CS F213
Object Oriented Programming

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

String Handling in Java
Ch.17 of T2

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

Strings In Java

- String is sequence of characters.
- In Java, strings are treated as objects.
- The String class is used to create a string object.
- String is immutable in nature. It means we can't modify the string once it is created.
- Whenever we change any string, a new instance is created.
- Mutable strings can be created by using StringBuffer and StringBuilder classes.
- We can create String object either by using string literal or using new keyword.

Creating Strings

There are two ways to create a String in Java

a. Using string literal

b. Using new keyword

Using String Literals

Assigning a String literal to a String reference.

Example:

String str1 = "Hello";

String str2 = "Hello";

Here we have not created any string object using the new keyword.

The compiler does that task for us, it creates a string object having the string literal (that we have provided, in

this case it is “Hello”) and assigns it to the provided string reference.

Using new keyword

But if the object already exist in the memory (string constant pool) it does not create a new

Object rather it assigns the same old object to the new reference, that means even though we have two string references above(str1 and str2) compiler only created one string object (having the value “Hello”) and the reference to the pool instance is returned to both the references. For example there are 10 string references that have same value, it means that in memory there is only one object having the value and all the 10 string references would be pointing to the same object.

String str1 = new String("Welcome");

String str2 = new String("Welcome");

This would create two different objects with same string value.

String class Constructors

innovate achieve lead

1) *String()* - This initializes a newly created String object so that it represents an empty character sequence.

2) *String(char[] value)* - This allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

3) *String(char[] value, int offset, int count)* - This allocates a new String that contains characters from a subarray of the character array argument.

4) *String(String original)* - This initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

5) *String(StringBuffer buffer)* - This allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

6) *String(StringBuilder builder)* - This allocates a new string that contains the sequence of characters currently contained in the string builder argument.

String Class Methods:

1) *char charAt(int index)* - Returns the character at the specified index.
2) *int compareTo(String anotherString)* - This method compares two strings lexicographically.
3) *String concat(String str)* - This method appends one String to the end of another. The method returns a String with the value of the String passed into the method, appended to the end of the String, used to invoke this method.
4) *int length()* - Returns the length of this string.
5) *String toLowerCase()* - Converts all of the characters in this String to lower.
6). *String toUpperCase()* - Converts all of the characters in this String to upper.

Ex.

```
class StringDemo
{
    public static void main(String args[])
    {
            String s1="hello";
            System.out.println(s1.length());  // 5
            System.out.println(s1.charAt(0));//index of h is 0
            String s2="job"; String s4="god";
            String s3=s1.concat(s2);System.out.println(s3);
            int c=s1.compareTo(s2); System.out.println(c);//-2
            c=s1.compareTo(s4); System.out.println(c);//1
            System.out.println(s1.toUpperCase());
            String s5="hello"; String s6=new String("hello");
            System.out.println((s1==s5));//true
            System.out.println((s1==s6));//false
    }
}
```

StringBuffer class

StringBuffer supports a modifiable string. As you know, String represents fixed-length,immutable character sequences. In contrast, StringBuffer represents growable and writable character sequences. StringBuffer may have characters and substrings inserted in

the middle or appended to the end. StringBuffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

StringBuffer Constructors

1) *StringBuffer()* - It is the default constructor and reserves room for 16 characters without reallocation.

2) *StringBuffer(int size)* - It accepts an integer argument that explicitly sets the size of the buffer.

3) *StringBuffer(String str)* - The third version accepts a String argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.

Methods

1) *int capacity()* - It is used to return the current capacity.

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

3)*synchronized StringBuffer replace(int startIndex, int endIndex, String str)* - It is used to replace the string from specified startIndex and endIndex.

