**String in Java-**

String is the class that represents sequence of character.

Package is Java.lang.

String class implements Serializable, comparable, char sequence interface.

String is the immutable, once string object is created, it cannot changed but new string object is created.

Example-1

**public** **class** StringDemo1 {

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

String s = "ram";

s=s.concat("patil");// concat() method appends the string at the end

System.***out***.println(s);// will print ram patil

}

}

Now what will happen here

First it will create the object with ram then later

It will create another object with rampatil. So here we cant directly assign patil value to s variable because string are immutable.

For mutable class, you can use String buffer and String builder class.

How to create the string object?

1. String literal
2. By new keyword
3. String literal-

It is created by using double quotes.

Example- String s=” codetech”;

Each time when you create string literal, the JVM check string constant pool first, if the string object is already present in the pool, reference to pooled instance is returned. If string does not present in the pool, new string instance is created and placed in pool.

Example- String s1=”codetech”;

String s2=” codetech”; //will not create the new instance.

In the above example, only one object will be created, firstly JVM will not find any string object with value “codetech” in the string constant pool, so it will create new object. After that it will find string with value= “codetech” in pool, it will not create the new object but will return reference to same instance.

Why Java uses the concept of string literal?

To make the java more memory efficient (because no new object is created if it exist already in string constant pool.)

1. By new Keyword-

Example- String s= new String(“pune”);

/\* create two objects \*/

In such case, JVM will create the new String object in normal(non-pool) heap memory and literal “pune” will be placed in string constant pool. The variable s refer to object in heap(non-pool).

Example- 1

**package** com.test;

**public** **class** StringDemo {

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

String s1 = "codetech";

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

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

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

}

}

Example- 2

**package** com.test;

**public** **class** StringDemo {

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

String s1 = "codetech";

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

s2=s1;

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

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

}

}

Output-

Example-3

**package** com.test;

**public** **class** StringDemo {

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

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

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

s2 = s1;

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

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

}

}

Output-

Example- 4

**package** com.test;

**public** **class** StringDemo {

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

String s1 = "codetech";

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

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

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

}

}

Output-

Example-5

**package** com.test;

**public** **class** StringDemo {

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

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

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

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

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

}

}

Output-

Example-6

**package** com.test;

**public** **class** StringDemo {

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

String s1="CODETECH";

String s2="codetech";

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

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

}

}

Example-7

**package** com.arrays;

**public** **class** StringDemo {

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

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

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

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

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

}

}

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |
| 3) | StringBuffer was introduced in Java 1.0 | StringBuilder was introduced in Java 1.5 |

**public class** BufferTest {

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

StringBuffer buffer=new StringBuffer("hello");

buffer.append("java");

System.out.println(buffer);

     }

}

Output : hellojava

**public class** BuilderTest {

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

StringBuilder build=new StringBuilder("hello");

build.append("java");

System.out.println(build);

     }

}

Output : hellojava