**String Handling in Java**

The basic aim of **String Handling** concept is storing the string data in the main memory (RAM), manipulating the data of the String, retrieving the part of the String etc. **String Handling** provides a lot of concepts that can be performed on a string such as concatenation of string, comparison of string, find sub string etc.

**Character**

It is an identifier enclosed within single quotes (' ').  
Example: 'A', '$', 'p'

**String:**

String is a sequence of characters enclosed within double quotes (" ") is known as **String.**  
Example: "Java Programming".

In java programming to store the character data we have a fundamental datatype called **char**. Similarly to store the string data and to perform various operation on String data, we have three predefined classes they are:

* String
* StringBuffer
* StringBuilder

## String class

It is a predefined class in java.lang package can be used to handle the String. String class is **immutable** that means whose content can not be changed at the time of execution of program.

**String** class object is immutable that means when we create an object of String class it never changes in the existing object.  
Example:

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s=**new** String("java");

s.concat("software");

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

}

}

## Output

java

**Explanation:**Here we can not change the object of String class so output is only java not java software.

### Methods of String class

### length()

**length():**This method is used to get the number of character of any string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

**int** l;

String s=**new** String("Java");

l=s.length();

System.**out**.println("Length: "+l);

}

}

## Output

Length: 4

### charAt(index)

**charAt():**This method is used to get the character at a given index value.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

**char** c;

String s=**new** String("Java");

c=s.charAt(2);

System.**out**.println("Character: "+c);

}

}

## Output

Character: v

### toUpperCase()

**toUpperCase():**This method is use to convert lower case string into upper case.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="Java";

System.**out**.println("String: "+s.toUpperCase());

}

}

## Output

String: JAVA

### toLowerCase()

**toLowerCase():**This method is used to convert lower case string into upper case.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="JAVA";

System.**out**.println("String: "+s.toLowerCase());

}

}

## Output

String: java

### concat()

**concat():**This method is used to combined two string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="Raddy";

System.**out**.println("Combined String: "+s1.concat(s2));

}

}

## Output

Combined String: HiteshRaddy

### equals()

**equals():**This method is used to compare two strings, It return true if strings are same otherwise return false. It is case sensitive method.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="Raddy";

String s3="Hitesh";

System.**out**.println("Compare String: "+s1.equals(s2));

System.**out**.println("Compare String: "+s1.equals(s3));

}

}

## Output

Compare String: false

Compare String: true

### equalsIgnoreCase()

**equalsIgnoreCase():**This method is case insensitive method, It return true if the contents of both strings are same otherwise false.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="HITESH";

String s3="Raddy";

System.**out**.println("Compare String: "+s1.equalsIgnoreCase(s2));

System.**out**.println("Compare String: "+s1.equalsIgnoreCase(s3));

}

}

## Output

Compare String: true

Compare String: false

### compareTo()

**compareTo():**This method is used to compare two strings by taking unicode values, It return 0 if the string are same otherwise return +ve or -ve integer values.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="Raddy";

**int** i;

i=s1.compareTo(s2);

**if**(i==0)

{

System.**out**.println("Strings are same");

}

**else**

{

System.**out**.println("Strings are not same");

}

}

}

## Output

Strings are not same

### compareToIgnoreCase()

**compareToIgnoreCase():**This method is case insensitive method, which is used to compare two strings similar to compareTo().

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="HITESH";

**int** i;

i=s1.compareToIgnoreCase(s2);

**if**(i==0)

{

System.**out**.println("Strings are same");

}

**else**

{

System.**out**.println("Strings are not same");

}

}

}

## Output

Strings are same

### startsWith()

**startsWith():**This method return true if string is start with given another string, otherwise it returns false.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="Java is programming language";

System.**out**.println(s.startsWith("Java"));

}

}

## Output

true

### endsWith()

**endsWith():**This method return true if string is end with given another string, otherwise it returns false.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="Java is programming language";

System.**out**.println(s.endsWith("language"));

}

}

## Output

true

### subString()

**subString():**This method is used to get the part of given string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="Java is programming language";

System.**out**.println(s.substring(8)); // 8 is starting index

}

}

## Output

programming language

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="Java is programming language";

System.**out**.println(s.substring(8, 12));

}

}

## Output

prog

### indexOf()

**indexOf():**This method is used find the index value of given string. It always gives starting index value of first occurrence of string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="Java is programming language";

System.**out**.println(s.indexOf("programming"));

}

}

## Output

8

### lastIndexOf()

**lastIndexOf():**This method used to return the starting index value of last occurence of the given string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Java is programming language";

String s2="Java is good programming language";

System.**out**.println(s1.lastIndexOf("programming"));

System.**out**.println(s2.lastIndexOf("programming"));

}

}

## Output

8

13

### trim()

**trim():**This method remove space which are available before starting of string and after ending of string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s=" Java is programming language ";

System.**out**.println(s.trim());

}

}

## Output

Java is programming language

### split()

**split():**This method is used to divide the given string into number of parts based on delimiter (special symbols like @ space , ).

