**String:**

String is basically an object that represents sequence of characters. An array of characters works same as String

Char ch[] = {‘r’,’a’,’m’,u’};

String s = new String(ch);

is same as

String s =”ramu”;

String class provides lot of methods to perform operations such as compare(), concat(), equals(), length(), substring() ….etc

Java.lang.String class implements Serializable, Comparable and CharSequence interfaces

CharSequence Interface – used to represent sequence of characters

String, StringBuffer and StringBuilder classes implement it i.e. we can create Strings using these three classes

Two ways to create String object - String literal and by new keyword

String literal: created by using double quotes

String s =”welcome”;

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 is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1 =”welcome”;

String s2=”welcome”; //it doesn’t create new instance

In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance

Java uses the concept of String literal to make java memory efficient (because no new objects are created if it exists in the string constant pool)

By new keyword:

String s = new String(“welcome”); //creates two objects and one reference variable

JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

Below is an example of String

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

**char** ch[] = {'r','a','m','u'};

String s2 = **new** String(ch);

String s3 = **new** String("ramu");

System.***out***.println(s1+" "+s2+" "+s3);

}

}

Immutable String: Strings are immutable. Immutable means unmodifiable or unchangeable

Once String is created its data or state can’t be changed but a new String is created.

Below is an example where string value didn’t change even after concatenation

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

s1.concat("kumar");

System.***out***.println(s1);

}

}

Below is an example where new object is created for String s1 to concatenate

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

s1 = s1.concat("kumar");

System.***out***.println(s1);

}

}

In above case “s1” points to “ramukumar” but “ramu” object is not modified

Below are the features which makes String object immutable

ClassLoader – uses String objects as argument. If String object is modifiable then value might be changed and class that supposed to be loaded might be different. To avoid this string is immutable

Thread Safe – since String is immutable we don’t need to take care of synchronization that is required while sharing an object across multiple threads

Security - As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable. This can make the application program more secure.

Heap space - The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space efficiently.

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

String Comparison:

Three ways – By equals() method, By using == operator, By comapreTo() method

By equals() method example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 = "ramu";

**if**(s1.equals(s2)) {

System.***out***.println("strings are equal");

}**else** {

System.***out***.println("Strings are not equal");

}

}

}

Another By equals() method example with ignore case:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 = "RAMU";

**if**(s1.equalsIgnoreCase(s2)) {

System.***out***.println("strings are equal");

}**else** {

System.***out***.println("Strings are not equal");

}

}

}

By == operator example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 = "ramu";

**if**(s1 == s2) {

System.***out***.println("strings are equal");

}**else** {

System.***out***.println("Strings are not equal");

}

}

}

By comapreTo() method example:

If String1 and String2 are same it returns 0

If String1 is greater than String2 it returns positive value (1)

If String1 is less than String2 it returns negative value (-1)

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 = "ramu";

String s3 = "ram";

**int** a1 = s1.compareTo(s2);

**int** a2 = s1.compareTo(s3);

**int** a3 = s3.compareTo(s1);

System.***out***.println(a1+" "+a2+" "+a3);

**if**(a1 == 0) {

System.***out***.println("String s1 and s2 are equal");

}

}

}

Another By comapreTo() method with ignore case:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 = "RAMU";

String s3 = "ram";

**int** a1 = s1.compareToIgnoreCase(s2);

**int** a2 = s1.compareToIgnoreCase(s3);

**int** a3 = s3.compareToIgnoreCase(s1);

System.***out***.println(a1+" "+a2+" "+a3);

**if**(a1 == 0) {

System.***out***.println("String s1 and s2 are equal");

}

}

}

String concatenation: forms a new String that is combination of multiple strings

Two ways – By + operator , By concat() method

By + operator:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu" + "kumar";

System.***out***.println(s1);

}

}

Another By + operator:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 = 30 + "ramu" + "kumar" + 50;

System.***out***.println(s1);

}

}

By concat() method:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 ="kumar";

String s3 =s1.concat(s2);

System.***out***.println(s3);

}

}

String concatenation using StringBuilder class: provides append() method to perform concatenation operation. Most popular and fastest way to concatenate strings. It is mutable class which means values stored in StringBuilder objects can be updated or changed

Example:

