Chapter (4.4, 10.10, 10.11)

Characters and Strings

Dr. Asem Kitana
Dr. Abdallah Karakra

The String Type

The char type only represents **one character**. To represent a string of characters, use the data type called **String**. For example,

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

The String type is not a primitive type. It is known as **a *reference type*.**

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.

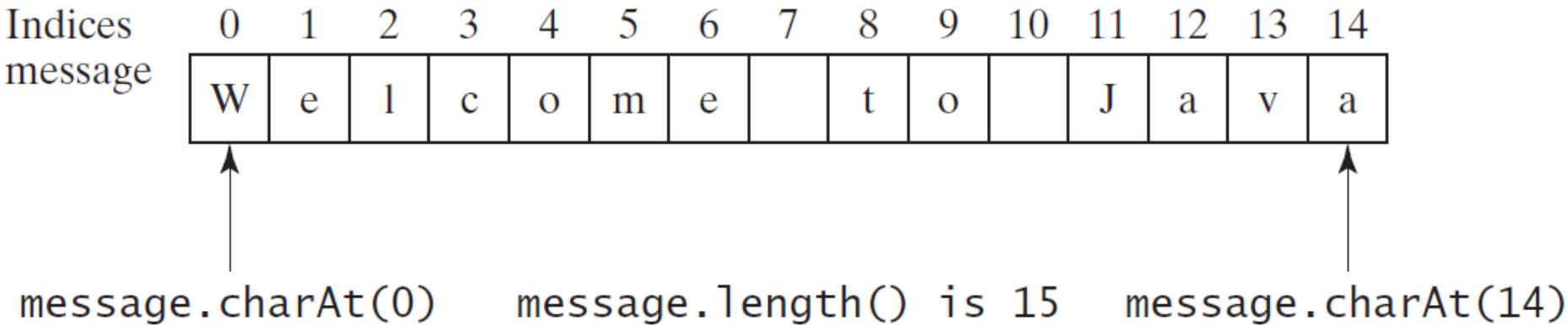
```
String s="Welcome to java";
String s1="Hi All";

System.out.println(s.length());//15
System.out.println(s.charAt(0));//W
System.out.println(s.charAt(2));//l
System.out.println(s.concat(s1));//Welcome to javaHi All
System.out.println((s.concat(" ")).concat(s1));//Welcome to java Hi All
System.out.println(s.toUpperCase());// WELCOME TO JAVA
System.out.println(s.toLowerCase());//welcome to java
System.out.println("    Hello All    ".trim());//Hello All
```

Getting String Length

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

The length of Welcome to Java is 15

Getting Characters from a String



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

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

The first character in message is W

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
```

concat

```
String str1= "Welcome ";
String str2="Comp 231";
String str3;
System.out.println(str1);
str1.concat(str2);
System.out.println(str1);
str3= str1.concat(str2);
System.out.println(str3);
```

Welcome

Welcome

Welcome Comp 231

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);
```

next() : Reads a string that ends before a whitespace character.
nextLine() : Reads a line of text (i.e., a string ending with the Enter key pressed).

Reading a String from the Console

```
Enter three words separated by spaces: Welcome to Java ↵Enter
s1 is Welcome
s2 is to
s3 is Java
```

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a line: ");
String s = input.nextLine();
System.out.println("The line entered is " + s);
```

```
Enter a line: Welcome to Java ↵Enter
The line entered is Welcome to Java
```

Example

```java
import java.util.Scanner;
public class Test{

    public static void main(String []args){

        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);


    }
}
```

```
Enter three words separated by spaces: hello   comp  231
s1 is hello
s2 is comp
s3 is 231
```

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);
```

```
Enter a character: Welcome
The character entered is W
```

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.

