Chapter 9
Character and String

Objectives

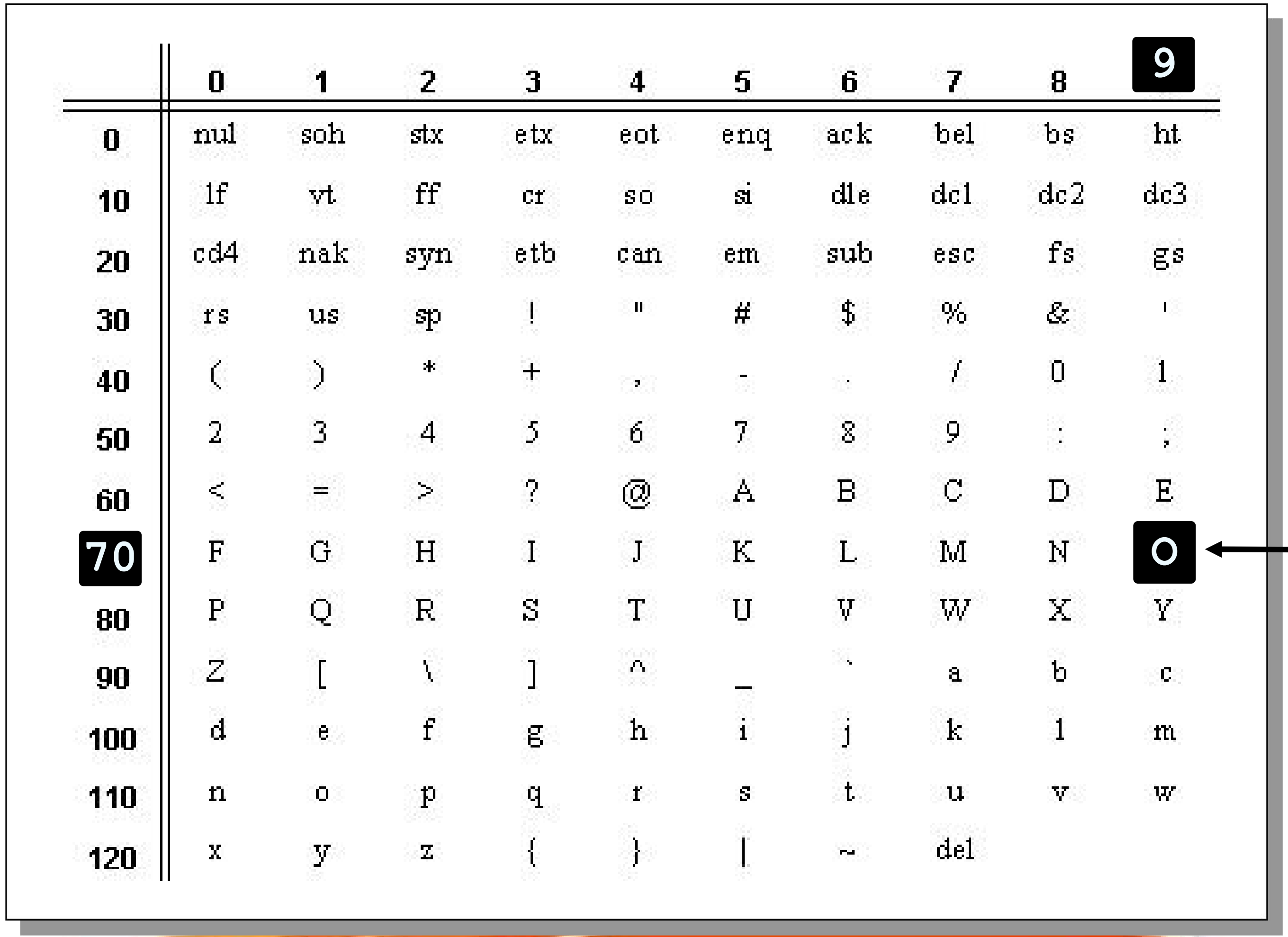
- After you have read and studied this chapter, you should be able to
  - Declare and manipulate data of the char data type.
  - Write string processing program, applicable in areas such as bioinformatics, using String, StringBuilder, and StringBuffer objects.
  - Specify regular expressions for searching 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.
- ASCII, using 8 bits, 265 symbols can represent
- To accommodate the character symbols of non-English languages, Java uses Unicode, which includes ASCII, for representing **char** constants, as a total of 34,168 distinct characters (2 bytes)

ASCII Encoding

- Ascii encoding table



	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);
```

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.

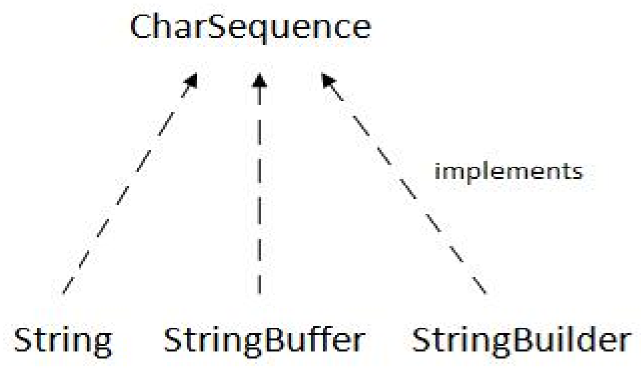
String

- String is basically an object that represents sequence of char values.
- An array of characters works same as Java string.

```
char[] ch={‘w',’e’,’l',’c’,’o’,’m',’e’};
String s=new String(ch);
            Or
String s=“welcome”;
```

CharSequence Interface

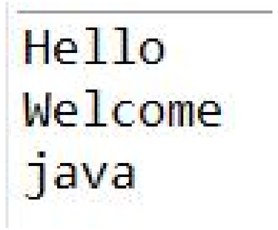
- The CharSequence interface is used to represent the sequence of characters.
- String, StringBuffer and StringBuilder classes implement it.
- It means, we can create strings in java by using these three classes.



- Java String is **immutable** which means it cannot be changed.
- StringBuffer and StringBuilder is **mutable**.

Example

```
public static void main(String[] args) {
   String s1="Hello"; // using literal
   String s2= new String("Welcome");// using new keyword
   char [] ch= {'j','a','v','a'};
   String s3= new String(ch); // convert char array to string
   System.out.println(s1);
   System.out.println(s2);
   System.out.println(s3);
}
```



```
Hello
Welcome
java
```

Example : String, StringBuffer and StringBuilder

```
public class string_eg {
 public static void concat1(String s1)
    {
        s1 = s1 + " of Computer Studies";
    }

    public static void concat2(StringBuilder s2)
    {
        s2.append(" of Computer Studies");
    }

    public static void concat3(StringBuffer s3)
    {
        s3.append(" of Computer Studies");
    }
```

```
public static void main(String[] args)
    {
    String s1 = "University";
        concat1(s1);
        System.out.println("String: " + s1);

