STRING HANDLING

1) The String characters - (7)
2) String length - (1)
3) Special string operations
   - String literals
   - String concatenation
   - String concatenation with other data types
   - String conversion and toString()

4) Character Extraction
   - charAt()
   - getBytes()
   - getChars()
   - toCharArray ()

5) String Comparison
   - equals() & equalsIgnoreCase()
   - regionMatches()
   - startsWith() and endsWith()
   - equals() Versus =
   - compareTo()

6) Searching Strings
   - indexOf()
   - lastIndexOf()

7) Modifying a String
   - subString()
   - replace()
   - concat()
   - trim()

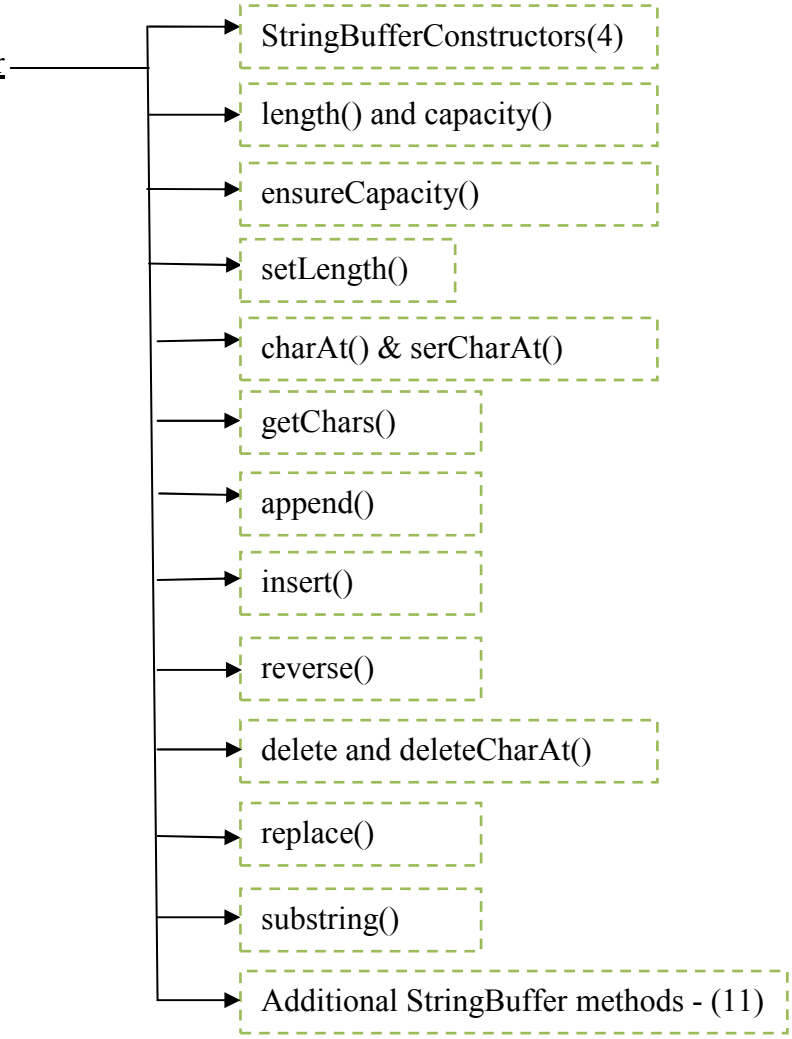
STRING HANDLING

8) Data Conversion Using valueOf() - (4)

9) Changing the Case Of Characters within a String - (2)

10) Additional String Methods - (15)

11) String Buffer



12) String Builder

STRING HANDLING

Let us discussed in detail about the essential about string handling topic :-

1) In JAVA, string is a sequence of characters as like in most other programming languages. But, unlike many other languages that implement string as character arrays, JAVA implements Strings as objects of type "**String**".

2) Implementing strings as built in objects allows JAVA to provide a full complement of features that make string handling convenient.
For Example : JAVA has a method
   - ➔ To compare 2 strings
   - ➔ Search for a substring
   - ➔ Concatenate 2 strings and
   - ➔ Change the case of letters within 2string
   - ➔ Also , **String** objects can be constructed a number of ways, making it easy to obtain a string when needed.

3) JAVA provides 2 options :**StringBuffer** and **StringBuilder.** Both holds Strings that can be modified after they are created and this classes are defined in java.lang package.

(I)THE STRING CONSTRUCTORS :

1) To create an empty string, you call the default constructor.
For EXAMPLE:

```
String s = new String();
```

Will create an instance of String with no characters in it.

2) To create a String initialized by an array of charact here:

```
String(char chars[])
```

For EXAMPLE:

```
char chars[ ]  = {'a','b','c'};
String s = new String (chars);
```

This constructor initializes **s** with the string "abc".

