**Q1. Explain** [**JVM, JRE and JDK**](https://www.geeksforgeeks.org/differences-jdk-jre-jvm/)**?**

**JVM (Java Virtual Machine):** JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a Java code. JVM is a part of JRE(Java Runtime Environment).

**JRE (Java Runtime Environment):** JRE refers to a runtime environment in which Java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. Basically, it’s an implementation of the JVM which physically exists.

**JDK(Java Development Kit):** It is the tool necessary to compile, document and package Java programs. The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. In short, it contains JRE + development tools.

**Q2. Explain public static void main(String args[]).**

**Public:** Public is an access modifier. Public means that this Method will be accessible by any Class.

**static :** It is a keyword in java which identifies it is class-based i.e it can be accessed without creating the instance of a Class. Since we want the main method to be executed without any instance also, we use static.

**Void:** It is the return type of the method. Void defines the method which will not return any value.

**main:** This is the first method executed by JVM. The signature of the method must be the same.

**Q3. Why Java is platform independent?**

Platform independent practically means “write once run anywhere”. Java is called so because of its **byte codes** which can run on any system irrespective of its underlying operating system.

**Q4. Why is Java not pure Object-oriented?**

Java is not considered pure Object-oriented because it supports primitive data-types such as boolean, byte, char, int, float, double, long, short.

**Q5. Define class and object. Explain them with an example using java.**

**Class:** A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

**Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

**Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.

**Object:** It is a basic unit of Object Oriented Programming and represents the real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

**State :** It is represented by attributes of an object. It also reflects the properties of an object.

**Behavior :** It is represented by methods of an object. It also reflects the response of an object with other objects.

**Identity :** It gives a unique name to an object and enables one object to interact with other objects.

**Q6.What is a method? Provide several signatures of the methods**

A Java method is a set of statements to perform a task. A method is placed in a class.

**Q7.Explain the difference between instance variable and a class variable.**

An instance variable is a variable which has one copy per object/instance. That means every object will have one copy of it.

A class variable is a variable which has one copy per class. The class variables will not have a copy in the object.

**Q8. Which class is the superclass of all classes?**

[java.lang.Object](https://www.geeksforgeeks.org/object-class-in-java/) is the root class for all the java classes and we don’t need to extend it.

**Q9.What are constructors in Java?**

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.

If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor.

This default constructor calls the class parent’s no-argument constructor (as it contains only one statement i.e. super();), or the Object class constructor if the class has no other parent (as Object class is a parent of all classes either directly or indirectly).

There are two types of constructors:

1. Default constructor
2. Parameterized constructor

**Q10. What are the different ways to create objects in Java?**

There are many different ways to create objects in Java. Please see [5 Different ways to create objects in Java](https://www.geeksforgeeks.org/different-ways-create-objects-java/)

**Q11. What’s the purpose of Static methods and static variables?**

When there is a requirement to share a method or a variable between multiple objects of a class instead of creating separate copies for each object, we use static keyword to make a method or variable shared for all objects.

**Static variable:** Static variables are also known as Class variables.

These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.

Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.

Static variables are created at the start of program execution and destroyed automatically when execution ends.

To access static variables, we need not create an object of that class.

Static methods: A static method can be accessed without creating objects. Just by using the Class name the method can be accessed. The static method can only access static variables and not local or global non-static variables.

**Q12. Why static methods cannot access non-static variables or methods?**

Ans) A static method cannot access non-static variables or methods because static methods can be accessed without instantiating the class, so if the class is not instantiated the variables are not initialized and thus cannot be accessed from a static method.

**Q13.What is a static class?**

A class can be said to be static class if all the variables and methods of the class are static and the constructor is private. Making the constructor private will prevent the class to be instantiated. So the only possibility to access is using the Class name only.

**Q14. How many types of Variable? Explain.**

There are three types of variables in Java:

1. Local Variables
2. Instance Variables
3. Static Variables

**Local Variables:** A variable defined within a block or method or constructor is called local variable.

**Instance Variables:** Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.

**Explain Final keyword in java?**

Final keyword in java is used to restrict usage of variable, class and method.

