List of Concepts Involved:

- String introduction
- Types of String
- Immutable Strings and Memory Map

#⇒String is a JAVA class present inside the "java.lang" package of JAVA which does not require importing unlike "java.util" package which requires importing
ex:- import java.util.Scanner;

Topic 1: String Introduction

String it refers to an Object in java present in package called java.lang.String.
String refers to collection of characters.

anything under double quote is a String Object
'asit' → not String
"asit" → String

#anything starts with capital letter is a class in JAVA
ex:- Scanner, String.

Example

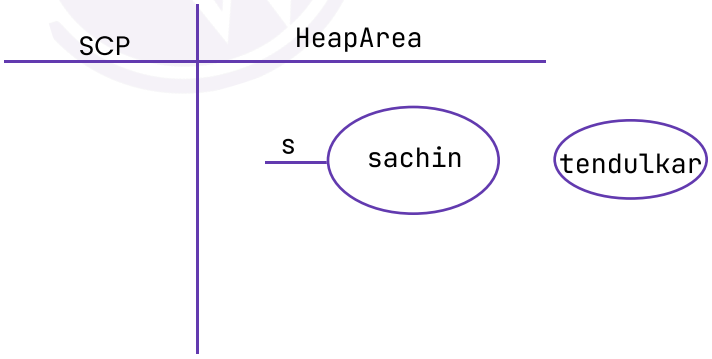
```
String s= "sachin";
System.out.println(s);//sachin

String s =new String("sachin");
System.out.println(s);//sachin
```

In java String object is by default immutable,meaning once the object is created we cannot change the value of the object, if we try to change then those changes will be reflected on the new object not on the existing object.

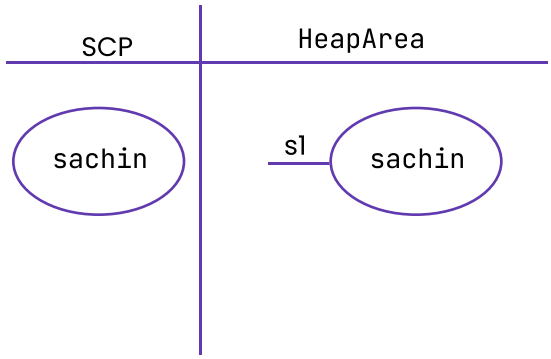
→ as String is a class ∴ the String value is stored inside the object create inside Heap.

Example

```
String s= "sachin";
s.concat("tendulkar");//(new object got created with modification so immutable)
System.out.println(s);// sachin
```

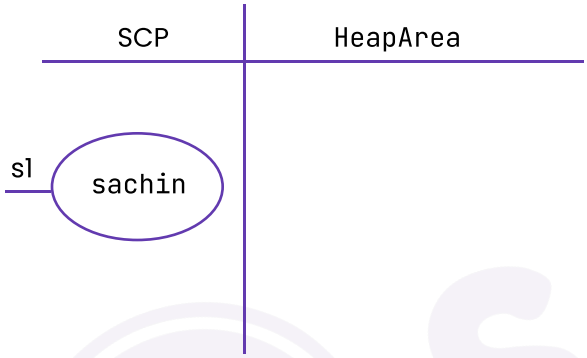


Note:
String s1 =new String("sachin");
In this case 2 objects will be created one in the heap and the other one in the String Constant Pool, the reference will always point to Heap.




```
String s1 ="sachin";
```

In this case only one object will be created in the SCP and it will be referred by our reference.



Note

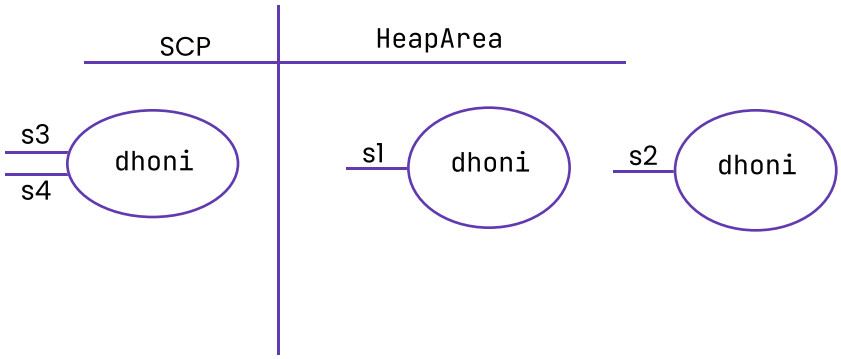
- Object creation in SCP is always optional,1st JVM will check if any object already created with required content or not.
- If it is already available then it will reuse the existing object instead of creating the new Object.
- If it is not available only then a new object will be created, so we say in SCP there is no chance of existing 2 objects with the same content.
- In SCP duplicates are not permitted.
- Garbage Collector cannot access SCP Area, Even though Object does not have any reference still object is not eligible for GC.
- All SCP objects will be destroyed only at the time of JVM ShutDown.

Example

```
String s1=new String("dhoni");
String s2=new String("dhoni");
String s3="dhoni";
String s4="dhoni";
```

Output

Two objects are created in the heap with data as "dhoni" with reference as S1,S2.
One object is created in SCP with the reference as S3,S4.



Note:

Importance of SCP

- In our program if any String object is required to use repeatedly then it is not recommended to create multiple objects with same content iit reduces performance of the system and affects memory utilisation.
- We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilisation. We can achieve this by using "scp".
- In SCP several references pointing to the same object the main disadvantage in this approach is by using one reference if we are performing any change the remaining references will be impacted. To overcome this problem sun people implemented immutability concept for String objects.
- According to this once we create a String object we can't perform any changes in the existing object if we are trying to perform any changes with those changes a new String object will be created hence immutability is the main disadvantage of scp.