        StringBuilder s2 = new StringBuilder("University");
        concat2(s2);
        System.out.println("StringBuilder: " + s2);

        StringBuffer s3 = new StringBuffer("University");
        concat3(s3);
        System.out.println("StringBuffer: " + s3);
    }
}
```

```
String: University
StringBuilder: University of Computer Studies
StringBuffer: University of Computer Studies
```

Some useful methods of java.lang.String class

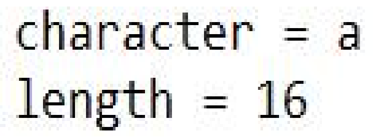
NO	Methods	Descriptions
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	String substring(int beginIndex)	returns substring for given begin index.
4	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
5	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.
6	boolean isEmpty()	checks if string is empty.
7	String concat(String str)	concatenates the specified string.
8	String replace(char oldChar, char newChar)	replaces all occurrences of the specified char value.
9	String replace(CharSequence target, CharSequence replacement)	replaces all occurrences of the specified CharSequence.
10	equalsIgnoreCase	compares another string. It doesn't check case.
11	equal	checks the equality of string with the given object.

Example

```
import java.util.*;
public class CountJava {
public static void main (String[] args) {
    Scanner scanner = new Scanner(System.in);
    int javaCount = 0;
    String word;
    System.out.print("Enter sentence: ");
    while (true) {
        word = scanner.next( );
        if (word.equals("STOP")) {
                break;
        } else if (word.equalsIgnoreCase("Java")){
                javaCount++;
        }
    }
    System.out.println("'Java' count: " + javaCount );
}
}
```

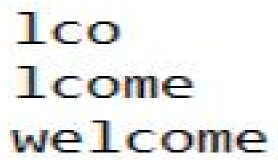
Example

```java
String s1="Java Programming";
System.out.println("character = " +s1.charAt(3));
System.out.println("length = " +s1.length());
```



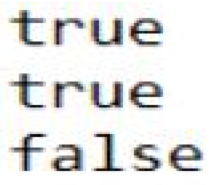
```
character = a
length = 16
```

```java
String s2="welcome";
System.out.println(s2.substring(2, 5));
System.out.println(s2.substring(2));
System.out.println(s2.substring(0));
```



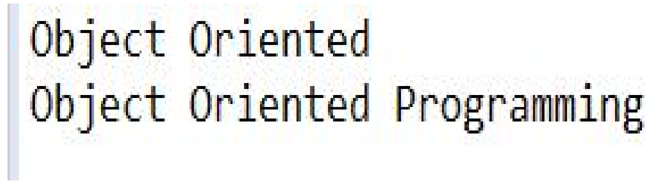
```
lco
lcome
welcome
```

```
String s3="Are you studying?";
System.out.println(s3.contains("Are you"));
System.out.println(s3.contains("study"));
System.out.println(s3.contains("going"));
// true if sequence of char value exists, otherwise false

System.out.println(s3.isEmpty());
// isEmpty() method checks if this string is empty or not.
// It returns true, if length of string is 0 otherwise false..
```



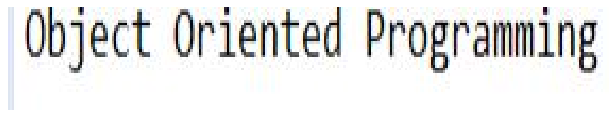
```
true
true
false
```

```
String s5="Object Oriented ";
s5.concat("Programming");
System.out.println(s5);
s5=s5.concat("Programming");
System.out.println(s5);
```



```
Object Oriented
Object Oriented Programming
```

```
String str1="Object";
String str2=" Oriented";
String str3=" Programming";
String str4=str1.concat(str2).concat(str3);
System.out.println(str4);
```



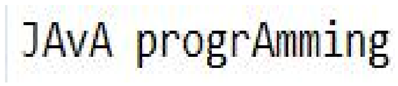
```
Object Oriented Programming
```

- **concat()** method *combines specified string at the end of this string. It* append another string.
- **public** String concat(String anotherString)

There are two type of replace methods in java string.

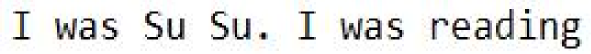
(1) public String replace(**char** oldChar, **char** newChar)
(2) public String replace(CharSequence target, CharSequence replacement)

(1)
```
String s6="Java programming";
String s=s6.replace('a', 'A’);
System.out.println(s);
```



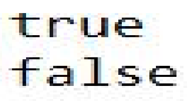
```
JAvA progrAmming
```

(2)
```
String s7="I am Su Su. I am reading";
String ss=s7.replace("am", "was");
System.out.println(ss);
```



```
I was Su Su. I was reading
```

```
String s1="Java";
String s2="JAVA";
System.out.println(s1.equalsIgnoreCase(s2));
System.out.println(s1.equals(s2));
```



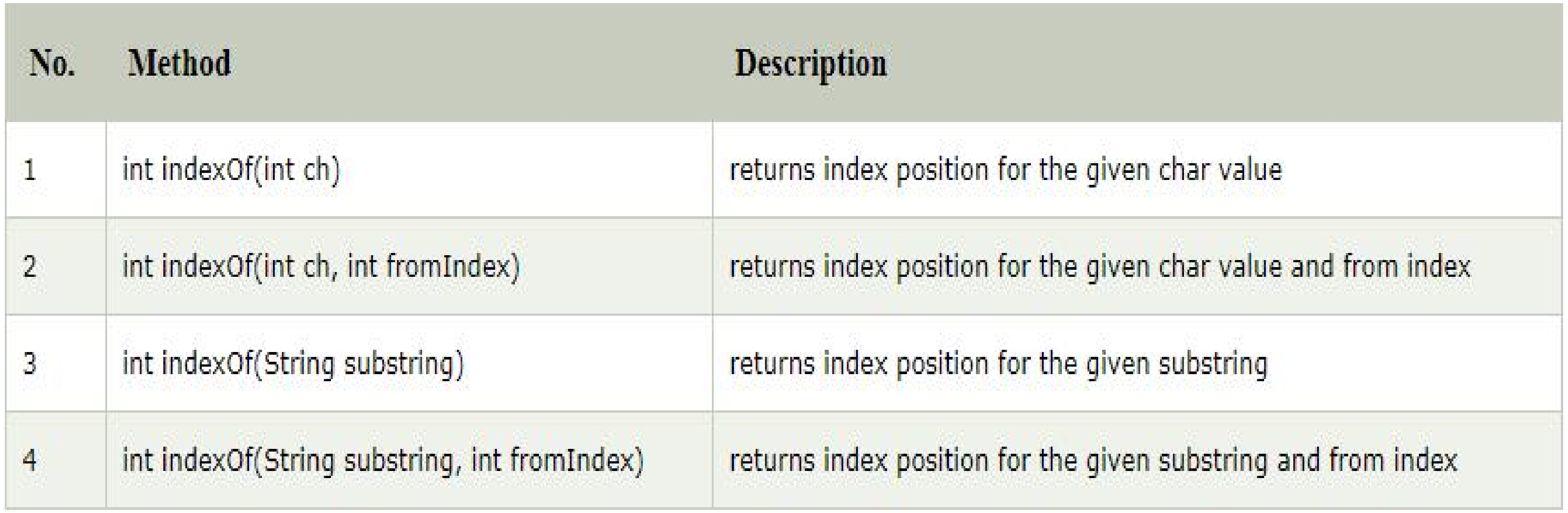
```
true
false
```

(3) String replaceAll(String regex, String replacement): It replaces all the substrings that fits the given regular expression with the replacement String.