**When is the super keyword used?**

Super keyword is used to refer:

* immediate parent class constructor,
* immediate parent class variable,
* immediate parent class method.

**What is the difference between StringBuffer and String?**

String is an Immutable class, i.e. you can not modify its content once created. While StringBuffer is a mutable class, means you can change its content later. Whenever we alter content of String object, it creates a new string and refer to that,it does not modify the existing one. This is the reason that the performance with StringBuffer is better than with String.

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

Java supports multiple inheritance but not through classes, it supports only through its interfaces. The reason for not supporting multiple inheritance is to avoid the conflict and complexity arises due to it and keep Java a Simple Object Oriented Language.

**Can a top level class be private or protected?**

Top level classes in java can’t be private or protected, but inner classes in java can. The reason for not making a top level class as private is very obvious, because nobody can see a private class and thus they can not use it. Declaring a class as protected also doesn’t make any sense.

**What is the difference between ‘throw’ and ‘throws’ in Java Exception Handling?**

Following are the differences between two:

* throw keyword is used to throw Exception from any method or static block whereas throws is used to indicate that which Exception can possibly be thrown by this method
* If any method throws a checked Exception, then the caller can either handle this exception(using try catch block )or can re-throw it by declaring another ‘throws’ clause in method declaration.
* throw clause can be used in any part of code where you feel a specific exception needs to be thrown to the calling method

**What is the finalize() method?**

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class.

**Difference in Set and List interface?**

Set and List both are child interfaces of Collection interface. There are following two main differences between them

* List can hold duplicate values but Set doesn’t allow this.
* In List interface data is present in the order you inserted but in the case of Set insertion order is not preserved.

**What will happen if you put System.exit(0) on a try or catch block? Will finally block execute?**

By Calling System.exit(0) in the try or catch block, we can skip the finally block. System.exit(int) method can throw a SecurityException. If System.exit(0) exits the JVM without throwing that exception then the finally block will not execute. But, if System.exit(0) does throw a security exception then finally the block will be executed.

# **Different ways to create objects in Java**

There are many different ways to create objects in Java. Let us list them later, discussinglater individually with the help of programs to illustrate internal working by which we can create objects in Java.

1. Using new keyword
2. Using new instance
3. Using clone() method: - In order to use the clone() method on an object we need to implement [Cloneable](https://www.geeksforgeeks.org/cloneable-interface-in-java/) and define the [*clone() method*](https://www.geeksforgeeks.org/clone-method-in-java-2/) in it.
4. Using deserialization
5. Using newInstance() method of Constructor class

There are two ways to create an object of string in java:

1. **By string literal**
2. **By new keyword**

### **i) By string literal:**

This is done using **double-quotes.**

Every time when a string literal is created, JVM will check whether that string already exists in the *string constant pool* or not. If the string already exists in the string literal pool then a reference to the pooled instance is returned. If the string does not exist, then a new string instance is created in the pool. Hence, only one object will get created.

Here, the JVM is not bonded to create a new memory.

### **ii) By new keyword:**

This is done using a **new** keyword.

Every time when a string object is created, JVM will create it in a heap memory. In this case, the JVM will not check whether the string already exists or not. If a string already exists , then also for every string object the memory will get created separately.

Here, the JVM is bound to create a new memory.

# **Passing and Returning Objects in Java**

Although Java is strictly passed [by value](https://www.geeksforgeeks.org/g-fact-31-java-is-strictly-pass-by-value/),

# **Lambda Expressions in Java 8**

lambda expressions are added in Java 8 and provide below functionalities.

* Enable to treat functionality as a method argument, or code as data.
* A function that can be created without belonging to any class.
* A lambda expression can be passed around as if it was an object and executed on demand.

