**6.1 String Fundamentals**

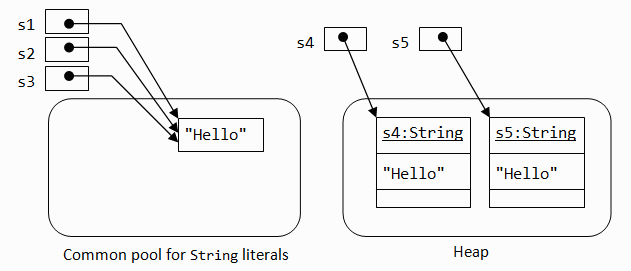
* String is a sequence of characters.
* Unlike other languages that implements strings as character arrays, Java implements strings as object of type String.
* String is immutable object, the contents of strings instance cannot be altered after it has been created.
* The String class is declared *final*, which means that can’t be sub classed.

**6.1.1 Constructing Strings**

* Java maintains special memory called: String literal pool
* Is a pool of unique Strings: avoid duplicate
* **Using**
  + String literal
  + *new* keyword

1. **String literal**

* Strings are maintained in String literal Pool of a Heap. ( String constant pool)
* ***String literal Pool*** is a pool of unique Strings, It avoids duplicate String objects in heap memory.
* **String str =”Hello”;**
* Whenever compiler encounters a string literal, it creates a String object with given string value.
* JVM performs String pool check with following actions
  + *first checks String pool for the string literal.*
  + *if the literal is exist, return its reference.*
  + *If literal not exist, create the new literal in pool.*

****

1. **Using new keyword**

* **String str = new String( );**
* JVM doesn't perform String pool check if you create object using new operator.
* Always creates new string object.

**Example of Strings created**

|  |
| --- |
| String s1 = “Hello” ; // String literal |
| String s2 = “Hello”; // String literal |
| String s3 = “Hello”; // String literal |
| String s4 = new String(“Hello”); // String Object |
| String s5 = new String(“Hello”); // String Object |

**6.2 The String Constructors**

* Different forms of Constructors.

1. **String ( )**
   1. Creates object with no characters
   2. example: String str = new String( );
2. **String (char charArr[ ]) or**
3. **String (char charArr[ ], int startindex, int numchars)**
   1. Creates string using char Array
   2. Example:

char charArr[ ] = { ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’ };

String str = new String( charArr );

System.out.println( str ); // output: abc

String str = new String( charArr, 2, 3);

System.out.println( str ); // output: cde

1. **String (String strObj)**
   1. Creates string using another string object
   2. Example

String s1 = new String (“abc”); //uses constructor

String s2 = new String (s1); //uses constructor

**6.3 Three String-Related Language Features**

* Java supports three especially useful string features within the syntax of the language.
* They are:
  1. **String literal**
  2. **String concatenation**
  3. **Overriding toString( )**

**a. String literal**

* A string literal is created by specifying a quoted string. For each string literal in your program, Java automatically constructs a String object.

***Example:*** String str = “MCA in RNSIT”;

**b. String concatenation:**

* In java, string concatenation forms a new string that is the combination of multiple strings.
* There are two ways to concatenate string in java:
  + 1. Using operator: +(string concatention) operator
    2. Using method : concat( )

Example 1:

class TestStringConcatenation1 {

public static void main(String args[ ]) {

*String s = "Sachin" + " Tendulkar";*

*System.out.println(s);*

}

}

* The String concat() method concatenates the specified string to the end of current string.

Example 2:

class TestStringConcatenation3 {

public static void main(String args[]) {

*String s1="Sachin ";*

*String s2="Tendulkar";*

*String s3=s1.concat(s2);*

*System.out.println(s3);*

}

}

* The string concatenation operator can concat not only string but primitive values also.

Example 3:

class TestStringConcatenation2 {

public static void main(String args[]) {

int age=36;

String str=“he is “ + age + “years old”;

System.out.println(str);

}

}

**c.Overriding toString( ):**

* If you want to represent any object as a string, toString() method comes into existence.
* The toString() method returns the string that describes the object.
* 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.

Class Student

{

int rollno;

String name;

Student(int rollno, String name){

this.rollno=rollno;

this.name =name;

}

public String toString( ) // *overriding the toString() method*

{

return rollno + “ “+ name;

}

public static void main(String[] args){

Student s =new Student(101,”Anil”);

System.out.println(s);

}

}

**Output:**

101 Anil

**6.4 The Length( ) Method**

* The method length( ) returns the length of the string.
* A string’s length is the number of characters that it contains.
* General form

*int length( )*

* Example:

String str =”*RNSIT”*

System.out.println(***str.length( )*** ); // displays 5

**6.5 Obtaining the characters within a string**

* String class provides three ways in which character can be obtained from a string.