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s="contact@tutorial4us.com";

String[] s1=s.split("@"); // divide string based on @

**for**(String c:s1) // foreach loop

{

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

}

}

}

## Output

contact

@tutorial4us.com

### replace()

**replace():**This method is used to return a duplicate string by replacing old character with new character.

**Note:**In this method data of original string will never be modify.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="java";

String s2=s1.replace('j', 'k');

System.**out**.println(s2);

}

}

## Output

kava

## StringBuffer

It is a predefined class in java.lang package can be used to handle the String, whose object is mutable that means content can be modify.   
StringBuffer class is working with thread safe mechanism that means multiple thread are not allowed simultaneously to perform operation of StringBuffer.

**StringBuffer** class object is **mutable** that means when we create an object of StringBulder class it can be change.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("java");

sb.append("software");

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

}

}

## Output

javasoftware

**Explanation:**Here we can changes in the existing object of StringBuffer class so output is javasoftware.

## Difference between String and StringBuffer

|  |  |  |
| --- | --- | --- |
|  | **String** | **StringBuffer** |
| 1 | The data which enclosed within double quote (" ") is by default treated as String class. | The data which enclosed within double quote (" ") is not by default treated as StringBuffer class |
| 2 | String class object is immutable | StringBuffer class object is mutable |
| 3 | When we create an object of String class by default no additional character memory space is created. | When we create an object of StringBuffer class by default we get 16 additional character memory space. |

## Similarities between String and StringBuffer

* Both of them are belongs to public final. so that they never participates in inheritance that is is-A relationship is not possible but they can always participates in As-A and Uses-A relationship.
* We can not override the method of String and StringBuffer.

### Methods of StringBuffer class

### reverse()

**reverse():**This method is used to reverse the given string and also the new value is replaced by the old string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("java code");

System.**out**.println(sb.reverse());

}

}

## Output

edoc avaj

### insert()

**insert():**This method is used to insert either string or character or integer or real constant or boolean value at a specific index value of given string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("this is my java code");

System.**out**.println(sb.insert(11, "first "));

}

}

## Output

this is my first java code

### append()

**append():**This method is used to add the new string at the end of original string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("java is easy");

System.**out**.println(sb.append(" to learn"));

}

}

## Output

java is easy to learn

### replace()

**replace()** This method is used to replace any old string with new string based on index value.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("This is my code");

System.**out**.println(sb.replace(8, 10, "java"));

}

}

## Output

This is java code

**Explanation:**In above example java string is replaced with old string (my) which is available between 8 to 10 index value.

### deleteCharAt()

**deleteCharAt():**This method is used to delete a character at given index value.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("java");

System.**out**.println(sb.deleteCharAt(3));

}

}

## Output

jav

### delete()

**delete():**This method is used to delete string form given string based on index value.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("java is easy to learn");

StringBuffer s;

s=sb.**delete**(8, 13);

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

}

}

## Output

java is to learn

**Explanation:**In above example string will be deleted which is existing between 8 and 13 index value.

### toString()

**toString():**This method is used to convert mutable string value into immutable string.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

StringBuffer sb=**new** StringBuffer("java");

String s=sb.toString();

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

s.concat("code");

}

}

## Output

java

## String Compare in Java

There are three way to compare string object in java:

* By equals() method
* By == operator
* By compreTo() method

## equals() Method in Java

equals() method always used to comparing contents of both source and destination String. It return true if both string are same in meaning and case otherwise it returns false. It is case sensitive method.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="Raddy";

String s3="Hitesh";

System.**out**.println("Compare String: "+s1.equals(s2));

System.**out**.println("Compare String: "+s1.equals(s3));

}

}

## Output

Compare String: false

Compare String: true

## == or Double Equals to Operator in Java

== Operator is always used for comparing references of both source and destination objects but not their contents.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1=**new** String("java");

String s2=**new** String("java");

**if**(s1==s2)

{

System.**out**.println("Strings are same");

}

**else**

{

System.**out**.println("Strings are not same");

}

}

}

## Output

Strings are not same

## compareTo() Method in Java

comapreTo() method can be used to compare two string by taking Unicode values. It returns 0 if the string are same otherwise returns either +ve or -ve integer.

## Example

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Hitesh";

String s2="Raddy";

**int** i;

i=s1.compareTo(s2);

**if**(i==0)

{

System.**out**.println("Strings are same");

}

**else**

{

System.**out**.println("Strings are not same");

}

}

}

## Output

Strings are not same

### Difference between equals() method and == operator

equals() method always used to comparing contents of both source and destination String.