Testing String Equality

```
if (string1.equals(string2))
  System.out.println("string1 and string2 have the same contents");
else
  System.out.println("string1 and string2 are not equal");
```

For example, the following statements display `true` and then `false`.

```
String s1 = "Welcome to Java";
String s2 = "Welcome to Java";
String s3 = "Welcome to C++";
System.out.println(s1.equals(s2)); // true
System.out.println(s1.equals(s3)); // false
```

Testing String Equality

- **"str1==str2"** on String objects compares memory addresses, not the contents
- Always use **"str1.equals(str2)"** to compare contents

compareTo

```
String str1=new String("Hello World!");
String str2=new String("Hello World!");
String str3=new String("bad");
String str4= new String ("bd");
int res=str1.compareTo(str2);
int res2=str3.compareTo(str4);
System.out.println(res); // 0
System.out.println(res2); // -3
```

stratsWith

```
String str1=new String("Hello World!");
String str2=new String("Hello World!");

boolean res=str1.startsWith("HE");
boolean res2=str1.startsWith("He");
boolean res3=str1.startsWith("H");
System.out.println(res + " " + res2 + " " + res3);
```

false true true

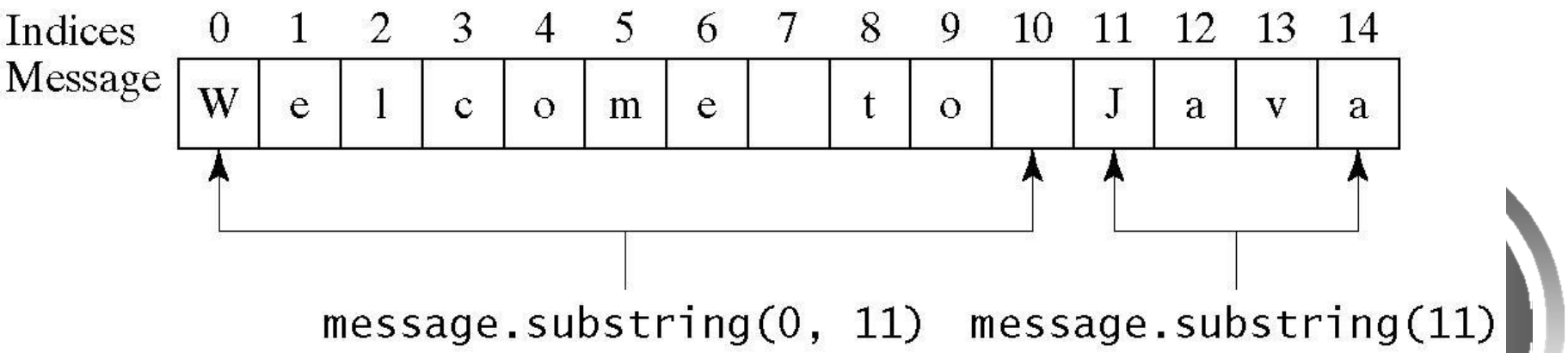
endsWith

```
String str1=new String("Hello World!");
String str2=new String("Hello World!");

boolean res=str1.endsWith("World");
boolean res2=str1.endsWith("ld");
boolean res3=str1.endsWith("ld!");
System.out.println(res + " " + res2 + " " + res3);
```

false false true

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,
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, Note that the character at `endIndex` is not part of the substring.



substring

```
String str1=new String("Hello World!");

String res=str1.substring(3,7); // lo W
String res2=str1.substring(2,5); //llo
String res3=str1.substring(2); //llo World!
String res4=str1.substring(8);//rld!
```

Examples: substring

```
String text = "Espresso";
```

`text.substring(6,8)`	→	`"so"`
`text.substring(0,8)`	→	`"Espresso"`
`text.substring(1,5)`	→	`"spre"`
`text.substring(3,3)`	→	`""`
`text.substring(4,2)`	→	`error`

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.