STRING HANDLING

3) You can specify a sub range of a character array as an initializer using the following constructor:

```
String(char chars[],int startIndex,int numChars)
```

Index at which the sub range begins — no.of characters to use

For Example:

```
char chars[ ] = {'a','b','c','d','e','f'};
String s = new String(chars, 2, 3);
```

cde

4) You can construct a String object that contains the same charactersequence as another String object using this constructor:

```
String(String  strObj[])
```

String object

For example:

```
class makestring
{
        public static void main(String args[])
        {
                Char c[]={'J','a','v','a'};
                String s1=new String(c);
                String s2=new String(s1);
                System.out.println(s1);
                System.out.println(s2);
        }
}
```

output

```
Java
Java
```

5) The String class provides constructors that initialize a string when given a byte array.

```
String(byte asciiChars[ ])

String(byte asciiChars[ ], int startIndex,int numChars)
```

STRING HANDLING

Array of bytes specify sub range

```
class substringcons
{
        public static void main(String args[])
        {
                Byte ascii[ ] ={65, 66, 67, 68, 69, 70};
                String s1 = new String(ascii);
                System.out.println(s1);
                String s2 = new String(ascii,2,3);
                System.out.println(s2);
        }
}
```

output

```
ABCDEF
CDE
```

NOTE: In each of these constructors, the byte-to-character conversion is done by using the default character encoding of the platform. In which you can specify the character encoding that determines how bytes are converted to characters.

6) It supports the extended Unicode Character set and ius shown here :

```
String(int codePoints[ ],int startIndex,int numChars);
```

Array that contains unicode code points range that begins no.of chars

7) It supports the new StringBuilder class.

```
String(StringBuilder strBuildObj);
```

This constructs a String from the StringBuilder passed in strBuildObj.

II) STRING LENGTH

The length of a string is the number of characters that it contains.

Syntax:

```
int length()
```

EXAMPLE:

```
char chars[ ] = {'a','b','c'};
```

OUTPUT

```
String s = new String (chars);                    3

System.out.println(s.length());
```

III) SPECIAL STRING OPERATIONS

These operations include

- ➔ The automatic creation of new string instances from string literals.
- ➔ Concatenation of multiple String objects by use of the + operator.
- ➔ The conversion of other data types to a string representation.

There are explicit methods available to perform all these functions, but JAVA does them automatically as a convenience for the programmer and to add clarity.

1) String Literals
   (a) The earlier examples showed how to explicitly create a String instance from an array of characters by using the new operator.
   (b) However there is an easier way to do this using a string literal in your program, Java automatically constructs a String object.
   (c) Thus you can use a string literal to initialize a string object.
   (d) For EXAMPLE:

```
char chars[ ]  = {'a','b','c'};
String s1 = new String (chars);
String s2 = "abc";                // use string literal
```

Because a String object is created foe every string literal, you can use a string literal any place you can use a String object.

For EXAMPLE: you can call methods directly on a quoted string as if it were an object reference, as the following statements shows :

```
System .out.println("abc".length());
```

2) String Concatenation
   The + operator, which concatenates 2 strings, producing a String object as the result.This allows you to chain together a seres of + operations.
   For Example:

```
String age = "9";
String s = "he is" + age + "years old." ;
System .out.println(s);
```

OUTPUT

```
he is 9 years old.
```

STRING HANDLING

One practical use of string concatenation is found when your are creating very long strings. Instead of letting long strings wrap around within your source code, you can break them into smaller pieces, using the + to concatenate them.

For EXAMPLE:

```
class concat
{
        public static void main(String args[])
        {
                String longstr = "This could have been " +
                                  "a very long line that would have " +
                                 "wrapped around. But string concatenation" +
                                "prevents this.";
                System.out.println(longstr);
        }
}
```

3) String concatenation with other data types:
   You can concatenate strings with other types of data.
   For EXAMPLE: Be careful when you mix with other types of operations with string concatenation expressions.

   Consider the following :

```
String s = "four:" +2 + 2;
System.out.println(s);
```

   this fragments displays four: 22 rather than four: 4

To complete the addition first, you must use parenthesis, like this:

```
String s = "four:" + (2 + 2);
```

4) String conversion and toString():

IV) CHARACTER EXTRACTION

The String class provides a no.of ways in which characters can be extracted from a String object.

1) charAt() : To extract a single character from a String.you can refer directly to an individual character via the charAt() method.

STRING HANDLING

Syntax :

```
char charAt(int where)
```

Example : char ch
ch=”abc”.charAt(1);
it assigns the value “b” to ch.

2) getChars( ) : if you need to extract more than one characters one at a time,you can use the getChars() method.