Types of String

In java Strings are classified into 2 types

1. Mutable String – using String Builder, String Buffer
2. Immutable String – using String Object

Mutable String

Once if we create a String, on that String if we try to perform any operation and if those changes get reflected in the same object then such strings are called “Mutable String”.

Example: StringBuffer, StringBuilder

Immutable String

Once if we create a String, on that String if we try to perform any operation then those changes won’t be reflected in the same object, rather a new object will be created.Such type of String is called as “Immutable String”.

Example: Strin

String class Constructor

- String s =new String()
  Creates an Empty String Object

- String s =new String(String literals)
  Creates an Object with String literals on Heap

- String s =new String(StringBuffer sb)
  Creates an equivalent String object for StringBuffer

- String s =new String(char[] ch)
  Creates an equivalent String object for character array

- String s =new String(byte[] b)
  Creates an equivalent String object for byte array

Example

```
char[] ch={'a','b','c'} ;
String s=new String(ch);
System.out.println(s);
```

Example

```
byte[] b={100,101,102};
String s=new String(b);
System.out.println(s)//def
```

Immutable String and Memory Map:-

Diff ways of creating String:

① Using String type variable :- `String name = "Asit";`

② Using String Obj method :-

a) → `String name = new String("Asit");`

b) → `char array[] = {'A', 'B', 'C', 'D'};`
`String name = new String(array);` } String Obj using array

Q) What is the difference b/w the above two method of creating String

Understanding String Using Memory Map:-

As String is an Object in JAVA ∴ it is stored inside the Heap area and inside heap area there is a special region c/a "String Constant Pool(SCP)" whenever a String Object is created the memory allocation for it is done inside String Constant Pool (which used to be present inside method area previously)

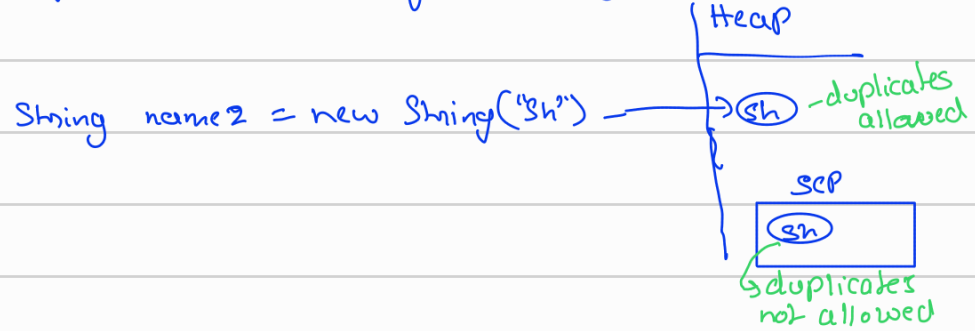
(☆) In "String Constant Pool(SCP)" no duplicates are allowed

In method 1:- `String name1 = "Asit"` using this an object will be created inside SCP only

where as,

In method 2:- `String name2 = new String("Shastri")` using this method Object is created in both scp and in Heap are and it refers to obj in Heap area

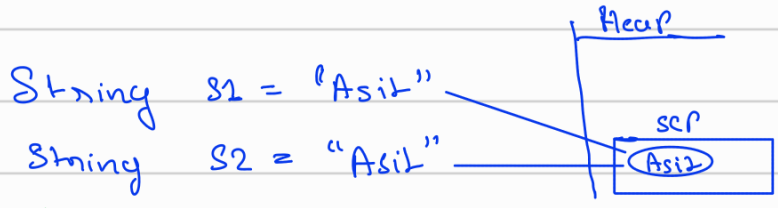
Ex:



Q2y Advantages and Disadvantages of SCP?

Sol^n → ⊙ Advantage :- As SCP does not allow any duplicate values any String variable created using 1st method having same value, all will refer to same memory location inside Heap memory.

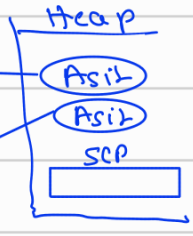
Ex: →



// Sop (S1 == S2); op → True

in method two i.e using new keword 2 diff Object containing similar value will be created inside the heap.

Ex: String S3 = new String("Asit")
String S4 = new String("Asit")



// Sop (S3 == S4); op → False

⊙ Disadvantage :- Garbage Collector is not able to collect memory inside SCP

List of Concepts Involved:

- Way to compare
- Inbuilt methods in String class
- Concatenation

Different ways of Comparison of String

To compare 2 Strings in java we use following approach

a. == operator

It compares the reference of the Object.

b. equals()

It compares the contents of two objects.

Example

```
String s1 = new String("sachin");
String s2 = new String("sachin");
System.out.println(s1==s2); //false
System.out.println(s1.equals(s2));//true
```

Important methods of String

1. public char charAt(int index)

Example

```
String s="sachin";
System.out.print(s.charAt(0));//s
System.out.print(s.charAt(-1));//StringArrayIndexOutOfBoundsException
System.out.print(s.charAt(10));//StringArrayIndexOutOfBoundsException
```

2. public String concat(String str)

Example

```
String s="sachin";
System.out.println(s.concat("tendulkar"));
s+="IND";
=s+"MI";
System.out.print(s);
```

3.public boolean equals(Object o)

It is used for Content Comparison,In String class equals() method is Overridden to check the content of the object.