```
String s8="Java Programming";
System.out.println(s8.replaceAll("a", "b"));                    //Jbvb Progrbmming

String s13="My .com is java.com";
String s14=s13.replaceAll(".com", ".net");
System.out.println(s14);                    //My .net is java.net

String s11="University of Computer Studies , Yangon";
String s12=s11.replaceAll("(.*)Computer(.*)", "Computer University");
System.out.println(s12);               // Computer University
```

indexOf() method

- There are 4 types of indexOf method in java.
- **indexOf()** method returns index of given character value or substring



No.	Method	Description
1	int indexOf(int ch)	returns index position for the given char value
2	int indexOf(int ch, int fromIndex)	returns index position for the given char value and from index
3	int indexOf(String substring)	returns index position for the given substring
4	int indexOf(String substring, int fromIndex)	returns index position for the given substring and from index

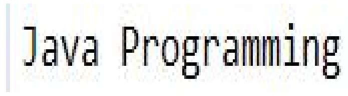
indexOf() and lastIndexOf()

```
String s6= "University of Computer Studies, Yangon : Computer Science";
System.out.println(s6.indexOf('o’));   //11
System.out.println(s6.indexOf('o', 12));  //15
System.out.println(s6.indexOf("Computer"));  //14
System.out.println(s6.indexOf("Computer", 20));  //41


System.out.println(s6.lastIndexOf('i')); //52
System.out.println(s6.lastIndexOf('i',50));//27
```

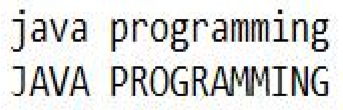
trim()

- **trim()** is a built-in function that eliminates leading and trailing spaces.
- The trim() method in java checks this Unicode value before and after the string, if it exists then removes the spaces and returns the omitted string.

```
String st="  Java Programming";
System.out.println(st.trim());

String s4="JAVA PROGRAMMINg";
System.out.println(s4.toLowerCase());
 System.out.println(s4.toUpperCase());
```



```
Java Programming
```

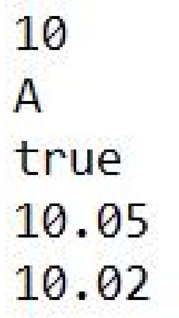


```
java programming
JAVA PROGRAMMING
```

valueOf()

- The **java string valueOf()** method converts different types of values into string.
- By the help of string valueOf() method, you can convert int to string, long to string, boolean to string, character to string, float to string, double to string, object to string and char array to string.

```
	int value=10; char ch='A';
	boolean flag=true; float f  = 10.05f;     double d = 10.02;
System.out.println(String.valueOf(value));
System.out.println(String.valueOf(ch));
System.out.println(String.valueOf(flag));
System.out.println(String.valueOf(f));
System.out.println(String.valueOf(d));
```

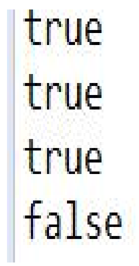


```
10
A
true
10.05
10.02
```

startsWith()/ endsWith()

- The **java string startsWith()** method checks if this string starts with given prefix.
- It returns true if this string starts with given prefix else returns false.
- The **java string endsWith()** method checks if this string ends with given suffix.
- It returns true if this string ends with given suffix else returns false.

```
String s1="java string example";
System.out.println(s1.startsWith("ja"));
System.out.println(s1.startsWith("java string"));
System.out.println(s1.endsWith("le"));
System.out.println(s1.endsWith("exam"));
```



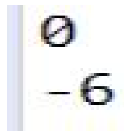
```
true
true
true
false
```

String compareTo()

- is used for comparing two strings lexicographically.
- Each character of both the strings is converted into a Unicode value for comparison.
- If both the strings are equal then this method returns 0 else it returns positive or negative value.
- The result is positive if the first string is lexicographically greater than the second string else the result would be negative.

