**JAVA**

1. **String ::**

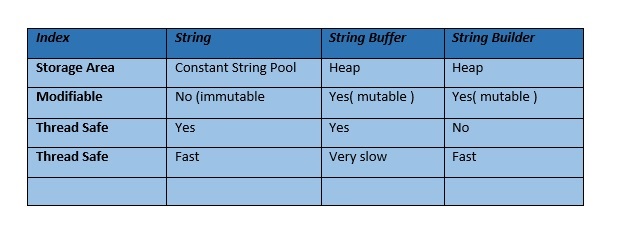
* Srting Is class in java and defined In java.lang package
* String class represents character Strings.
* String in [immutable](https://www.journaldev.com/129/how-to-create-immutable-class-in-java) and final in Java and JVM uses String Pool to store all the String objects.  
  Some other interesting things about String is the way we can instantiate a String object using double quotes and overloading of “+” operator for concatenation
* **Security**: parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. If it were mutable, these parameters could be easily changed.
* **Synchronization** and concurrency: making String immutable automatically makes them thread safe thereby solving the synchronization issues
* Ways to create string in java :

We can create String object using new operator like any normal java class or we can use double quotes to create a String object.

* String str = new String("abc");
* String str1 = "abc";

2 **String builder string buffer ::**

StringBuilder is much faster than StringBuffer because It's non synchronized. Better use StringBuilder since it is not synchronized and therefor better performance. StringBuilder is a drop-in replacement of the older StringBuffer. StringBuffer is synchronized, but StringBuilder is not.



String builder : StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously

StringBuilder is more efficient than StringBuffer. String buffer is opposite

String abc = “Hello”

abc.concat(“hi”);

but the value of abc will not change because string is immutable wee have to assign the value into another variable.

But in stringbuffer and buffer we can append the value.

3) Java is a **programming language** and a **platform**.

Java is a high level, robust, object-oriented and secure programming language.

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

**JAVA SE :** java standard edition

It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. I

**JAVA J2EE :: java enterprise edition**

**1)**

**JVM::**java virtual machine provide runtime environment in which java byte code can be excecuted

**JRE::** java run time environment. Which implements jvm and provides all the class libraries and other files that jvm uses at runtime.

**JDK ::** java development kit. Physically it contains jre + development tolls. Necessary to compile the java program.

2) **Synchronization ::** process which keeps all concurrent threads in execution in sync. Ex one movie ticket so many user trying to book that ticket but the first person lock the ticket than other can not book and will get the message.

3) **Process and thread::** A process, in the simplest terms, is an executing program. One or more threads run in the context of the process. A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread.

4) **Wrapper Class::** wrap the java primitive data type into and object of that class

**int** i = 10;

Integer iref = **new** Integer(i); // Boxing construction the object

System.***out***.println(iref);

**int** j = iref.intValue(); // unboxing converting object into the value

Integer kref = i; // autoBoxing

**int** k = kref.intValue(); // autounboxing

// boxing is used to convert the primitive data type into the object and strically following the OOPS



5) **final ::**  final is used to apply restriction on class method and variable

* Final class can’t inherited
* Final method cant be overridden
* Final variable value cant be changed

**Is final method inherited?**

Ans) Yes, final method is inherited but you cannot override it. For Example:

**Can we initialize blank final variable?**

Yes, but only in constructor. For example:

**Can we declare a constructor final?**

No, because constructor is never inherited.

**finally ::** finally used in exception it always executed weather exception is occurred or not. Without catch we can write finally.

**finalize:**: used to perform clean up processing just before object is garbage collected.

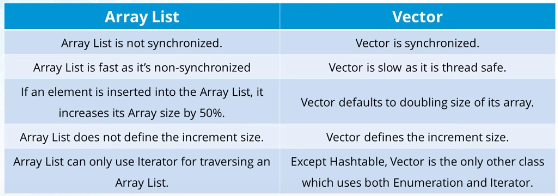
System.gc();

Java Runtime class is used to interact with java runtime environment. The java.lang.Runtime.gc() method runs the garbage collector. Calling this method suggests that the JVM (Java virtual machine) expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse.