== Operator is always used for comparing references of both source and destination objects but not their contents.

**String Concatenation**

There are two way to concat string object in java:

* By + (string concatenation) operator
* By concat() method

**By + operator**

Using Java string concatenation operator (+) you can combined two or more strings.

**Example**

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s= "Java" + "Code";

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

}

}

**Output**

JavaCode

**By concat() method**

concat() method is used to combined two strings.

**Example**

**class** StringHandling

{

**public** **static** **void** main(String arg[])

{

String s1="Java";

String s2="Code";

String s3=s1.concat(s2);

System.**out**.println(s3);

}

}

**Output**

JavaCode

**StringBuilder**

It is a predefined class in java.lang package can be used to handle the String. StringBuilder class is almost similar to to StringBuffer class. It is also a mutable object.  
The main difference StringBuffer and StringBuilder class is StringBuffer is thread safe that means only one threads allowed at a time to work on the String where as StringBuilder is not thread safe that means multiple threads can work on same String value.

**Difference between StringBuffer and StringBuilder**

All the things between StringBuffer and StringBuilder are same only difference is StringBuffer is synchronized and StringBuilder is not synchronized. synchronized means one thread is allow at a time so it thread safe. Not synchronized means multiple threads are allow at a time so it not thread safe.

|  |  |  |
| --- | --- | --- |
|  | **StringBuffer** | **StringBuilder** |
| 1 | It is thread safe. | It is not thread safe. |
| 2 | Its methods are synchronized and provide thread safety. | Its methods are not synchronized and unable to provide thread safety. |
| 3 | Relatively performance is low because thread need to wait until previous process is complete. | Relatively performance is high because no need to wait any thread it allows multiple thread at a time. |
| 1 | Introduced in 1.0 version. | Introduced in 1.5 version. |

**When we use String, StringBuffer and StringBuilder**

* If the content is fixed and would not change frequently then we use String.
* If content is not fixed and keep on changing but thread safety is required then we use StringBuffer
* If content is not fixed and keep on changing and thread safety is not required then we use StringBuilder

## StringTokenizer in Java

It is a pre defined class in **java.util** package can be used to split the given string into tokens (parts) based on delimiters (any special symbols or spaces).

Suppose that we have any string like "Features of Java\_Language" when we use stringTokenizer this string is split into tokens whenever spaces and special symbols present. After split string are :

## Example

Features

of

Java

Language

## Methods of StringTokenizer

* hasMoreTokens()
* nextToken()

## hasMoreTokens()

It is predefined method of StringTokenizer class used to check whether given StringTokenizer having any elements or not.

## nextToken()

Which can be used to get the element from the StringTokenizer.

### Example of StringTokenizer:

## Example of StringTokenizer

**import** java.util.\*;

**class** Stringtokenizerdemo

{

**public** **static** **void** main(String args[])

{

String str="He is a gentle man";

StringTokenizer st=**new** StringTokenizer(str," ");

System.**out**.println("The tokens are: ");

**while**(st.hasMoreTokens())

{

String one=st.nextToken();

System.**out**.println(one);

}

}

}

## Output

The tokens are:

He

is

a

gentle

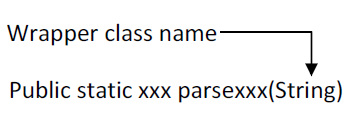
man

## Data Conversion

In java programming we have six data conversion technique they are:.

### 1 Converting numeric string type data into numerical / fundamental type values

In order to convert numerical string into numerical or fundamental values we use the following generalized predefined method which is present in wrapper classes.



Here xxx represent any fundamental data type.

## Example

String s1="100";

**int** x=Integer.parseInt(s1);

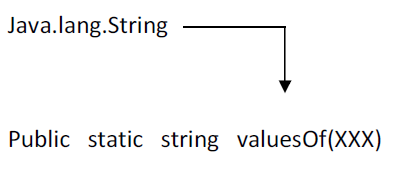
## Example

String s2="100.75f";

Float y=Float.parseFloat(s2);

## 2 Converting numeric / fundamental type values into string type values

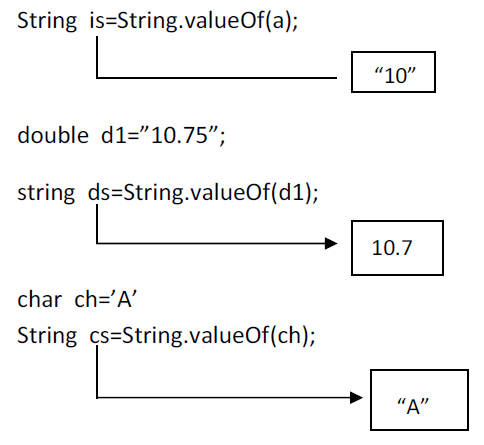
In order to convert numeric / fundamental type values into string values, we use the following predefined static overloaded method.