**package** com.lokesh;

**import** java.util.Scanner;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuilder s1 =**new** StringBuilder("ramu");

StringBuilder s2 =**new** StringBuilder("kumar");

StringBuilder s3 =s1.append(s2);

System.***out***.println(s3);

}

}

String concatenation using String.format() method: allows to concatenate using format specifier ‘%s’ followed by string values or objects

Example:

**package** com.lokesh;

**import** java.util.Scanner;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 ="kumar";

String s3 =String.*format*("%s%S",s1,s2);

System.***out***.println(s3);

}

}

String concatenation using String.join() method:

**package** com.lokesh;

**import** java.util.Scanner;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

String s2 ="kumar";

String s3 =String.*join*("",s1,s2);

System.***out***.println(s3);

}

}

String concatenation using StringJoiner class: StringJoiner class has all the functionalities of string.join() method. In advance its constructors can also accept original arguments

Example:

**package** com.lokesh;

**import** java.util.Scanner;

**import** java.util.StringJoiner;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringJoiner s1 = **new** StringJoiner(" ");

s1.add("ramu");

s1.add("kumar");

System.***out***.println(s1);

}

}

String concatenation using Collectors.joining() method:

**package** com.lokesh;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.StringJoiner;

**import** java.util.stream.Collector;

**import** java.util.stream.Collectors;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

List<String> lst =Arrays.*asList*("abc","def","xyz");

String s1 =lst.stream().collect(Collectors.*joining*(":"));

System.***out***.println(s1);

}

}

Substring: A part of String is called substring.

Two methods to do substring

Public String substring (int startindex)

Public String substring (int startindex, int endindex)

Example:

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu kumar";

System.***out***.println("substring using only start index "+s1.substring(3));

System.***out***.println("substring using start index and end index "+s1.substring(2,8));

}

}

String class methods: provides lot of built in methods that are used to manipulate strings

toUpperCase() and toLowerCase() method: toUpperCase() converts string to uppercase while toLowerCase() converts to lowercase

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="Ramu";

System.***out***.println(s1.toUpperCase());

System.***out***.println(s1.toLowerCase());

}

}

Trim() method: eliminates white spaces before and after string

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 =" Ramu ";

System.***out***.println(s1.trim());

}

}

StartsWith() and endsWith() : checks whether String starts with letter passed as argument and in same way checks ends with letter passed as arguments if correct then it gives true otherwise false

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="Ramu";

System.***out***.println(s1.startsWith("R")); //true

System.***out***.println(s1.endsWith("s")); //false

}

}

charAt() method: returns character at specific index

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramukumar";

System.***out***.println(s1.charAt(0)); // returns 'r'

System.***out***.println(s1.charAt(4)); //returns 'k'

}

}

length() method: returns length of specified string

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 ="ramu";

System.***out***.println(s1.length()); // returns 4

}

}

String valueOf() method: converts type such as int, long, float, double, boolean, char and char array to string

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

**int** a =20;

String s1 =String.*valueOf*(a);

System.***out***.println(s1+10); // display 2010

}

}

replace() method: replaces all occurences of first sequence of character with second sequence of character

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

String s1 = "ramu is playing";

String s2 =s1.replace("ramu", "raju");

System.***out***.println(s2);

}

}

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

StringBuffer class append() method: concatenates the given argument with string

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuffer s1 =**new** StringBuffer("ramu ");

s1.append("kumar");

System.***out***.println(s1);

}

}

StringBuffer class insert() method: insert the given string with this string at given position

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuffer s1 =**new** StringBuffer("ramu");

s1.insert(2, "raju");

System.***out***.println(s1); //rarajumu

}

}

StringBuffer class replace() method: replaces the given string with provided string from specified begin and end index

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuffer s1 =**new** StringBuffer("ramu");

s1.replace(0, 2, "uuu");

System.***out***.println(s1); //uuumu

}

}

StringBuffer class delete() method: deletes the String from specified begin index to end index

Index starts from 1

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuffer s1 =**new** StringBuffer("ramu");

s1.delete(1, 2);

System.***out***.println(s1); //ru

}

}

StringBuffer reverse() method: reverses the current string

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuffer s1 =**new** StringBuffer("hello");

s1.reverse();

