JAVA

1. Variable : Is something that hold temporary memory

Char always denote in single quota

Ex:- char gender = ‘M’;

2. Scanner defined in java.util package

Scanner sc = new Scanner(System.in);

3. comments in java two way

Single line : //

Multiple line: /\* \*/

4. DataType conversion

int , double ,Boolean to String

String name = String.valueOf(intValParameter);

String , double convert to int

Int age = Integer.parseInt(StringVal);

Int age1 = (int) doubleVal;

Same for int string to double

5. Pre-increment post-increment :

i++ , ++i

i-- , --i

6 Nested for loop for loop inside for loop

7) Continue : continue in for loop ignore the below statement in code and break kill the repeatation on for loop

8 ) We have so much values i can not assing value to diffrent variable for this we use array

array work on index position

array size is fixed

9) Polymorphism : Same method , same name ,same dataType but different no of inputs called polymorphism. Overloading , Overridning

10) Global and local variable: local variable declare and access inside the function but global variable can be used inside the program.

11) Garbage Collection: Java used garbage collection means collect the garbage that will not use.

12) Why Java has no destructor :- Because java use garbage collector it does not require any destructor like C++

13) Constructor: its member method.

A constructor in Java is a block of code similar to a method that's called when an instance of an object is created. Constructor has doesn’t return type and and name should be same as class name.

Constructor can be more than one but parameter should be diffrent

Constructor can be normal or parameterised.

14) Inheritance: Inheritance in Java is a mechanism in which one object acquires all

the properties and behaviours of a parent object

15) Encapsulation : is a process of wrapping code and data together into a single unit Limit access to some variable or valuable using private public or pratiuqe.

This limit the direct access. First we have to create getter and setter for that particular private variable.

**Method Overloading** is a feature that allows a class to have more than one **method** having the same name, if their argument lists are different.

An **overloaded method** may or may not have different **return types**. But **return type** alone is not sufficient for the compiler to determine which **method** is to be executed at run time. It's not possible to have a **method** with **same** parameters and different **return type**. Compiler throws error in the below case(Duplicate **method**).

16) Overriding:-

* In java, a method can only be written in Subclass, not in same class.
* The argument list should be exactly the same as that of the overridden method.
* The return type should be the same or a subtype of the return type declared in the original overridden method in the super class.
* Use @override keyword to override the method

17)  **this keyword:**

It (this) works as a reference to the current Object, whose Method or constructor is being invoked.

18) **Super** keyword is used to access the main class method or variable.

**Java** Programming/Keywords/**super**. It is used inside a sub-class method definition

to call a method defined in the **super** class. ... Only public and protected methods can be called by the **super** keyword. It is also used by class constructors to invoke

constructors of its parent class

19) **Abstract :** if you have declare class as abstract that means you don’t want to allow

Anyone to create the instance of that class. if anyone wants to use he has to inherit first.

If method is abstract class will be abstract.

abstract method has no body.

It’s not necessary to have abstract class to have abstract method.

If abstract class doesn’t have any method implementation, its better to use interface because java doesn’t support multiple class inheritance.

**Imp:**

If abstract class doesn’t have any method implementation, its better to use interface because java doesn’t support multiple class inheritance.

**public abstract class Abstract1**

**Interface:**

Interface need to implement

Interface in java provide a way to achieve [abstraction](https://www.journaldev.com/1582/abstract-class-in-java). Java interface is also used to define the contract for the subclasses to implement.

**Public interface Shape{**

**}**

We can’t instantiate an interface in java.

Interfaces can’t have constructors because we can’t instantiate them and interfaces can’t have a method with body.

By default any attribute of interface is **public**, **static** and **final**, so we don’t need to provide access modifiers to the attributes but if we do, compiler doesn’t complain about it either.

Interface cannot be private

Interface cant extend interface only implement and class can extend

Implements keyword is used by classes to implement an interface.

Interface can implements more than one interface in java.