Examples: indexOf

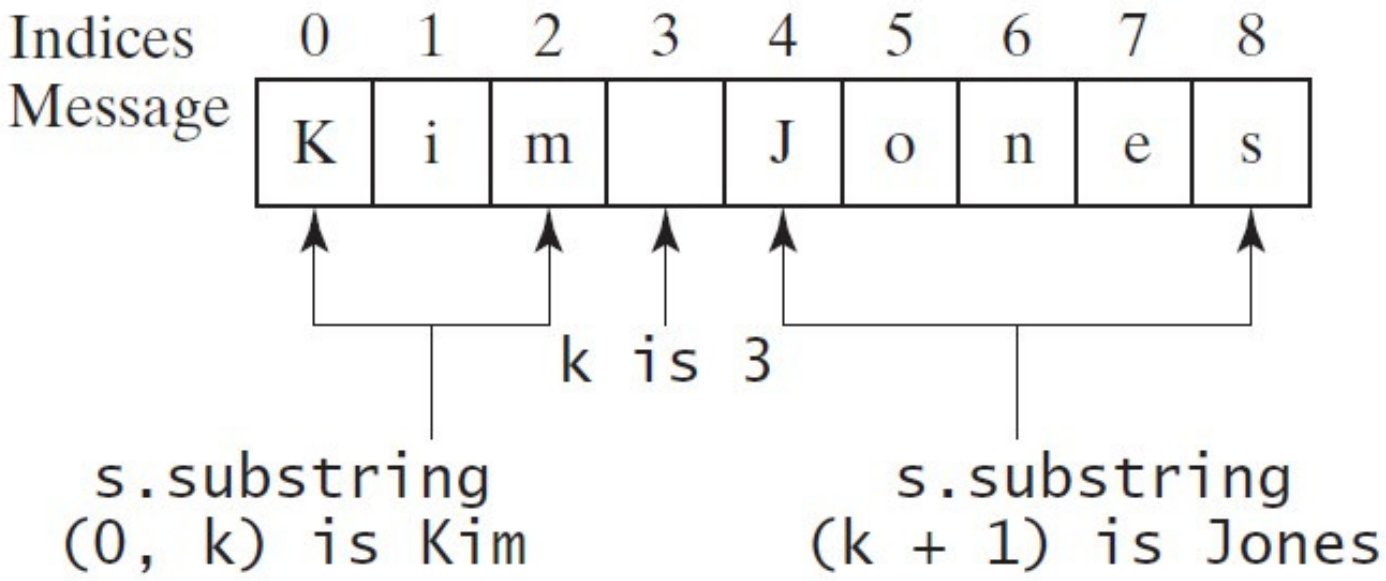
```
String str;
str = "I Love Java and Java loves me." ;
```

3 7 21

```
str.indexOf( "J" )          → 7
str2.indexOf( "love" )      → 21
str3. indexOf( "ove" )      → 3
str4. indexOf( "Me" )       → -1
```

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);
```



Examples

```
String str = "Hello, Comp 231";
System.out.println(str.indexOf("231")); //12
System.out.println(str.indexOf('l',1)); //2
System.out.println(str.indexOf('l',2)); //2
System.out.println(str.indexOf('o',5)); //8
System.out.println(str.indexOf('o',4));//4
System.out.println(str.indexOf("Comp",4));//7
System.out.println(str.indexOf("hello"));//-1
System.out.println(str.indexOf("comp",4));//-1
System.out.println(str.lastIndexOf("Comp"));//7
System.out.println(str.lastIndexOf('o'));//8
System.out.println(str.lastIndexOf('l'));//3
System.out.println(str.lastIndexOf('l',7));//3
System.out.println(str.lastIndexOf(' ',8));//6
System.out.println(str.lastIndexOf(' '));//11
```

Conversion between Strings and Numbers

You can convert a numeric string into a number. To convert a string into an `int` value, use the `Integer.parseInt` method, as follows:

```
int intValue = Integer.parseInt(intString);
```

where `intString` is a numeric string such as `"123"`.

To convert a string into a `double` value, use the `Double.parseDouble` method, as follows:

```
double doubleValue = Double.parseDouble(doubleString);
```

where `doubleString` is a numeric string such as `"123.45"`.

If the string is not a numeric string, the conversion would cause a runtime error. The `Integer` and `Double` classes are both included in the `java.lang` package, and thus they are automatically imported.

You can convert a number into a string, simply use the string concatenating operator as follows:

```
String s = number + "";
```

Examples

```
System.out.println(Integer.parseInt("123")); //123

System.out.println(Double.parseDouble("123.98"));//123.98

System.out.println(Double.parseDouble("123.98h"));
```

java.lang.NumberFormatExceptio

```
System.out.println(Double.parseDouble("123")); // 123.0

System.out.println(Integer.parseInt("123.4"));
```

java.lang.NumberFormatException

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";
```

Constructing Strings

Also a string can be created from an array of characters.

For example, the following statements create the string **"Good Day"**:

```
char[] charArray = {'G', 'o', 'o', 'd', ' ', 'D', 'a', 'y'};
String message = new String(charArray);
```

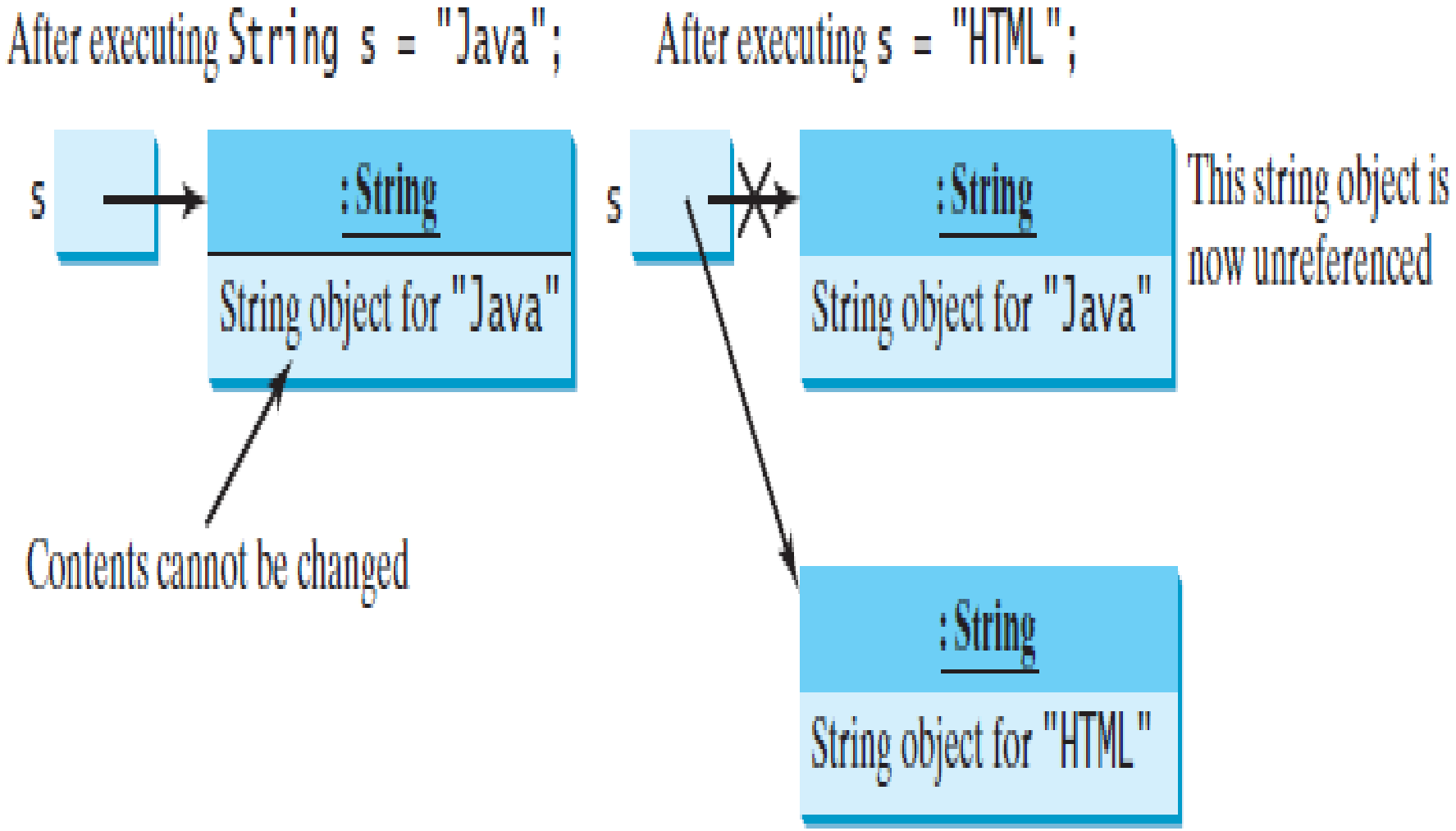
Strings Are Immutable

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";
```

Strings Are Immutable

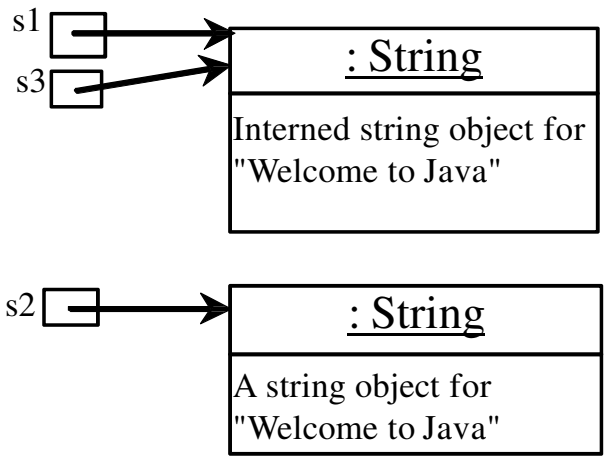


Interned Strings

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*.

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.

Check Point

```
public class HelloWorld {
  public static void main(String[] args) {
  String m="hello";
  String n="hello";
  n=n+"1";
  System.out.println(m);
  System.out.println(n);
  }
}
```

// find the output

```
hello
hello1
```

Replacing and Splitting Strings

"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.

```
String s="Espresso";
s.replace('s','l');
System.out.println(s);//Espresso