It is used for Content Comparison without comparing the case.

5. public String subString(int begin)
It gives the String from the beginning index to the end of the String.

Example
```
String s="iNeuron";
System.out.print(s.substring(2));//searching from 2 to end of the string
```

6. public String subString(int begin,int end)
It gives the String from the begin index to end-1 of the String.

Example
```
String s="INeuron";
System.out.print(s.substring(2,6));//searching from 2 to 5 will happen
```

7. public int length()
It returns the no of characters present in the String.

Example
```
String s="INeuron";
System.out.print(s.length());//7
System.out.print(s.length);//Compile time error
```

8. public String replace(char old, char new)

Example
```
String s="ababab";
System.out.print(s.replace('a','b')); //bbbbbb
```

9. public String toLowerCase()
10. public String toUpperCase()
11. public String trim()
To remove the blank spaces present at the beginning and end of string but not the blank spaces present at the middle of the String.

12. public int indexOf(char ch)
It returns the index of 1st occurance of the specified character if the specified character is not available then it returns -1.

Example
```
String s="sachinramesh";
System.out.print(s.indexOf('a'));//1
System.out.print(s.indexOf('z'));//-1
```

13. public int lastIndexOf(char ch)
It returns the index of last occurrence of the specified character if the specified character is not available then it returns -1.

Example
```
String s="sachinramesh";
System.out.print(s.lastIndexOf('a'));//7
System.out.print(s.lastIndexOf('z'));//-1
```

Note:
Because of runtime operation, if there is a change in the content with those changes a new Object will be created only on the heap, but not in SCP.
If there is no change then the same object will be reused.
This rule is applicable for Objects present in both SCP and Heap.

Example1
```
String s1="sachin"
String s2=s1.toUpperCase();
String s3=s1.toLowerCase();
System.out.print(s1==s2);//false
System.out.print(s1==s3);//true
```

Example2
```
String s1="sachin";
String s2=s1.toString();
System.out.print(s1==s2);//true
```

Example3
```
String s1=new String("sachin");
String s2=s1.toString();
String s3=s1.toUpperCase();
String s4=s1.toLowerCase();
String s5=s1.toUpperCase();
String s6=s1.toLowerCase();
System.out.print(s1==s6);//true
System.out.print(s3==s5);//false
```

Concatenation

Concatenation is the process of combining two or more strings into a single string. This can be done in multiple ways, including using the "+" operator or the concat() method.

Example 1:
Using the "+" operator:
String str1 = "Hello";
String str2 = "World";
String str3 = str1 + " " + str2;
System.out.println(str3); //Output: Hello World

Example 2:
Using the concat() method:

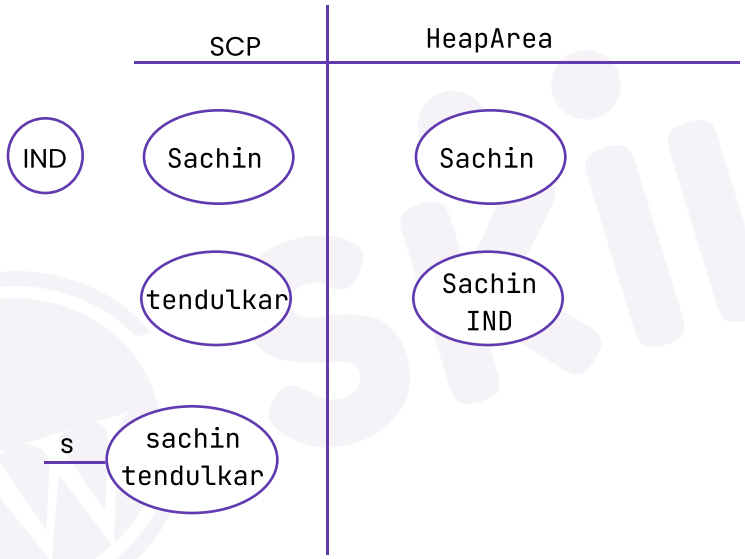
```
String str1 = "Hello";
String str2 = "World";
String str3 = str1.concat(" ").concat(str2);
System.out.println(str3);    //Output: Hello World
```

Example3: ☆

```
String s = new String("sachin");
s.concat("tendulkar");
s=s.concat("IND");
s="sachintendulkar";
```

Output

Direct literals are always placed in SCP. Because of runtime operations, if an object is required to be created compulsorily, that object should be placed on the heap, not in SCP.

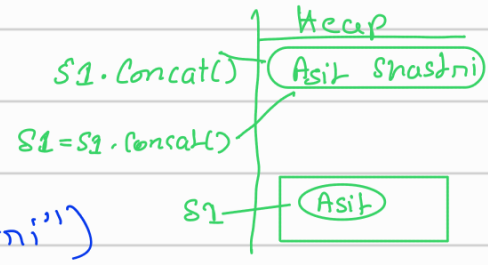


Example 4:

```
String s1= new String("sachin");
s1.concat("tendulkar");
s1+="IND";
String s2=s1.concat("MI");
System.out.println(s1);
```

Q) Explain 2 methods of String Concatination.

Soln Method 1 :- String s1 = "Asit" ;
String s2 = "Shastri" ;
String s3 = s1 + s2 ;



Method 2 :- String s1 = "Asit" ;
s1 = s1.concat(" Shastri")

↳ Even though the new keyword is not used still the concat method will create another object in Heap.
↳ s1 = s1.concat(" Shastri") shift / changes the s1 reference to the reference of the obj. created by the concat() method

Example :- String s1 = "Asit" ;
String s2 = s1.concat(" Shastri");
String s3 = new String("Asit") ;
s3 = s3.concat(" Shastri");
Sop (s1); // op→ Asit
Sop (s2); // op→ Asit Shastri
Sop (s3); // op→ Asit Shastri

Q) What happens if we concatinate int d-type with String d-type using '+' operator. Also give Advantages of using "+" operator for String Concatination.

