append() method is a overloaded method , method name is same arguments are different

public StringBuffer append(int i)

public StringBuffer append(long l)

public StringBuffer append(boolean b)

public StringBuffer append(double d)

public StringBuffer append(folat f)

Eg: String\_Append\_Method\_Overloading

public StringBuffer insert(int index , int i );

public StringBuffer insert(int index , float f );

public StringBuffer insert(int index , long l );

public StringBuffer insert(int index , double );

public StringBuffer insert(int index , boolean b );

public StringBuffer insert(int index , Object o )

Eg: String\_Insert\_Method\_Overloading\_35

public StringBuffer delete( int begin , int end ) // deletes data from specified index to end-1

public StringBuffer deleteCharAt(int index) // deletes character at specified index

Eg: StringBuffer\_Delete\_Eg37

public StringBuffer reverse()

Eg:StringBuffer\_Reverse\_Eg38

// go through the code

public StringBuffer setLength()

// used to set only specified no of characters and remove all remaining characters

Eg: StringBuffer\_Set\_Length\_Eg39

public void trimToSize()

This is method is used to deallocate the extra allocated free memory such that capacity and size are equal .

Eg: StringBuffer\_trimToSize()\_Eg40

public void ensureCapacity(int capacity)

It is used to increased the capacity dynamically based on our requirement . because normal StringBuffer and every time jvm reaches the maximum capacity for object and after that increasing its size is a costly operation . so to avoid that ensureCapacity() method is used .

Eg: StringBuffer\_ensureCapacity\_Eg41

StringBuffer

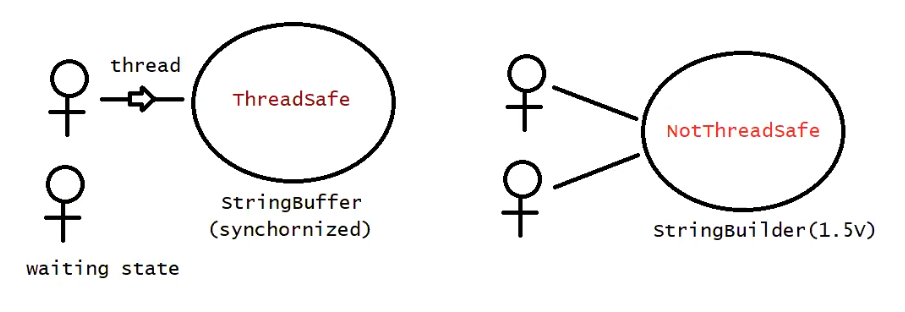
Every method present in StringBuffer is synchronized, so that only one thread is allowed to use the object at a time ,it would create performance problems , so to overcome this we use StringBuilder

