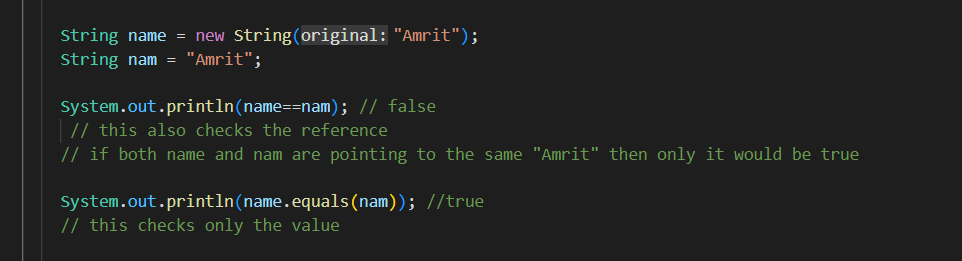
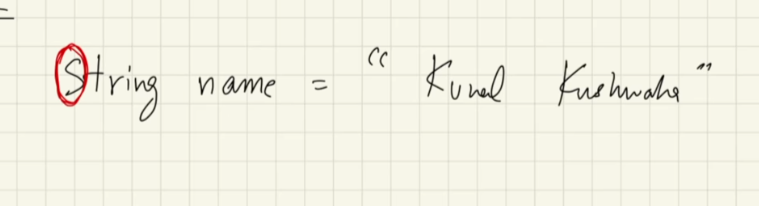
**An important concept**

In every language the double equals to == checks both the reference and the value

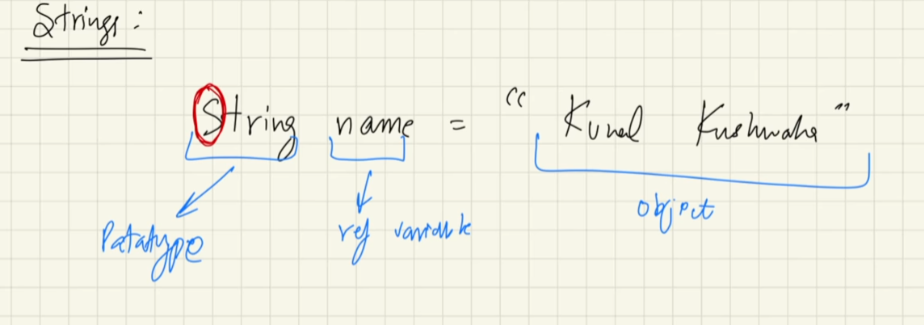
but only to check the values we aways have some other methods



String is the most commonly used class in the java’s class library



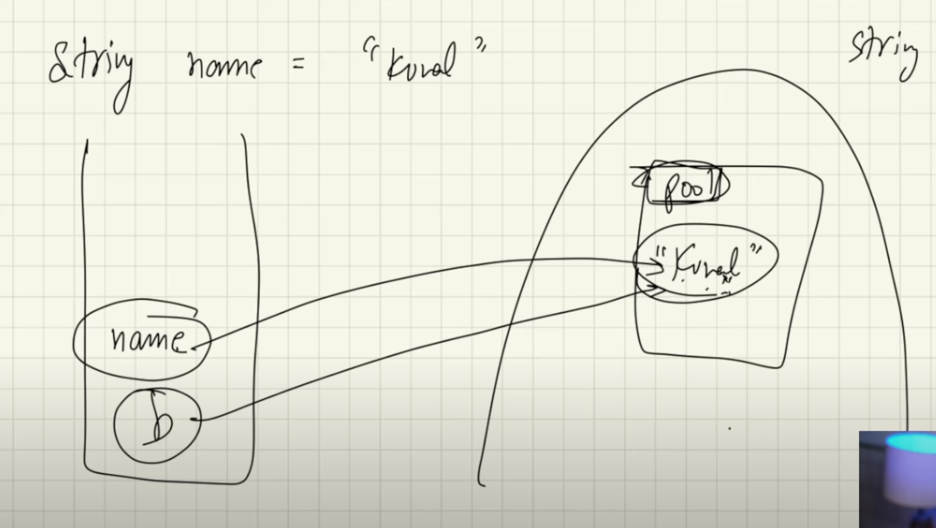
Every strings that we create is actually an object of type string



**How strings are stored in memory ?**

Since strings are also objects, so like with every objects in java, the reference variable lives in the stack and points to the object that lives in the heap

But here is something called String pool

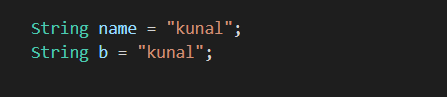


here name is a reference variable that lives in the stack, and “kunal” is string data that lives in the pool in the heap

pool is a small segment of memory in the heap

**What is the useful of pool ?**

When two same strings are created then there won’t be two similar strings in the heap meory, rather both of the reference variable will point to the same string in the single string pool



