Java.lang.String

String is used to represent group of characters or character array enclosed with in the double quotes

There are two ways to create string in Java:

String literal

```
String s = "Java Programming";
```

Using new keyword

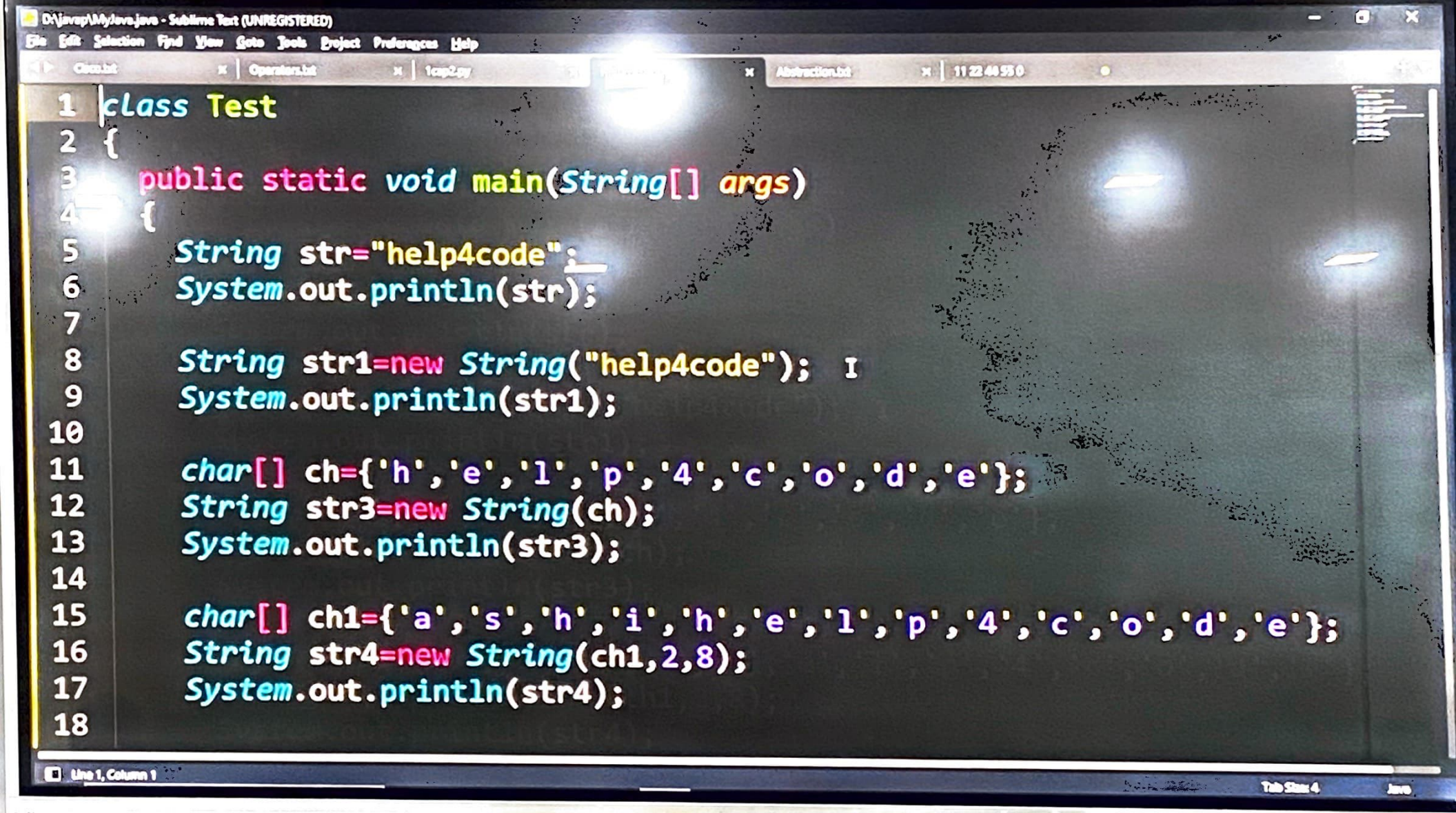
```
String s = new String ("Java Programming");
```