Soln Advantages
① Can join multiple strings ex:
String b = "Shastri".
String a = "Asit"
String c = "24"
String d = a + b + c
Sop (d) = AsitShastri24

② Concatinate int & String :- int a=10
String num = "Asit"
Sop (num+a) — op: Asit10.

List of Concepts Involved:

- Reversing String Different cases
- Palindrome
- Anagram program
- Pangram program

Reversing String Different cases

Q1) "Asit Shastri" ⟶ "irtsahS tisA"

```
String str1 = "Asit Shastri";
String str2 = "";
for(int i = str1.length()-1; i>=0; i--)
{
    str2 = str2 + str1.charAt(i);
}
Sop(str); op: "irtsahS tisA"
```

Q2) "Asit Shastri" ⟶ "Shastri Asit". {use String.split() method}

```
String str1 = "Asit Shastri";
String str2 = "";
String sArray[] = str1.split() // Splits the string with space and stores in array {"Asit", "Shastri"}
for(int i = sArray.length-1; i>=0; i--)
{
    str2 = str2 + sArray[i] + " ";
}
Sop(str2) op -> Shastri Asit
```

Q1) "Asit Shastri" ⟶ "tisA irtsahS"

Soln ->

Palindrome

ex: 2552, NITIN

```
Class RevRev{
    public String reving(String Str1)
    {
        String Str2 = "";
        for(int i= Str1.length()-1; i>=0; i--)
        {
            Str2 += Str1.charAt(i);
        }
        return str2;
    }
```

```
Class Palindrome{
    pubic void isPalindrome (String str1)
    {
        RevRev rr = new RevRev();
        String revstr = rr.reving(Str1);
        if (Str1.equalsIgnoreCase(revstr))
        { sop("Palindrome");
        }
        else{
            sop("not Palindrome");}
        }
```

```
public class PalindromeCheck{
    public static void main(String[] args){
        String input = "2552";
        Palindrome p = new Palindrome();
        p.isPalindrome(input);
    }
}
```

Pangram Program

String · where all A-Z · letters are involved

Ex:- "The Quick Brown Fox Jumps Over Lazy Dog"

☆ ASCII value of characters — 'a' → 97, 'A' → 65 } char in JAVA is defined by single quotes

☆ Declaring a char :- char ch = 'a';

☆ char and int operations :- ch - 97 = 0

↳ using these concepts to solve Pangram concept

Sol → 1 → Take a string

2 → replace all the spaces

3 → Convert all to Lower or Upper case ← Lower case — 'a' = 97, Upper case — 'A' = 65

4 → Convert str to char array

5 → Create an empty int Array of size 26

6 → Iterate over every char element of (4th) and use

arr[(char array name)[i]-97]++; if all lower case an,

arr[(char array name)[i]-65]++; if all Upper case

| empt array at (5)

Q) index of method()

Sol → used to know the idex of a particular element in an array

Anagram Program

List of Concepts Involved:

- Mutable String
- String Buffer vs String Builder
- Inbuilt Methods

Mutable String

Once if we create a String, on that String if we try to perform any operation and if those changes get reflected in the same object then such strings are called "Mutable String".

Example: StringBuffer, StringBuilder

How to create a mutable string?

We can use the StringBuffer and StringBuilder classes to generate mutable strings. Which class we should use entirely depends on the scenario. Both classes generate a mutable object of string.
If the string needs to be thread-safe and you wish to operate in a multithreading environment, you should use the StringBuffer class. On the other hand, StringBuilder is not necessary if you don't want a multithreading environment.
However, if speed is your top priority, StringBuilder outperforms StringBuffer in terms of speed.

StringBuffer

- If the content will change frequently then it is not recommended to go for String object becoz for every new change a new Object will be created.
- To handle this type of requirement, we have a StringBuffer/StringBuilder concept.

1.StringBuffer sb=new StringBuffer();

- Creates an empty StringBuffer object with default initial capacity of "16".
- Once StringBuffer reaches its maximum capacity a new StringBuffer Object will be created.
  new capacity = (current capacity+1) * 2;

Example

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());//16
sb.append("abcdefghijklmnop");
System.out.println(sb.capacity());//16
sb.append('q');
System.out.println(sb.capacity());//34
```

- Capacity gets boy (16+1)2 = 34

2.StringBuffer sb=new StringBuffer(initialCapacity);

It creates an Empty String with the specified initial capacity.

Example

```
StringBuffer sb = new StringBuffer(19);
System.out.println(sb.capacity());//19
```

3. StringBuffer sb=new StringBuffer(String s);

It creates a StringBuffer object for the given String with the
capacity = s.length() + 16;

Example

capacity = 6 + 16 = 22

```
StringBuffer sb = new StringBuffer("sachin");
System.out.println(sb.capacity());//22
```

Important methods of StringBuffer

a. public int length()
b. public int capacity()
c. public char charAt(int index)
d. public void setCharAt(int index,char ch)
e. public StringBuffer append(String s)
f. public StringBuffer append(int i)
g. public StringBuffer append(long l)
h. public StringBuffer append(boolean b)
i. public StringBuffer append(double d)
j. public StringBuffer append(float f)
k. public StringBuffer append(int index,Object o)

Example