```
 String st1="java";
 String st2="programming";
 String st3="java";
 System.out.println(st1.compareTo(st3));
  System.out.println(st1.compareTo(st2));
```

How about st2.comparteTo(st1)?



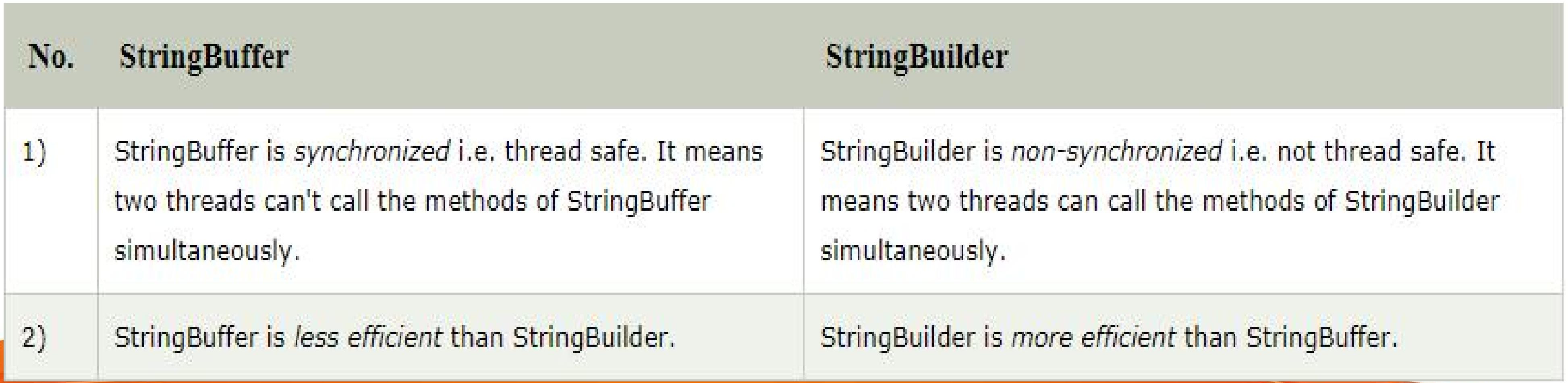
```
0
-6
```

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

StringBuffer and String Builder

- Since String is immutable in Java, whenever we do String manipulation like concatenation, substring etc, it generates a new String and discards the older String for garbage collection.
- These are heavy operations and generate a lot of garbage in heap.
- So Java has provided StringBuffer and StringBuilder class that should be used for String manipulation

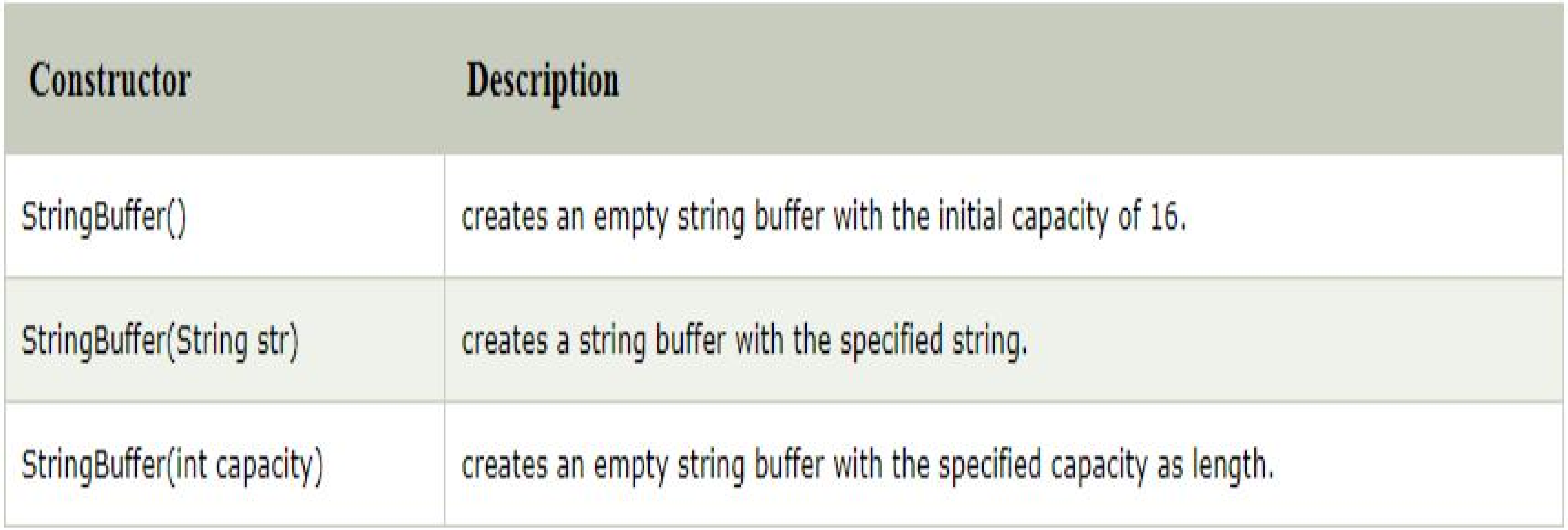


No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

Example

```
StringBuffer st=new StringBuffer("Good");
st.append(" Morning");
System.out.println(st);

StringBuilder str=new StringBuilder("Welcome");
str.append(" UCSY");
System.out.println(str);
```

Constructor of StringBuffer Class



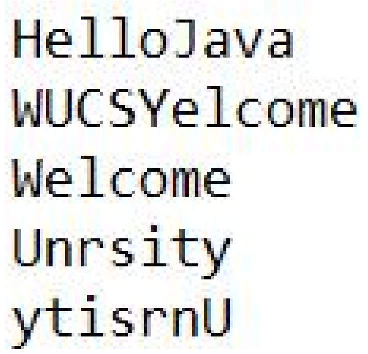
Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

Some useful methods of StringBuffer class

Methods	Descriptions
append(String s)	is used to append the specified string with this string. append(char), append(boolean), append(int), append(float), append(double) etc.
insert(int offset, String s)	is used to insert the specified string with this string at the specified position. insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double)
replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
reverse()	is used to reverse the string.
charAt(int index)	is used to return the character at the specified position.
length()	is used to return the length of the string i.e. total number of characters.
substring(int beginIndex)	is used to return the substring from the specified beginIndex.
substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Example

```
StringBuffer s1=new StringBuffer("Hello");
System.out.println(s1.append("Java"));

StringBuffer s2=new StringBuffer("Welcome");
System.out.println(s2.insert(1, "UCSY"));
System.out.println(s2.replace(1, 5, ""));

StringBuffer s3=new StringBuffer("University");
System.out.println(s3.delete(2, 5));
System.out.println(s3.reverse());
```

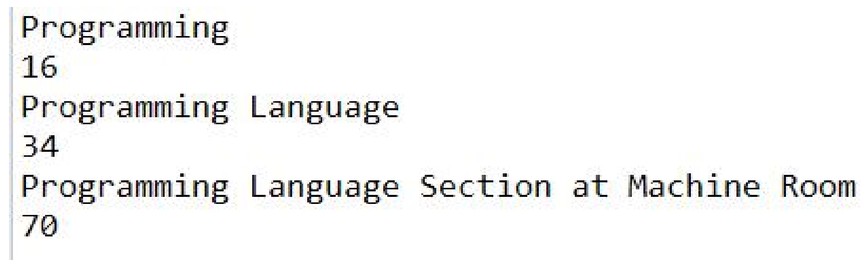


```
HelloJava
WUCSYelcome
Welcome
Unrsity
ytisrnU
```

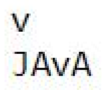
Example

```
StringBuffer s4=new StringBuffer();
System.out.println(s4.capacity());
System.out.println(s4.append("Programming"));
System.out.println(s4.capacity());
System.out.println(s4.append(" Language"));
System.out.println(s4.capacity());
System.out.println(s4.append(" Section at Machine Room"));
System.out.println(s4.capacity());

StringBuffer s5=new StringBuffer("Java");
System.out.println(s5.charAt(2));
s5.setCharAt(1, 'A');
s5.setCharAt(3, 'A');
System.out.println(s5);
```

The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.



