

Strings and String Buffer Classes

Overview of Strings:

Strings represents a sequence of Characters, Strings can be represented using an array of characters.

Ex: `char a[]= new char [10];`

`String a= new String ("hello");`

1. In java, Strings are class objects and implemented using two class String and String Buffer.
2. A java String is an instantiated object of the String class.
3. String may be declared and created as follows:-

`String string_name;`

`String name=new String ("string");`

Examples:-

`String a;`

`a=new String ("hello");`

‘Or’

`String a= new String ("hello");`

4. The String, String Buffer and String Builder class are defined in java.lang.

String arrays:

We can also create and use arrays that contain Strings. The Statement

`String itemArray [] = new String [3];`

Will create an itemArray of size 3 to hold three String constants. We can assign the Strings to the item array element by element using three different statements or more efficiently using a for loop.

String manipulation operations:

The String class defines a no. of methods that allow us to accomplish a various String manipulations operation. Consider an example of two strings S1 & S2.

String S1="Hello";

String S2="hello";

Following are the various String manipulations methods:-

- **length ()**:- the length of a string is the number of characters that will contains.

To obtain this value.

Syntax:- Stringname.length();

Ex:- S1.length(); //returns 5;

- **charAt()** :- To extract a single character from a string. You can refer directly to an individual character via the charAt() method.

Syntax:- charAt(pos);

Here, pos is the index of the character that you want to obtain.

The charAt() returns the character at the specified location.

Example:- char ch= s1.charAt(1); //assigns e to ch

- **equals()**:- to compare two strings for equality, use equals().

Syntax:- boolean a=Stringname. equals (String str);

Example:- boolean a=s1.equals(s2); //returns false

→ It returns true if the String contain the same characters in the same order, & false otherwise.

→ The comparison is case Sensitive.

- **equalsIgnoreCase()**:- to perform comparison that ignore that ignore case difference call equalsIgnoreCase(). When it compare two string.

→ it consider A-Z to be the same as a-z

Syntax:- boolean a=Stringname. equalsIgnoreCase(String str);

Example:- boolean a=s1.equalsIgnoreCase (s2); //returns true

→ returns true if the strings contains some characters in the same order and false

Otherwise.

- **concat():** - you can concatenate two strings using concat().

Syntax:- String concat(String str);

→ This method creates a new object that contains the invoking string with the contents of str appended to the End.

Example:- String s1= "one";

String s2= s1.concat("two");

Puts the string "onetwo" into s2.

concat() performs the same function as +.

- **substring():** - you can extract a substring using substring() method.

Syntax:- String substring(int startIndex);

→ the Second form of substring allows you to specify both the beginning and index of Substring.

String substring(int startindex, int Endindex);

Here,

Startindex specifies the beginning index and End Index specifies the stopping point.

Example:- String s= "small aim is crime";

System.out.println(s.substring(7));

System.out.println(s.substring(3,9));

OUTPUT:-

im is crime

ll aim

- **trim():** - The trim() method returns a copy of the invoking String from which any leading and trailing whitespace has been removed.

Syntax:- String trim();

Ex:- String s= "Hello World".trim();

OUTPUT:-

HelloWorld

- **toLowerCase() and toUpperCase():**- the method toLowerCase() converts all the character in a string from uppercase to Lowercase

The method toUpperCase() converts all the character in a string from Lowercase to Uppercase.

Syntax:- String toUpperCase();

String toLowerCase();

Example:- String s= "This is a text";

System.out.println(s.toUpperCase());

System.out.println(s.toLowerCase());

Output:-

THIS IS A TEXT

this is a text

- **compareTo():**- The compareTo() method is used to compare Strings.

Syntax: s1.compareTo(s2);

Example:s1.compareTo(s2);

It will return negative if s1<s2, positive if s2<s1 and zero if s1 is equal to s2.

- **indexOf():**- The indexOf() method is used to search for the first occurrence of a character or Substring.

Syntax:- int indexOf(int ch);

Here, ch is the character being searched (index)

→To, search for the first occurrence of a substring.

int indexOf(int str);

you can specify the string point for the search using this form.

int indexOf(int ch, int startIndex());

Example:

```
String s= "Now is the time for all good";
```

```
System.out.println(s.indexOf('t'));
```

```
System.out.println(s.index("time"));
```

```
System.out.println(s.indexOf('t',10));
```

OUTPUT:-

7

11

11

- **lastIndexOf():-** the lastindexOf method is used to search for the last occurrence of a character or substring.

Syntax: - int lastIndexOf(int ch);

→to search for the last occurrence of a substring

int lastIndexOf(String ch);

you can specify a starting point for the search.

int lastIndexOf (int ch, int startindex);

Example:-

```
String s= "Now is the Time for all the good";
```

```
System.out.println(s.lastIndexOf("the"));
```

```
System.out.println(s.lastIndexOf('t',10));
```

Output:-

7

7

- **replace():** - the replace method replace all occurrence of one character in the invoking string with another character.