System.***out***.println(s1); //olleh

}

}

StringBuilder class: used to create mutable (modifiable) String. StringBuilder is same as StringBuffer expect it is non-synchronized

Just like StringBuffer class, StringBuilder class also have append(), insert(), replace(), delete(), reverse() and other methods

In below I have covered all method examples in one program

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringBuffer s1 =**new** StringBuffer("hello");

//append

s1.append("world");

System.***out***.println(s1); //helloworld

//insert

s1.insert(0, "ooo");

System.***out***.println(s1); //ooohelloworld --- String s1 is changed to helloworld because it is modifiable

//replace

s1.replace(5, 8,"oooo");

System.***out***.println(s1); //oooheooooworld

//delete

s1.delete(0, 3);

System.***out***.println(s1); //heooooworld

//reverse

s1.reverse();

System.***out***.println(s1); //dlrowooooeh

}

}

The difference between String and StringBuffer is StringBuffer is faster

The difference between StringBuffer and StringBuilder is StringBuilder is faster

Creating immutable class: there are many immutable classes like Boolean, Byte, Short, Int…etc. In short all wrapper classes and String class are immutable. We can also create immutable class by creating final class that have final data members. Below is example

**package** com.lokesh;

**public** **final** **class** HelloWorld {

**final** String name;

**public** HelloWorld(String name) {

**this**.name =name;

}

**public** String getname() {

**return** name;

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

HelloWorld hw =**new** HelloWorld("ramu");

String s1 =hw.getname();

System.***out***.println(s1);

}

}

Java toString() method: if you want to represent any object as String , toString() method comes into existence.

Without toString() method: hw prints the hashcode values of the objects it doesn’t print values of these object

**package** com.lokesh;

**public** **class** HelloWorld {

**int** id;

String name;

**public** HelloWorld(**int** id, String name) {

**this**.id=id;

**this**.name =name;

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

HelloWorld hw =**new** HelloWorld(1,"ramu");

System.***out***.println(hw); [//com.lokesh.HelloWorld@3d012ddd](mailto://com.lokesh.HelloWorld@3d012ddd) ….compiler writes hw.toString() internaly

}

}

toString() method: values of object are printed

**package** com.lokesh;

**public** **class** HelloWorld {

**int** id;

String name;

**public** HelloWorld(**int** id, String name) {

**this**.id=id;

**this**.name =name;

}

@Override //overiding tostring() method

**public** String toString() {

**return** "HelloWorld [id=" + id + ", name=" + name + "]";

}

}

**package** com.lokesh;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

HelloWorld hw =**new** HelloWorld(1,"ramu");

System.***out***.println(hw); //HelloWorld [id=1, name=ramu]....compiler write hw.toString() internally

}

}

StringTokenizer: allows you to break string into tokens

Useful methods

Boolean hasMoreTokens() – checks if more tokens available

String nextToken() – returns next token from StringTokenizer object

String nextToken() – returns next token based on delimiter

Boolean hasMoreElements() – same as hasMoreTokens() method

Object nextElement() – same as nextToken() method but its return type is object

Int countTokens() – returns total number of tokens

Example: StringTokenizer class that Tokenizes string “i am ramu” on basis of white spaces

The below demonstrates use of hasMoreTokens() amd nextToken() methods

**package** com.lokesh;

**import** java.util.StringTokenizer;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringTokenizer s1 =**new** StringTokenizer("i am ramu");

**while**(s1.hasMoreTokens()) {

System.***out***.println(s1.nextToken());

}

}

}

StringTokenizer class nextElement() method: it is same like nextToken() method

**package** com.lokesh;

**import** java.util.StringTokenizer;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringTokenizer s1 =**new** StringTokenizer("i am ramu");

**while**(s1.hasMoreTokens()) {

System.***out***.println(s1.nextElement());

}

}

}

StringTokenizer class countTokens() method: it returns the number of tokens in tokenizer string

**package** com.lokesh;

**import** java.util.StringTokenizer;

**public** **class** HelloWorld2 {

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

// **TODO** Auto-generated method stub

StringTokenizer s1 =**new** StringTokenizer("i am ramu");

System.***out***.println("total number of tokens "+s1.countTokens());

}

}