String k=s.replace('s','l');
System.out.println(k);//Elprello
```

Splitting a String

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

Output

Java

HTML

Perl

```java

public class Split1 {

    public static void main(String[] args) {
        String[] tokens = "Java#HTML#Perl".split("#", 1);
        for (int i = 0; i < tokens.length; i++)
            System.out.println(tokens[i]);

    }

}

```

Problems | Javadoc | Declaration | Console

<terminated> Split1 [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Jan 3, 2023, 4:39:32 AM – 4:39:33

```
Java#HTML#Perl
```

```java

public class Split1 {

    public static void main(String[] args) {
        String[] tokens = "Java#HTML#Perl".split("#", 2);
        for (int i = 0; i < tokens.length; i++)
            System.out.println(tokens[i]);

    }

}

```

Problems Javadoc Declaration Console

<terminated> Split1 [Java Application] C:\Program Files\Java\jdk-17.0.5\bin\javaw.exe (Jan 3, 2023, 4:41:18 AM – 4:41:19 AM

```
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* (regex).

```
"Java".matches("Java");   //true
"Java".matches("java");   //false
"Java".equals("Java");   //true

"Java  is  fun".matches("Java.*");  //true
"Java  is  cool".matches("Java.*");// true
"Java is cool".matches("java.*");// false
"Hello hi bye".matches("hi.*");//false
```

Matches Example

```
public class TestMatches{

    public static void main(String args[]) {
      String Str = new String("Welcome to Tutorialspoint.com");

      System.out.print("Return Value :" );
      System.out.println(Str.matches(".*Tutorials.*"));

      System.out.print("Return Value :" );
      System.out.println(Str.matches("Tutorials"));

      System.out.print("Return Value :" );
      System.out.println(Str.matches("Welcome.*"));
    }
}
```

```
Return Value :true
Return Value :false
Return Value :true
```

