Java Summer 18 Class 6 Notes
Characters
and
Strings

Objectives

- After you have read and studied this chapter, you should be able to
  - Declare and manipulate data of the char data type.
  - Write string processing program, applicable in areas such as bioinformatics, using String, StringBuilder, and StringBuffer objects.
  - Differentiate the three string classes and use the correct class for a given task.
  - Specify regular expressions for searching a pattern in a string.
  - Use the Pattern and Matcher classes.
  - Compare the String objects correctly.

Characters

- In Java, single characters are represented using the data type **char**.
- Character constants are written as symbols enclosed in single quotes.
- Characters are stored in a computer memory using some form of encoding.
- *ASCII*, which stands for *American Standard Code for Information Interchange*, is one of the document coding schemes widely used today.
- Java uses Unicode, which includes ASCII, for representing **char** constants.

ASCII Encoding

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
10	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
20	cd4	nak	syn	etb	can	em	sub	esc	fs	gs
30	rs	us	sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}	|	~	del		

For example, character 'O' is 79 (row value 70 + col value 9 = 79).

Unicode Encoding

- The *Unicode Worldwide Character Standard* (*Unicode*) supports the interchange, processing, and display of the written texts of diverse languages.
- Java uses the Unicode standard for representing **char** constants.

```
char ch1 = 'X';

System.out.println(ch1);
System.out.println( (int) ch1);
```

→ X
→ 88

Character Processing

```
char ch1, ch2 = 'X';
```

Declaration and initialization

```
System.out.print("ASCII code of character X is " +
                 (int) 'X' );

System.out.print("Character with ASCII code 88 is "
        + (char)88 );
```

Type conversion between int and char.

```
'A' < 'c'
```

This comparison returns true because ASCII value of 'A' is 65 while that of 'c' is 99.

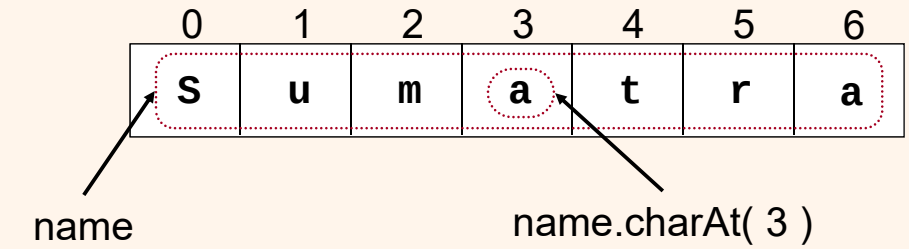
Strings

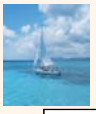
- A *string* is a sequence of characters that is treated as a single value.
- Instances of the **String** class are used to represent strings in Java.
- We can access individual characters of a string by calling the **charAt** method of the **String** object.

Accessing Individual Elements

- Individual characters in a String accessed with the charAt method.

```
String name = "Sumatra";
```





Example: Counting Vowels

```
char     letter;
System.out.println("Your name:");
String   name = scanner.next(); //assume 'scanner' is created properly
int      numberOfCharacters = name.length();
int      vowelCount  = 0;

for (int i = 0; i < numberOfCharacters; i++) {
     letter = name.charAt(i);

     if (    letter == 'a' || letter == 'A' ||
             letter == 'e' || letter == 'E' ||
             letter == 'i' || letter == 'I' ||
             letter == 'o' || letter == 'O' ||
             letter == 'u' || letter == 'U'
                                            ) {
          vowelCount++;
     }
}
System.out.print(name + ", your name has " + vowelCount + " vowels");
```

Here's the code to count the number of vowels in the input string.

Example: Counting 'Java'

```
int         javaCount      = 0;
boolean     repeat         = true;
String      word;
Scanner     scanner = new Scanner(System.in);

while ( repeat ) {
     System.out.print("Next word:");
     word = scanner.next();

     if ( word.equals("STOP") )     {
          repeat = false;

     } else if ( word.equalsIgnoreCase("Java") ) {
          javaCount++;
     }
}
```

Continue reading words and count how many times the word Java occurs in the input, ignoring the case.

Notice how the comparison is done. We are not using the == operator.

