MIRACLE SOFTWARE SYSTEMS, INC.

String Handling

Table of Contents

1.String

String Handling provides a lot of concepts that can be performed on a string such as concatenating string, comparing string, substring etc.

In java, string is basically an immutable object. We will discuss about immutable string later. Let's first understand what is string and how we can create the string object.

String
Generally string is a sequence of characters. But in java, string is an object. String class is used to create string object.

How to create String object?
There are two ways to create String object:

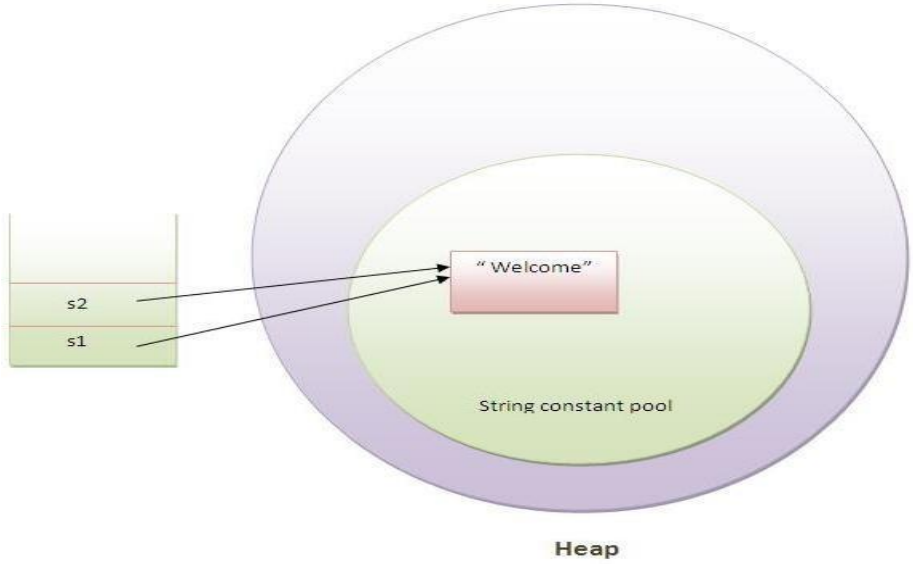
1. By string literal
2. By new keyword

1) String literal
String literal is created by double quote.For Example:

1. String s="Hello";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance returns. If the string does not exist in the pool, a new String object instantiates, then is placed in the pool.For example:

1. String s1="Welcome";
2. String s2="Welcome";//no new object will be created



In the above example only one object will be created. First time JVM will find no string object with the name "Welcome" in string constant pool,so it will create a new object. Second time it will find the string with the name "Welcome" in string constant pool,so it will not create new object whether will return the reference to the same instance. To make Java more memory efficient (because no new objects are created if it exists already in string constant pool) java uses concept of string literal.

Note: **String objects are stored in a special memory area known as string constant pool inside the Heap memory.**

2) By new keyword

1. String s=new String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new String object in normal(nonpool) Heap memory and the literal "Welcome" will be placed in the string constant pool.The variable s will refer to the object in Heap(nonpool).

1.1 String Immutability

In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.

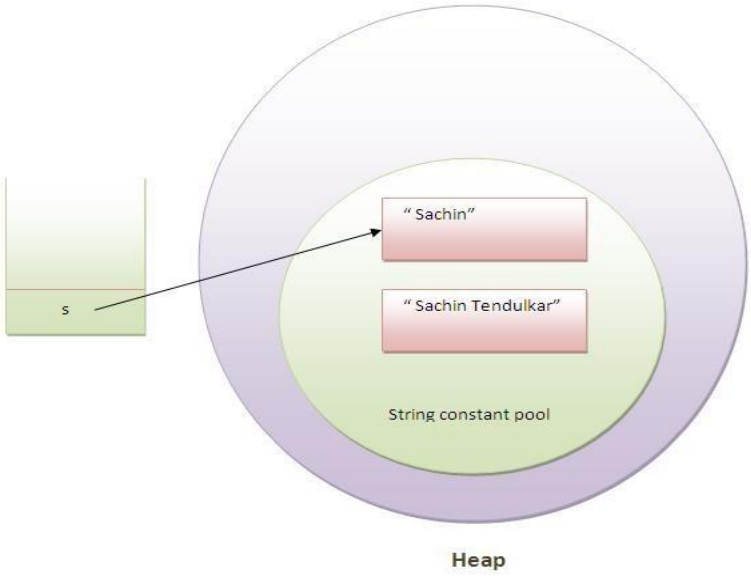
Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

```
class Simple{
 public static void main(String args[]){
  String s="Sachin";
  s.concat(" Tendulkar");     //concat() method appends the string at the end
  System.out.println(s);      //will print Sachin because strings are immutable objects
 }
}
```

