Java interview preparation:

### **Q-01: What is Java?**

Java is a programming language and computing platform first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Java program which prints Hello World!

|  |  |
| --- | --- |
| 1  2  3  4  5 | public class MyClass {    public static void main(String args[]) {      System.out.println("Hello World!");    }  } |

### **Q-02: Mention some features of Java?**

Some of the features which play an important role in the popularity of java are as follows:

* Simple: Java is easy to learn. Even though Java is based on C++ , it was developed by eliminating poor programming practices of C++.
* Object-Oriented: Java is an object-oriented programming language. Everything in Java is an Object.

[Learn more here.](https://www.softwaretestingmaterial.com/oops-concept-in-java/)

* **Portable:** Java run time environment uses a byte code verification process to make sure that code loaded over the network doesn’t violate Java security constraints.
* **Platform independent:** Java is platform-independent. Java is a write once, run anywhere language. Without any modifications, we can use a program on different platforms.
* **Secured:** Java is well known for its security. It delivers virus-free systems.
* **High Performance:** Java enables high performance with the use of JIT (Just-In-Time) compilers
* **Multithreaded:** Java multithreaded features allows us to write programs that can perform many tasks simultaneously. The multithreading concept of Java shares a common memory area. It doesn’t occupy memory for each thread.

**Q-03: Differences between JDK, JRE and JVM**

### **JDK :** JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools.

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

### **JRE :** JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.

### **JVM :** JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java byte code can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](https://www.javatpoint.com/os-tutorial) is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

The JVM performs the following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

Learn more : <https://www.javatpoint.com/jvm-java-virtual-machine>

### **Q4: Explain public static void main(String args[]) in Java.**

main () in Java is the entry point for any Java program. It is always written as **public static void main(String[] args)**.

* **public**: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
* **static**: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as **main**() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
* **void**: It is the return type of the method. Void defines the method which will not return any value.
* **main**: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
* **String args[]**: It is the parameter passed to the main method.

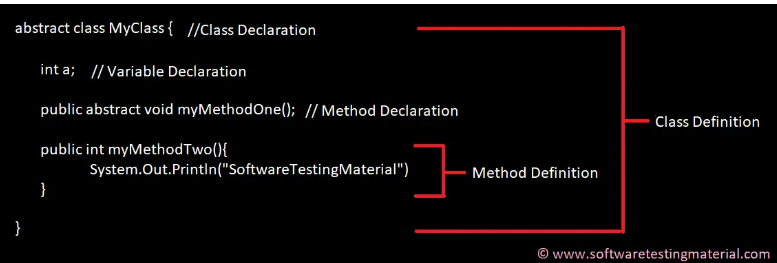
### **Q5: What is the difference between Declaration and Definition in Java?**

**Declaration:** If you just declare a class or method/function or variable without mentioning anything about what that class or method/function or variable looks like is called a declaration in Java.

**Definition:** If you define how a class or method/function or variable is implemented then it is called definition in Java.

When we create an interface or abstract class, we simply declare a method/function but not define it.

For a clear understanding, check the below image



### **Q6. Why Java is platform independent?**

Java is called platform independent because of its byte codes which can run on any system irrespective of its underlying operating system.

### **Q7. Why Java is not 100% Object-oriented?**

Java is not 100% Object-oriented because it makes use of eight primitive data types such as boolean, byte, char, int, float, double, long, short which are not objects.

### **Q5. What are wrapper classes in Java?**

Wrapper classes convert the Java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class..

### **Q8. 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.

There are two types of constructors:

1. **Default Constructor:** In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.
2. **Parameterized Constructor:** The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

### **Q9. What is singleton class in Java and how can we make a class singleton?**

Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

### **Q10. What is the difference between Array list and vector in Java?**

|  |  |
| --- | --- |
| **ArrayList** | **Vector** |
| Array List is not synchronized. | Vector is synchronized. |
| Array List is fast as it’s non-synchronized. | Vector is slow as it is thread safe. |
| If an element is inserted into the Array List, it increases its Array size by 50%. | Vector defaults to doubling size of its array. |
| Array List does not define the increment size. | Vector defines the increment size. |
| Array List can only use Iterator for traversing an Array List. | Vector can use both Enumeration and Iterator for traversing. |

### **Q11. What is the difference between equals() and == in Java?**

Both equals() method and the == operator are used to compare two objects in Java. == is an operator and equals() is method. But == operator compares reference or memory location of objects in a heap, whether they point to the same location or not.  
Whenever we create an object using the operator new, it will create a new memory location for that object. So we use the == operator to check memory location or address of two objects are the same or not.

