Java StringBuffer class

Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

***Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.***

Important Constructors of StringBuffer class

1. **StringBuffer():** creates an empty string buffer with the initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer class

1. **public synchronized StringBuffer append(String s):** is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. **public synchronized StringBuffer insert(int offset, String s):** is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. **public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** is used to replace the string from specified startIndex and endIndex.
4. **public synchronized StringBuffer delete(int startIndex, int endIndex):** is used to delete the string from specified startIndex and endIndex.
5. **public synchronized StringBuffer reverse():** is used to reverse the string.
6. **public int capacity():** is used to return the current capacity.
7. **public void ensureCapacity(int minimumCapacity):** is used to ensure the capacity at least equal to the given minimum.
8. **public char charAt(int index):** is used to return the character at the specified position.
9. **public int length():** is used to return the length of the string i.e. total number of characters.
10. **public String substring(int beginIndex):** is used to return the substring from the specified beginIndex.
11. **public String substring(int beginIndex, int endIndex):** is used to return the substring from the specified beginIndex and endIndex.

What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

**class** A{

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

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

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

**class** A{

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

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

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

**class** A{

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

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

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

**class** A{

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

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

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

**class** A{

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

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

sb.reverse();

System.out.println(sb);//prints olleH

}

}

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. 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. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** A{

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

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

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** A{

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

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

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

sb.ensureCapacity(10);//now no change

System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34\*2)+2

System.out.println(sb.capacity());//now 70

}

}

# Java StringBuilder class

Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

## **Important Constructors of StringBuilder class**

1. **StringBuilder():** creates an empty string Builder with the initial capacity of 16.
2. **StringBuilder(String str):** creates a string Builder with the specified string.
3. **StringBuilder(int length):** creates an empty string Builder with the specified capacity as length.

## **Important methods of StringBuilder class**

|  |  |
| --- | --- |
| Method | Description |
| public StringBuilder append(String s) | is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc. |
| public StringBuilder insert(int offset, String s) | is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc. |
| public StringBuilder replace(int startIndex, int endIndex, String str) | is used to replace the string from specified startIndex and endIndex. |
| public StringBuilder delete(int startIndex, int endIndex) | is used to delete the string from specified startIndex and endIndex. |
| public StringBuilder reverse() | is used to reverse the string. |
| public int capacity() | is used to return the current capacity. |
| public void ensureCapacity(int minimumCapacity) | is used to ensure the capacity at least equal to the given minimum. |
| public char charAt(int index) | is used to return the character at the specified position. |
| public int length() | is used to return the length of the string i.e. total number of characters. |
| public String substring(int beginIndex) | is used to return the substring from the specified beginIndex. |
| public String substring(int beginIndex, int endIndex) | is used to return the substring from the specified beginIndex and endIndex. |

## **Java StringBuilder Examples**

Let's see the examples of different methods of StringBuilder class.

### 1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

**class** A{

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

StringBuilder sb=**new** StringBuilder("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

### 2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

**class** A{

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

StringBuilder sb=**new** StringBuilder("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}

### 3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

**class** A{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

### 4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

**class** A{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

### 5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

**class** A{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}

### 6) StringBuilder capacity() method

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** A{

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

StringBuilder sb=**new** StringBuilder();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

### 7) StringBuilder ensureCapacity() method

The ensureCapacity() method of StringBuilder class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** A{

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

StringBuilder sb=**new** StringBuilder();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

sb.ensureCapacity(10);//now no change

System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34\*2)+2

System.out.println(sb.capacity());//now 70

}

}

# Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

|  |  |  |
| --- | --- | --- |
| No. | String | StringBuffer |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

## **Performance Test of String and StringBuffer**

**public** **class** ConcatTest{

**public** **static** String concatWithString()    {

        String t = "Java";

**for** (**int** i=0; i<10000; i++){

            t = t + "Tpoint";

        }

**return** t;

    }

**public** **static** String concatWithStringBuffer(){

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

**for** (**int** i=0; i<10000; i++){

            sb.append("Tpoint");

        }

**return** sb.toString();

    }

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

**long** startTime = System.currentTimeMillis();

        concatWithString();

        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");

        startTime = System.currentTimeMillis();

        concatWithStringBuffer();

        System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms");

    }

}

Time taken by Concating with String: 578ms

Time taken by Concating with StringBuffer: 0ms

## **String and StringBuffer HashCode Test**

As you can see in the program given below, String returns new hashcode value when you concat string but StringBuffer returns same.

