**Day 8**

**Strings in Java**

* In Java, a string is a sequence of characters.
* In Java, string is an object that represents a sequence of characters.

Syntax:

Char[] ch = {‘a’,’b’,’c’,’d’};

String s= new String(ch);

Or

String s= “abcd”;

* **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

**How to create a string?**

Strings in Java can be created by two methods. They are:

1. By string literal
2. By new keyword

**By String literal:**

Java String literal is created by using double quotes.

For Example:

String s=”java”;

* 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 a 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.

* **String objects are stored in a special memory area known as the “String constant pool”.**
* To make Java more memory efficient, Java uses the concept String literal (because no new objects are created if it exists already in the string constant pool).

**By new keyword:**

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

In such case, 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).

**Immutable string**

* In Java, String objects are immutable. Immutable simply means unmodifiable or unchangeable.
* Once String object is created its data or state can't be changed but a new String object is created.

Example

class S

{

public static void main(String args[])

{

String s=”hello”;

s.concat(“World”);

}

}

* A ClassLoader in Java uses a String object as an argument. Consider, if the String object is modifiable, the value might be changed and the class that is supposed to be loaded might be different. To avoid this kind of misinterpretation, String is immutable.
* As the String object is immutable, we don't have to take care of the synchronization that is required while sharing an object across multiple threads. That is, it is thread safe.
* 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.
* 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.
* String Buffer: String is modifiable/ mutable . Thread safe (cannot access parallelly)

StringBuffer sb= new StringBuffer(“hello”);

Sb.append(“java”)

Output : hellojava

* String Builder: Mutable, Not thread safe (can access parallelly). The is the most efficient string class.

**String Methods**

Length();

IsEmpty();

Substring();

Equal();

Contain()

toUpperCase()

toLowerCase()

Sample Program:

class StringMethod

{

public static void main(String args[])

{

int i=5;

String s1= "Software";

String s2="Tester";

s1.concat("Engineer");

System.out.println("s1.length()==: "+s1.length());

System.out.println("s1.substring()==: "+s1.substring(2,4));

System.out.println("s1.valueOf(i)==: "+s1.valueOf(i));

System.out.println(s1.valueOf(i).length());

String sf=s1.valueOf(1.1);

System.out.println("sf==: "+sf);

System.out.println("s1.equals(s2)==: "+s1.equals(s2));

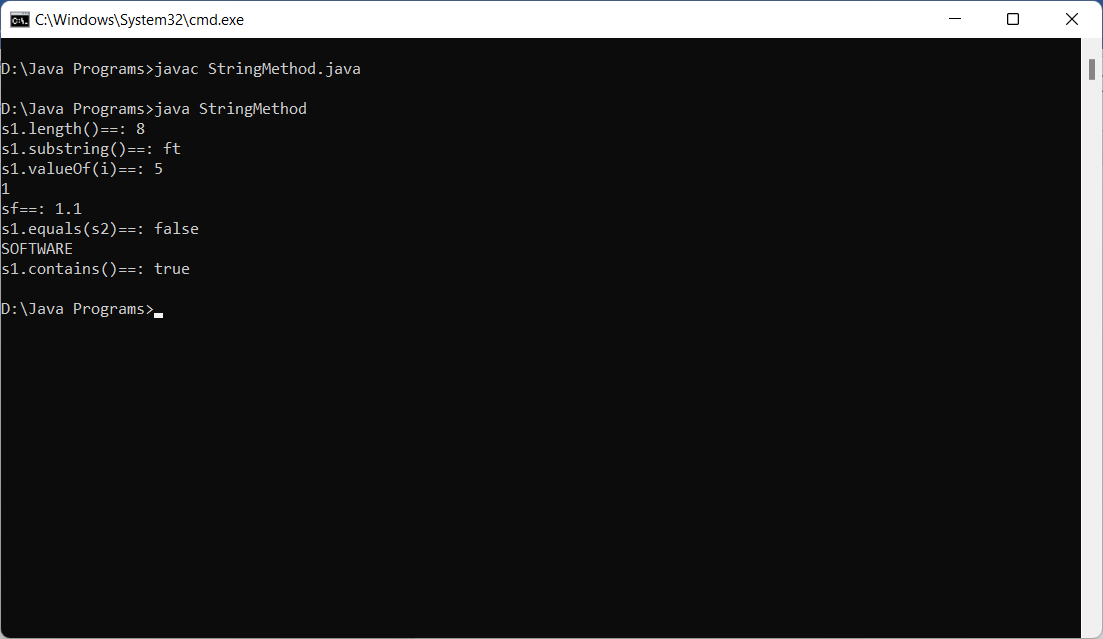
System.out.println(s1.toUpperCase());

System.out.println("s1.contains()==: "+s1.contains("of"));

}

}

Output:



String is immutable

Eg:

class Stringnotmutable

{

public static void main(String args[])

{

String s1="Hello";

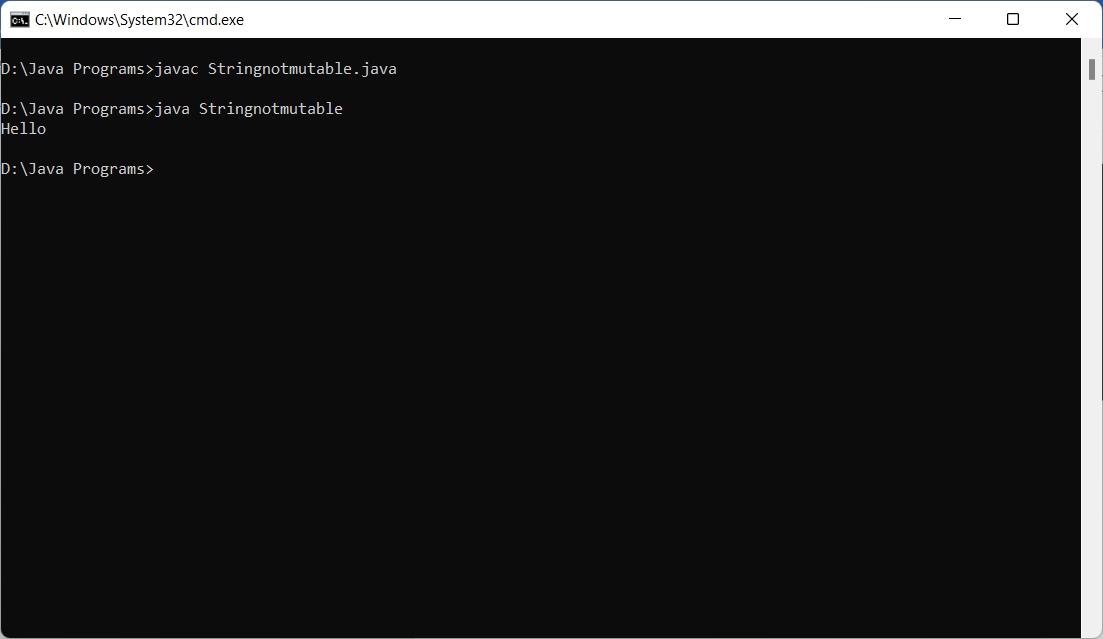
s1.concat("World");

System.out.println(s1);

}

}

**Output: Since the string is immutable the concat value is not updated**



StringBuffer and StringBuilder

Eg:

class StringBuffermutable

{

public static void main(String args[])

{

StringBuffer s1=new StringBuffer("Hello");

s1.append(" World");

System.out.println(s1);

StringBuilder s2=new StringBuilder("Java");

s2.append(" Program");

System.out.println(s2);

}

}

Output: The strings are mutable and the value is getting updated