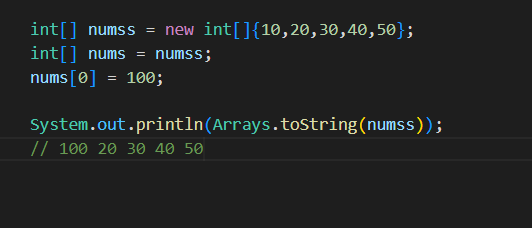
Here both of the reference variables name and b points to the same string “kunal” present in the string pool in heap memory

Although, both reference variable points to the same value, changing value via, “name” won’t have any impact on the “b”

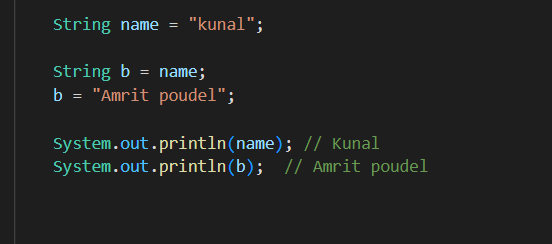
Because strings are immutable

**mutable and immutable**

Arrays are mutable, see below



**But strings are immutable for security reasons**



here both reference variables “b” and “name” points to the same string “kunal” initially

But we change b to “Amrit poudel”

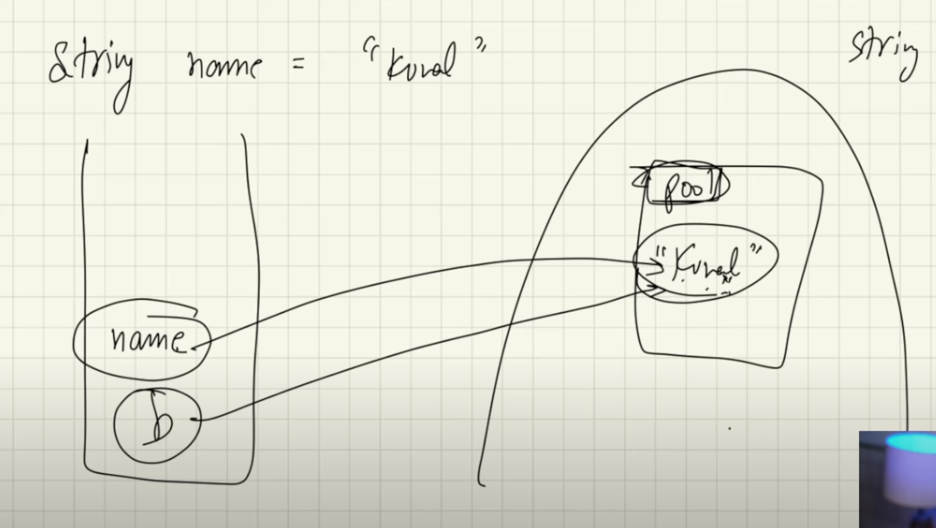
Now the string pointed by name will not change

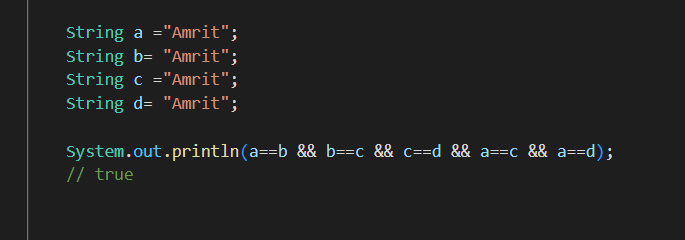
rather a new string of “Amrit poudel” will be created in string pool pointed by b

When we have many reference variables pointing to same string value, then unlike in other Objects we won’t have as many string values as many reference varlables

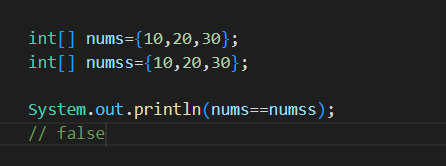
There is going to be only one string value, and that is inside a memory segment called “String pool” inside heap memory

And all of those reference variables point to the same single string value present in the String pool in the heap



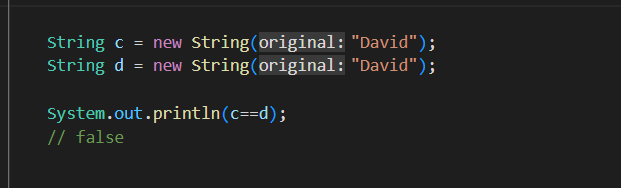


And Since strings are immutable, changing value via one reference variable won’t have any impact on other reference variables