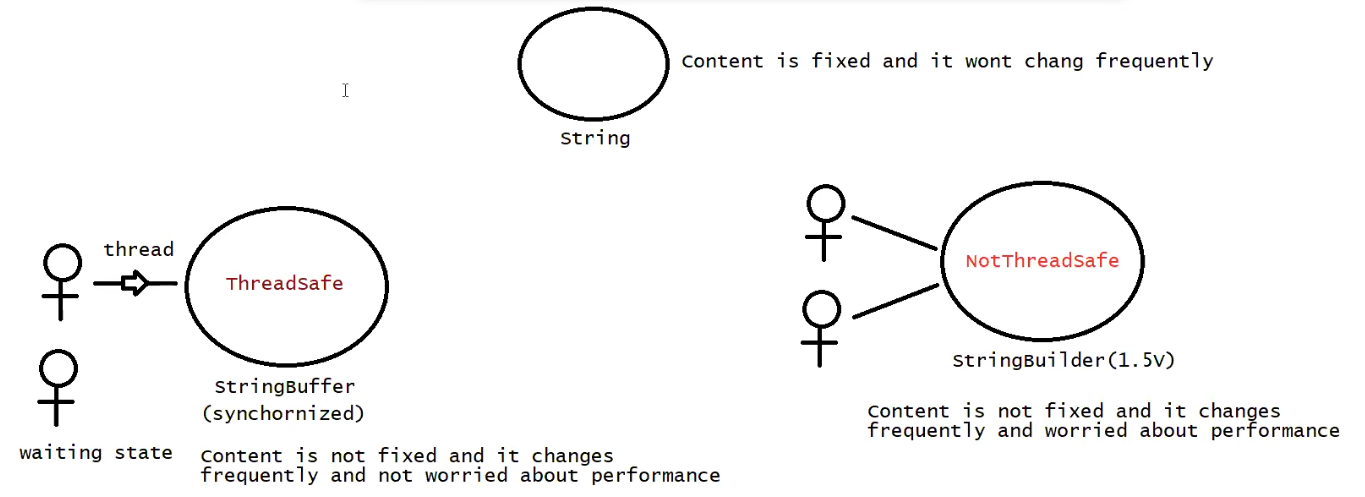
StringBuilder

It is introduced in the jdk 1.5 version

At a time more than one thread can use the object so it is not thread safe .

Threads are not required to wait ,so the performance is high .





String vs StringBuffer vs StringBuilder

String -> we can choose if the content is fixed and not changed frequently

StringBuffer -> we can choose if the content is not fixed and changes frequently and it should thread safe (commonly used in banking applications )

StringBulilder -> we can choose if the content is not fixed and changes frequently and it is not thread safe

Method Chaining

Most of the methods in the String, StringBuffer , StringBuilder the return type is same . hence after applying method on result we can call another method which forms method chaining .

Note : here the after method works based on the return type of the before method , and all of them works on the same object

Eg: Method\_Chaning\_Eg42

Eg: Copy\_Of\_String\_Eg43

// length property is for array and length() is of String type

Eg: Capital\_String\_Eg44

// go through the program.

Eg: SmallerCase\_String\_Eg45

// go through the program

Eg: Opposite\_Case\_Converting\_Eg46

// go through the program

Eg: Reverse\_The\_String\_Eg47

// go through the program

Eg: Reverse\_The\_String\_2\_Eg48

// go through the program

String\_object . split(“ “)

Splits this string around the matches of the given regular expression( should be given in bracket)