In general, both equals() and “==” operators in[Java](https://www.geeksforgeeks.org/java-tutorial/) are used to compare objects to check equality, but here are some of the differences between the two:

1. The main difference between the [.equals() method](https://www.geeksforgeeks.org/method-class-equals-method-in-java/) and ==[operator](https://www.geeksforgeeks.org/operators-in-java/) is that one is a method, and the other is the 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.

### **Q12. When can you use the super keyword?**

In Java, the super keyword is a reference variable that refers to an immediate parent class object.

When you create a subclass instance, you’re also creating an instance of the parent class, which is referenced to by the super reference variable.

The uses of the Java super Keyword are-

1. To refer to an immediate parent class instance variable, use super.
2. The keyword super can be used to call the method of an immediate parent class.
3. Super() can be used to call the constructor of the immediate parent class.

### Q13. What makes a HashSet different from a TreeSet?

|  |  |
| --- | --- |
| **HashSet** | **TreeSet** |
| It is implemented through a hash table. | TreeSet implements SortedSet Interface that uses trees for storing data. |
| It permits the null object. | It does not allow the null object. |
| It is faster than TreeSet especially for search, insert, and delete operations. | It is slower than HashSet for these operations. |
| It does not maintain elements in an ordered way. | The elements are maintained in a sorted order. |
| It uses equals() method to compare two objects. | It uses compareTo() method for comparing two objects. |
| It does not permit a heterogenous object. | It permits a heterogenous object. |

**Q14. What are the differences between HashMap and HashTable in Java?**

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| It is non synchronized. It cannot be shared between many threads without proper synchronization code. | It is synchronized. It is thread-safe and can be shared with many threads. |
| It permits one null key and multiple null values. | It does not permit any null key or value. |
| is a new class introduced in JDK 1.2. | It was present in earlier versions of java as well. |
| It is faster. | It is slower. |
| It is traversed through the iterator. | It is traversed through Enumerator and Iterator. |
| It uses fail fast iterator. | It uses an enumerator which is not fail fast. |
| It inherits AbstractMap class. | It inherits Dictionary class. |

**Q15. How to not allow serialization of attributes of a class in Java?**

The Non-Serialized attribute can be used to prevent member variables from being serialized.  
You should also make an object that potentially contains security-sensitive data non-serializable if possible. Apply the Non-Serialized attribute to certain fields that store sensitive data if the object must be serialized. If you don’t exclude these fields from serialization, the data they store will be visible to any programs with serialization permission. Transient

**Q16. Can you call a constructor of a class inside another constructor?**

Yes, we can call a constructor of a class inside another constructor. This is also called as constructor chaining. Constructor chaining can be done in 2 ways-

1. **Within the same class:** For constructors in the same class, the this() keyword can be used.
2. **From the base class:** The super() keyword is used to call the constructor from the base class.  
   The constructor chaining follows the process of inheritance. The constructor of the sub class first calls the constructor of the super class. Due to this, the creation of sub class’s object starts with the initialization of the data members of the super class. The constructor chaining works similarly with any number of classes. Every constructor keeps calling the chain till the top of the chain.

**Q17. Contiguous memory locations are usually used for storing actual values in an array but not in ArrayList. Explain.**

An array generally contains elements of the primitive data types such as int, float, etc. In such cases, the array directly stores these elements at contiguous memory locations. While an ArrayList does not contain primitive data types. An arrayList contains the reference of the objects at different memory locations instead of the object itself. That is why the objects are not stored at contiguous memory locations.

**Q18. How is the creation of a String using new() different from that of a literal?**  
When we create a string using new(), a new object is created. Whereas, if we create a string using the string literal syntax, it may return an already existing object with the same name.

**Q19. Why is synchronization necessary? Explain with the help of a relevant example.**

Java allows multiple threads to execute. They may be accessing the same variable or object. Synchronization helps to execute threads one after another.  
It is important as it helps to execute all concurrent threads while being in sync. It prevents memory consistency errors due to access to shared memory. An example of synchronization code is-

### Programming & Frameworks Training

|  |  |
| --- | --- |
| 1  2  3  4 | public synchronized void increment()  {  a++;  } |

As we have synchronized this function, this thread can only use the object after the previous thread has used it.

**Q20. Why is it said that the length() method of String class doesn’t return accurate results?**

The length() method of String class doesn’t return accurate results because  
it simply takes into account the number of characters within in the String. In other words, code points outside of the BMP (Basic Multilingual Plane), that is, code points having a value of U+10000 or above, will be ignored.

The reason for this is historical. One of Java’s original goals was to consider all text as Unicode; yet, Unicode did not define code points outside of the BMP at the time. It was too late to modify char by the time Unicode specified such code points.