Output:Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

```
class Simple{
 public static void main(String args[]){
  String s="Sachin";
  s=s.concat(" Tendulkar");
  System.out.println(s);
 }
}
```

Output:Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

Why string objects are immutable in java?
Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

1.2 String class methods

java.lang.String class provides a lot of methods to work on string. Let's see the commonly used methods of String class.

Method	Description
1)public boolean equals(Object anObject)	Compares this string to the specified object.
2)public boolean equalsIgnoreCase(String another)	Compares this String to another String, ignoring case.
3)public String concat(String str)	Concatenates the specified string to the end of this string.
4)public int compareTo(String str)	Compares two strings and returns int
5)public int compareToIgnoreCase(String str)	Compares two strings, ignoring case differences.
6)public String substring(int beginIndex)	Returns a new string that is a substring of this string.
7)public String substring(int beginIndex,int endIndex)	Returns a new string that is a substring of this string.
8)public String toUpperCase()	Converts all of the characters in this String to upper case
9)public String toLowerCase()	Converts all of the characters in this String to lower case.
10)public String trim()	Returns a copy of the string, with leading and trailing whitespace omitted.
11)public boolean startsWith(String prefix)	Tests if this string starts with the specified prefix.
12)public boolean endsWith(String suffix)	Tests if this string ends with the specified suffix.
13)public char charAt(int index)	Returns the char value at the specified index.
14)public int length()	Returns the length of this string.
15)indexof()	Finding the position of particular string

16)valueof()	To convert primitive numbers into strings.

1.3 Examples

Now Let's take the example of methods:
Example of equals() method:

```
public class MyStringEquals {
  public static void main(String a[]){
    String x = "JUNK";
    String y = "junk";
    if(x.equals(y)){
      System.out.println("Both strings are equal.");
    } else {
      System.out.println("Both strings are not equal.");
    }
    if(x.equalsIgnoreCase(y)){
      System.out.println("Both strings are equal.");
    } else {
      System.out.println("Both strings are not equal.");
    }
  }
}
```

output:Both strings are not equal.
Both strings are equal.

Example of equalsIgnoreCase() method:

```
String s1=”Hello”;
String s2=”hello”; if(s1.equals(s2))
--->false
if(s1.equalsIgnoreCase(s2))--->true
```

Example of concat()method:

```
public class Test {

  public static void main(String args[]) {
   String s = "Strings are immutable";
   s = s.concat(" all the time");
   System.out.println(s);
  }
}
```

output:

```
Strings are immutable all the time
```

Example of compareTo()method:

```
public class Test{

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

```
   Integer x = 5;
   System.out.println(x.compareTo(3));
   System.out.println(x.compareTo(5));
   System.out.println(x.compareTo(8));
   }
}
```

output: 1

0

-1

Example of compareToIgnoreCase()method:

```
public class Test {


  public static void main(String args[]) {
   String str1 = "Strings are immutable";
   String str2 = "Strings are immutable";
   String str3 = "Integers are not immutable";

   int result = str1.compareToIgnoreCase( str2 );
   System.out.println(result);

   result = str2.compareToIgnoreCase( str3 );
   System.out.println(result);

   result = str3.compareToIgnoreCase( str1 );
   System.out.println(result);
  }
}
```

output: 0

10

-10

Example of substring() method:

String subs=”television”.substring(2);---->levision

String subs=”television”.substring(2,5);--->lev

Example of toUpperCase() and toLowerCase() method:

```
class Simple{
public static void main(String args[]){ String
 s="Sachin";
 System.out.println(s.toUpperCase());//SACHIN
 System.out.println(s.toLowerCase());//sachin
 System.out.println(s);//Sachin(no change in original)
 }
}
```

```
Output: SACHIN
 sachin Sachin
```

Example of trim() method:

```
class Simple{
public static void main(String args[]){
 String s=" Sachin ";
 System.out.println(s);// Sachin
 System.out.println(s.trim());//Sachin
 }
}
Output: Sachin
      Sachin
```

Example of startsWith() and endsWith() method:

```
class Simple{
public static void main(String args[]){ String
 s="Sachin";
 System.out.println(s.startsWith("Sa"));  //true
 System.out.println(s.startsWith("n"));    //true
 }
}
Output: true
      true
```

Example of charAt() method:

```
class Simple{
public static void main(String args[]){
 String s="Sachin";
 System.out.println(s.charAt(0));//S
 System.out.println(s.charAt(3));//h
 }
}
Output: S
      h
```

Example of length() method:

```
class Simple{
```

```
public static void main(String args[]){
 String s="Sachin";
 System.out.println(s.length());//6
 }
}
```

Output: 6

Example of indexOf()method:

String name="President George Washington"

name.indexOf('p');--->0

name.indexOf('e');--->2

name.laseindexOf('e');--->15

Example of valueOf()method:

String s1=String.valueOf(123); --->"123"

String s2=String.valueOf(123.12);--->"123.12"

2. StringBuffer class

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

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

Commonly used Constructors of StringBuffer class:

StringBuffer(): creates an empty string buffer with the initial capacity of 16.

StringBuffer(String str): creates a string buffer with the specified string.

StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

2.1 Commonly used methods

1. public synchronized StringBuffer append(String s): is used to append the specified string with this string.

2. public synchronized StringBuffer insert(int offset, String s): is used to insert the specified string with this string at the specified position

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.

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.

2.2 Examples

simple example of StringBuffer class by append() method

```
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
}
}
```

output: Hello Java

Example of insert() method of StringBuffer class

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
}
}
```

output: HJavaello

Example of replace() method of StringBuffer class

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
}
}
```

Output:HJavalo

Example of delete() method of StringBuffer class

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
        }
}
```

output:Hlo

Example of reverse() method of StringBuffer class

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
        }
```

```
}
output: olleH
```

Example of capacity() method of StringBuffer class:

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
}
}
```

Output: 34

3. StringBuilder class

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

Commonly used Constructors of StringBuilder class:

StringBuilder(): creates an empty string Builder with the initial capacity of 16.

StringBuilder(String str): creates a string Builder with the specified string.

StringBuilder(int length): creates an empty string Builder with the specified capacity as length.

3.1 Commonly used methods

1.public StringBuilder append(String s): is used to append the specified string with this string.

2.public StringBuilder insert(int offset, String s): is used to insert the specified string with

this string at the specified position. .

3. public StringBuilder replace(int startIndex, int endIndex, String str): is used to replace the string from specified startIndex and endIndex.

4. public StringBuilder delete(int startIndex, int endIndex): is used to delete the string from specified startIndex and endIndex.

5. public StringBuilder 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.

3.2 Examples

Simple program of StringBuilder class by append() method:

The 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
}
}
```

output:Hello Java

Example of insert() method of StringBuilder class

The 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
}
}
```

Example of replace() method of StringBuilder class

The 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
}
}
```

Example of delete() method of StringBuilder class

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
}
}
```

Example of reverse() method of StringBuilder class

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
}
}
```

Example of ensureCapacity() method of StringBuilder class

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

}
}
```

4.StringTokenizer

This 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:

There are 3 constructors defined in the StringTokenizer class.

Constructors	
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:

The 6 useful methods of StringTokenizer class are as follows:

Public method	Description
Boolean hasMoreToken()	Checks if there is more tokens available.
String nextToken()	Return if there is more tokens available.
String nextToken(String delim)	Returns the next token based on the delimiter.
Boolean hasMoreElements()	Checks if there is more elements available.
Object nextElement()	Return if there is more elements available but return types is object.
Int countTokens()	Returns total number of tokens.

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

4.1 Examples

```
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:Nexttokenis:my