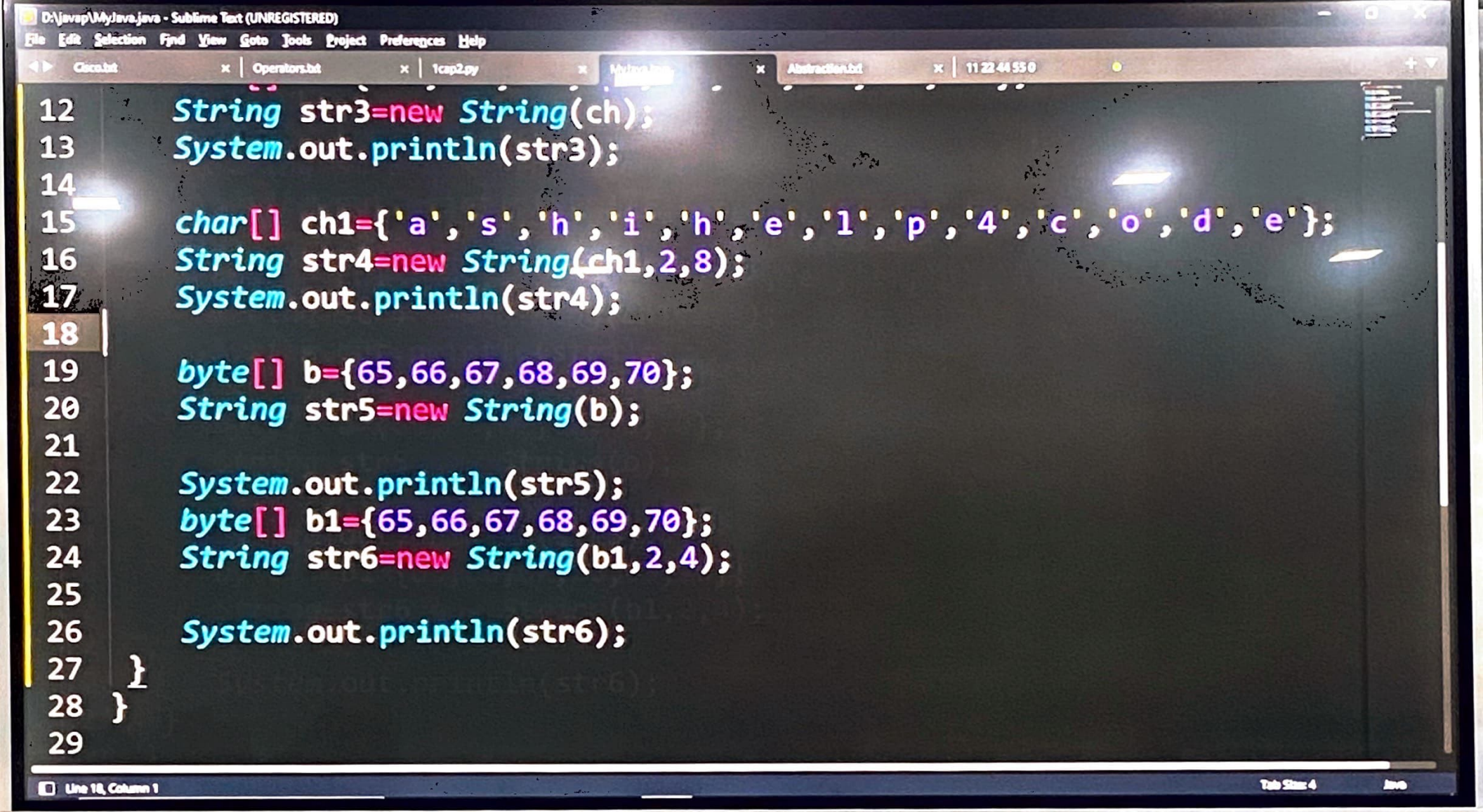
Ex1:

String is a sequence of characters placed in double quotes (“ ”).
Performing different operations on strings is called **string handling**.

In String manipulations we are going to learn following classes

- Java.lang.String
- Java.lang.StringBuffer
- Java.lang.StringBuilder
- Java.util.StringTokenizer

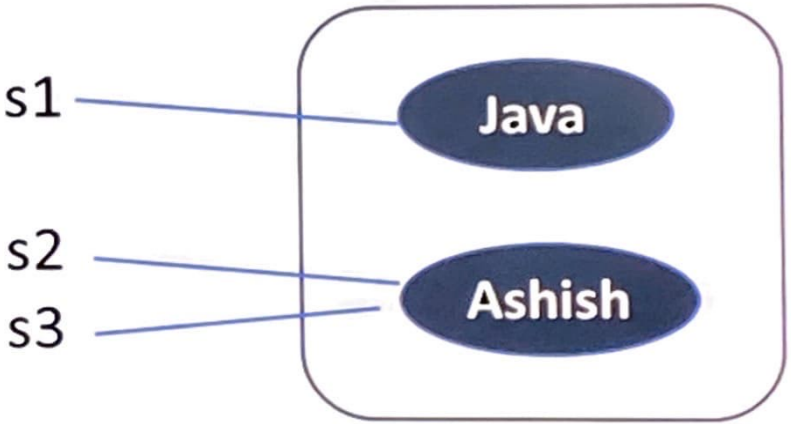

D:\javap\MyJava.java - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Class.txt
Operator.txt
1cap2.py
Abstraction.txt
class Test
{
public static void main(String[] args)
{
String str="help4code";
System.out.println(str);
String str1=new String("help4code");
System.out.println(str1);
char[] ch={'h','e','l','p','4','c','o','d','e'};
String str3=new String(ch);
System.out.println(str3);
char[] ch1={'a','s','h','i','h','e','l','p','4','c','o','d','e'};
String str4=new String(ch1,2,8);
System.out.println(str4);
Line 1, Column 1
Tab Size: 4
Java