Syntax :

```
void getChars(int sourceStart,int sourceEnd,char target[ ] ,int targetStart)
```

Index of the beginning Of the substring

index i.e., one past the end of the desired substring

array that will receive the characters is specified by target

Index within target at which the substring will be copied is passed in targetStart

Example :

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

        String s="this is a demo of the getchars method.";
        int start=10;
        int end=14;
        char buf[ ] =new char[end-start];
        s.getChars(start,end,buf,0);
        System.out.println(buf);
        }
}
```

STRING HANDLING

3) getBytes( ) : It uses the default character-to-byte conversions provided by the platform.

Syntax :

```
byte[ ] getBytes( )
```

4) toCharArray( ) : If you want to convert all the characters in a String object into character array, the easiest way is to call toCharArray().

➔ It returns acters for the entire string.

Syntax :

```
char[ ] toCharArray( )
```

Note: It is possible to use getChars() to achieve the same result.

V) STRING COMPARISION

The string class includes several method that compare strings or substrings within strings.

1) Equals( ) and equalsIgnoreCase( ) : Compare two Strings

Syntax :

```
boolean equals(Object str)
```

Returns : **True** -> if the string contain the same characters in the same order, otherwise

False.

Note: To perform a comparision that ignores case differences,call **equalsIgnoreCase( )** .

General Form :

```
boolean equalsIgnoreCase(Object str)
```

Example:

```
class equalsdemo
{
public static void main(String args[ ])
{
String s1="Hello";
String s2="Hello";
String s3="Good-bye";
String s1="HELLO";
System.out.println(s1 + "equals " +s2+"->"+s1.equals(s2));
System.out.println(s1 + "equals " +s3+"->"+s1.equals(s3));
System.out.println(s1 + "equals " +s4+"->"+s1.equals(s4));
System.out.println(s1 + "equalsignorecase "
```

OUTPUT

```
Hello equals Hello - > true

Hello equals Good-bye - > false

Hello equals HELLO - > false

Hello equalsignorecase HELLO - > true
```

STRING HANDLING

2) regionMatches( ) : Compares a specific region inside a string with another specific region in another string.

Syntax :

```
boolean regionMatches(int startIndex,String str2,int str2StastIndex,int numChars)
```

(or)

```
boolean regionMatches(boolean ignoreCase,int startIndex,String str2,int str2StastIndex,int numChars)
```

- Specifies the index at which the region begins
- string being compared
- Comparision will start Within str2 is specified here
- Length of the substring being compared

3) **startsWith( ) and endsWith( )** :

- This method determiners whether a given string begins with a specified string
- this method determiners whether a given string ends with a specified string

Syntax :

```
boolean startsWith(String str)
```

```
boolean endsWith(String str)
```

(or)

```
boolean startsWith(String str,int startIndex)
```

RETURNS : If String matches -> true
Otherwise false.

EXAMPLE : "foobar".endsWith("bar") → true
"foobar".startWith("foo") → true
"foobar".startWith("bar" , 3) → true

4) equals( ) Versus = = :

compares a characters inside a String object	compares two object references to see whether they refer to the same instance.

Example :

```
class equalsnotequalto
{
public static void main(String args[ ])
{
String s1="Hello";
String s2=new String(s1);
System.out.println(s1 + "equals " +s2+"->"+s1.equals(s2));
System.out.println(s1 + "= = " +s2+"->"+(s1= =s2));
}
}
```

OUTPUT

```
Hello equals Hello -> true

Hello = = Hello -> false
```

5) compareTo( ) : It is not enough to simply know whether 2 strings are identical.For sorting applications, you need to know which is less than,equal to,greather than the next.
   - A string is less than another if it comes before the other in dictionary order.
   - A string is greater than another if it comes after the other in dictionary order.

Syntax:

```
int compareTo(String str)
```

VI) SEARCHING STRINGS

The String class provides 2 methods that allow you to search a string for a specified character or substring:

- indexOf( ) : Searches for the first occurrence of a character or substring.
- lastIndexOf( ) : Searches for the last occurrence of a character or substring.

Syntax:

```
int indexOf(int ch)
```

```
int lastIndexOf(int ch)
```

To search for the first or last occurrence of a substring, use

- int indexOf(String str)
- int lastIndexOf(String str)

STRING HANDLING

You can specify a starting point for the search using these forms :

- int indexOf(String str,int startIndex)
- int lastIndexOf(String str,int startIndex)

VII) MODIFYING A STRING

1) substring( ) : You can extract a substring using substring( )
It has 2 forms :
(a)
```
String substring(int startIndex)
```
(b)
```
String substring(int startIndex,int endIndex)
```

2) concat( ) : You can concatenate 2 strings using concat( )
syntax :
```
String concat(String str)
```

example : String s1 = "one" ; (or) String s1 = "one" ;
String s2 = s1.concat("two"); String s2 = s1+ "two";

3) replace( ) : It has 2 forms :
(a) The first replaces all occurrences of one character in the invoking string with another character.
```
String replace(char original,char replacement)
```
Example : String s = "Hello".replace('l','w');
Output : Hewwo

(b) It replaces one character sequence with another.
```
String replace(charSequence original,charSequence replacement)
```

4) trim( ) : It returns a copy of the invoking string from which any leading and trailing whitespace has been removed.
```
String trim()
```

STRING HANDLING

Syntax :

Example : String s = " Hello world ".trim( ); **OUTPUT:** Hello World

VIII) DATA CONVERSION USING valueOf( )

- This method converts data from its internal format into a human readable form.
- It is a static method that is overloaded within String for all of Java's built in types so that each type can be converted properly into a string.
- Its general forms are as follows :