**Q21. What are the differences between Heap and Stack Memory in Java?**

The major difference between Heap and Stack memory are:

|  |  |  |
| --- | --- | --- |
| **Features** | **Stack** | **Heap** |
| **Memory** | Stack memory is used only by one thread of execution. | Heap memory is used by all the parts of the application. |
| **Access** | Stack memory can’t be accessed by other threads. | Objects stored in the heap are globally accessible. |
| **Memory Management** | Follows LIFO manner to free memory. | Memory management is based on the generation associated with each object. |
| **Lifetime** | Exists until the end of execution of the thread. | Heap memory lives from the start till the end of application execution. |
| **Usage** | Stack memory only contains local primitive and reference variables to objects in heap space. | Whenever an object is created, it’s always stored in the Heap space. |

### **Q22. What is a package in Java? List down various advantages of packages.**

Packages in Java, are the collection of related classes and interfaces which are bundled together. By using packages, developers can easily modularize the code and optimize its reuse. Also, the code within the packages can be imported by other classes and reused. Below I have listed down a few of its advantages:

* Packages help in avoiding name clashes
* They provide easier access control on the code
* Packages can also contain hidden classes which are not visible to the outer classes and only used within the package
* Creates a proper hierarchical structure which makes it easier to locate the related classes

### **Q23. Why pointers are not used in Java?**

Java doesn’t use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation, thus in order to avoid direct access to memory by the user,  pointers are discouraged in Java.

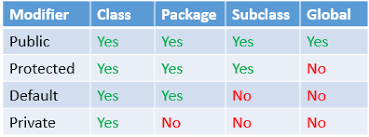
### **Q24. What is JIT compiler in Java?**

JIT stands for Just-In-Time compiler in Java. It is a program that helps in converting the Java bytecode into instructions that are sent directly to the processor. By default, the JIT compiler is enabled in Java and is activated whenever a Java method is invoked. The JIT compiler then compiles the bytecode of the invoked method into native machine code, compiling it “just in time” to execute. Once the method has been compiled, the JVM summons the compiled code of that method directly rather than interpreting it. This is why it is often responsible for the performance optimization of Java applications at the run time.

### **Q25. What are access modifiers in Java?**

In Java, access modifiers are special keywords which are used to restrict the access of a class, constructor, data member and method in another class. Java supports four types of access modifiers:

1. Default
2. Private
3. Protected
4. Public



### **Q26. Define a Java Class.**

A class in Java is a blueprint which includes all your data.  A class contains fields (variables) and methods to describe the behavior of an object. Let’s have a look at the syntax of a class.

|  |  |
| --- | --- |
| 1  2  3 | class Abc {  member variables // class body  methods} |

### **Q27. What is an object in Java and how is it created?**

An object is a real-world entity that has a state and behavior. An object has three characteristics:

1. State
2. Behavior
3. Identity

An object is created using the ‘new’ keyword. For example:

ClassName obj = new ClassName();

### **Q28. What is Object Oriented Programming?**

Object-oriented programming or popularly known as OOPs is a programming model or approach where the programs are organized around objects rather than logic and functions. In other words, OOP mainly focuses on the objects that are required to be manipulated instead of logic. This approach is ideal for the programs large and complex codes and needs to be actively updated or maintained.

### **Q29. What are the main concepts of OOPs in Java?**

Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

1. Inheritance: Inheritance is a process where one class acquires the properties of another.
2. Encapsulation: Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.
3. Abstraction: Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.
4. Polymorphism: Polymorphism is the ability of a variable, function or object to take multiple forms.

### **Q30. What is the difference between a local variable and an instance variable?**

In Java, a **local variable** is typically used inside a method, constructor, or a **block** and has only local scope. Thus, this variable can be used only within the scope of a block. The best benefit of having a local variable is that other methods in the class won’t be even aware of that variable.

#### **Example**

|  |  |
| --- | --- |
| 1  2  3  4 | if(x > 100)  {  String test = "Edureka";  } |

Whereas, an **instance variable** in Java, is a variable which is bounded to its object itself. These variables are declared within a **class**, but outside a method. Every object of that class will create it’s own copy of the variable while using it. Thus, any changes made to the variable won’t reflect in any other instances of that class and will be bound to that particular instance only.

|  |  |
| --- | --- |
| 1  2  3  4 | class Test{  public String EmpName;  public int empAge;  } |

### **Q31. Differentiate between the constructors and methods in Java?**

|  |  |
| --- | --- |
| **Methods** | **Constructors** |
| 1. Used to represent the behavior of an object | 1. Used to initialize the state of an object |
| 2. Must have a return type | 2. Do not have any return type |
| 3. Needs to be invoked explicitly | 3. Is invoked implicitly |
| 4. No default method is provided by the compiler | 4. A default constructor is provided by the compiler if the class has none |
| 5. Method name may or may not be same as class name | 5. Constructor name must always be the same as the class name |

In case you are facing any challenges with these Core Java interview questions, please comment on your problems in the section below.