```
Programming
16
Programming Language
34
Programming Language Section at Machine Room
70
```



```
v
JAvA
```

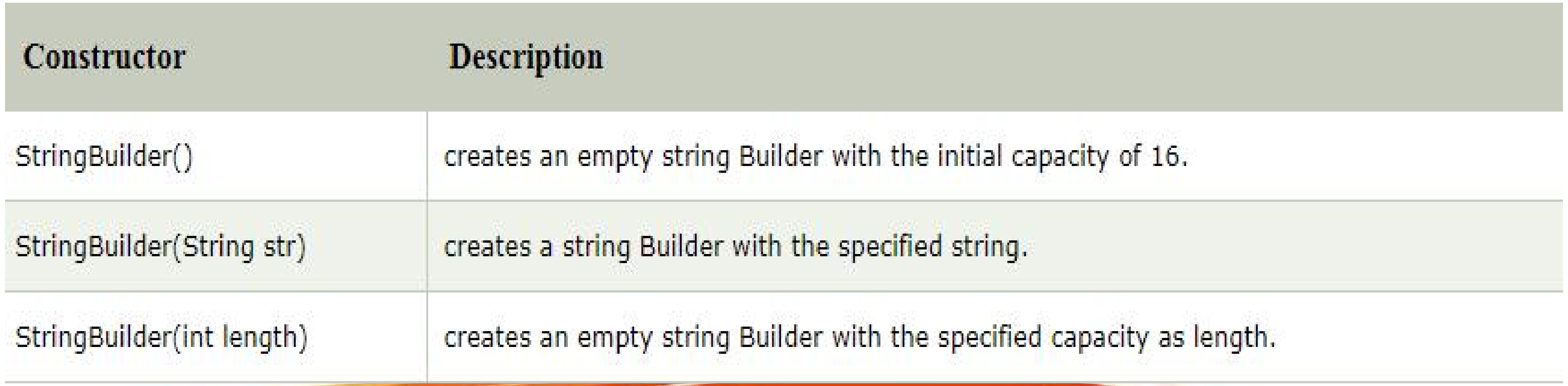
Example

```
ımport java.util.*;
class ReplaceVowelsWithX {
public static void main (String[] args)
{
    Scanner scanner = new
    Scanner(System.in);
    scanner.useDelimiter(System.getProper
    ty("line.separator"));
    StringBuffer tempStringBuffer;
    String inSentence;
    char letter;
    System.out.println("Sentence: ");
    inSentence = scanner.next();
    tempStringBuffer = new
    StringBuffer(inSentence);
```