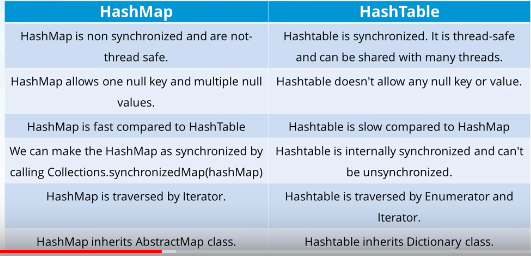
6) **Stack and Heap Memory::**

* Memory: when we use new keyword construct the memory location in heap, anything is constructed dynamically at run time in program resided in hea
* Lifetime: stack exists until the end of thread execution , heap memory lives from start till end of the application.
* Usages: stack is used for contains local primitives and reference variable , but whenever is object created stored in heap

7) **ArrayList and Vector**::



8) **HashMap and HashTable:**



9) Diff blw equals() and == operator

== used to comparing the refrences

.equals used to compare the values

1. Main difference between .equals() method and == operator is that one is method and other is operator.
2. We can use == operators for reference comparison (address comparison) and .equals() method for content comparison. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not override the equals method, then by default it uses equals(Object o) method of the closest parent class that has overridden this method. See this for detail

Coding Example:

// Java program to understand

// the concept of == operator

public class Test {

public static void main(String[] args)

{

String s1 = new String("HELLO");

String s2 = new String("HELLO");

System.out.println(s1 == s2);

System.out.println(s1.equals(s2));

}

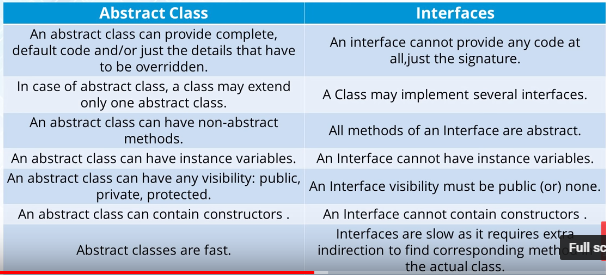
}

ANS false

True

**10) Diff blw abstract and interface::**

Both are used to achieve the run time polymorphism



11) **Polymorphism ::** one interface many implementation polymorphism means many form

12) can we override the private and static method in java

ANS: NO, private can not be override since it is not visible

That called class to class overriding knows as hiding

13) java support multiple inheritance

ANS: NO

14) why java Is popular

ANS: 1) platform independent 2) object oriented language

You're right, **platform independence means** that the same program works on any**platform** (operating system) without needing any modification. In the case of **Java**the application runs in a **Java** Virtual Machine which itself isn't **platform independent,** build once run anywhere

15 java and C++

* java platform dependent c++ not
* java java based on oops but c++ based on structured programming language
* c++ has pointers java has no concept
* c++ support multiple inheritance but java not support

16  **Constructor :**: In Java, a constructor is a block of codes similar to the method. It is called when an instance of the object is created, and memory is allocated for the object

Every time an object is created using new() keyword, at least one constructor is called. It calls a default constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

**Constructor Overloading in Java**

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Que: constructor return any value ?

Yes it return the current class instance

Que : can constructor perform other task instead of initiation

Yes object creation starting of thread calling method etc.

17) **Static ::** The **static keyword** in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class

* **Static variable:** The static variable gets memory only once in the class area at the time of class loading.It makes your program **memory efficient** (i.e., it saves memory).
* **Static Method:**

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

**Restrictions for the static method**

* The static method can not use non static data member or call non-static method directly.
* this and super cannot be used in static context.

**Why is the Java main method static?**

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation

**Java static block**

* Is used to initialize the static data member.
* It is executed before the main method at the time of classloading.
* Static block is called at the time of class loading

class A2{

static{System.out.println("static block is invoked");}

public static void main(String args[]){

System.out.println("Hello main");

}

}

**Q) Can we execute a program without main() method? ANS ::**NO

**18) this::**

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usages of ths keywords::

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

**this**();  directly invocked the constructor of the class

19) **Inheritance::**

* **single level :: A -- > B**
* **multilevel :: A 🡪 B 🡪 C**
* **hierarchical 🡪 A🡪 B A🡪C**

**Why multiple inheritance is not supported in java?**

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

**Example :** Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

20) **Super ::** The super keyword in Java is a reference variable which is used to refer immediate parent class object

## Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.