4) *synchronized StringBuffer reverse()* - It is used to reverse the string.

```
class StringBufferDemo
{   public static void main(String args[])
    {       StringBuffer sb1=new StringBuffer("hello");
            StringBuffer sb3= new StringBuffer("TEMPLE");
            System.out.println(sb1);  //hello
            System.out.println(sb1.capacity());  // 21
            System.out.println(sb1.length());  // 5
            StringBuffer sb2=sb1.insert(1,"tr");
            System.out.println(sb2);//htrello
            sb3=sb3.reverse();
            System.out.println(sb3); //ELPMET
    }
}
```

StringBuilder class

Introduced by JDK 5, StringBuilder is a relatively recent addition to Java's string handling capabilities.

StringBuilder is similar to StringBuffer except for one important difference: it is not synchronized, which means that it is not thread-safe.

The advantage of StringBuilder is faster performance.

However, in cases in which a mutable string will be accessed by multiple threads, and no external synchronization is employed, you must use StringBuffer rather than StringBuilder.

When to use which one :

- If a string is going to remain constant throughout the program, then use a String class object because a String object is immutable.
- If a string can change (example: lots of logic and operations in the construction of the string) and will only be accessed from a single thread, using a StringBuilder is good enough.
- If a string can change, and will be accessed from multiple threads, use a StringBuffer because StringBuffer is synchronous so you have thread-safety.

StringTokenizer class

The java.util.StringTokenizer class allows you to break a string into tokens.

It is simple way to break a string.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc.

```
Import java.util.*;
class STDemo
{
    public static void main(String args[])
    {     String s; String delim;
          s=args[0]; delim=args[1];
          StringTokenizer st1 = new StringTokenizer(s,delim);
          while (st1.hasMoreTokens())
          { System.out.println(st1.nextToken());     }
     }
}
```

```
Import java.util.*;
class STDemo
{
    public static void main(String args[])
    {     String s=“Hello$how$are$you”; String “$”;
          StringTokenizer st1 = new StringTokenizer(s,delim);
          while (st1.hasMoreTokens())
          { System.out.println(st1.nextToken()); }
     }
}
```

Constructors

1)*StringTokenizer(String str)* - Creates StringTokenizer with specified string.

2)*StringTokenizer(String str, String delim)* - Creates StringTokenizer with specified string and delimiter.

3) *StringTokenizer(String str, String delim, boolean returnValue)* - Creates StringTokenizer with specified string, delimiter 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

1)*boolean hasMoreTokens()* - Checks if there are more tokens available.

2) *String nextToken()* - Returns the next token from the StringTokenizer object.

3) *String nextToken(String delim)* - Returns the next token based on the delimiter.

4) *boolean hasMoreElements() –*

Same as hasMoreTokens() method.

5) *Object nextElement()* - Same as nextToken() but its return type is Object.

6) *int countTokens()* - Returns the total number of tokens.

Regular Expressions

innovate achieve lead

- A regular expression is a sequence of characters that forms a search pattern.
- When you search for a data in a text, you can use this search pattern to describe what you are searching for.
- These can be used for '**text search**' as well as '**text replace**' operations.
- In java, **java.util.regex** package supports regular expression processing. It lets you define a pattern for searching or manipulating strings.
- These are mainly used for password and email validation in real life applications.
- Java regex API provides 1 interface and 3 classes in **java.util.regex** package.

Java regex package

The java.util.regex package provides following classes and interfaces for regular expressions:-

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class

Example Regular expressions for some patterns

innovate achieve lead

Pattern	Regular expression
Any string with numeric and alphabets, starting with alphabets and ending with a number	[a-zA-Z][a-zA-Z0-9]*[0-9]
Positive integer does not start with zero	[1-9][0-9]*
String starting with 3 numbers [0-9] and followed by 4 alphabets	[0-9]{3}[a-z]{4}
String starting with **a** or **b**	[ab][a-z]*
Zero or more **a**'s followed by one **b**	a*b

Example

```
import java.util.regex.*;
public class RegexDemo
{
   public static void main(String args[])
   {
   boolean b3 = Pattern.matches("[+]?[0-9]+", "++456");
   System.out.println("Matching status:"+b3);
   }
}
```

Summary

1. String class
2. String constructors and methods
3. StringBuffer
4. StringBuilder
5. StringTokenizer class
6. Regular expressions