String Builder in Java

Author: Nitish k

Java String Builder Class

Java StringBuilder class is used to create mutable string. The StringBuilder class is same as StringBuffer class except that it is non synchronized.

Important Constructors of StringBuilder class

Constructor	Description
StringBuilder()	It creates an empty String Builder with the initial capacity of 16.
StringBuilder(String str)	It creates a String Builder with the specified string.
StringBuilder(int length)	It creates an empty String Builder with the specified capacity as length.

Important methods of StringBuilder class

Method	Description
public StringBuilder append(String s)	It 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)	It 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)	It is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public StringBuilder reverse()	It is used to reverse the string.
public int capacity()	It is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	It is used to return the character at the specified position.
public int length()	It is used to return the length of the string i.e. total number of characters.

public String substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

Examples

1) For append() method

```
package stringbuilder;
public class Example {

	public static void main(String[] args) {
		StringBuilder sb = new StringBuilder("Hello");
		sb.append(" World");
		System.out.println(sb);
	}
}
```

2) For insert() method

Inserts a string at given position

Eg:

```
package stringbuilder;

public class Example2 {

	public static void main(String[] args) {
		StringBuilder sb = new StringBuilder("Hello");
		sb.insert(5, " World!!");
		System.out.println(sb);
	}
}
```

3) For replace() method

The StringBuilder replace() method replaces the given string from the specified beginindex and endindex

Eg:

```
package stringbuilder;
```

```
public class Example3 {

	public static void main(String[] args) {
		StringBuilder sb = new StringBuilder("hello");
		sb.replace(1, 3, "Java");
		System.out.println(sb);
	}
}
```

4) For delete() method
The delete() method of StringBuilder class deletes the string from the specified beginindex to endindex.
Eg:

```
package stringbuilder;
public class Example4 {
	public static void main(String[] args) {
		StringBuilder sb = new StringBuilder("hello");
		sb.delete(1, 3);
		System.out.println(sb);
	}
}
```

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

Case 1:
To reverse the entire string of input
Eg:

```
package stringbuilder;
public class Example5 {
	public static void main(String[] args) {
		StringBuilder sb = new StringBuilder("This is a Sentence");
		sb.reverse();
		System.out.println(sb);
	}
}
```

Case 2:
To reverse the string word by word
Steps:

i. Tokenize each word using String.split() method.
ii. Loop through string array and use StringBuilder.reverse() method to reverse each word.
iii. Join all reversed word to get back the word

```
package stringbuilder;
import java.io.*;
public class Example5 {

	public static void main(String[] args) throws IOException{
		// TODO Auto-generated method stub
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		System.out.println("Enter a String");
		String str = br.readLine();
		System.out.println("Initial String is:");
		System.out.println(str);
		StringBuilder sb = new StringBuilder();
		String [] words = str.split(" ");
		for(String word : words)
		{
			String reverseWord = new
StringBuilder(word).reverse().toString();
			sb.append(reverseWord+" ");
		}
		System.out.println("Reversed String is: ");
		System.out.println(sb);
	}
}
```

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.
Eg:

```
class StringBuilderExample6{
	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.

Eg:

```
class StringBuilderExample7{
        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
        }
}
```

Java StringBuffer Class

Java StringBuffer is used to create mutable String objects. The StringBuffer class in java is the same as String class except it is mutable.

Important Constructors of StringBuffer Class

Constructor	Description
StringBuffer()	It creates an empty String buffer with the initial capacity of 16.
StringBuffer(String str)	It creates a String buffer with the specified string..
StringBuffer(int capacity)	It creates an empty String buffer with the specified capacity as length.

Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	It 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 synchronized StringBuffer	insert(int offset, String s)	It 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 synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	It is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	It is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	It is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	It is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	It is used to return the character at the specified position.
public int	length()	It is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	It is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	It is used to return the substring from the specified beginIndex and endIndex.

1) Using append method

   The append() method concatenates the given argument with this String

   Eg:

```
package stringbuilder;

public class Example6 {

	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer("Hello");
		sb.append(" World");
		System.out.println(sb);
	}

}
```

2) Using insert() method

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

Eg:

```
class StringBufferExample2{
	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) Using replace() method:

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

Eg:

```
class StringBufferExample3{
	public static void main(String args[]){
		StringBuffer sb=new StringBuffer("Hello");
		sb.replace(1,3,"Java");
		System.out.println(sb);//prints HJavalo
	}
}
```

4) Using delete() method:
The delete() method of the StringBuffer class deletes the String from the specified beginIndex to endIndex.
Eg:

```
class StringBufferExample4{
	public static void main(String args[]){
```

```
                StringBuffer sb=new StringBuffer("Hello");
                sb.delete(1,3);
                System.out.println(sb);//prints Hlo
        }
}
```

5) Using reverse() method:
The reverse() method of the StringBuilder class reverses the current String.

Eg:

```
class StringBufferExample5{
        public static void main(String args[]){
                StringBuffer sb=new StringBuffer("Hello");
                sb.reverse();
                System.out.println(sb);//prints olleH
        }
}
```

6) Using capacity method
The capacity() method of the 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.

Eg:

```
class StringBufferExample6{
        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) Using ensureCapacity() method
The ensureCapacity() method of the 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.

Eg:

```
class StringBufferExample7{
```

```
    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
    }
}
```

Immutable Classes in Java

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. All the wrapper classes and String class is immutable. We can create immutable classes by creating **final** class that have final data members

Eg:

```
package practice;
class Employeer
{
    final String PanCardNumber;
    public Employeer(String PanCardNumber)
    {
            this.PanCardNumber = PanCardNumber;
    }
    public void getPancard()
```

```
    {
        System.out.println(PanCardNumber);
    }
}
public class Employee {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Employeer obj = new Employeer("ABC12345");
        obj.getPancard();
    }
}
```

Important Points to note from above example:

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

Java toString() Method

If u 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. depending on implementation.

Advantage

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.

Eg:

```
package practice;

public class Student {

	String Name;
	String city;
	int age;

	public Student(String Name,String city,int age)
	{
		this.Name = Name;
		this.city = city;
		this.age = age;
	}
	public String toString()
	{
		return Name+" "+city+" "+age;
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Student s1 =  new Student("Raj","Bangalore",21);
		Student s2 = new Student("Verma","Bombay",22);
		System.out.println(s1);
		System.out.println(s2);
	}
}
```