1. charAt( )
2. getChars( )
3. toCharArray( )
4. **charAt( )**
   * To obtain single character from a string.
   * Syntax:

char **charAt (** int **index )**

* + *returns* a single character from a string at specified index*.*
  + Example:

char ch;

ch = "ab**z**de".charAt( **2** ); *//assigns the value “****z****” to ch.*

1. **getChars( )**

* *Syntax:*

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

* *reads the set of characters from index* ***sourceStart*** *to the index* ***sourceEnd*** *and store in the* ***target*** *character array from a given index* ***targetStart****.*
* *Example:*

String str =”Programming is both art and science”;

char [ ] buf =new char[7]

str.getChars(2, 9, buf, 0);  *// “ogrammi”*

1. **toCharArray( )**

* Convertsall characters in a string object into a character array.
* **Syntax**

**char[ ] toCharArray()**

* Returns an array of characters for the entire string
* **Example:**

char [ ]charArr = “**RNS**”.***toCharArray()*** *// charArr = [‘R’,’N’,’S’]*

for( char ch: charArr)

System.out.println(**charArr**);

1. **getBytes**

byte**[]** **getBytes()**

* Syntax:
* returns the array of bytes by encoding the source String.
* This method throws an exception **UnsupportedEncodingException,** and it should be surrounded with try-catch block.
* Example:

byte [] b = "Aa".getBytes(); *// b = [ 65, 97 ]*

**for**( **byte** x : b )

System.*out*.println( x ); *// prints 65 97*

**6.6 String comparison**

* Different methods used for string comparison are
  1. equals() and equalsIgnoreCase()
  2. regionMatches()
  3. startsWith() and endsWith()
  4. equals() VS ==
  5. compareTo()

**6.6.1 equals( ) and equal IgnoreCase( )**

* Syntax :

boolean **equals(String str)**  // case sensitive

boolean ***equalsIgnoreCase* ( String str )**  // ignore the case

* Compares **source** string with target strings **str** for equality.
* returns **true** if the strings contain the **same** characters in the same order, **false** otherwise.
* Example:

System.out.println(“Hello”.equals (“Hello”)) // true

System.out.println(“Hello”.equals (“HELLO”)) // false

System.out.println(“Hello”.equalsIgnoreCase (“HELLO”)) // true

**6.6.2 regionMatches( )**

* *compares* a specific region inside a string with another specific region in another string.
* *It has two overloaded forms:*

*boolean* regionMatches ( int *startIndex*, String *str2*, int *str2StartIndex*, int *length*)

*boolean* regionMatches ( boolean *IgnoreCase,* int *startIndex*, String *str2*, int *str2StartIndex*,

int *length*)

* **Parameter Values**

|  |  |
| --- | --- |
| ***startIndex*** | specifies the index at which the region begins within the invoking String  object. |
| ***str2*** | The String being compared is specified by str2. |
| ***str2StartIndex*** | The index at which the comparison will start within str2 . |
| ***Length*** | The length of the substring being compared is passed in numChars. |
| ***IgnoreCase*** | if ignoreCase is true, the case of the characters is ignored. Otherwise,  case is significant. |

* **Example**

Class CompareRegion{

public static void main(String [ ] args){

String str1 =”Standing at river’s edge”;

String str2 =”Running at river’s edge”;

if(str1.regionMatches(9, str2, 8, 12))

System.out.println(“Regions Matches”);

If(!str1.regionMatches(0, str2, 0, 12))

System.out.println(“Regions do not Match”);

}

}

**6.6.3 startsWith( ) and endsWith( )**

* The **startsWith**() method determines whether a given String begins with a specified string.
* The **endsWith**() determines whether the String in question ends with a specified string.
* They have the following general forms:

boolean *startsWith* ( String str )

boolean *endsWith* ( String str )

* If *the* string matches, true is returned. Otherwise, false is returned.
* **Example**

"Foobar"**.endsWith**("bar"); // true

"Foobar"**.startsWith**("Foo"); // true

**6.6.4 equlas( ) vs ==**

* **equals( ): *Compares the characters*** inside a String object.
* *The* **==** *operator* ***compares the references*** to see ***whether both refer to the same instance.***
* *Example*

String s1 = **new** String ("ABC");

String s2 = **new** String ("ABC");

System.out.println(**s1.**equals(**s2**) ); // true

System.out.println( **s1==s2** ); //false

**6.6.5 compareTo( )**

* **Syntax:**

int compareTo (String str)

* Compares source string with target string **str.**
* The result of the comparison is one of the following:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| Less than zero | the invoking string is less than str. |
| Greater than zero | the invoking string is greater than str. |
| Zero | then the two strings are equal. |

* Used in conditions to sort strings
* **Example:**

**class StringSort {**

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

String list[] = { "Java", "is", "Fun", "for", "all" };