Other Useful String Operators

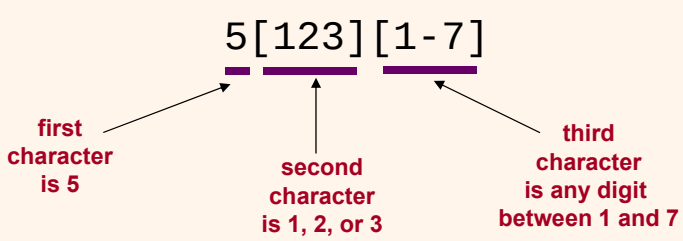
Method	Meaning
compareTo	Compares the two strings. `str1.compareTo( str2 )`
substring	Extracts the a substring from a string. `str1.substring( 1, 4 )`
trim	Removes the leading and trailing spaces. `str1.trim( )`
valueOf	Converts a given primitive data value to a string. `String.valueOf( 123.4565 )`
startsWith	Returns true if a string starts with a specified prefix string. `str1.startsWith( str2 )`
endsWith	Returns true if a string ends with a specified suffix string. `str1.endsWith( str2 )`

- See the String class documentation for details.

Pattern Example

- Suppose students are assigned a three-digit code:
  - The first digit represents the major (5 indicates computer science);
  - The second digit represents either in-state (1), out-of-state (2), or foreign (3);
  - The third digit indicates campus housing:
    - On-campus dorms are numbered 1-7.
    - Students living off-campus are represented by the digit 8.

The 3-digit pattern to represent computer science majors living on-campus is



Regular Expressions

- The pattern is called a *regular expression*.
- Rules
  - The brackets [ ] represent choices
  - The asterisk symbol * means zero or more occurrences.
  - The plus symbol + means one or more occurrences.
  - The hat symbol ^ means negation.
  - The hyphen – means ranges.
  - The parentheses ( ) and the vertical bar | mean a range of choices for multiple characters.

Regular Expression Examples

Expression	Description
`[013]`	A single digit 0, 1, or 3.
`[0-9][0-9]`	Any two-digit number from 00 to 99.
`[0-9&&[^4567]]`	A single digit that is 0, 1, 2, 3, 8, or 9.
`[a-z0-9]`	A single character that is either a lowercase letter or a digit.
`[a-zA-z][a-zA-Z0-9_$]*`	A valid Java identifier consisting of alphanumeric characters, underscores, and dollar signs, with the first character being an alphabet.
`[wb](ad|eed)`	Matches `wad, weed, bad`, and `beed`.
`(AZ|CA|CO)[0-9][0-9]`	Matches `AZxx`,`CAxx`, and `COxx`, where `x` is a single digit.

The replaceAll Method

- The replaceAll method replaces all occurrences of a substring that matches a given regular expression with a given replacement string.

Replace all vowels with the symbol @

```
String originalText, modifiedText;

originalText = ...;    //assign string

modifiedText =
          originalText.replaceAll("[aeiou]","@");
```


The Pattern and Matcher Classes

- The matches and replaceAll methods of the String class are shorthand for using the Pattern and Matcher classes from the java.util.regex package.
- If str and regex are String objects, then

```
str.matches(regex);
```

is equivalent to

```
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(str);
matcher.matches();
```

The compile Method

- The compile method of the Pattern class converts the stated regular expression to an internal format to carry out the pattern-matching operation.
- This conversion is carried out every time the matches method of the String class is executed, so it is more efficient to use the compile method when we search for the same pattern multiple times.

The find Method

- The find method is another powerful method of the Matcher class.
  - It searches for the next sequence in a string that matches the pattern, and returns true if the pattern is found.
- When a matcher finds a matching sequence of characters, we can query the location of the sequence by using the start and end methods.

The String Class is Immutable

- In Java a String object is immutable
  - This means once a String object is created, it cannot be changed, such as replacing a character with another character or removing a character
  - The String methods we have used so far do not change the original string. They created a new string from the original. For example, substring creates a new string from a given string.
- The String class is defined in this manner for efficiency reason.

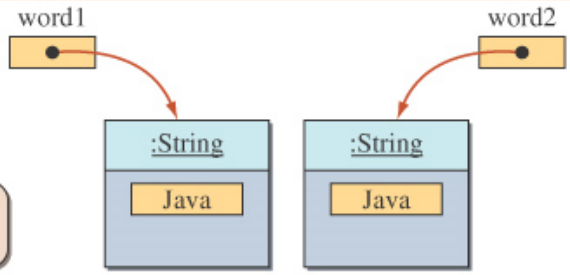
Effect of Immutability

```
String word1, word2;

word1 = new String("Java");

word2 = new String("Java");
```

Whenever the **new** operator is used, there will be a new object.



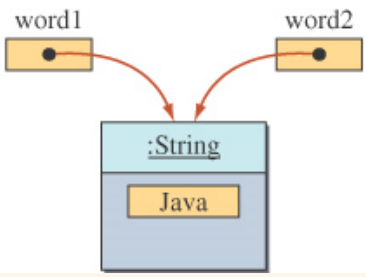

```
String word1, word2;

word1 = "Java";

word2 = "Java";
```

We can do this because String objects are immutable.