Syntax:-

String replace(char original, char replacement);

Here, original specifies the character to be replaced by the character specified by replacement.

Example:- String s= "Hello".replace('l', 'w');

Output:- Hewwo.

/*Program to Sort String

import java.util.;*

class StringSort

{

public static void main(String args[])

{

int n;

String temp;

System.out.println("Enter the number of names:");

Scanner s=new Scanner(System.in);

n=s.nextInt();

String names[]=new String[n];

System.out.println("Enter the Names you Desire:");

for(int i=0;i<n;i++)

{

names[i]=s.next();

}

//Sorting

```
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(names[i].compareTo(names[j])<0)
                {
                    temp=names[i];
                    names[i]=names[j];
                    names[j]=temp;
                }
            }
        }
        System.out.println("The Sorted String of name is:");
        for(int i=0;i<n;i++)
        {
            System.out.println(names[i]);
        }
    }
}
```

Output:-

Enter the number of names:

4

Enter the Names you Desire:

naruto

sasuke

hinata

kurama

The Sorted String of name is:

hinata

kurama

naruto

sasuke

String buffer class

1. String buffer is a peer class of String which creates Strings to flexible length that can be modified in terms of both length and content.
2. String created using String buffer class can modified by inserting characters 'or' substrings int the middle of a string 'or' append another string to the end.
3. Java manipulate string buffer behind the scene by using the overloaded + operator.

String buffer methods:-

String buffer class supports all the methods of string class with the following conditional methods.

- **setCharAt():**-You can set the value of a character within a string buffer using setCharAt().

Syntax:- void setCharAt(int where, char ch);

Here, where specifies the index of character being set and ch specifies the new value of that characters.

Example:-

```
StringBuffer sb=new StringBuffer("Hello");
```

```
Sb.setCharAt(1,'i');
```

```
System.out.println(sb);
```

Output:-

Hillo

- **append():**- the append method concatenate the string representation of any other type of data to the end of the invoking string buffer object.

Syntax:-

```
StringBuffer append(String str);
```

Example:-

```
StringBuffer sb= new StringBuffer("Hiii");  
StringBuffer s;  
StringBuffer a= new StringBuffer(" John");  
s=sb.append(a);  
System.out.println(s);
```

Output:-

Hiii John

→The append() method is most often called when the + operator is used on string objects.

- **insert():-** the insert() method inserts one String into another.

Syntax:- StringBuffer insert(int index, String str);

Here, index specifies the index of at which point the String will be inserted into the invoking String Buffer Objects.

Example:-

```
StringBuffer sb=new StringBuffer("I Java");  
  
sb.insert(2, "Like");  
System.out.println(sb);
```

Output:-

I Like Java

- **delete() and deleteCharAt():-** you can delete character within a stringBuffer by using the method delete() and deleteCharAt().

Syntax:-

```
StringBuffer delete (int startindex, int endindex);
```

```
StringBuffer deleteCharAt(int 100);
```

Example:-

```
StringBuffer s= new StringBuffer("This is a List");
```

```
s.delete(4,7);
```

```
System.out.println(s);
```

```
s.deleteCharAt(0);
```

```
System.out.println(s);
```

Output:-

```
This a List
```

```
his a List
```

<i>String class</i>	<i>String Buffer class</i>
1. Creates objects which are of fixed length.	1. String objects are of flexible length
2. Can't be modified both in terms of length and content.	2. Can be modified in terms of length and content.
3. Strings values are resolved at run time.	3. String Buffer values are resolved at compile time.
4. String is not thread safe.	4. String Buffer is thread Safe.
5. We can't append a new String.	5. We can append a new String.

- **setLength():** - To set the length of the String within a string buffer object uses setLength().

Syntax:- void setLength(int len);

Here, len specifies the length of the string.

Example:-

```
S1.setLength(10);
```

if $n < S1.length()$, S1 is truncated

if $n > S1.length()$, null characters are added to end.

- **reverse():-** you can reverse the characters within a string buffer Object using reverse().