**Disadvantages:**

1. We need to choose interface methods very carefully at the time of designing our project because we can’t add of remove any methods from the interface at later point of time, it will lead compilation error for all the implementation classes. Sometimes this leads to have a lot of interfaces extending the base interface in our code that becomes hard to maintain.

**Collection:**

Collection :Its interface

Collections :-its class

arrayList implements list and extends Collection

we got generic in 1.5 its denoted in <> agle brackets

collections not support indexing.

We can sort the list by Collection.sort(list);

Reverse : Collection.reverse(list);

**Vector :** Initial capacity of vector Is 10

**Vector is dynamic its increase the size by 100%**

20) **ArrayList**: The **ArrayList** class extends AbstractList and implements the List interface. **ArrayList** supports dynamic arrays that can grow as needed. Standard **Java** arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold.

ArrayList<String> ar = **new** ArrayList<String>(); //generaic arrayList by string

ar.add("viin");

ar.add("rahul");

ar.add("abhi");

ar.remove(2);

**for**(String ar1 : ar){

System.***out***.println("Name::"+ar1);

}

**Diff:**

1. Vector increase memory by 100% but arraylist by 50% that means it save the memory
2. Vector thread safe but arryList not.
3. Vector is slow and arrayList is fast

**arrayList is best as compare to vector.**

**How to print duplicate value of arrayList:**

ArrayList<String> ar = **new** ArrayList<String>();

ar.add("vipin");

ar.add("abhis");

ar.add("rahul");

ar.add("vipin");

Set<String>set =**new** HashSet<>();

**for**(String s : ar){

**if**(set.add(s)==**false**){

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

}

}

**21 LinkedList:**

**Linked List in Java**. A **linked list** is a data structure that consists of a group of nodes representing a sequence together.

* When lot of delete and update required use linked list because array list take time to shift
* When you have read lots of data use array list
* Linked not not worked on index basics it work on basics of elements.
* Linked List are linear data structures where the elements are not stored in contiguous locations and every element is a separate **object with a data part and address part.** The elements are linked using pointers and addresses. **Each element is known as a node.** Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. It also has few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach to a node we wish to access.

In computer programming, an **iterator** is an object that enables a programmer to traverse a container,

LinkedList<String>ar = **new** LinkedList<String>();

ar.add("rahul");

ar.add("Vipin");

ar.add("abhishek");

Iterator<String>itr= ar.iterator();

**while**(itr.hasNext()){

System.***out***.println(itr.next());

}

**ArrayList and LinkedList** are two popular collection classes in Java and Major **difference between ArrayList and LinkedList** is on there implementation while **ArrayList** uses index number based Array **LinkedList** is implemented using Doubly **Linked List** Data Structure.

**Comparator:**

**Java Comparator interface** is used to order the objects of user-defined class.

**Its interface so need to implement the interface**

**public** **int** compare(Integer o1, Integer o2) {

**if**(o1%10>o2%10){

**return** 1;

}

**return** -1;

}

Return 1 means ascending order else -1 means descending order.

**Comparable:**

Java Comparable interface is used to order the objects of user-defined class.This interface is found in java.lang package and contains only one method named compareTo(Object). It provide single sorting sequence only i.e. you can sort the elements on based on single data member only. For example it may be rollno, name, age or anything else.

**Difference:**

**Comparator** is used to **compare** two datatypes are objects. If you see then logical **difference between** these two is **Comparator** in Java **compare** two objects provided to him, while **Comparable interface**compares "this" reference with the object specified.**Comparable** in Java is used to implement natural ordering of object.

**22. HashMap:**

**Map not support duplicate keys.**

**HashMap** is a part of **Java's** collection since **Java**1.2. It provides the basic implementation of Map interface of **Java**. It stores the data in (Key, Value) pairs. To access a value one must know its key.**HashMap** is known as **HashMap** because it uses a technique called Hashing.