D:\javap\MyJava.java - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Coco.txt
Operators.txt
1cap2.py
MyJava.java
Abstraction.txt
11 22 44 55 0
12 String str3=new String(ch);
13 System.out.println(str3);
14
15 char[] ch1={'a','s','h','i','h','e','l','p','4','c','o','d','e'};
16 String str4=new String(ch1,2,8);
17 System.out.println(str4);
18
19 byte[] b={65,66,67,68,69,70};
20 String str5=new String(b);
21
22 System.out.println(str5);
23 byte[] b1={65,66,67,68,69,70};
24 String str6=new String(b1,2,4);
25
26 System.out.println(str6);
27 }
28 }
29
Line 18, Column 1
Tab Size: 4
Java

Creating a string without using new operator

When we create String object without using **new** operator the objects are created in String constant pool area.

```
String s1 = "Java";
String s2 = "Ashish";
String s3 = "Ashish";
```

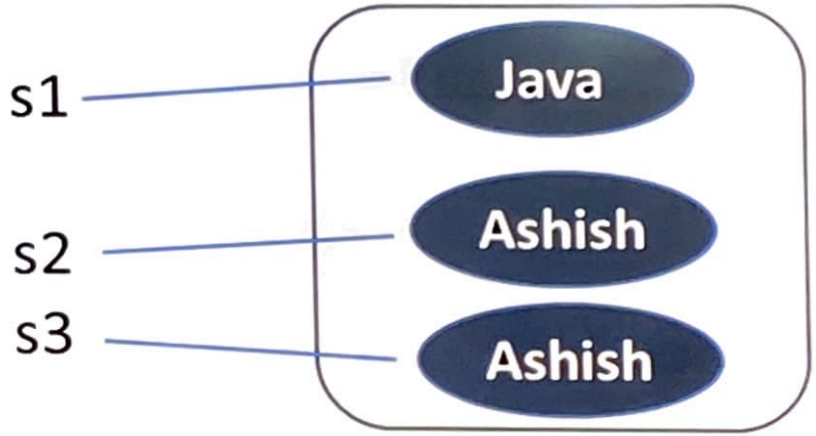


- If previous object is available with the same content then it won't create new object, that reference variable will point to existing object.
- If previous objects are not available then JVM will create new object.

Creating a string with using new operator

Whenever we are creating String object by using new operator the object created in heap area.

```
String s1 = "Java";
String s2 = "Ashish";
String s3 = "Ashish";
```



- When we create object in Heap area instead of checking previous objects it directly creates new objects
- Heap memory allows duplicate objects

```
class Test
{
  public static void main(String[] args)
  {
        Test t1 = new Test();
        Test t2 = new Test();
        System.out.println(t1==t2);
        String str1="Ashish";
        String str2="Ashish";
        System.out.println(str1==str2);
        String s1 = new String("help4code");
        String s2 = new String("help4code");
        System.out.println(s1==s2);
  }
}
```

Output

```
false
true
false
```

In java, objects of String are immutable which means a constant and cannot be changed in the same memory after they are created. Hence String is defined as an immutable sequence of characters.

Immutability vs. Mutability

String is **immutable** class it means once we are creating String objects it is not possible to perform modifications on existing object.

StringBuffer & StringBuilder are **mutable** classes it means once we are creating StringBuffer objects, it is possible to perform modification on that existing object

Difference between StringBuffer and StringBuilder

StringBuffer	StringBuilder
It is Synchronized	It is not synchronized
Object must be used in single thread programming model application	Object must be used in multithreaded programming model application

When should we go for *String, StringBuffer and StringBuilder?*

- If we don't want to store string modification in same memory, must use String
- If we want to store modification in same memory, must use StringBuffer or StringBuilder

String is immutable for several reasons

- Security
- Synchronization and concurrency
- Caching
- Class loading

```
class Test
{
 public static void main(String[] args)
 {
  //conversion of String to StringBuffer
       String str1="Ashish";
       StringBuffer sb1 = new StringBuffer(str1);
       System.out.println(sb1);
  //conversion of StringBuffer to String
       StringBuffer sb2 = new StringBuffer("Prashant");
       String str2 = sb2.toString();
       System.out.println(str2);
 }
}
```