FuncInterface fobj = (int x)->System.out.println(2\*x);

fobj.abstractFun(5);

**lambda operator -> body**

**where lambda operator can be:**

**Zero parameter:**

**() -> System.out.println("Zero parameter lambda");**

**One parameter:–**

**(p) -> System.out.println("One parameter: " + p);**

**It is not mandatory to use parentheses, if the type of that variable can be inferred from the context**

**Multiple parameters :**

**(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);**

# **Serialization and Deserialization in Java**

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

To make a Java object serializable we implement the **java.io.Serializable** interface.ObjectOutputStream class contains **writeObject()** method for serializing an Object.

**SerialVersionUID** The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization.

If the receiver has loaded a class for the object that has a different UID than that of corresponding sender’s class, the Deserialization will result in an **InvalidClassException**.

In case of **transient variables:-** A variable defined with transient keyword is not serialized during serialization process.This variable will be initialized with default value during deserialization.

Transient Vs Final:

**final** variables will be participated into serialization directly by their values.

Hence declaring a final variable as transient there is no use.

# **Garbage Collection in Java**

Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

The garbage collector is the best example of the [Daemon thread](https://www.geeksforgeeks.org/daemon-thread-java/) as it is always running in the background.

# Volatile Keyword in Java

Volatile keyword is used to modify the value of a variable by different threads. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem. The volatile keyword can be used either with primitive types or objects.

1. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
2. [**strictfp**](https://www.javatpoint.com/strictfp-keyword)**:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
3. [**transient**](https://www.javatpoint.com/transient-keyword)**:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
4. [**volatile**](https://www.javatpoint.com/volatile-keyword-in-java)**:** Java volatile keyword is used to indicate that a variable may change asynchronously.

# Java Functional Interfaces

An Interface that contains exactly one abstract method is known as a functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve a functional programming approach.

A functional interface can extend another interface only when it does not have any abstract method.

[**What happens to a thread as soon as it has completed its assigned task in java?**](https://stackoverflow.com/questions/19960798/what-happens-to-a-thread-as-soon-as-it-has-completed-its-assigned-task-in-java)

we should not create many threads to accomplish our task, it will give you many issues like OutofMemoryError. And also creation of thread is a costly task, so we should think of Thread pool i.e. ExecutorService in which we reuse the same threads again and again.

But any ways to answer your question after threads are created they die automatically i.e. it will be garbage collected, you don't need to do anything. Initially Java provided methods like stop() or destroy() but these are deprecated for good reason.

# **Atomic Variables in Java**

In Java, **atomic variables are used** in concurrency. If multiple threads access & update the value of the same variable. Hence, they may lead to inconsistency of the program. It is also used to make classes thread safe.

Ex. int count = 0;

public void run(){

int max = 1\_000\_00\_000;

for (int i = 0; i < max; i++) {

count++;

}}

If we execute multiple threads then the result of count will not be accurate because of concurrency.

So will use AtomicInteger count=new AtomicInteger();

run(){count.addAndGet(1);}

## Synchronized Vs. Atomic Vs. Volatile

| **Synchronized** | **Atomic** | **Volatile** |
| --- | --- | --- |
| It applies to methods only. | It applies to variables only. | It also applies to variables only. |

### **The volatile variable can not be used when one variable's value depends upon others or its own increment value.**

### What is flattening?

# **Flattening** is the process of converting several lists of lists and merge all those lists to create a single list containing all the elements from all the lists.

# flatMap() Method

**it is used to convert a Stream of Stream into a list of values.**

# Get distinct objects by key

***public static <T> Predicate<T> distinctByKey(Function<? super T, ?> keyExtractor) { Set<Object> seen = ConcurrentHashMap.newKeySet(); return t -> seen.add(keyExtractor.apply(t)); }***

**List<Person> distinctElements = list.stream().filter( distinctByKey(p -> p.getFname())).collect( Collectors.toList() );**

**Or**

list = people.stream().collect(Collectors.toMap(Person::getAge, Function.identity(),(person1, person2) -> person1)).values();

A lambda expression in Java is a concise block of code that can be used to implement a functional interface. Lambda expressions are similar to anonymous inner classes, but they are more concise and easier to read.

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

Stream API is **a way to express and process collections of objects**. Enable us to perform operations like filtering, mapping,reducing and sorting.