```
    for (int index = 0; index <
    tempStringBuffer.length(); 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');
    }
    }
    System.out.println("Input: " + inSentence);
    System.out.println("Output: " +
    tempStringBuffer);
    }
}
```

Constructor of StringBuilder Class

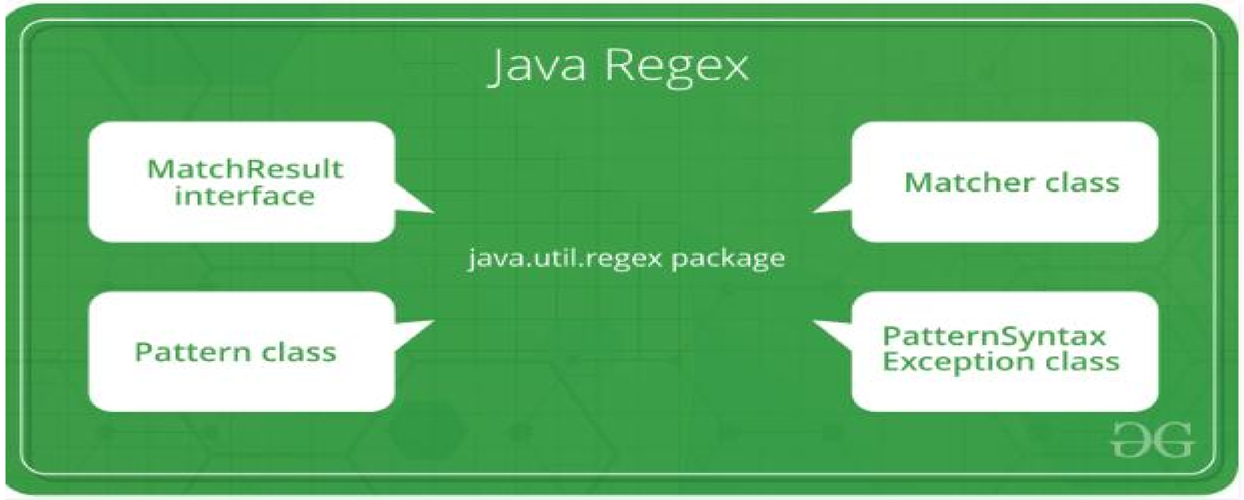
- is used to create mutable (modifiable) string.
- StringBuffer and StringBuilder support exactly the same set of methods, so they are interchangeable.
- *same as StringBuffer class* except that it is ***non-synchronized***.
- It is available since JDK 1.5.



Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
StringBuilder(int length)	creates an empty string Builder with the specified capacity as length.

Regular Expression

- **Java Regex** or **Regular Expression** is an API to *define a pattern for searching or manipulating strings*.
- Used to define the constraint on strings such as password and email validation.
- Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.
- java.util.regex.Pattern – Used for defining patterns
- java.util.regex.Matcher – Used for performing match operations on text using patterns



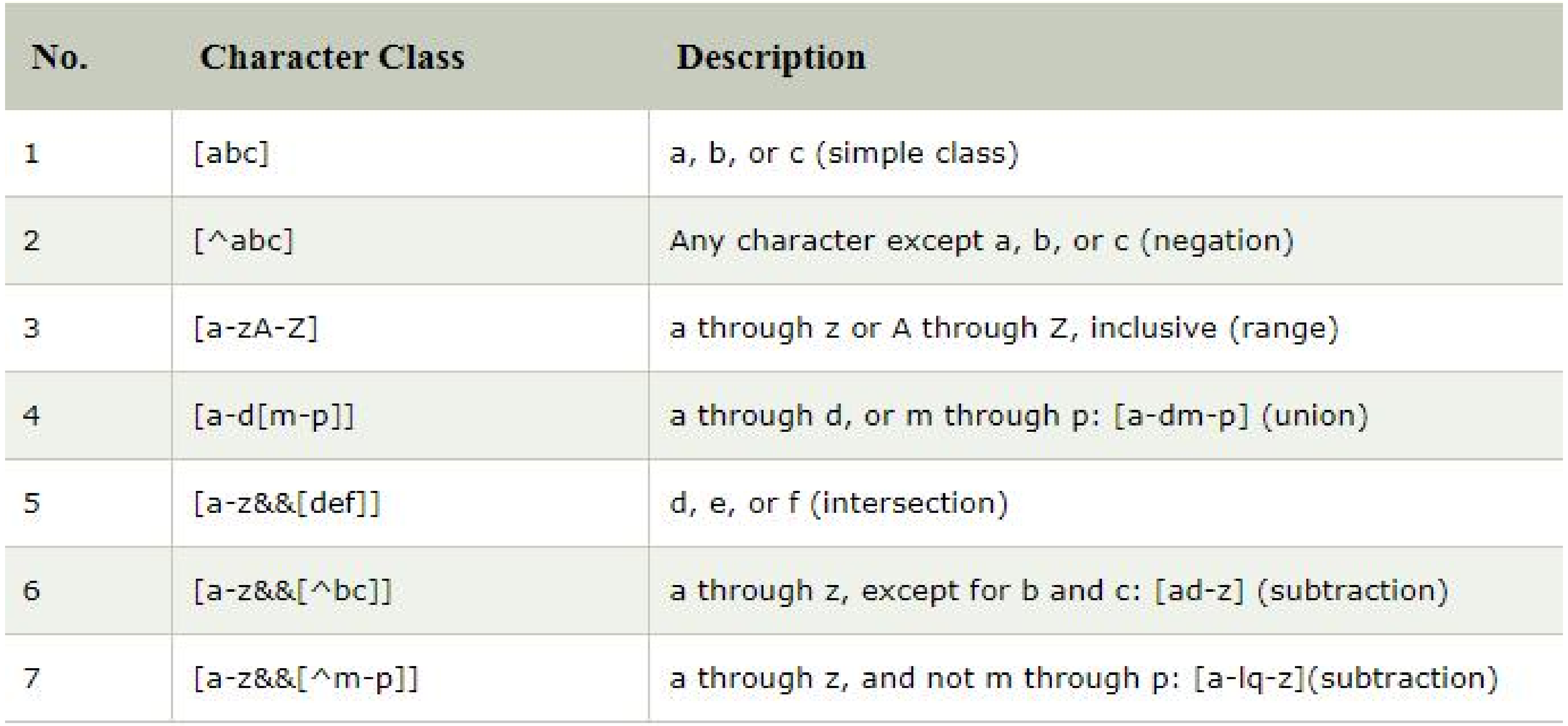
Picture : https://www.geeksforgeeks.org

Regular Expression

- Some 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.

Regex Character Class

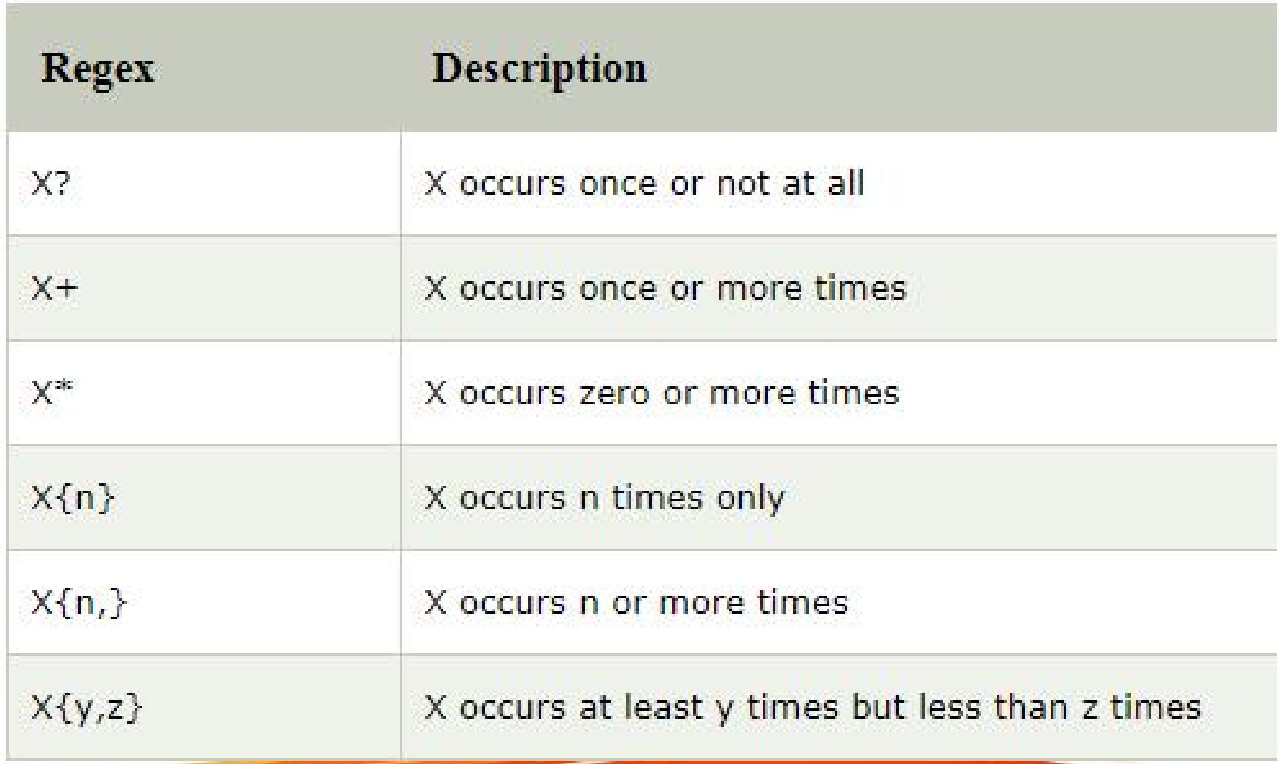
The Java regular expression syntax has a few predefined character classes



No.	Character Class	Description
1	[abc]	a, b, or c (simple class)
2	[^abc]	Any character except a, b, or c (negation)
3	[a-zA-Z]	a through z or A through Z, inclusive (range)
4	[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
5	[a-z&&[def]]	d, e, or f (intersection)
6	[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
7	[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

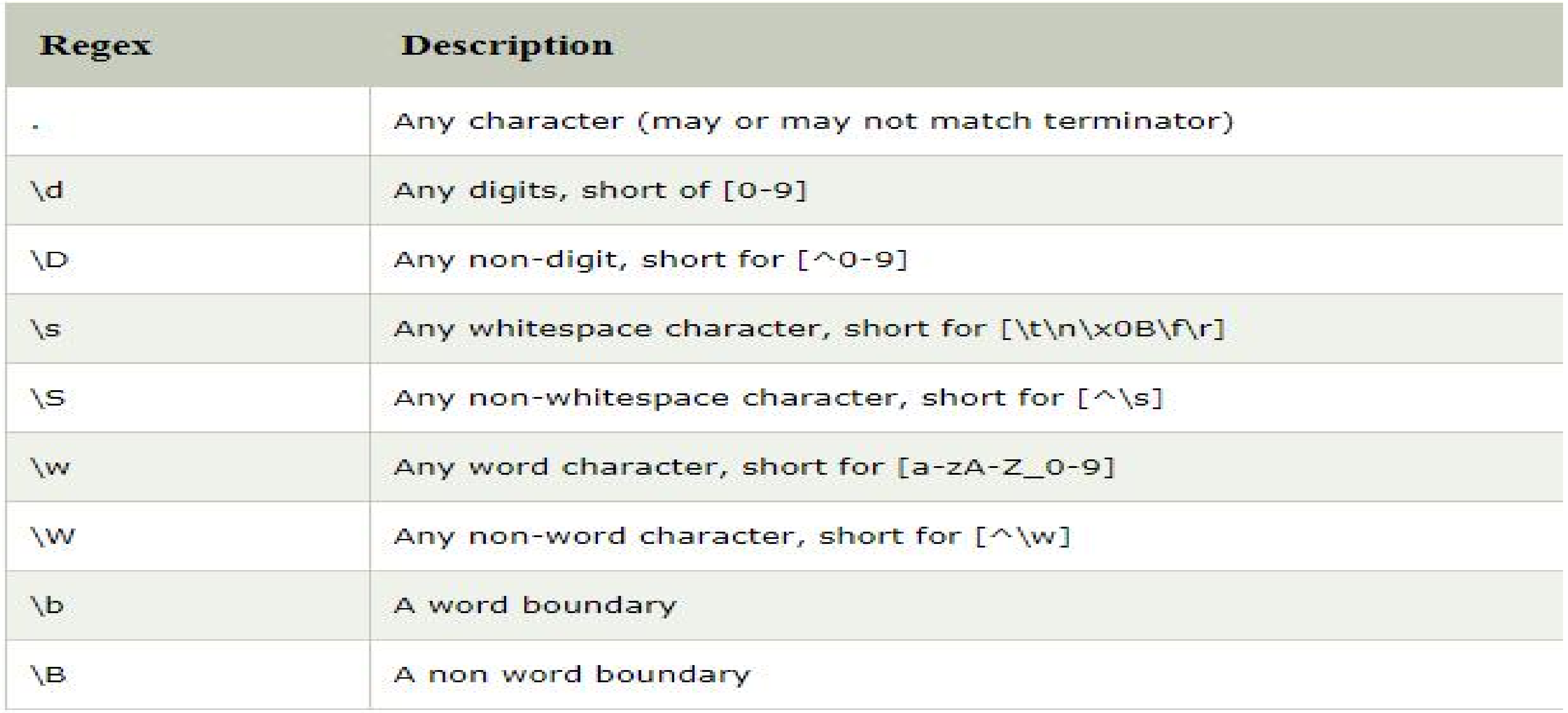
Regex Quantifiers

- The quantifiers specify the number of occurrences of a character.



Regex	Description
X?	X occurs once or not at all
X+	X occurs once or more times
X*	X occurs zero or more times
X{n}	X occurs n times only
X{n,}	X occurs n or more times
X{y,z}	X occurs at least y times but less than z times

Predefined Character Class (Regex Metacharacters)



Regex	Description
.	Any character (may or may not match terminator)
\d	Any digits, short of [0-9]
\D	Any non-digit, short for [^0-9]
\s	Any whitespace character, short for [\t\n\x0B\f\r]
\S	Any non-whitespace character, short for [^\s]
\w	Any word character, short for [a-zA-Z_0-9]
\W	Any non-word character, short for [^\w]
\b	A word boundary
\B	A non word boundary

The predefined character classes do not have to be enclosed in square brackets, but you can if you want to combine them.
\b matches any digit character
[\b\s] matches any digit or any white space character.

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.

Regular Expression Examples

Expression	Description
[wb](ad|eed)	Matches wad,weed,bad, and beed.
(pro|anti)-OO?	Matches pro-OOP and anti-OOP.
(AZ|CA|CO)[0–9]{4}	Matches AZxxxx,CAxxxx, and COxxxx, where x is a single digit.

Example