```
static String valueOf(double num)
```

```
static String valueOf(long num)
```

```
static String valueOf(Object ob)
```

```
static String valueOf(char chars[ ])
```

IX) Changing The Case Of Characters Within a String

- String toLowerCase( ) : Converts all the characters in a string from uppercase to lowercase.
- String toUpperCase( ) : Converts all the characters in a string from lowercase to uppercase.

X) ADDITIONAL STRING METHODS : (15 Methods)

(REFER THE TEXT BOOK JAVA COMPLETE REFERENCE PAGE NO : 374)

XI) STRING BUFFER

String Buffer will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

1) StringBuffer Constructors : It has 4 constructors :

   i. StringBuffer( ) : reserves room for 16 characters .

ii. StringBuffer(int size) : accepts a String arg that explicitly sets the size of the buffer.

iii. StringBuffer(String str) : accepts a String arg that sets the Initial contents of the StringBuffer allocates room for additional characters without reallocations.

iv. StringBuffer(CharSequence chars): creates an object that contains the character sequence contained in chars.

2) length( ) and capacity( ) :

- int length( ) : current length of a StringBuffer can be found.
- Int capacity( ) : total allocated capacity can be found through the capacity( ) method.

Example :

```
class length
{
public static void main(String args[ ])
{
StringBuffer sb = new StringBuffer( " Hello " );
System.out.println("buffer = "+sb);
System.out.println("length = "+sb.length());
System.out.println("capacity = "+sb.capacity());
}
}
```

OUTPUT

```
buffer = Hello

length = 5

capacity = 21 (16 added
automatically + 5(hello))
```

3) ensureCapacity( ) : To set the length of the buffer within a StringBuffer object, use setLength( )

syntax :

```
void setLength(int len)
```

note : len value must be non negative.

4) charAt and setCharAt( ) : The value of a single character can be obtained from a StringBuffer via the charAt( ).
You can set the value of a character within a StringBuffer using setCharAt( )

Syntax :

```
char charAt(int where)
```

```
void  setCharAt(int where,char ch)
```

STRING HANDLING

Example :

```
class setchar
{
public static void main(String args[ ])
{
StringBuffer sb = new StringBuffer("Hello");
System.out.println("Buffer before="+sb);
System.out.println("charAt(1)  before ="+sb.charAt(1));
sb.setCharAt(1,'i');
sb.setLength(2);
System.out.println("Buffer after="+sb);
System.out.println("charAt(1)  after ="+sb.charAt(1));
}
}
```

OUTPUT

```
buffer before = hello

charAt(1) before = e

buffer after = hi

charAt(1) after = i
```

5) getChars( ) : To copy a substring of a StringBuffer into an array, use the getChars() method.

Syntax :

```
void getChars(int sourceStart, int sourceEnd,char target[ ],int targetStart)
```

6) append( ) : It concatenates the string representation of any other type of data to the end of the invoking StringBuffer object.

Syntax :

```
StringBuffer append(String str)
```

```
StringBuffer append(int num)
```

```
StringBuffer append(Object obj)
```

7) insert( ) : It inserts one String into another.

Syntax :

```
StringBuffer insert(int index,String str)
```

```
StringBuffer insert(int index,char ch)
```

```
StringBuffer insert(int index,Object obj)
```

8) reverse( ) :You can reverse the characters within a StringBuffer object.

Syntax :

```
StringBuffer reverse()
```

9) delete and deleteCharAt( ) : You can delete characters within a StringBuffer by using these methods.

Syntax:

```
StringBuffer delete(int Startindex,int endIndex)
```

```
StringBuffer delete CharAt(int loc)
```

10) replace( ) : You can replace one set of characters with another set inside a StringBuffer object.

Syntax :

```
StringBuffer replace(int Startindex,int endIndex,String str)
```

11) substring( ) : You can obtain a portion of a StringBuffer by calling these method. It has 2 forms :

- String substring(int startIndex)
- String substring(int startIndex,int endIndex)

12) Additional StringBuffer Methods : (11methods)

XII) STRINGBUILDER

- It is identical 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 you are using multithreading , you can use StringBuffer rather than StringBuilder.