Here XXX represent any fundamental data type values

## Example

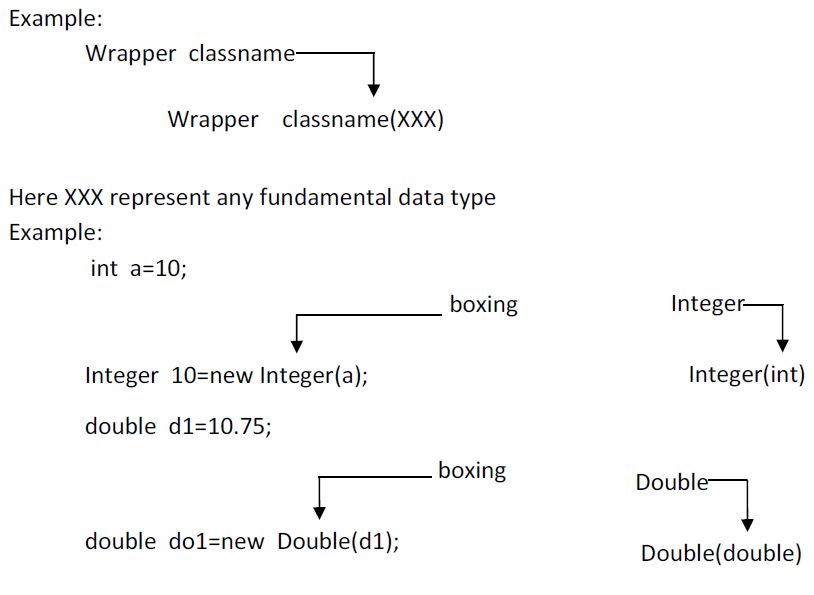
**int** a=10;



## 3 Converting fundamental type values into object type values:

In order to convert the fundamental data into equivalent wrapper class object type data we use the following generalized predefined parameterized constructor by taking fundamental data type as a parameter.

Example:



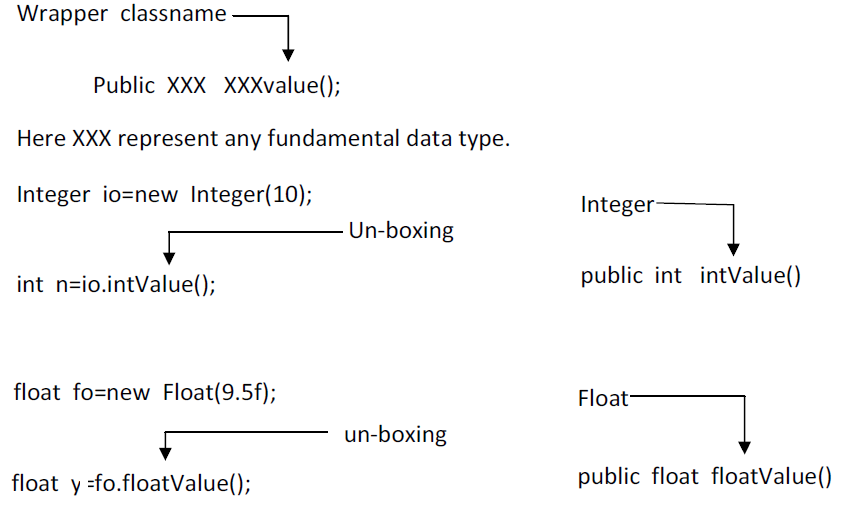
in JDK 1.4 converting fundamental data type values into wrapper class object is known as boxing. In the case of JDK 1.5 and in higher version it is optional to the java programmer to convert fundamental data type value into equivalent wrapper class object. That is implicitly taken care by JVM and it is known as auto boxing.

## Definition of auto boxing

The process of implicitly converting fundamental type value into equivalent wrapper class object is known as auto boxing.

## 4 Converting object type value into fundamental type value:

In order to convert wrapper class object data into fundamental type data, we use the following predefined instance method present in each and every wrapper class.



In case of JDK 1.5 and in higher version it is optional to the java programmer to convert object data into fundamental type data and this process is known as auto un-boxing and its takes care by JVM.

## Definition of auto un-boxing

In process of implicitly conversion objects type data into fundamental type data is known as auto un-boxing.

## 5 Converting String type data into object type data

In order to convert String type numeric data into equivalent wrapper class object, we use the following predefined parameterized constructor by each and every wrapper class except character class

Example

int a=10;

* String is=String.valueOf(a);
* Integer io=new Integer(is);
* int x=io.intValue();
* Integer io=new Integer(n);
* String so=io.toString();
* int x=Integer.parseInt(so);