```
import java.util.*;
class MatchJavaIdentifier {
private static final String STOP =
"STOP";
private static final String VALID =
"Valid Java identifier";
private static final String INVALID =
"Not a valid Java identifier";


private static final String
VALID_IDENTIFIER_PATTERN = "[a-zA-
Z][a-zA-Z0-9_$]*";
```

```
public static void main (String[] args) {
Scanner scanner = new Scanner (System.in);
String str, reply;
while (true) {
    System.out.print ("Identifier: ");
    str = scanner.next( );
    if (str.equals(STOP)) break;
    if
    (str.matches(VALID_IDENTIFIER_PATTERN)) {
        reply = VALID;
    } else {
        reply = INVALID;
    }
    System.out.println(str + ": " + reply +
    "\n");
} }
}
```

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.**
- See the sample programs Ch9MatchJavaIdentifierPM on Page 539 and Ch9CountJavaPM on Page 540

Example

```
import java.util.*;
import java.util.regex.*;
class MatchJavaIdentifierPM {
private static final String STOP =
"STOP";
private static final String VALID =
"Valid Java identifier";
private static final String INVALID =
"Not a valid Java identifier";

private static final String
VALID_IDENTIFIER_PATTERN = "[a-zA-
Z][a-zA-Z0-9_$]*";
```

```
Scanner scanner = new Scanner(System.in);
String str, reply;
Matcher matcher;
Pattern pattern =
Pattern.compile(VALID_IDENTIFIER_PATTERN);
while (true) {
    System.out.print("Identifier: ");
    str = scanner.next();
    if (str.equals(STOP)) break;
    matcher = pattern.matcher(str);
    if (matcher.matches()) {
        reply = VALID;
    } else {
        reply = INVALID;
    }
    System.out.println(str + ": " + reply + "\n");
}}
```

Three ways to write Java Regex

(1)
```
Pattern p= Pattern.compile(".b");   // . Means one character
Matcher m=p.matcher("ab");
boolean result = m.matches();
System.out.println(result);                    // true
```

(2)
```
boolean result1= Pattern.compile(".b").matcher("ab").matches();
System.out.println(result1);                 // true
```

(3)
```
boolean result2= Pattern.matches(".b", "ab");
System.out.println(result2);                 // true
```

Split() method Example (Pattern class)

- To split a text into multiple strings based on a delimiter, we can use Pattern.split() method

```
String text3="redisyellowIsgreenisblue";

Pattern p1=Pattern.compile("is", Pattern.CASE_INSENSITIVE);

String [] result=p1.split(text3);

for(String s: result)
{
      System.out.print(s);
}
```

```
red  yellow green blue
```

Find out multiple occurrences of Pattern (Pattern and Matcher Class)

```
String text4="AABBCAAADEEAA";
Pattern p2=Pattern.compile("AA");
Matcher m2=p2.matcher(text4);
while(m2.find())
{
System.out.println("Found at : "+m2.start()+" - "+m2.end());
}
```