**Q-32. What is the difference between a function and method?**

A function is a set of instructions or procedures to perform a specific task, and a method is a set of instructions that are associated with an object.

### **Q32. What is final keyword in Java?**

**final**is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

* **final variable**

When the final keyword is used with a variable then its value can’t be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.

#### **final method**

When a method is declared final then it can’t be overridden by the inheriting class.

#### **final class**

When a class is declared as final in Java, it can’t be extended by any subclass class but it can extend other class.

### **Q33. What is the difference between break and continue statements?**

|  |  |
| --- | --- |
| **break** | **continue** |
| 1. Can be used in switch and loop (for, while, do while) statements | 1. Can be only used with loop statements |
| 2. It causes the switch or loop statements to terminate the moment it is executed | 2. It doesn’t terminate the loop but causes the loop to jump to the next iteration |
| 3. It terminates the innermost enclosing loop or switch immediately | 3. A continue within a loop nested with a switch will cause the next loop iteration to execute |

***Example break:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for (int i = 0; i < 5; i++)  {  if (i == 3)  {  break;  }  System.out.println(i);  } |

***Example continue:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for (int i = 0; i < 5; i++)  {  if(i == 2)  {  continue;  }  System.out.println(i);  } |

### **Q34. What is an infinite loop in Java? Explain with an example.**

An infinite loop is an instruction sequence in Java that loops endlessly when a functional exit isn’t met. This type of loop can be the result of a programming error or may also be a deliberate action based on the application behavior. An infinite loop will terminate automatically once the application exits.

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class InfiniteForLoopDemo  {  public static void main(String[] arg) {  for(;;)  System.out.println("Welcome to Edureka!");  // To terminate this program press ctrl + c in the console.  }  } |

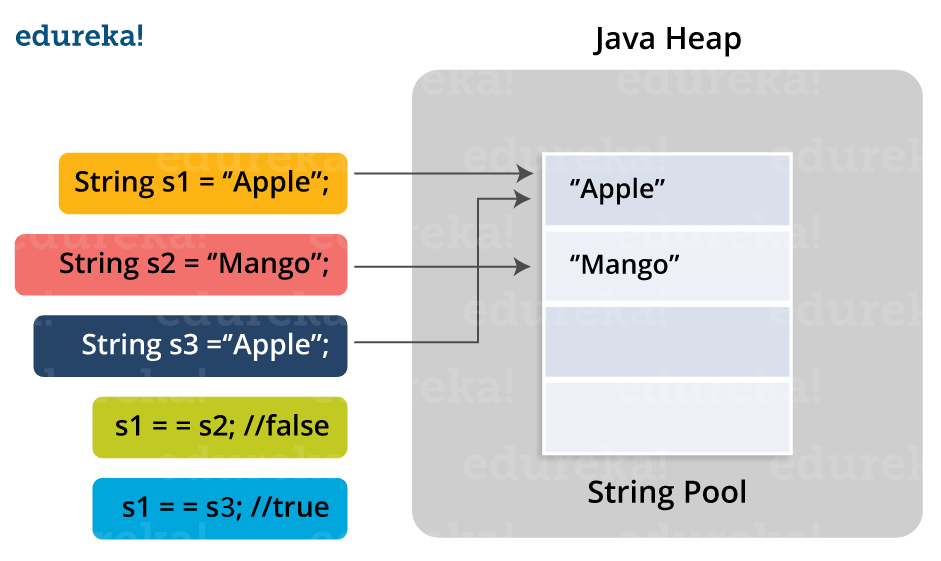
### **Q35. What is the difference between this() and super() in Java?**

In Java, super() and this(), both are special keywords that are used to call the constructor.

|  |  |
| --- | --- |
| **this()** | **super()** |
| 1. this() represents the current instance of a class | 1. super() represents the current instance of a parent/base class |
| 2. Used to call the default constructor of the same class | 2. Used to call the default constructor of the parent/base class |
| 3. Used to access methods of the current class | 3. Used to access methods of the base class |
| 4.  Used for pointing the current class instance | 4. Used for pointing the superclass instance |
| 5. Must be the first line of a block | 5. Must be the first line of a block |

### **Q36. What is Java String Pool?**

Java String pool refers to a collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.

