean</td>
<td>Returns true if the character is a lowercase letter</td>
</tr>
<tr>
<td>+isUpperCase(ch: char): boolean</td>
<td>Returns true if the character is an uppercase letter</td>
</tr>
<tr>
<td>+toLowerCase(ch: char): char</td>
<td>Returns the lowercase of the specified character</td>
</tr>
<tr>
<td>+toUpperCase(ch: char): char</td>
<td>Returns the uppercase of the specified character</td>
</tr>
</tbody>
</table>
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
Example 7.2
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.
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 object is fixed once the string is created.
## StringBuffer

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>StringBuffer()</code></td>
<td>Constructs an empty string buffer with capacity 16</td>
</tr>
<tr>
<td><code>StringBuffer(capacity: int)</code></td>
<td>Constructs a string buffer with the specified capacity</td>
</tr>
<tr>
<td><code>StringBuffer(str: String)</code></td>
<td>Constructs a string buffer with the specified string</td>
</tr>
<tr>
<td><code>append(data: char[])</code>: StringBuffer</td>
<td>Appends a char array into this string buffer</td>
</tr>
<tr>
<td><code>append(data: char[], offset: int, len: int): StringBuffer</code></td>
<td>Appends a subarray in data into this string buffer</td>
</tr>
<tr>
<td><code>append(v: aPrimitiveType): StringBuffer</code></td>
<td>Appends a primitive type value as string to this buffer</td>
</tr>
<tr>
<td><code>append(str: String): StringBuffer</code></td>
<td>Appends a string to this string buffer</td>
</tr>
<tr>
<td><code>capacity(): int</code></td>
<td>Returns the capacity of this string buffer</td>
</tr>
<tr>
<td><code>charAt(index: int): char</code></td>
<td>Returns the character at the specified index</td>
</tr>
<tr>
<td><code>delete(startIndex: int, endIndex: int): StringBuffer</code></td>
<td>Deletes characters from startIndex to endIndex</td>
</tr>
<tr>
<td><code>deleteCharAt(int index): StringBuffer</code></td>
<td>Deletes a character at the specified index</td>
</tr>
<tr>
<td><code>insert(index: int, data: char[], offset: int, len: int): StringBuffer</code></td>
<td>Inserts a subarray of the data in the array to the buffer at the specified index</td>
</tr>
<tr>
<td><code>insert(offset: int, data: char[])</code>: StringBuffer</td>
<td>Inserts data to this buffer at the position offset</td>
</tr>
<tr>
<td><code>insert(offset: int, b: aPrimitiveType): StringBuffer</code></td>
<td>Inserts a value converted to string into this buffer</td>
</tr>
<tr>
<td><code>insert(offset: int, str: String): StringBuffer</code></td>
<td>Inserts a string into this buffer at the position offset</td>
</tr>
<tr>
<td><code>length(): int</code></td>
<td>Returns the number of characters in this buffer</td>
</tr>
<tr>
<td><code>replace(int startIndex, int endIndex, String str): StringBuffer</code></td>
<td>Replaces the characters in this buffer from startIndex to endIndex with the specified string</td>
</tr>
<tr>
<td><code>reverse(): StringBuffer</code></td>
<td>Reverses the characters in the buffer</td>
</tr>
<tr>
<td><code>setCharAt(index: int, ch: char): void</code></td>
<td>Sets a new character at the specified index in this buffer</td>
</tr>
<tr>
<td><code>setLength(newLength: int): void</code></td>
<td>Sets a new length in this buffer</td>
</tr>
<tr>
<td><code>substring(startIndex: int): String</code></td>
<td>Returns a substring starting at startIndex</td>
</tr>
<tr>
<td><code>substring(startIndex: int, endIndex: int): String</code></td>
<td>Returns a substring from startIndex to endIndex</td>
</tr>
</tbody>
</table>
StringBuffer Constructors

- public StringBuffer()
  No characters, initial capacity 16 characters.