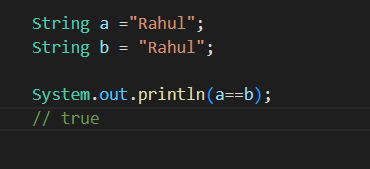
becoz there is no such thing as Arrays pool, these two arrays although they both have similar data, the reference variable points to the separate array data in the heap

**Using new keyword is the usual method of object creation**



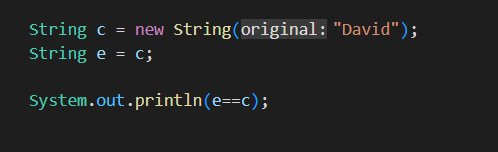
So on using new keyword string will not live in the string pool even if the value is same

**String will live in string pool only when created as string literals**

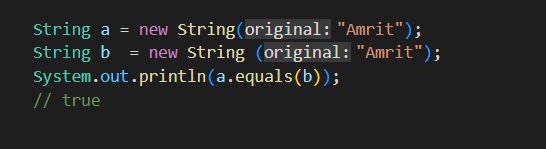


== will give true only when

1. both objects have the same value
2. both objects have the same reference variable



But to check only the value we have **.equals( ) method** in java



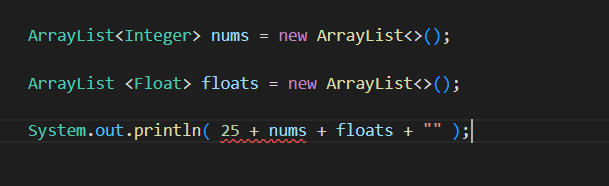
**.chatAt( index ) method**

It gives us the character at the index number

**A little dipper**

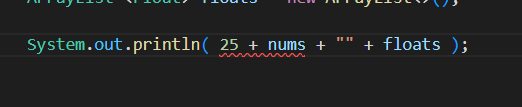
**In java string can be concatenated with any data type**

If there is a long expression involving string than it seems like the java will evaluate that expression from left to right



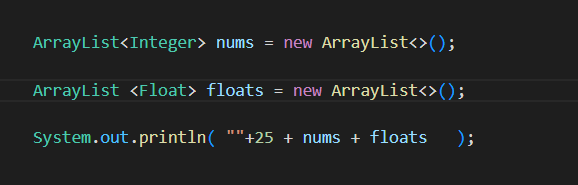
here we are getting error because, java first saw 25, than + operator than an Nums object of ArrayList

And these are two different types so could not concatenate



here also same problem arises

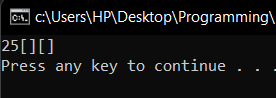
But here



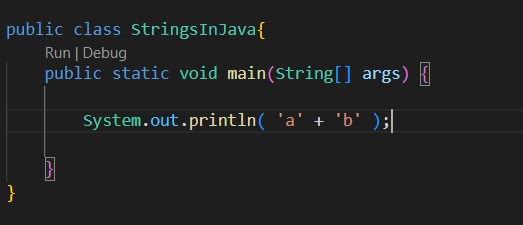
We know string can be concatenated with any types

And in expression java seems to move from left to right

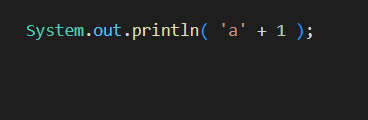
So first the java concatenate empty string and 25, and evaluates to a string, and that string is again concatenated to nums and evaluates to string, and again that string is concatenated to floats and thus final string is obtained



**A little bit about char**

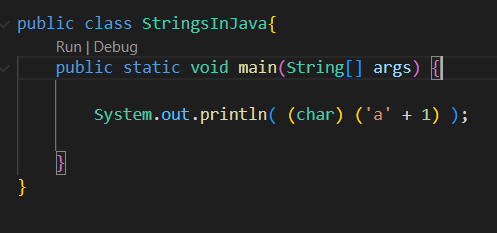


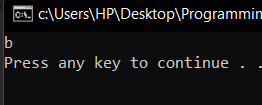
On doing this the ASCI codes of ‘a’ and ‘b’ will be added

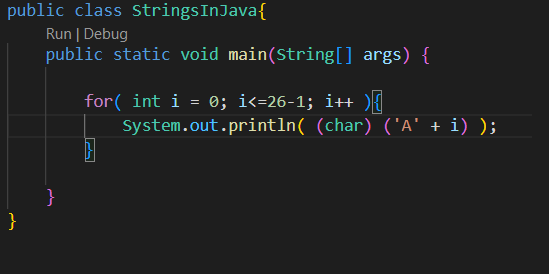


Here 1 will be added to the ASCI code of the ‘a’

