**StringBuffer**

* If the content will change frequently then never recommended to go for String object because for every change a new object will be created internally.
* To handle this type of requirement we should go for StringBuffer concept.
* The main advantage of StringBuffer over String is, all required changes will be performed in the existing object only instead of creating new object.

**Constructors:**

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

* Creates an empty StringBuffer object with default initialcapacity “16”.
* Once StringBuffer object reaches its maximum capacity a new StringBuffer object will be created with **Newcapacity=(currentcapacity+1)\*2.**

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer();

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

sb.append("abcdefghijklmnop");

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

sb.append("q");

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

}

}

1. **StringBuffer sb=new StringBuffer(int initialcapacity);**

* Creates an empty StringBuffer object with the specified initial capacity.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer(19);

System.out.println(sb.capacity());//19

}

}

1. **StringBuffer sb=new StringBuffer(String s);**

* Creates an equivalent StringBuffer object for the given String with capacity=s.length()+16;

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("bhaskar");

System.out.println(sb.capacity());//23

}

}

**Important methods of StringBuffer:**

1. **public int length();**

* Return the no of characters present in the StringBuffer.

1. **public int capacity();**

* Returns the total no of characters but a StringBuffer can accommodate(hold).

1. **public char charAt(int index);**

* It returns the character located at specified index.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("vijayabhaskarreddy");

System.out.println(sb.length());//18

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

System.out.println(sb.charAt(14));//e

}

}

1. **public void setCharAt(int index,char ch);**

* To replace the character locating at specified index with the provided character.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("vijayabhaskarreddy");

sb.setCharAt(6,'A');

System.out.println(sb);

}

}

1. **public StringBuffer append(String s);**

public StringBuffer append(int i);

public StringBuffer append(long l);

public StringBuffer append(boolean b); All these are overloaded methods.

public StringBuffer append(double d);

public StringBuffer append(float f);

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer();

sb.append("PI value is :");

sb.append(3.14);

sb.append(" this is exactly ");

sb.append(true);

System.out.println(sb);//PI value is :3.14 this is exactly true

}

}

1. **public StringBuffer insert(int index,String s);**

public StringBuffer insert(int index,int i);

public StringBuffer insert(int index,long l);

public StringBuffer insert(int index,double d); All are overloaded methods

public StringBuffer insert(int index,boolean b);

public StringBuffer insert(int index,float f);

* To insert at the specified location.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("vijaya");

sb.insert(6,"bhaskar");

sb.insert(13,"9");

System.out.println(sb);//vijayabhaskar9

}

}

1. **public StringBuffer delete(int begin,int end);**

* To delete characters from begin index to end n-1 index.

1. **public StringBuffer deleteCharAt(int index);**

* To delete the character locating at specified index.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("vijayabhaskar");

System.out.println(sb);//vijayabhaskar

sb.delete(6,13);

System.out.println(sb);//vijaya

sb.deleteCharAt(5);

System.out.println(sb);//vijay

}

}

1. **public StringBuffer reverse();**

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("vijayabhaskar");

System.out.println(sb);//vijayabhaskar

System.out.println(sb.reverse());//raksahbayajiv

}

}

1. **public void setLength(int length);**

* Consider only specified no of characters and remove all the remaining characters.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("vijayabhaskar");

sb.setLength(6);

System.out.println(sb);//vijaya

}

}

1. **public void trimToSize();**

* To deallocate the extra free memory such that capacity and size are equal.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer(1000);

System.out.println(sb.capacity());//1000

sb.append("bhaskar");

System.out.println(sb.capacity());//1000

sb.trimToSize();

System.out.println(sb.capacity());//7

}

}

1. **public void ensureCapacity(int initialcapacity);**

* To increase the capacity dynamically based on our requirement.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer();

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

sb.ensureCapacity(1000);

System.out.println(sb.capacity());//1000

}

}

**StringBuilder (1.5)**

* Every method present in StringBuffer is declared as synchronized hence at a time only one thread is allowed to operate on the StringBuffer object due to this, waiting time of the threads will be increased and effects performance of the system.
* To overcome this problem sun people introduced StringBuilder concept in 1.5v.
* StringBuilder is exactly same as StringBuffer except the following differences.

|  |  |
| --- | --- |
| **StringBuffer** | **StringBuilder** |
| 1. Every method present in StringBuffer is synchronized. | 1. No method present in StringBuilder is synchronized. |
| 1. At a time only one thread is allow to operate on the StringBuffer object hence StringBuffer object is Thread safe. | 1. Multiple Threads are allowed to operate simultaneously on the StringBuilder object hence StringBuilder is not Thread safe. |
| 1. It increases waiting time of the Thread and hence relatively performance is low. | 1. Threads are not required to wait and hence relatively performance is high. |
| 1. Introduced in 1.0 version. | 1. Introduced in 1.5 versions. |

**String vs StringBuffer vs StringBuilder:**

* If the content is fixed and won’t change frequently then we should go for String.
* If the content will change frequently but Thread safety is required then we should go for StringBuffer.
* If the content will change frequently and Thread safety is not required then we should go for StringBuilder.

**Method chaining:**

* For most of the methods in String, StringBuffer and StringBuilder the return type is same type only. Hence after applying method on the result we can call another method which forms method chaining.

**Example:**

s.m1().m2().m3()……………….

* In method chaining all methods will be evaluated from left to right.

**Example:**

class StringBufferDemo

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer();

sb.append("vijaya").insert(6,"bhaskarreddy").delete(13,17).reverse().append("solutions").insert(22,"abcdf").reverse();

System.out.println(sb);//sfdcbanoitulosvijayabhaskary

}

}