- public StringBuffer(int length)
  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");
Example 7.3
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.
The **StringTokenizer Class**

<table>
<thead>
<tr>
<th>java.util.StringTokenizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>+StringTokenizer(s: String)</td>
</tr>
<tr>
<td>+StringTokenizer(s: String, delimiters: String)</td>
</tr>
<tr>
<td>+StringTokenizer(s: String, delimiters: String, returnDelimiters: boolean)</td>
</tr>
<tr>
<td>+countTokens(): int</td>
</tr>
<tr>
<td>+hasMoreTokens(): boolean</td>
</tr>
<tr>
<td>+nextToken(): String</td>
</tr>
<tr>
<td>+nextToken(delimiters: String): String</td>
</tr>
</tbody>
</table>

Constructs a string tokenizer for the string.
Constructs a string tokenizer for the string with the specified delimiters.
Constructs a string tokenizer for the string with the delimiters and returnDelims.
Returns the number of remaining tokens.
Returns true if there are more tokens left.
Returns the next token.
Returns the next token using new delimiters.
Examples 1

String s = "Java is cool.";
StringTokenizer tokenizer = new StringTokenizer(s);

System.out.println("The total number of tokens is " +
tokenizer.countTokens());

while (tokenizer.hasMoreTokens())
    System.out.println(tokenizer.nextToken());

    The code displays

    The total number of tokens is 3
    Java
    is
    cool.
Examples 2

String s = "Java is cool.";
StringTokenizer tokenizer = new StringTokenizer(s, "ac");

System.out.println("The total number of tokens is "+
tokenizer.countTokens());

while (tokenizer.hasMoreTokens())
    System.out.println(tokenizer.nextToken());

The code displays

| The total number of tokens is 4 |
| J |
| v |
| is |
| ool. |
Examples 3

String s = "Java is cool.";
StringTokenizer tokenizer = new StringTokenizer(s, "ac", ture);

System.out.println("The total number of tokens is " +
tokenizer.countTokens());

while (tokenizer.hasMoreTokens())
    System.out.println(tokenizer.nextToken());

The code displays

The total number of tokens is 7
J
a
v
a
is
c
ool.
No no-arg Constructor in StringTokenizer

The `StringTokenizer` class does not have a no-arg constructor. Normally it is a good programming practice to provide a no-arg constructor for each class. On rare occasions, however, a no-arg constructor does not make sense. `StringTokenizer` is such an example. A `StringTokenizer` object must be created for a string, which should be passed as an argument from a constructor.
The Scanner Class

The delimiters are single characters in StringTokenizer. You can use the new JDK 1.5 java.util.Scanner class to specify a word as a delimiter.

```java
String s = "Welcome to Java! Java is fun! Java is cool!";
Scanner scanner = new Scanner(s);
scanner.useDelimiter("Java");
while (scanner.hasNext())
    System.out.println(scanner.next());
```

- Creates an instance of Scanner for the string.
- Sets “Java” as a delimiter.
- hasNext() returns true if there are still more tokens left.
- The next() method returns a token as a string.

**Output**

Welcome to
!
is fun!
is cool!
Scanning Primitive Type Values

If a token is a primitive data type value, you can use the methods `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `nextBoolean()` to obtain it. For example, the following code adds all numbers in the string. Note that the delimiter is space by default.

```java
String s = "1 2 3 4";
Scanner scanner = new Scanner(s);

int sum = 0;
while (scanner.hasNext())
    sum += scanner.nextInt();

System.out.println("Sum is " + sum);
```
Console Input Using Scanner

Another important application of the `Scanner` class is to read input from the console. For example, the following code reads an `int` value from the keyboard:

```java
System.out.print("Please enter an int value: ");
Scanner scanner = new Scanner(System.in);
int i = scanner.nextInt();
```

NOTE: `StringTokenizer` can specify several single characters as delimiters. `Scanner` can use a single character or a word as the delimiter. So, if you need to scan a string with multiple single characters as delimiters, use `StringTokenizer`. If you need to use a word as the delimiter, use `Scanner`. 
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.
Example 7.4  
Using Command-Line Parameters

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

```
java Calculator 2 + 3
java Calculator 2 - 3
java Calculator 2 / 3
java Calculator 2 “*” 3
```