```
Found at : 0 - 2
Found at : 5 - 7
Found at : 11 - 13
```

replaceAll (String replacement) (Matcher Class)

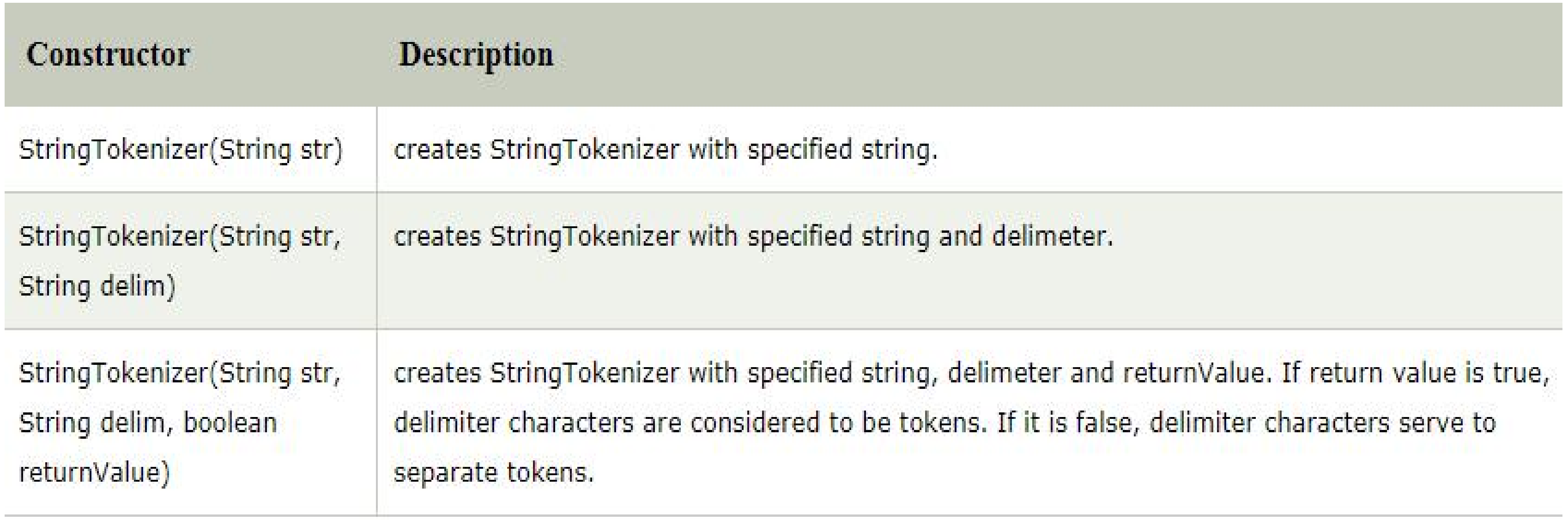
Replacing multiple spaces into single space.
It covers tab, new line, any kind of spaces replaces with single space.

```
Pattern p3=Pattern.compile("\\s+");
Matcher m3=p3.matcher("This   is my first  time");
String newString=m3.replaceAll(" ");
System.out.println("New String ="+newString);
```

```
New String = This is my first time
```

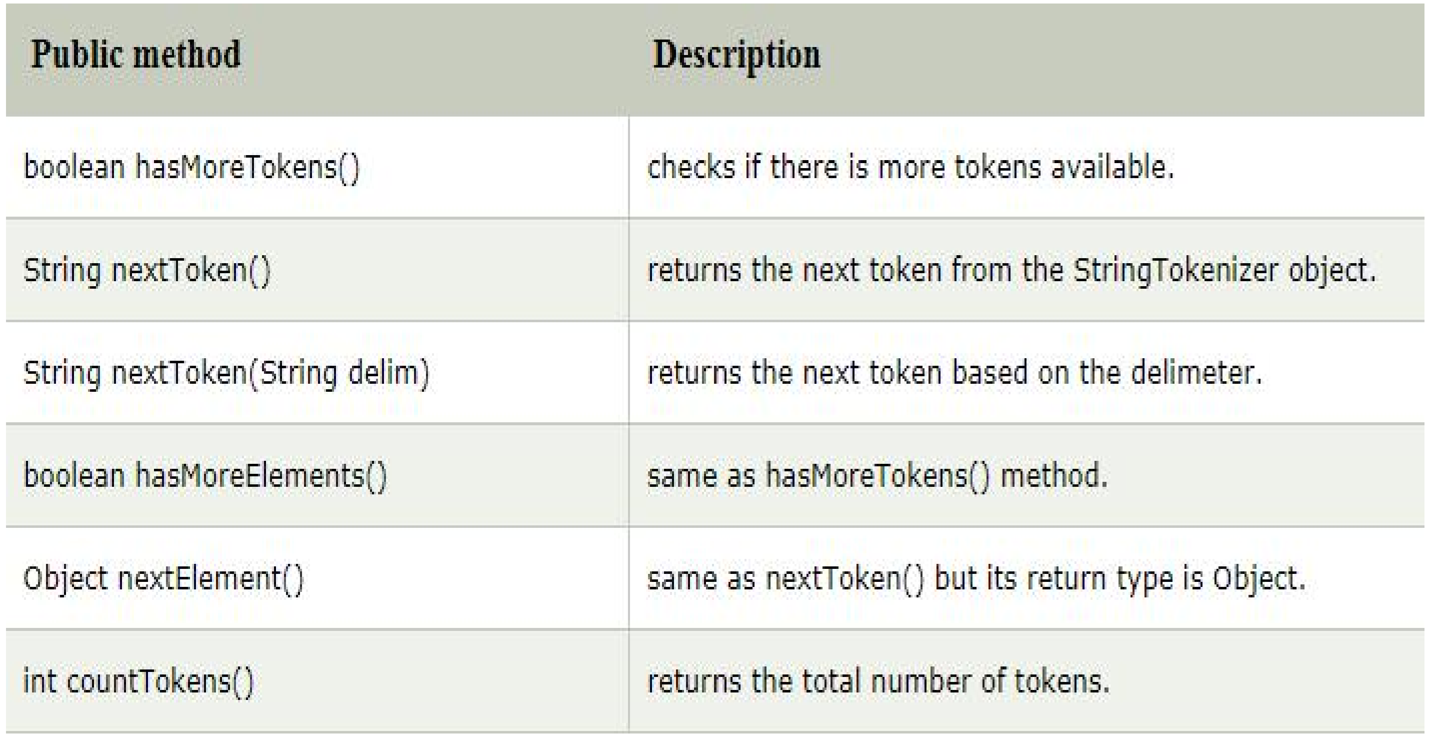
StringTokenizer

- **java.util.StringTokenizer** class allows you to break a string into tokens
- There are 3 constructors defined in the StringTokenizer class



Constructor	Description
StringTokenizer(String str)	creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	creates StringTokenizer with specified string and delimeter.
StringTokenizer(String str, String delim, boolean returnValue)	creates StringTokenizer with specified string, delimeter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of StringTokenizer class



Public method	Description
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.
String nextToken(String delim)	returns the next token based on the delimeter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

Examples

```
StringTokenizer st = new StringTokenizer("My name is Vasudev Adhikari."," ");
StringTokenizer st1 = new StringTokenizer("You can call me Moe Thiha as well.","a");
    while (st.hasMoreTokens()) {
       System.out.print(st.nextToken() + “,”);
     }
     while (st1.hasMoreTokens()) {
          System.out.println(st1.nextToken() + “-”);
     }
     // the output will be My,name,is,Vasudev,Adhiakri.,
     // You c-n c-ll me Moe Thih- -s well.
```

Exercises

4, 10, 11, 12, 19, 20 from reference book