**Hashing** is transforming a given entity (in **java** terms - an object) to some number

HashMap<Integer, String> hm = **new** HashMap<Integer, String>();

hm.put(1, "abhishek");

hm.put(2, "rahul");

hm.put(3, "vippipn");

**for**(Map.Entry m:hm.entrySet()){

System.***out***.println("key:"+m.getKey()+" value:"+m.getValue());

}

System.***out***.println(hm.size());

hm.put(4, "sharma");

hm.remove(4);

Inside map Entry is interface which gives the entry set

Hashtable is synchronised but map is not

Hashtable is threadsafe but map not

Map is fast

Synchronied menas wait to stop the current process.

Hashmap provide the value in sorted way on the basics of key if you want the value the in the form of insertion order use linkedHashMap.

**23) HashSet:** Same with linked list but set follow element can not be different.

24) **TreeSet :** Same as tree but **TreeSet** are sorted according to their natural ordering

**SET** is an interface which extends collections

Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).

**Hashset ::** no duplicate value

**Tress set ::** no duplicate value and sorted data

**LinkedHashSet** ::

A HashSet is unordered and unsorted Set. LinkedHashSet is the ordered version ofHashSet.The only difference between HashSet and LinkedHashSet is thatLinkedHashSet maintains the insertion order.

Set<String> hash\_Set = new HashSet<String>();

        hash\_Set.add("Geeks");

        hash\_Set.add("For");

        hash\_Set.add("Geeks");

        hash\_Set.add("Example");

        hash\_Set.add("Set");

        System.out.print("Set output without the duplicates");

        System.out.println(hash\_Set);

        // Set deonstration using TreeSet

        System.out.print("Sorted Set after passing into TreeSet");

        Set<String> tree\_Set = new TreeSet<String>(hash\_Set);

        System.out.println(tree\_Set);

25) **Enum:** Means you have some data that will never change like weekdays

enum Color

{

    RED, GREEN, BLUE;

}

Use enum in place of class

26) **for loop and enhanced for loop :**

In for loop first we count and than print according to the index but in enhanced for loop directly print.

For(int a : val){

}

In enhanced for loop we start from 0 and print all the values. but in for loop we can specify.

27) **Pakages :**

Inuput output class in java.io package

Network releated in java.net

Jdbc in java.sql

Collections in java.util

28) **Access modifier:**

Main class can not be private. Main class can be only public abstract and final

Without public can not access class outside the package

Protected can not access outside the package.

29) Convert java file to class file use javac for compile the class. And to to run the class use java than class name.

30) how to get the count of object in java.

Simple with the help of class variable

Vedio link: <https://www.youtube.com/watch?v=zZlqRkgNSrg&index=72&list=PLGwb7xZHg-oMv1pOlTHAqAEjw0EPALzlL>

**public** **class** Count\_Obj {

**static** **int** *i*;

**public** Count\_Obj(){

*i*++;

System.***out***.println("count"+*i*);

}

**void** run(){

}

}

31) static block is called when your class loaded inn jvm. No need to call static block

we can create more than one static block in same class.

Static{

System.out.pprintln(“hello”);

}

32) **Inner class: 1 member**

**2 static**

**3 anonymous**

We can not access inner class directly.

With the help of member class we can access inner class.

InnerClass a = **new** InnerClass();

a.nameA=00;

InnerClass.B b = a.**new** B();

How to use static inner class in java

<https://www.youtube.com/watch?v=okruEgWGVGU&list=PLGwb7xZHg-oMv1pOlTHAqAEjw0EPALzlL&index=146>

**33 Excetion handling:**

Exception we can handle but errors we cannot (ex jvm is out of memory cpu not able to handle the load)

Checked exception and unchecked exception

**Throable**

**Exception:**

**-Checked - IOException**

**- SQLException**

**-Unchecked – RunTimeException**

**Errors:**

If catch the exception execution not stop and program run continuously.

Throws keyword use with class name and throw method use in inside the method or program.