Eg: String\_Eg49

// go through the program

Eg: String\_Eg50

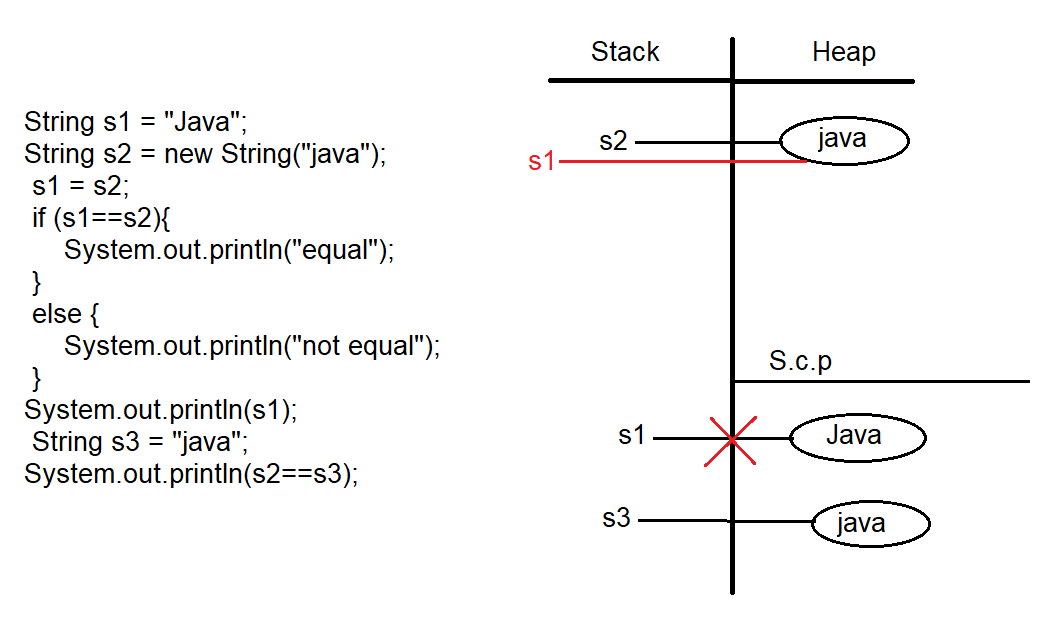
// go through the program

Eg: String\_Eg51

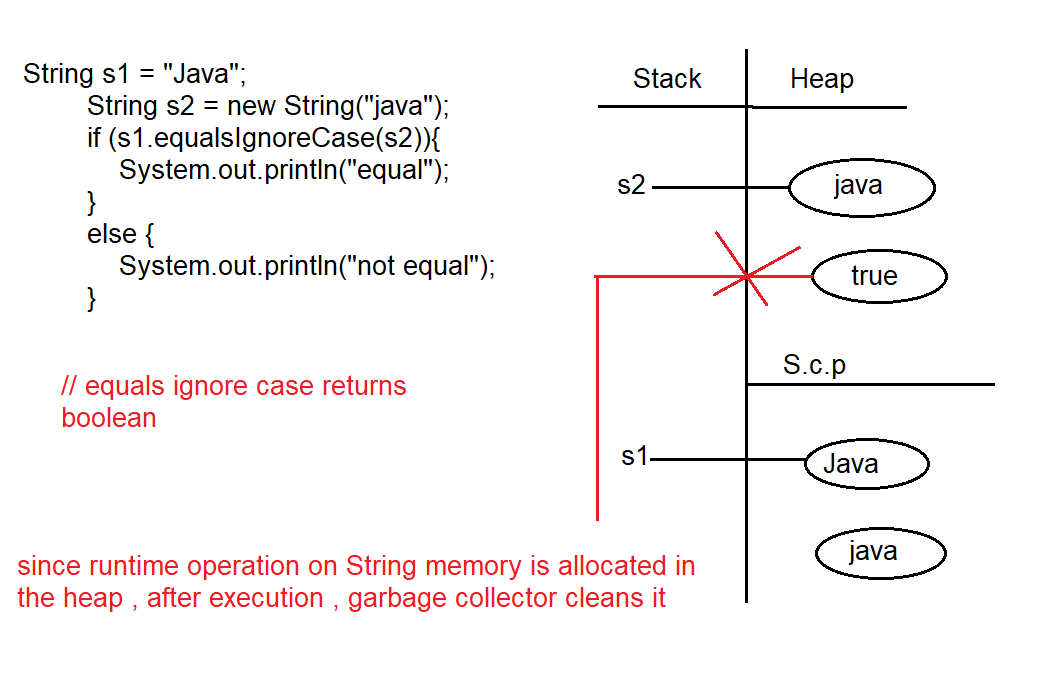
// go through the program

Eg: String\_Eg52

Since assigning s2 to s1 is a runtime operation memory is allocated in the heap.



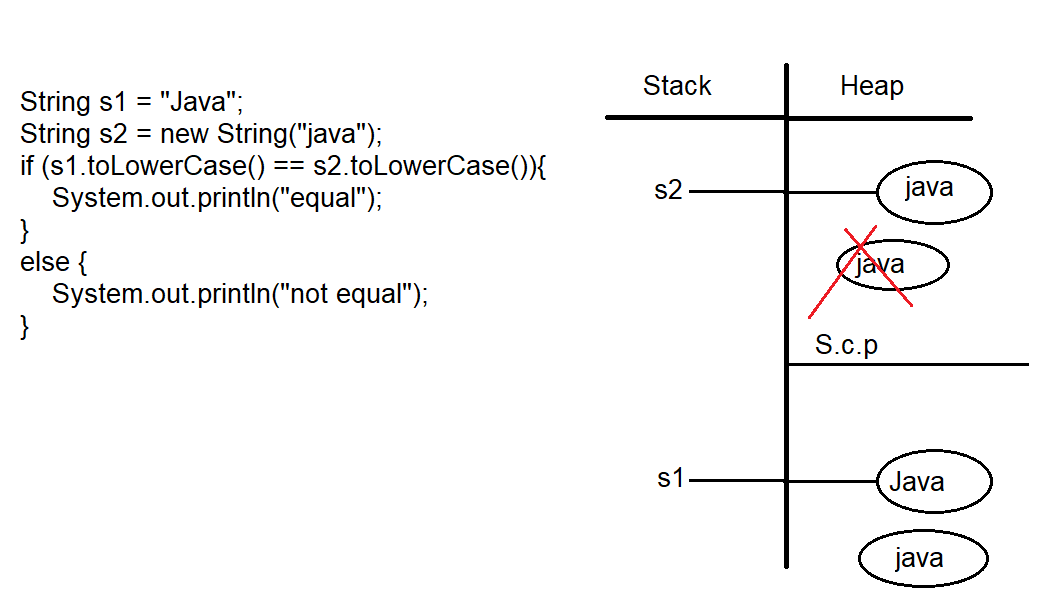
Eg: String\_Eg53



Eg: String\_Eg54

// go through the code

Eg: String\_Eg55



// while executing if condition since runtime operation and data is being changed a new object will be created in the heap . And even though runtime operation is made on s2 , and there is no change in its data s2 , no new object is created ,

The 2 objects in the heap have different address , else block is excueted

When else block is executed ,as the object is not collected by reference variable garbage collector cleans it .