Syntax:- StringBuffer reverse();

Example:-

```
StringBuffer s=new StringBuffer("abcde");
```

```
System.out.println(c);
```

```
s.reverse();
```

```
System.out.println(s);
```

Output:-

abcde

ebcda

/*Program to implement String and StringBuffer class methods*/

class S

{

public static void main(String args[])

{

String s1="hello world";

System.out.println("illustration of string class and its method\n");

System.out.println("length of s1="+s1.length());

System.out.println("Convert s1 to to uppercase="+s1.toUpperCase());

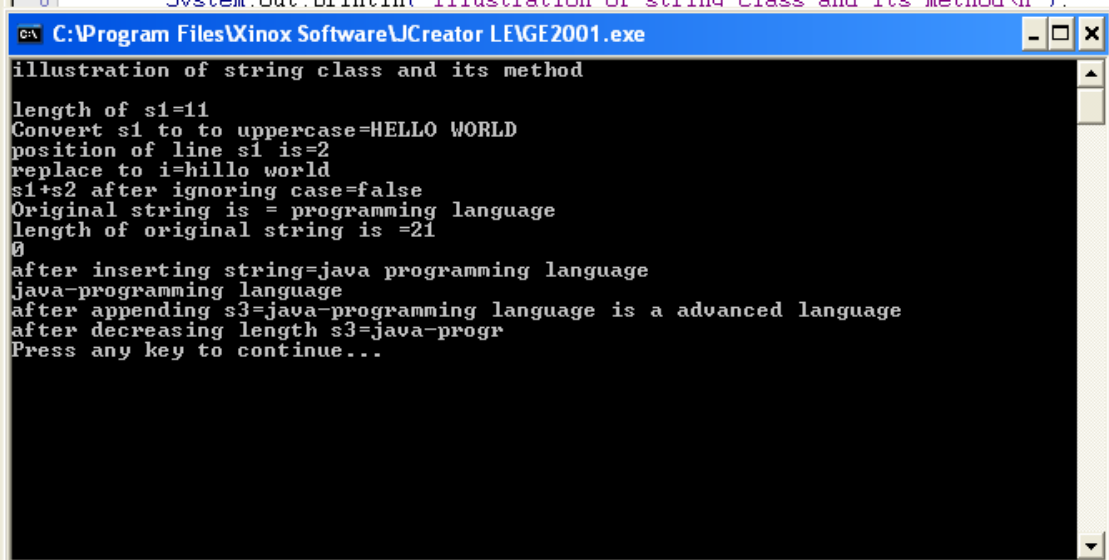
System.out.println("position of line s1 is="+s1.indexOf('l'));

System.out.println("replace to i="+s1.replace('e','i'));

String s2="hillo world";

System.out.println("s1+s2 after ignoring case="+s1.equalsIgnoreCase(s2));

```
StringBuffer s3=new StringBuffer(" programming language");
System.out.println("Original string is "+s3);
System.out.println("length of original string is "+s3.length());
int i=s3.indexOf(" programming");
s3.insert(i,"java");
System.out.println(i);
System.out.println("after inserting string="+s3);
s3.setCharAt(4,'-');
System.out.println(s3);
s3.append(" is a advanced language");
System.out.println("after appending s3="+s3);
s3.setLength(10);
System.out.println("after decreasing length s3="+s3);
}
}
```



```
illustration of string class and its method
length of s1=11
Convert s1 to to uppercase=HELLO WORLD
position of line s1 is=2
replace to i=hillo world
s1+s2 after ignoring case=false
Original string is = programming language
length of original string is =21
0
after inserting string=java programming language
java-programming language
after appending s3=java-programming language is a advanced language
after decreasing length s3=java-progr
Press any key to continue...
```

Vectors:

1. The vector is a built in class of java present in java utility (util) package.
2. A class can be used to create a dynamic array known as vector that can hold objects of any type and any number.

Syntax:- `Vector v=new Vector();`

3. Vector can be created with size
`Vector v=new Vector(7);`

Advantages of Vector

1. It is convenient to use vector to store objects.
2. A vector can be used to share a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

Disadvantages of Vector

1. Vector can't store simple datatypes directly
2. The simple datatype need to be converted to objects to store them in vectors. Hence the concept of 'wrapper class' are used to convert simple datatypes to objects.

Vectors methods

The Vector class supports a no. of methods that can be used to manipulate the vectors created.

The important vector methods are

`v.addElement(pos);`

→add the item given to v at the end.

`v.elementAt(pos);`

→gives the name of the item at specified position.

`v.size();`

→gives the no.of objects present in v

`v.removeElement(item);`
→remove the specified item from v

`v.removeAllElements();`
→removes all the elements from v

`v.removeElementAt();`
→removes the items stored at nth position.

`v.copyInto(array);`
→copies all items from v to array.

`v.insertElementAt(item,n);`
→insert the item at nth position.

`v.lastElement();`
→returns the last element in the vector.