```
StringBuilder sb = new StringBuilder("sachinrameshtendulkar");
System.out.println(sb.length());//21
System.out.println(sb.capacity());//21 + 16 = 37
System.out.println(sb.charAt(20));//'r'
System.out.println(sb.charAt(100));//StringIndexOutOfBoundsException
```

Example

```
StringBuilder sb = new StringBuilder("sachin");
sb.setCharAt(2, 'C');
System.out.println(sb);
```

Append method is an overloaded method, methodName is the same but changes in the argument type.

Example

```
StringBuffer sb = new StringBuffer();
sb.append("PI value is :: ");
sb.append(3.1414);
sb.append(" This is exactly ");
sb.append(true);
System.out.println(sb);// PI value is ::3.1414 This is exactly true
```

Overloaded methods

l. public StringBuffer insert(int index,String s)
m. public StringBuffer insert(int index,int i)
n. public StringBuffer insert(int index,long l)
o. public StringBuffer insert(int index,double d)
p. public StringBuffer insert(int index,boolean b)
q. public StringBuffer insert(int index,float s)
r. public StringBuffer insert(int index,Object o)

To insert the String at the specified position we use insert method
Example

```
StringBuffer sb = new StringBuffer("sacin");
sb.insert(3, 'h');
System.out.println(sb);//sachin
sb.insert(6, "IND");
System.out.println(sb);//sachinIND
```

Methods of StringBuffer

public StringBuffer delete(int begin,int end)
It deletes the character from the specified index to end-1.

public StringBuffer deleteCharAt(int index)
It deletes the character at the specified index.

Example:

```
StringBuffer sb=new StringBuffer("sachinrameshtendulkar");
sb.delete(6,12);
System.out.println(sb);//sachintendulkar
sb.deleteCharAt(13);
System.out.println(sb);//sachintndulkar
```

public StringBuffer reverse()
It is used to reverse the given String.

Example

```
StringBuffer sb=new StringBuffer("sachin");
sb.reverse();
System.out.println(sb);//nihcas
```

public void setLength(int Length)
It is used to consider only the specified no of characters and remove all the remaining characters.

Example

```
StringBuffer sb=new StringBuffer("sachinramesh");
sb.setLength(6);
System.out.println(sb);//sachin
```

public void trimToSize()

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

Example

```
StringBuffer sb = new StringBuffer(1000);
System.out.println(sb.capacity());//1000

sb.append("sachin");
System.out.println(sb.capacity());//1000

sb.trimToSize();

System.out.println(sb);//sachin
System.out.println(sb.capacity());//6
```

public void ensureCapacity(int capacity)

It is used to increase the capacity dynamically based on our requirements.

Example

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());//16
sb.ensureCapacity(1000);
System.out.println(sb.capacity());//1000
```

Note

EveryMethod present in StringBuffer is synchronized, so at a time only one thread can are allowed to operate on StringBuffer Object, it would increase the waiting time of the threads it would create performance problems, to overcome this problem we should go for StringBuilder.

StringBuilder(1.5v)

StringBuilder is same as StringBuffer(1.0V) with few differences

StringBuilder

- No methods are synchronized
- At a time more than one thread can operate so it is not ThreadSafe.
- Threads are not required to wait so performance is high.
- Introduced in jdk1.5 version

StringBuilder is a mutable sequence of characters in Java and provides several inbuilt methods to modify its contents. Here is a list of some commonly used methods in StringBuilder:

- **append(Object obj):** Adds the string representation of an Object to the end of the StringBuilder
- **append(String str):** Adds a String to the end of the StringBuilder
- **insert(int offset, Object obj):** Inserts the string representation of an Object into the StringBuilder at a specified position

- **insert(int offset, String str):** Inserts a String into the StringBuilder at a specified position
- **delete(int start, int end):** Removes the characters in the StringBuilder between the specified start and end positions
- **deleteCharAt(int index):** Removes the character at a specified position in the StringBuilder
- **replace(int start, int end, String str):** Replaces the characters in the StringBuilder between the specified start and end positions with the specified String
- **reverse():** Reverses the order of characters in the StringBuilder
- **toString():** Returns the String representation of the contents of the StringBuilder

String vs StringBuffer vs StringBuilder

String
we opt if the content is fixed and it wont change frequently
StringBuffer
we opt if the content changes frequently but ThreadSafety is required
StringBuilder
we opt if the content changes frequently but ThreadSafety is not required

MethodChaining

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

Example

```
StringBuffer sb = new StringBuffer();
sb.append("sachin").insert(6, "tendulkar").reverse().append("IND").
delete(0, 4).reverse();
System.out.println(sb);
```

List of Concepts Involved:

- static keyword
- Class loading and How Java program actually executes
- Different Members in the java program
- Static variables
- Static Methods
- Static block
- Difference with respect to static and non static members of a class

#object creation is same as instance creation.

'static' keyword

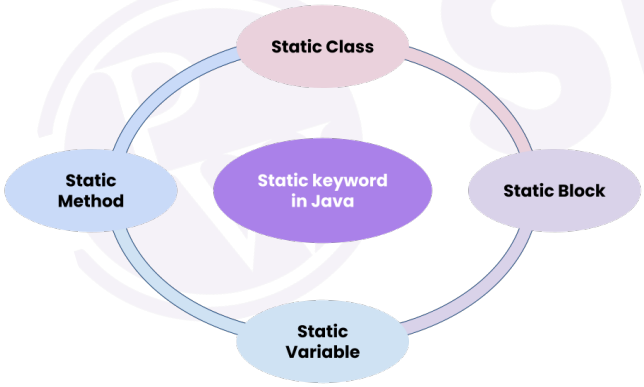
function
inner class

The static keyword is mainly used for memory management in Java. A static keyword can be applied to variables, blocks, methods, and classes. The static keyword is a property of a class rather than an instance of the class. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

Where is the "static" keyword applicable?