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);  //aNNNbNNNNNNc
```

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]);
```

Output

```
Java
C
C#
C++
```

Example

```
class StringSplit {
    public static void main(String args[]) {
        String s = "Apple-Banana-Orange-Water-Watermelon";
        String fruits[] = s.split("-"); // parse strings in between the dash character
        for(String temp : fruits ) {
            System.out.println(temp); // display array elements to the console
        }

        System.out.println("\n----Max 3----");
        fruits = s.split("-", 3); // parse up to 3 strings in between the dash character
        for(String temp : fruits ) {
            System.out.println(temp); // display array elements to the console
        }

    }
}
```

```
Apple
Banana
Orange
Water
Watermelon
```

```
----Max 3----
```

```
Apple
Banana
Orange-Water-Watermelon
```

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'.

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 Class

```
public class Test{

    public static void main(String []args){
        Character ch1 = new Character('b');
        Character ch2 =new Character ('a');
        char myChar= ch1.charValue();
        int res= ch1.compareTo(ch2);
        boolean res1= ch1.equals (ch2);
        boolean res2 = Character.isDigit ('4');
        boolean res3 = Character.isLetter ('4');
        boolean res4 = Character.isUpperCase ('A');
        char c = Character.toLowerCase('A');
        char c2 = Character.toUpperCase('a');
        System.out.println(myChar);  //b
        System.out.println(res); // 1
        System.out.println(res1); // false
        System.out.println(res2); // true
        System.out.println(res3); // false
        System.out.println(res4); // true
        System.out.println(c); // a
        System.out.println(c2); // A
    }
}
```

Examples : Characters class

Character c = new Character('b');

c.compareTo(new Character('a'))	returns **1**
c.compareTo(new Character('b'))	returns **0**
c.compareTo(new Character('c'))	returns **-1**
c.compareTo(new Character('d')	returns **-2**
c.equals(new Character('b'))	returns **true**
c.equals(new Character('d'))	returns **false**

`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. (immutable)

`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.

StringBuilder Constructors

```
StringBuilder sb  = new StringBuilder();            //(default capacity 16)
StringBuilder sb2 = new StringBuilder(10);         //(capacity 10)
StringBuilder sb3 = new StringBuilder("Java");     //(default 16+4)
StringBuilder sb4 = new StringBuilder("Welcome"); // (default 16 + 7)

System.out.println("Capacity = " + sb.capacity()); //16
System.out.println("Length = " + sb.length()); //0

System.out.println("Capacity = " + sb2.capacity()); //10
System.out.println("Length = " + sb2.length()); //0

System.out.println("Capacity = " + sb3.capacity()); //20
System.out.println("Length = " + sb3.length()); //4

System.out.println("Capacity = " + sb4.capacity()); //23
System.out.println("Length = " + sb4.length()); //7
```

Modifying Strings in the StringBuilder

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-1.
+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 into 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-1 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 sb = new StringBuilder(10);
```

0	1	2	3	4	5	6	7	8	9

```
sb.append("Hello...");
```

0	1	2	3	4	5	6	7	8	9
H	e	l	l	o	.	.	.		

```
sb.append('!');
```

0	1	2	3	4	5	6	7	8	9
H	e	l	l	o	.	.	.	!	