/*A java program to implement vector class and its method.*/

```
import java.util.*;

class Vec
{
public static void main(String args[])
{
    Vector v=new Vector();
    String s1[]=new String[3];
    s1[0]="hi";
    s1[1]="hello";
    s1[2]="you";
    for(int i=0;i<s1.length;i++)
    {
        v.addElement(s1[i]);
    }
    System.out.println("Elements in vector are");
}
```

```
for(int j=0;j<v.size();j++)
System.out.println(v.elementAt(j));
System.out.println("want to insert String");
v.insertElementAt("how are",2);
System.out.println("After inserting, Vector are:");
for(int k=0;k<v.size();k++)
System.out.println(v.elementAt(k));
System.out.println("No.of objects in vector are");
System.out.println(v.size());
int k=v.size();
String s2[]=new String[k];
v.copyInto(s2);
System.out.println("Copied Element are");
for(int l=0;l<s2.length;l++)
System.out.println(s2[l]);
}
}
```

Output:-

Elements in vector are
hi
hello
you
want to insert String
After inserting, Vector are:
hi
hello
how are
you
No.of objects in vector are

4

Copied Element are

hi

hello

how are

you

Wrapper class:

1. Wrapper class are the built in class of java present in java.lang package.
2. Wrapper class can be used to convert primitive datatypes like int, float, long, char and double into objects.
3. Wrapper class concept can be used when we want to add simple datatypes to a vector.
4. The wrapper classes have a number of unique methods for handling primitive datatypes and objects.
5. The following are the simple datatypes & their corresponding wrapper clas

Simple datatypes	Wrapper class
int float double long boolean char short byte	Integer Float Double Long Boolean Character Short Byte

/*Sample Wrapper Class Program*/

import java.io.*;

class wrap

{

public static void main(String args[])

{

int a=10;

float b=6.6f;

byte c=20;

double d=6.5;

System.out.println("Value of Integer Variable a="+a);

Integer a1=Integer.valueOf(a);

System.out.println("Value of integer object a1="+a1);

System.out.println("\nValue of Float variable f="+b);

Float b1=Float.valueOf(b);

System.out.println("Value of Float object="+b1);

System.out.println("\nValue of Byte Variable c="+c);

Byte c1=Byte.valueOf(c);

System.out.println("Value of Byte object="+c1);

System.out.println("\nValue of Double Variable="+d);

Double d1=Double.valueOf(d);

System.out.println("Value of Double Object="+d1);

}

}

Output:-

Value of Integer Variable a=10

Value of integer object a1=10

Value of Float variable f=6.6

Value of Float object=6.6

Value of Byte Variable c=20

Value of Byte object=20

Value of Double Variable=6.5

Value of Double Object=6.5

Difference between Arrays and Vectors

Array	Vector
<ol style="list-style-type: none">1. It contains all the elements of similar data types.2. The size of array is fixed.3. We can Store simple elements directly.4. We can directly add the simple datatypes elements in the array	<ol style="list-style-type: none">1. It contains the Elements that may be varying Datatype.2. The size of array is varying.3. We can store simple objects in the database.4. We can't add directly to the simple datatype element in the vector.

Enumerated Data Types

Explain the use of enum with an example?

1. The enumerated data types can be denoted by the keyword '**enum**'.
2. The **enum** helps to define the user defined data type.
3. The value can also be assigned to the elements in the **enum** data type.

/* Progrm to illustrate simple Enumerated Data Types

class weekdays

{

enum Days

{

sunday,

tuesday,

wednesday,

thursday,

friday,

saturday

}

public static void main(String args[])

{

for(Days d:Days.values())

{

System.out.println(d);

}

}

}

Output

sunday

tuesday

wednesday

thursday

friday

saturday

/ Progrm to illustrate simple Enumerated Data Types*

class weekdays

{

enum Days

{

sunday,

tuesday,

wednesday,

thursday,

friday,

saturday

}

public static void main(String args[])

{

for(Days d:Days.values())

{

if(d.equals(Days.sunday))

System.out.println(d + "is holiday");

else

System.out.println(d + "is working day");

}

}

}

OUTPUT:

Sunday is holiday

Monday is working day

Tuesday is working day

Wednesday is working day

Thursday is working day

Friday is working day

Saturday is working day

ANNOTATIONS

1. An annotation is a mechanism for associating a meta-tag with program elements.
2. Annotation can be applied in two steps- firstly annotation type definition is specified and then the annotation itself must be defined.
3. For an annotation type definition an 'at' (@) sign is followed by the **interface** keyword which is then followed by **annotation name**.
4. For an annotation definition an 'at' (@), is followed by the annotation type.
5. The annotations are supported in java.lang package.

Syntax:

```
@interface annotationname  
{  
  
}
```

Example:

```
@interface unitTest  
{  
String value();  
}
```

6. The annotation can be accessed using @ followed by annotation_name.
7. Example:

```
@unitTest(value=10);
```

Java contains the following annotations:

1. @Deprecated
2. @Overrides

Java also contains few meta annotations:

1. @Documented
2. @Inherited
3. @Retention
4. @Target