The static keyword is a non-access modifier in Java and is applicable for the following:

1. Variables
2. Methods
3. Blocks
4. Class



Static Variables

If we declare any variable as static, then it is called a static variable. When a variable is declared as static, then a single copy of that variable is created and shared among all of the objects at the class level. Static variables are global variables. All instances of the class share the same static variable.
We can create static variables at the class level only.

Why static?

It makes our program more efficient, as every object doesn't allocate separate memory to a static variable.

Static Method

A static method is a method that belongs to a class rather than an instance of a class. This means you can call a static method without creating an object of the class. Static methods are sometimes called class methods.

There are a few other reasons why you might want to use static methods:

- You can access static methods from outside of the class in which they are defined. This is not possible with non-static methods.
- Subclasses can override static methods, but non-static methods cannot.
- Static methods are executed when an instance of the class is created, whereas non-static methods are not.
- Static methods can be used to create utility classes that contain general-purpose methods.

Static Blocks

It is used to initialize static data members. It is used to initialize before the main method at the time of class loading. It gets executed only once when the class gets loaded. It is not necessary to execute it again when creating different objects after the first time.

Static Class

In Java, a "static class" is a class that can be instantiated without having to create an instance of the containing class. A static class is defined as a member of another class and can only access static members of the containing class.

How Java Program Actually executes:

inside Java Runtime Environment (JRE) by JVM.

Class Loading

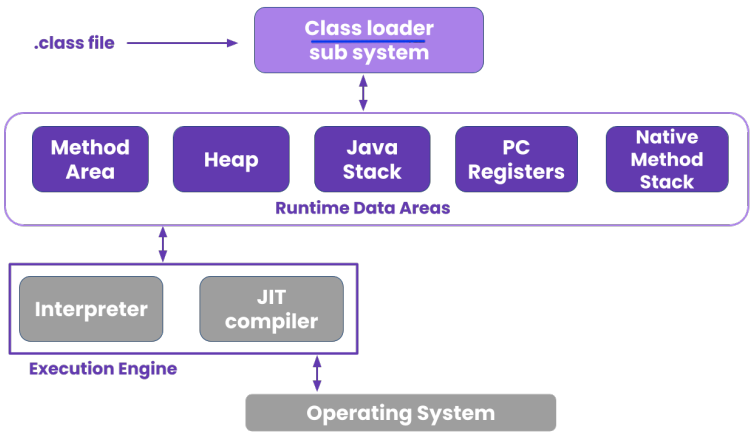
In Java, classloading is the process of loading class files into the JVM (Java Virtual Machine) at runtime. It is responsible for loading classes from various sources, such as the file system, network, and databases, and making them available to the JVM for execution.

The class loading process in Java is divided into three phases: loading, linking, and initialization.

1. **Loading:** In the loading phase, the classloader locates the class file using the fully qualified class name, reads the class file, and converts it into a Class object. The Class object contains the metadata of the class, such as the fields, methods, and constructors.

2. **Linking:** In the linking phase, the JVM performs several operations on the Class object, such as verifying the class file's integrity, resolving symbolic references, and allocating memory for the class variables.

3. **Initialization:** In the initialization phase, the JVM initializes the class variables with their default values, and runs the class's static initialization block (if any).



Principles of functionality of a Java classloader

A Java classloader's functionality is based on the following principles:

1. Delegation: A class loader delegates the responsibility of loading a class to its parent class loader before attempting to load it itself. This allows the class loader to take advantage of any classes that have already been loaded by the parent class loader.

2. Visibility: A class loaded by a class loader is only visible to that class loader and its child class loaders. This allows different class loaders to load different versions of the same class without interfering with each other.

3. Uniqueness: Each class is identified by its fully-qualified name (FQN) and is loaded by only one class loader. This ensures that the same class is not loaded multiple times by different class loaders.

4. Caching: A class loader caches the classes that it loads to improve performance. This allows the class loader to quickly return a class that has already been loaded, rather than having to load it again.

5. Security: A class loader enforces Java's security model by only allowing classes to access resources and execute code that is within their scope of permissions.

6. Extensibility: Java class loaders are extensible, allowing developers to create custom class loaders that can be used to load classes from specific locations or with specific behaviours.

7. Dynamic nature: Class loading is a dynamic process and can happen at runtime. Classes can be loaded, unloaded, and reloaded as the application runs.

8. Classpath: Java classloaders use the classpath to locate the classes that they need to load. The classpath is an ordered list of directories and JAR files that contain the class files.

9. Loading order: Java classloaders follow a specific order to load classes, first it checks the memory, if class is not found in memory it checks the cache, if class is not found in cache it checks the local file system, if class is not found in local file system it checks the network.

