**java.lang package – Part-07**

* **StringBuffer:**

If the content is fixed and won’t change frequently then it is recommended to go for String.

If the content is not fixed and keep on changing then it is not recommended to use String. Because, for every change a new object will be created which affects performance of the system.

To handle this requirement we should go for StringBuffer.

The main advantage of StringBuffer over String is all required changes will be performed in the existing object only.

* **Constructors:**

1. StringBuffer sb = new StrinbBuffer();

Creates an empty StringBuffer object with default initial capacity 16.

Once StringBuffer reaches its max capacity, a new StringBuffer object will be created with new capacity with the below formula.

Newcapactiy = (currentcapacity + 1) \* 2

Example:

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 specified initialCapacity.

Note:

This constructor will improve the system performance, as we are giving the initialCapacity, so it will not go through the series of creating new StringBuffer object copy the old values to the new object as and when it reaches the currentCapacity.

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

Creates an equivalent StringBuffer for the given String with capacity as below.

Capacity = s.length() + 16

Example:

StringBuffer sb = new StringBuffer(“durga”);

System.out.println(sb.capactiy()); //21

* **Important methods of StringBuffer:**

1. public int length()
2. public int capacity()
3. public char charAt(int index)

StringBuffer sb = new StringBuffer(“durga”);

System.out.println(sb.charAt(3));// g

System.out.println(sb.charAt(30)); // RE

RE: StringIndexOutOfBoundsException

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

To replace the character located at specified index with provided character.

1. public StringBuffer append(String s);
2. public StringBuffer append(int i);
3. public StringBuffer append(long l)
4. public StringBuffer append(char c);
5. public StringBuffer append(boolean b); … etc.

StringBuffer sb = new StringBuffer();

sb.append(“PI value is : ”);

sb.append(3.14);

sb.append(“ It is exactly : ”);

sb.append(true);

System.out.println(sb);

Output:

PI value is : 3.14 It is exactly : true

Note: append method is overloaded.

Note: Append method always add the new value at the end.

1. pubic StringBuffer insert(int index, String s);
2. public StringBuffer insert(int index, int i);
3. public StringBuffer insert(int index, double d);
4. public StringBuffer insert(int index, char ch);

All the insert methods are overloaded.

StringBuffer sb = new StringBuffer(“abcdefgh”);

sb.insert(2, “xyz”);

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

Note: Insert always add the value before the existing content.

1. public StrinbBuffer delete(int begin, int end);

To delete characters located from begin index to end-1 index.

1. public StringBuffer deleteCharAt(int index);

To delete the character located at specified index.

1. public StringBuffer reverse();

StringBuffer sb = new StringBuffer(“durga”)

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

1. public void setLength(int length);

StringBuffer sb = new StringBuffer(“aishwaryaabhi”);

sb.setLength(8);

System.out.println(sb);

1. public void ensureCapacity(int capacity);

To increase capacity on the fly, based on our requirement.

Example:

StringBuffer sb = new StringBuffer();

sb.capacity();

sb.ensureCapacity(1000);

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

1. public void trimToSize();

To deallocate extra allocated free memory.

Example:

StringBuffer sb = new StringBuffer(1000);

sb.append(“abc”);

sb.trimToSize();

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

* **StringBuilder:**

Every method present in StringBuffer is “synchronized” and hence only one thread is allowed to operate on StringBuffer object at a time. Which may create performance problems. To handle this requirement Sun people introduced

StringBuilder concept in 1.5 version

StringBuilder is exactly same as StringBuffer except the following differences:

|  |  |  |
| --- | --- | --- |
| S.No | StringBuffer | StringBuilder |
| 1 | Every method present in StringBuffer is synchronized | Every method present in StringBuilder is Non-synchronized |
| 2 | At a time only one thread is allowed to operate on StringBuffer object and hence StringBuffer object is thread safe. | At a time multiple threads are allowed to operate on StringBuilder object and hence StringBuilder is not thread safe. |
| 3 | Threads are required to wait to operate on StringBuffer object and hence relatively performance is low. | 4 Threads are not required to wait to operate on StringBuilder object and hence relatively performance is high. |
| 4 | Introduced in 1.0 version | Introduced in 1.5 version. |

**Note: Except above differences everything is same in StringBuffer and StringBuilder (including methods and constructors).**

* **String vs StringBuffer vs StringBuilder:**

1. If the content is fixed and won’t change frequently, then we should go for String.
2. If the content is not fixed and keep on changing but thread-safety is required, then we should go for StringBuffer.
3. If the content is not fixed and keep on changing but thread-safety is not required then we should go for StringBuilder.

* **Method Chaining:**

For most of the methods in String, StringBuffer and StringBuilder returns types are same type, hence after applying a method on the result, we can call another method which forms method chaining.

sb.m1().m2().m3().m4()…

In method chaining method calls will be executed from left to right.

StringBuffer sb = new StringBuffer();

sb.append(“durga”).append(“software”).append(“solutions”).insert(2, “xyz”).reverse().delete(2, 10);

System.out.println(sb);