Literal string constant such as "Java" will always refer to the one object.

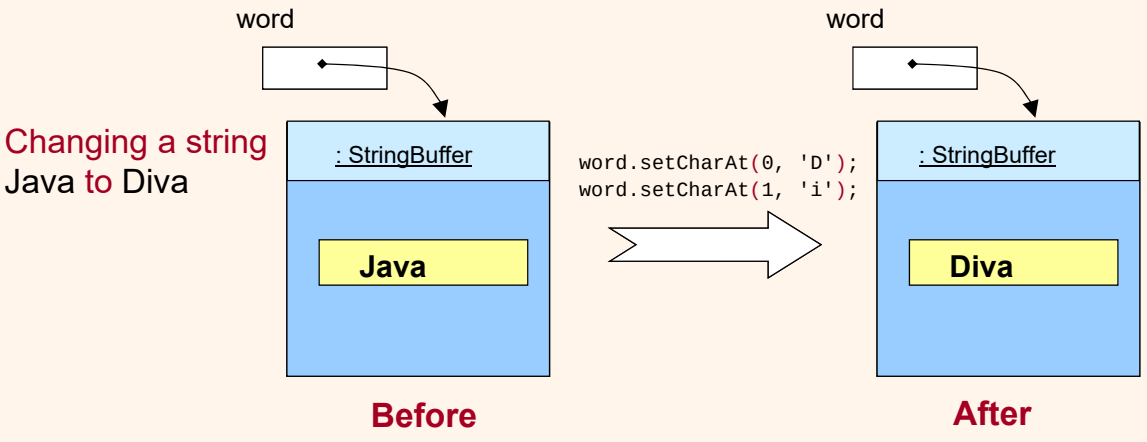


The StringBuffer Class

- In many string processing applications, we would like to change the contents of a string. In other words, we want it to be mutable.
- Manipulating the content of a string, such as replacing a character, appending a string with another string, deleting a portion of a string, and so on, may be accomplished by using the **StringBuffer** class.

StringBuffer Example

```
StringBuffer word = new StringBuffer("Java");
word.setCharAt(0, 'D');
word.setCharAt(1, 'i');
```



Sample Processing

Replace all vowels in the sentence with 'X'.

```
char         letter;
String       inSentence   = JOptionPane.showInputDialog(null, "Sentence:");
StringBuffer tempStringBuffer   = new StringBuffer(inSentence);
int          numberOfCharacters = tempStringBuffer.length();

for (int index = 0; index < numberOfCharacters; index++) {

     letter = tempStringBuffer.charAt(index);

     if ( letter == 'a' || letter == 'A' || letter == 'e' || letter == 'E' ||
          letter == 'i' || letter == 'I' || letter == 'o' || letter == 'O' ||
          letter == 'u' || letter == 'U'  ) {
          tempStringBuffer.setCharAt(index,'X');
     }
}
JOptionPane.showMessageDialog(null, tempStringBuffer );
```


The append and insert Methods

- We use the append method to append a String or StringBuffer object to the end of a StringBuffer object.
  - The method can also take an argument of the primitive data type.
  - Any primitive data type argument is converted to a string before it is appended to a StringBuffer object.
- We can insert a string at a specified position by using the insert method.

The StringBuilder Class

- This class is new to Java 5.0 (SDK 1.5)
- The class is added to the newest version of Java to improve the performance of the StringBuffer class.
- StringBuffer and StringBuilder support exactly the same set of methods, so they are interchangeable.
- There are advanced cases where we must use StringBuffer, but all sample applications in the book, StringBuilder can be used.
- Since the performance is not our main concern and that the StringBuffer class is usable for all versions of Java, we will use StringBuffer only in this book.

Bioinformatics

- Bioinformatics is a field of study that explores the use of computational techniques in solving biological problems.
- Genes are made of DNA (deoxyribonucleic acid), which is a sequence of molecules called *nucleotides or bases.*
- DNA provides instructions to the cell, so it serves a role similar to a computer program.
  - A cell is a computer that produces proteins (output) by reading instructions in DNA (program).
- The genetic information in DNA is encoded as a sequence of four chemical bases—adenine (A), guanine (G), cytosine (C), and thymine (T).

String Processing and Bioinformatics

- DNA is encoded as a sequence of bases.
- This information can be represented as a string of four letters—A, T, G, and C.
- Common operations biologists perform on DNA sequences can be implemented as string processing programs.