**for**( int j = 0; j < list.length; j++)

**{**

**for** ( int i = j + 1; i < list.length; i++)

**{**

**if ( list[i]. compareTo( list[j]) < 0) {**

String t = list[ j ] ;

list[ j ] = list[ i ] ;

list[ i ] = t;

**}**

**}**

**}**

for ( int i =0; i < list.length; i++ ) {

System.out.println( list[i] );

**}**

**}**

**6.7 using indexOf( ) and lastIndexOf( )**

* Methods for searching String

|  |  |
| --- | --- |
| **int *indexOf* ( *searchString*, [*startIndex*])** | returns the position of ***first occurrence*** *of*  ***searchString***. |
| **int *lastIndexOf* ( *searchString*, *start* )** | returns the position of ***last occurrence*** *of*  ***searchString***. |

* *Returns the position of* ***searchString*** *found in source string.*
* *Second parameter is optional which tells position from where search to begin.*
* *Both* method returns **-1** if string is not found.
* *Both methods are case sensitive*
* *Example*

String str = "alpha beta gamma theta zeta”

*System.out.prinln(The first index of t is “+str.****indexOf****(‘t’); // returns 8*

*System.out.prinln* (*The last index of t is “+str.****lastIndexOf****(‘t’)*); // returns 25

**6.8 Changing the case of characters within a string**

* Returns a copy of this string converted to lowercase or uppercase.
* If no conversions are necessary, these methods return the original string.
* General form:

String *toLowerCase( )*

String *toUpperCase( )*

* Example:

class StringUpperCaseDemo {

public static void main(String[] args) {

String name = "Rns";

System.out.println("Name in Upper Case: " + name.***toUpperCase()*** );

System.out.println("Name in Lower Case: " + name.***toLowerCase*()** );

System.out.println("Original name: " + name);

}

}

Output:

Name in Upper Case : RNS

Name in Lowercase: Rns

Orinigal Name: Rns

**6.8 Obtaining a modified String**

Different methods for searching a string

a) substring()

b) concat()

c) replace()

d) trim()

e) valueOf()

a) **substring()**

* ***Syntax***

Form1: **public String substring(**int **beginIndex)**

Form2: **public String substring(**int **beginIndex,** int **endIndex)**

* returns a new sub string extracted from the source string from index beginIndex till the index specified by endindex .
* *Example:* Write program to check email belongs to which domain (like: gmail, yahoo,rediff .etc)

class Demo {

public static void main(String args[]){

String email = "Rns@rediffmail.com";

int startIndex = email.indexOf("@") + 1;

int lastIndex = email.indexOf(".") ;

System.*out*.print("Domain Name :" );

System.*out*.println(email.substring(startIndex, lastIndex));

}

}

**b) concat( )**

* ***Syntax***:

**String** concat(String s)

* appends one String to the end of another.
* Example:

String s = "Strings are immutable";

s = s.concat(" all the time");

System.out.println(s);

**c) replace( )**

* ***Syntax***

public String **replace**(char oldChar, char newChar)

* It returns a string derived from this string *by replacing every occurrence of* ***oldChar*** *with* ***newChar.***
* ***Example:***

import java.io.\*;

public class Test{

public static void main(String args[]){

String Str = new String("Welcome to RNSIT");

System.out.print("Return Value :" );

System.out.println(Str.replace('o', 'T'));

System.out.print("Return Value :" );

System.out.println(Str.replace('l', 'D'));

}

}

**d) trim( )**

* returns a copy of the string, with leading and trailing whitespace omitted.
* Syntax:

**public String trim()**

* **Example:**

public class Test{

public static void main(String args[]){

String Str = " Welcome to rnsit.ac.com ";

System.out.print("Return Value :" );

System.out.println(Str.trim( ) );

}

}

**e)valueOf( )**

* The valueOf method returns the relevant Number Object holding the value of the argument passed. The argument can be a primitive data type, String, etc.
* This method is a static method. The method can take two arguments, where one is a String and the other is a radix.
* Syntax:
* static Integer valueOf(int i)
* static Integer valueOf(String s)
* Example

class Test{

public static void main(String args[]){

Integer x =Integer.valueOf(9);

Double c = Double.valueOf(5);

Float a = Float.valueOf("80");

Integer b = Integer.valueOf("444",16);

System.out.println(“x=” + x); // 9

System.out.println(“c=” + c); // 5.0

System.out.println(“a=” + a); //80.0

System.out.println(“b=” + b); //1092

}

}

**6.9 StringBuffer and String Builder.**

* StringBuffer and StringBuilder is a mutable class; means one can change the value of the object
* The object created through StringBuffer and StringBulder is stored in the heap.
* StringBuffer has the same methods as the StringBuilder, but each method in StringBuffer is synchronized
* Due to this it does not allow two threads to simultaneously access the same method. Each method can be accessed by one thread at a time.
* String Buffer can be converted to the string by using