```
sb.insert(8," Java");
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
H	e	l	l	o	.	.	.		J	a	v	a	!	

```
sb.delete(5,8);
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
H	e	l	l	o		J	a	v	a	!				

Examples

```
public class TestStringBuilder{

    public static void main(String []args){
        StringBuilder sb = new StringBuilder("Welcome to ");
        System.out.println (sb);//Welcome to
        sb.append("Java");
        System.out.println (sb);//Welcome to Java
        sb.insert(11, "HTML and ");
        System.out.println (sb);//Welcome to HTML and Java
        sb.delete(8, 11) ;
        System.out.println (sb);//Welcome HTML and Java
        sb.deleteCharAt(8) ;
        System.out.println (sb);//Welcome TML and Java
        sb.reverse() ;
        System.out.println (sb);//avaJ dna LMT emocleW
        sb.replace(11, 15, "HTML") ;
        System.out.println (sb);//avaJ dna LMHTMLocleW
        sb.setCharAt(0, 'w') ;
        System.out.println (sb);//wvaJ dna LMHTMLocleW
    }
}
```

Example

```
public class Example{

    public static void main(String []args){

        StringBuilder stb = new StringBuilder("hi");
        char x[]={'a','b','c',' '};
        char y[]={'z','n','m','w'};
        stb.append(x);
        System.out.println(stb);//hiabc
        stb.append(y,2,2);
        System.out.println(stb);//hiabc mw
        stb.append(5);
         System.out.println(stb);//hiabc mw5
         stb.insert(2,y,2,1);
          System.out.println(stb);//himabc mw5
    }
}
```

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.

Example

```
public class TestStringBuilder{

    public static void main(String []args){
        StringBuilder sb = new StringBuilder("Welcome to ");
        System.out.println (sb.toString());//Welcome to
        System.out.println (sb.capacity());//27
        System.out.println (sb.length());//11
        System.out.println (sb.charAt(0));//W
        System.out.println (sb.charAt(5));//m
        sb.setLength(15);
        System.out.println (sb.length());// 15
        System.out.println (sb.charAt(13));//" "
        System.out.println (sb.substring(1));//elcome to
        System.out.println (sb.substring(8));//to
        System.out.println (sb.substring(1,6));//elcom
        System.out.println (sb.substring(0,7));//Welcome
        sb.trimToSize();
        System.out.println (sb.capacity());//15
        System.out.println (sb.length());//15
    }
}
```

Note

- The **capacity()** is the number of characters the string builder is able to store without having to increase its size.
- The **length()** method returns the number of characters actually stored in the string builder.
- The **trimToSize()** method is used to reduce the capacity to the actual size.

Check Point

```
//find output
public class Test {
  public static void main(String[] args) {
    String s = "Java";
    StringBuilder buffer = new StringBuilder(s);
    change(s);
    System.out.println(s);
  }

  private static void change(String s) {
    s = s + " and C";
  }
}
```

Java

Check Point

```
//Correct the error
public class Test {
  public static void main(String[] args) {
    String s = "Java";
    StringBuilder buffer = new StringBuilder(s);
    change(buffer);
    System.out.println(s);
  }

  private static void change(String s) {
    s = s + " and  C";
  }
}
```

s instead of buffer inside change method

Check Point

//find output

```
public class HelloWorld{
  public static void main(String[] args)  {
    StringBuilder strBuf = new StringBuilder("ABCDEFG");
    strBuf.delete(1, 4);
    System.out.println(strBuf);
  }
}
```

AEFG

Check Point

Suppose that s1, s2, s3, and s4 are four strings, given as follows:

```
String s1 = "Welcome to Java";
String s2 = s1;
String s3 = new String("Welcome to Java");
String s4 = "Welcome to Java";
```

What are the results of the following expressions?

```
a.      s1 == s2
b.      s1 == s3
c.      s1 == s4
d.      s1.equals(s3)
e.      s1.equals(s4)
f.      "Welcome to Java".replace("Java", "HTML")
g.      s1.replace('o', 'T')
h.      s1.replaceAll("o", "T")
i.      s1.replaceFirst("o", "T")
j.      s1.toCharArray()
```

a. true
b. false
c. true
d. true
e. true
f. Welcome to HTML
g. WelcTme tT Java
h. WelcTme tT Java
i. WelcTme to Java
j.toCharArray() returns an array of characters consisting of
W, e, l, c, o, m, e, , t, o, , J, a, v, a