Eg: String\_Eg56

// go through the program

Eg: String\_Eg57

// go through the program

Eg: String\_Eg58

// go through the code

Eg: String\_Eg59

// since s1 and s2 are of String type we are assigning s2 data to s1 so no compilation error .

Eg:String\_Eg60

// String is immutable , we cannot add data on same object , if we to try to perform any operation on String (using method ) a new object is created for that change , and if that object is not collected by reference variable it is cleared by garbage collector .

Eg: String\_Eg61

// StringBuilder is mutable , we can add data on same object .

Eg: String\_Eg62

// go through the code

Eg: String\_Eg63

// whenever you try to print a reference , internally toString() method is called.

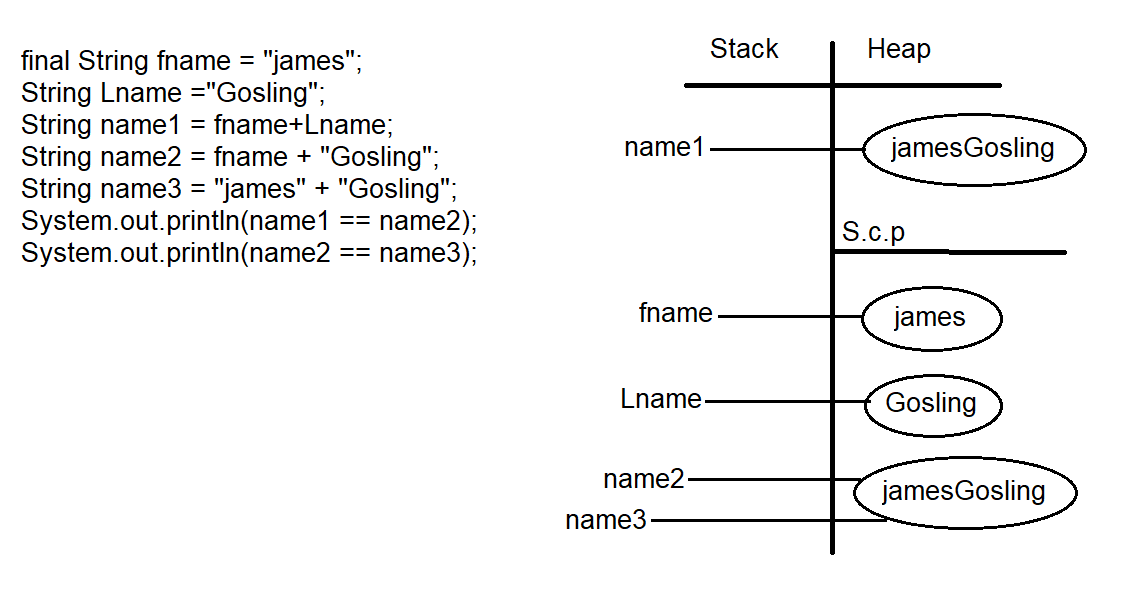
Note: if a class is final then that class cannot be inherited or extended , String class is final .

Eg:String\_Eg64

“” (no space is considered ) and “ “ are different

// go through the program

Eg: String\_Eg65



final variables are evaluated at compile time

Eg: Palindrome\_Using\_String\_Methods

// String is internally stored as char array , to access that we use charAt() method

Eg: Converting \_String\_To\_Char\_Array

// go through the program

Eg: Anagram

Anagram means characters in 1st string should match characters in second string

// go through the program

Eg: panagram

The string should contain all the 26 alphabets irrespective of their repetation .

Go through the program

Eg: String\_Eg66

in the expression the first one is String the remaining expression is also treated as String even though if there are other type literals.

Eg: String\_Eg67

In the expression if the first one is other than String literal , the expression is evaluated as normal .

Eg: String\_Eg68

In String objects only + operator is allowed other operators will result in the CompileTimeError

Eg: String\_Eg69

// go through the code