Different Members in the Java program

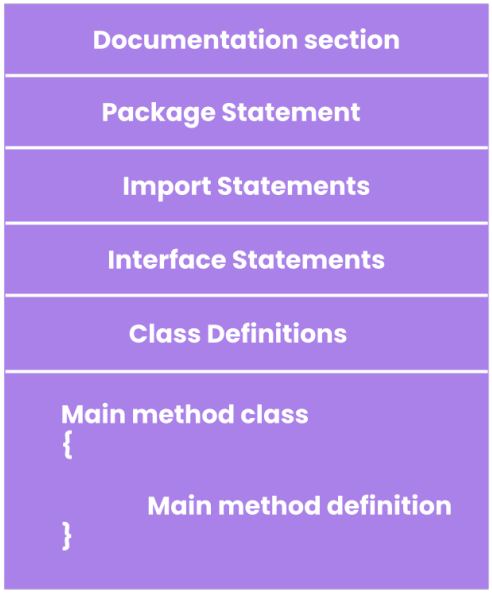
A Java program consists some of Members are:

1. Instance Member
2. Static Member

Instance Member: An instance member is essentially anything within a class that is not marked as static. That is, that it can only be used after an instance of the class has been made (with the new keyword). This is because instance members belong to the object, whereas static members belong to the class.

Static Member: Static members are those which belong to the class and you can access these members without instantiating the class. The static keyword can be used with methods, fields, classes (inner/nested), blocks.

Structure of Java Program:



Static variables

- If the value of a variable is not varied from object to object such type of variables is not recommended to be declared as instance variables.
- We have to declare such types of variables at class level by using static modifiers.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables for the entire class only one copy will be created and shared by every object of that class.
- Static variables will be created at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly the same as the scope of the .class file.
- Static variables will be stored in the method area. Static variables should be declared within the class directly but outside of any method or block or constructor.
- Static variables can be accessed from both instance and static areas directly. We can access static variables either by class name or by object reference but usage of class name is recommended.
- But within the same class it is not required to use class names we can access directly.

We can access static variables in 2 ways

1. Using className
2. Using reference variables

Note: Static variables also known as class level variables or fields.

Static methods

Methods which are available at the class level are referred to as “static methods”.
These methods are referred to as utility methods.
Inside the static methods we can access only static variables.
If we try to access the instance variables directly then it would result in **“CompileTimeError”.**

static block

- These are the blocks which gets executed automatically at the loading the .class files
- If we want to perform any activity at the time of loading a .class file we have to define that activity inside the static block.
- We can write any no of static blocks, those static blocks will be executed from top to bottom.
- Normally a static block is used to perform initialization of the static variables.

Difference with respect static and non static members of a class static

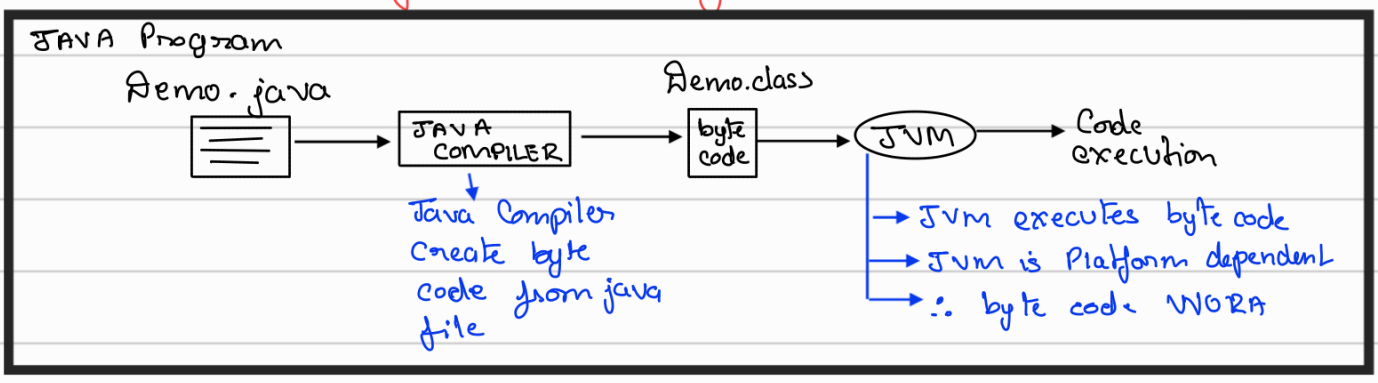
- These variables are called “class variables”.
- These variables will get memory in the method area.
- If the value does not change from object to object then we need to use static variables.
- Inside a static area we can access static variables only.
- Static variables are created using static keywords.