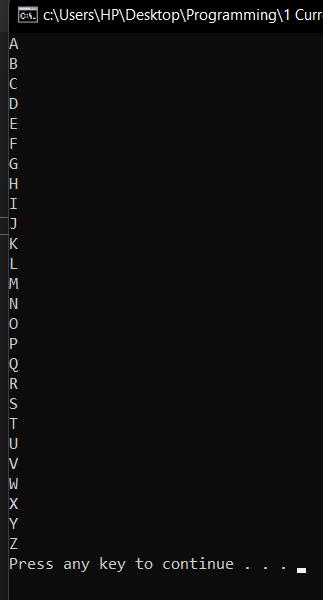
If we convert it to the char than we will get ‘b’

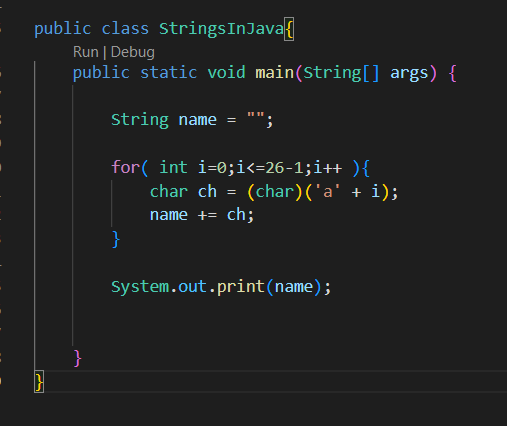


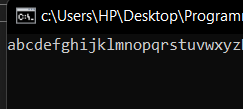




this will print all the English alphabets







Actually this is very bad, since here java is doing a lot of work and memory is being wasted

We know strings in java are immutable

So lets see the above code, what is happening these

First of all we have an empty string object

At first iteration an string “a” is created

Second “ab” is created

Third “abc” is created

… … …

And at last “abcdefghijklmnopqrstuvwxyz” is created

Since the strings are immutable each time new string objects are created and thereference variable name is assigned to the newly formed string

**All because the strings in java are immutable**

So there is a wastage of memory because all the strings are also there in memory

**Need of String builder in java**

So to solve this problem there is a separate data type known as the string builder

In string builder the change is made to the same object,

We can also say that this is a mutable string

StringBuilder is actually a class and we will be making objects of this type and that object is mutable unlike Strings



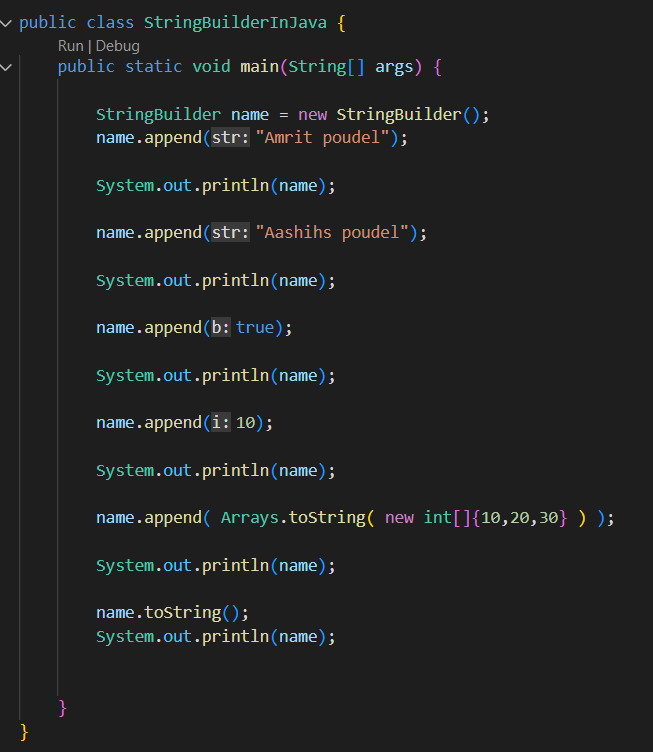
**.append( )** it is actually a method provided by the StringBuilder class

We an always look at the available methods by typing . (dot) after the object name

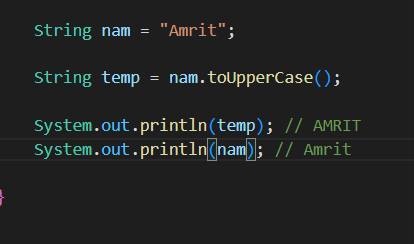
Also it looks like that in .append( ) method we could pass various data types

**It seems like the .append( ) method is internally overloaded by the java**

**.toString( )** we can convert the StringBuilder type object in to the String type via this method



See all the methods of the StringBuilder class, just do object\_name. and VS code will list all the available methods



Strings in java are immutable, here no any method changes the original string