**public** **static** **void** main(String[] args) **throws** Exception {

**try**

{

**int** i=5;

**int** j=0;

**int** k = i/j;

**throw** **new** ArithmeticException();

}

**catch**(Exception e){

System.***out***.println("error"+e);

}

Finally{

System.***out***.println("elase part");

}

}

}

User defined exception class

**public** **class** UserDefinedException {

**public** **static** **void** main(String[] args) {

**int** i=3;

**try**{

**if**(i<4){

**int** k = 10/i;

**throw** **new** myException("errro");

}

}**catch**(myException e){

}

}

}

**class** myException **extends** Exception{

**public** myException(String msg) {

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

}

}

34 **MultiThreading:**

**Thread:**unit of process

**Using complete power of CPU**

**1** if you want to create thread class it necessary to extends thread

than create the object of thread

Thread t1 = new Thread();

t1.start();

**2** Runnable is interface

3 If you create thread you can run your process parallel.

4 runnable interface have only one method call run()

5 if we create thread by runnable we have to create the two thread and runnable object

<https://www.youtube.com/watch?v=xvXbvrUUGMM&index=3&list=PLsyeobzWxl7rmuFYRpkqLanwoG4pQQ7oW>

6 isAlive(): metod is used to check the thread is alive or dead. output in form of true or false

7) join(): method : This java thread join method puts the current thread on wait until the thread on which it’s called is dead. If the thread is interrupted, it throws InterruptedException.

8) thread name and thread prioriry

Thread.setName();

Thread.getName();

Thread.serPriority(1 to 10);

1 is least priority and 10 is highest priority and 5 for NOM priority

We can set Thread.MAX\_PRIORITY , Thread.MIN\_PRIORITY , NOM\_PRIORITY

9) synchronized : if you not declare method as synchronised that means your method is not thread safe.

**JAVA 8 features**

1. Till java 1.7 we can declare method in interface but in 8 we can

By default interface methods are public abstract final.

1. If class extends one class and implements one interface with same method priority will be class.
2. If we have 2 interface with same method than we have to implements the method in class else will get completion error.
3. Only **default** method can be create in interface java 8
4. Now in 8 we can create static method and directly call by class name dot method name.
5. Example of method define

**interface** phone{

**void** message(); //java 1.7

**default** **void** drawPatern(){

System.***out***.println("drawPatter-->>"); // java 1.8

}

}

1. forEach Method :

for loop and for each is external loop but java launched forEach internal method

which is faster than for and for each loop

List<Integer>ar = Arrays.asList(1,2,3,4);

ar.forEach(i ->System.out.println(i));

1. Lambda expression:

**Why use Lambda Expression**

To provide the implementation of Functional interface.

Less coding.

**Syntax:**

(argument-list) -> {body}

Java lambda expression is consisted of three components.

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

3) Body: It contains expressions and statements for lambda expression.

Consumer is function interface which is used in for Each method and its support accept method with one integer variable.

//Without Lambda Expression

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("Implementation without Lambda Expressions");

}

}).start();

//With Lambda Expression

new Thread( () -> System.out.println("Implementation with Lambda Expression") ).start();

1. **StreamAPI: ??**

Introduced in Java 8, the Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

1. **Date Time API:**

Why new api because in old version two packages support date feature

Java.util;.Date;

Java.sql.Data;

If we want to format the data new package need to be import- java.text.\*

LocalDate and the package is:- java.time.localDate;

LocalDate d = LocalDate.*now*();

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

System.***out***.println("day"+d.getDayOfMonth());

LocalDate d1 = LocalDate.*of*(1993, Month.***JULY***, 17);

System.***out***.println(d1);

LocalTime t = LocalTime.*now*();

System.***out***.println("t:"+t);

LocalTime t1 = LocalTime.*of*(5, 45);

System.***out***.println("t1:"+t1);

LocalTime t2 = LocalTime.*now*(ZoneId.*of*("Turkey"));

System.***out***.println("t2:"+t2);

1. **Method refrence:**

**?? navin reddy**