**public** **class** InstanceTest{

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

        System.out.println("Hashcode test of String:");

        String str="java";

        System.out.println(str.hashCode());

        str=str+"tpoint";

        System.out.println(str.hashCode());

        System.out.println("Hashcode test of StringBuffer:");

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

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

        sb.append("tpoint");

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

    }

}

Hashcode test of String:

3254818

229541438

Hashcode test of StringBuffer:

118352462

118352462

# Difference between StringBuffer and StringBuilder

There are many differences between StringBuffer and StringBuilder. A list of differences between StringBuffer and StringBuilder are given below:

|  |  |  |
| --- | --- | --- |
| 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. |

## **StringBuffer Example**

**public** **class** BufferTest{

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

        StringBuffer buffer=**new** StringBuffer("hello");

        buffer.append("java");

        System.out.println(buffer);

    }

}

Hellojava

## **StringBuilder Example**

**public** **class** BuilderTest{

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

        StringBuilder builder=**new** StringBuilder("hello");

        builder.append("java");

        System.out.println(builder);

    }

}

Hellojava

## **Performance Test of StringBuffer and StringBuilder**

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

**public** **class** ConcatTest{

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

**long** startTime = System.currentTimeMillis();

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

**for** (**int** i=0; i<10000; i++){

            sb.append("Tpoint");

        }

        System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");

        startTime = System.currentTimeMillis();

        StringBuilder sb2 = **new** StringBuilder("Java");

**for** (**int** i=0; i<10000; i++){

             sb2.append("Tpoint");

         }

        System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");

    }

}

Time taken by StringBuffer: 16ms

Time taken by StringBuilder: 0ms

How to create Immutable class?

|  |
| --- |
| There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable. We can also create immutable class by creating final class that have final data members as the example given below: |

Example to create Immutable class

|  |
| --- |
| In this example, we have created a final class named Employee. It have one final datamember, a parameterized constructor and getter method. |

**public** **final** **class** Employee{

**final** String pancardNumber;

**public** Employee(String pancardNumber){

**this**.pancardNumber=pancardNumber;

}

**public** String getPancardNumber(){

**return** pancardNumber;

}

}

The above class is immutable because:

* The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
* The class is final so we cannot create the subclass.
* There is no setter methods i.e. we have no option to change the value of the instance variable.

These points makes this class as immutable.

# Java toString() method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the string representation of the object.

If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

## **Advantage of Java toString() method**

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

### Understanding problem without toString() method

Let's see the simple code that prints reference.

**class** Student{

**int** rollno;

  String name;

  String city;

  Student(**int** rollno, String name, String city){

**this**.rollno=rollno;

**this**.name=name;

**this**.city=city;

  }

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

    Student s1=**new** Student(101,"Raj","lucknow");

    Student s2=**new** Student(102,"Vijay","ghaziabad");

    System.out.println(s1);//compiler writes here s1.toString()

    System.out.println(s2);//compiler writes here s2.toString()

  }

}

Output:Student@1fee6fc

Student@1eed786

As you can see in the above example, printing s1 and s2 prints the hashcode values of the objects but I want to print the values of these objects. Since java compiler internally calls toString() method, overriding this method will return the specified values. Let's understand it with the example given below:

## **Example of Java toString() method**

Now let's see the real example of toString() method.

**class** Student{

**int** rollno;

  String name;

  String city;

  Student(**int** rollno, String name, String city){

**this**.rollno=rollno;

**this**.name=name;

**this**.city=city;

  }

**public** String toString(){//overriding the toString() method

**return** rollno+" "+name+" "+city;

  }

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

    Student s1=**new** Student(101,"Raj","lucknow");

    Student s2=**new** Student(102,"Vijay","ghaziabad");

    System.out.println(s1);//compiler writes here s1.toString()

    System.out.println(s2);//compiler writes here s2.toString()

  }

}

Output:101 Raj lucknow

102 Vijay ghaziabad

# StringTokenizer in Java

1. [StringTokenizer](http://www.javatpoint.com/string-tokenizer-in-java)
2. [Methods of StringTokenizer](http://www.javatpoint.com/string-tokenizer-in-java)
3. [Example of StringTokenizer](http://www.javatpoint.com/string-tokenizer-in-java)

The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

**Constructors of StringTokenizer class**

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**

The 6 useful methods of StringTokenizer class are as follows:

|  |  |
| --- | --- |
| 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. |

Simple example of StringTokenizer class

Let's see the simple example of StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

**import** java.util.StringTokenizer;

**public** **class** Simple{

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

    StringTokenizer st = **new** StringTokenizer("my name is khan"," ");

**while** (st.hasMoreTokens()) {

System.out.println(st.nextToken());

      }

    }

}

Output:my

name

is

khan

Example of nextToken(String delim) method of StringTokenizer class

**import** java.util.\*;

**public** **class** Test {

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

       StringTokenizer st = **new** StringTokenizer("my,name,is,khan");

      // printing next token

      System.out.println("Next token is : " + st.nextToken(","));

   }

}

Output:Next token is : my

***StringTokenizer class is deprecated now. It is recommended to use split() method of String class or regex (Regular Expression).***