Non-static

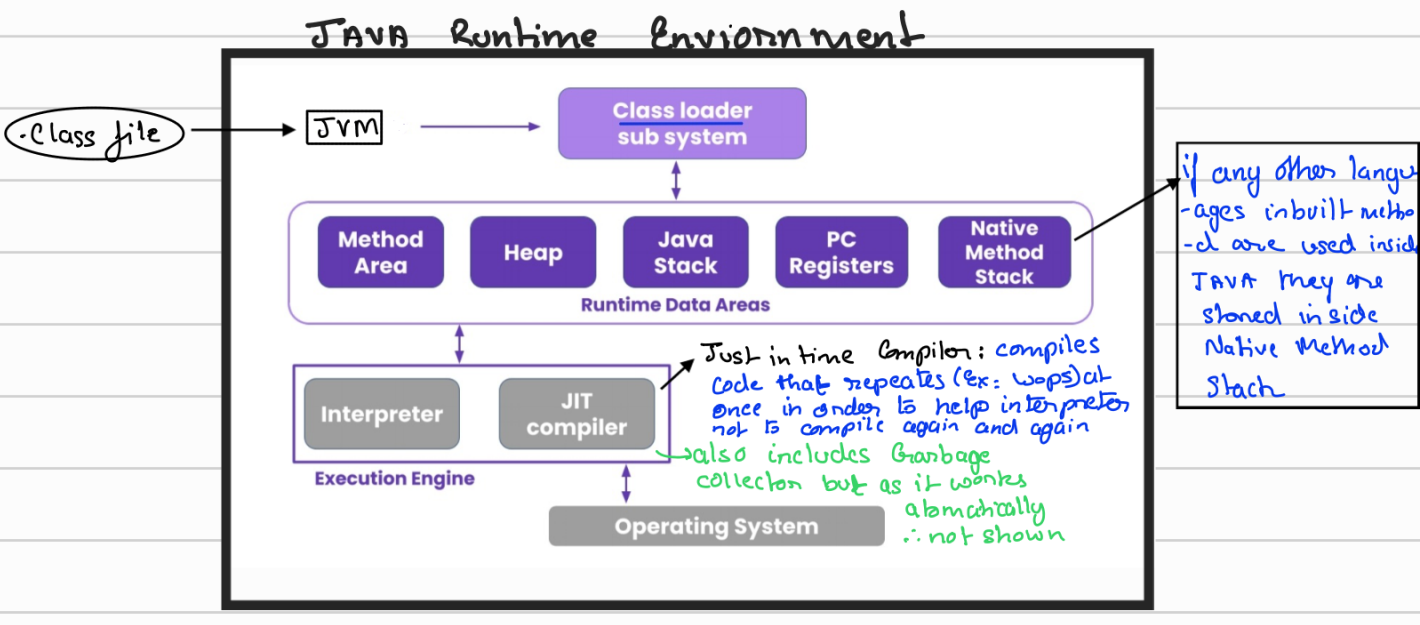
- These variables are called “instance variables”.
- These variables will get memory in the heap area.
- If the value changes from object to object then we need to use “non-static” variables.
- Inside a nonstatic area we can access both static and non-static variables.
- Non-static variables are created without using the “static” keyword.

Static Keyword:-

Q) How JAVA Program actually executed by JVM?



JVM is present inside the JAVA Runtime Environment (JRE). Whenever JVM encounters a class file, to execute the file JVM brings the file inside the JRE in order to execute it.



Q) Is Java a compiled or interpreted programing language?

Sol^n → Complied Programming Languages: Compiles the whole code first and then executes it

Ex: C

Interpreted Programming languages:- Interprets / Executes the code line by line.

Java is both compiled and Interpreted Programming language.

Q) How Classloader performs Class loading?

Sol^n -> Class loader performs Class loading.

Class loading is a process of loading .class file into JVM

Class loading is performed in 3 phases

① Loading —

② Linking

③ Initialization

① Loading: 1st loads all java packages (like util, long etc.), 2nd application class loader loades the .class file, then it loads all the imported files insed it etc.

② Linking:- 2.1) Verifying :- Once .class file loaded class loader verifies if all the code is written in JAVA standard or not. If not the verify exception is given

2.2) Prepare: Preparation is an imp. stage, where memory is allocated inside the Heap area for any static variable and then the default value is given.

2.3) Resolution: at this stage all the subclasses or class are resolved

③ Initialization:- initializes class variable with default value
from static execution block.

Static variable are requride to be initialized inside this block ex:-

```
static int age;
static{
    age = 10;
}
```

Note: if static variable is initialize inside static variable block then it is initialized in the loader stage.

Code execution Hierarchy:-

automatically called → Static variable (1st)

Static block

Only executes when we call for it

Static method (main block) (last)

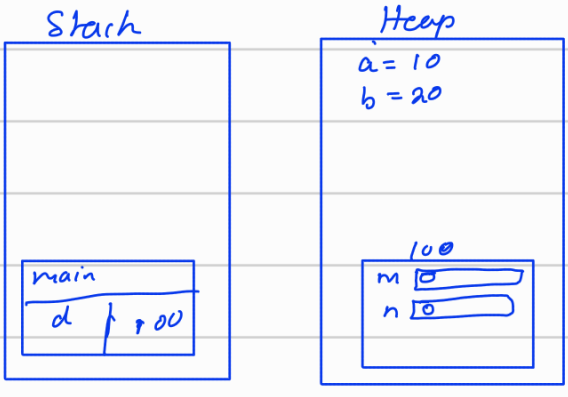
non static (instance)

non static block

non static method

```
Class demo{
    static int a;
    static int b;
    int m;
    int n;
    static {
            a=10;
            b=20;
            sop("Control in static block");
           }

    {
       m=100;
       n=200;
       sop("Control in non-static block");
    static void disp1()
    {
       sop(" value of static var: "+a);
      sop(" value of static var: "+b);
    }
    void disp2()
      {
        sop(" valu of non-static var: "+m);
        sop(" valu of non-static var: "+n);
      }
}

public class x{

    p s m v(String []args)
    {
       demo d = new demo();
       demo.disp1();
       d.disp2();
    }
}
```

- class file → JVM



Q) Can we create multiple objects using single class in JAVA?

Soln → Yes!

Q) What is the advantage of using static variable instead of class instances.

Soln -> As we know static variable are created inside the Heap area by class loader at the time of loading and they are created only once.

While the class instances (m, & n in above case) created every time an object is created which means every time an object is created different memory locations are allocated for the same instances.

So,

If there is a common value b/w all the objects of a same class then that variable can be created under static keyword (to save memory). "That is why static variables are also called "Class members" cuz common copy of data will be shared among all the objects memory will be allocated only once in the heap area at the time of class loading."

Q) Can non-static instances be called in static methods?

Soln -> Non-static instances inside Static :- As non-static instances are created at the instance of creation of an object so, they can not be called inside a static block or a static method because static block / variable / methods are created at the time of class loading.

Static variable insid Non-static :- Yes! cuz they are loaded at the time of class loading.