****

### **Q37. Differentiate between static and non-static methods in Java.**

|  |  |
| --- | --- |
| **Static Method** | **Non-Static Method** |
| 1. *The static* keyword must be used before the method name | 1. No need to use the static keyword before the method name |
| 2. It is called using the class (className.methodName) | 2. It is can be called like any general method |
| 3. They can’t access any non-static instance variables or methods | 3. It can access any static method and any static variable without creating an instance of the class |

### **Q39. What is constructor chaining in Java?**

In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass’ constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways:

1. Within the same class using this()
2. From base class using super()

**Q40. Difference between String, StringBuilder, and StringBuffer.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **String** | **StringBuilder** | **StringBuffer** |
| Storage Area | Constant String Pool | Heap Area | Heap Area |
| Mutability | Immutable | Mutable | Mutable |
| Thread Safety | Yes | No | Yes |
| Performance | Fast | More efficient | Less efficient |

If you think this article on Java Interview Questions is helpful, you can check out Edureka’s [Java Training in Chennai](https://www.edureka.co/java-j2ee-training-course-chennai) as well.

### **Q41. What is a classloader in Java?**

The **Java ClassLoader** is a subset of JVM (Java Virtual Machine) that is responsible for loading the class files. Whenever a Java program is executed it is first loaded by the classloader. Java provides three built-in classloaders:

1. Bootstrap ClassLoader
2. Extension ClassLoader
3. System/Application ClassLoader

### **Q42. Why Java Strings are immutable in nature?**

In Java, string objects are immutable in nature which simply means once the String object is created its state cannot be modified. Whenever you try to update the value of that object instead of updating the values of that particular object, Java creates a new string object. Java String objects are immutable as String objects are generally cached in the String pool. Since String literals are usually shared between multiple clients, action from one client might affect the rest. It enhances security, caching, synchronization, and performance of the application.

### **Q43. What is the difference between an array and an array list?**

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Cannot contain values of different data types | Can contain values of different data types. |
| Size must be defined at the time of declaration | Size can be dynamically changed |
| Need to specify the index in order to add data | No need to specify the index |
| Arrays are not type parameterized | Arraylists are type |
| Arrays can contain primitive data types as well as objects | Arraylists can contain only objects, no primitive data types are allowed |

### **Q44. What is a Map in Java?**

In Java, Map is an interface of Util package which maps unique keys to values. The Map interface is not a subset of the main Collection interface and thus it behaves little different from the other collection types. Below are a few of the characteristics of Map interface:

1. Map doesn’t contain duplicate keys.
2. Each key can map at max one value.

### **Q45. What is collection class in Java? List down its methods and interfaces.**

In Java, the collection is a framework that acts as an architecture for storing and manipulating a group of objects. Using Collections you can perform various tasks like searching, sorting, insertion, manipulation, deletion, etc. Java collection framework includes the following:

* Interfaces
* Classes
* Methods

The below image shows the complete hierarchy of the Java Collection.



Want to upskill yourself to get ahead in Career? Check out this video

Core Java Interview Quesions And Answers for Freshers and Experienced

### **13. What is Local Variable and Instance Variable?**

**Local Variable:**

A local variable is a variable that we declare inside a Method. A method will often store its temporary state in local variables.

It can be accessible only inside a block, function, or constructor.



|  |  |
| --- | --- |
| 1  2  3  4  5 | public void website() {  String websiteName;  double websiteLoadTime;  int webisteAge;  } |

String websiteName, double websiteLoadTime, int websiteAge are Local variables in above example.

**Instance Variable (Non-static):**

An instance variable is a variable that is declared inside a Class but outside a Method. We don’t declare this variable as Static because these variables are non-static variables.

It can be accessible by all the methods in the class.



|  |  |
| --- | --- |
| 1  2  3  4  5 | class website() {  public String websiteName;  public double websiteLoadTime;  public int webisteAge;  } |

websiteName, websiteLoadTime, websiteAge are Instance variables in above example.

### **26. Why are strings immutable in Java?**

In Java, String is immutable to make sure that the string value doesn’t change. String literals are usually shared between multiple clients. If the value of the string changes (from “STM” to “stm”), it will affect all reference variables and cause severe discrepancies.  
Hence, strings are immutable in Java. Making string immutable enhances security, caching, synchronization, and performance of the application.

### **42. How to read a file line by line in Java?**

We can read a file line by line in Java in two ways.

1. BufferedReader Class  
2. Scanner Class

**Using BufferedReader Class:**

BufferedReader Class belongs to java.io package and it provides readLine() method to read a file line by line in Java.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package softwareTestingMaterial;    import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;    public class ReadLineByProgram {    public static void main(String[] args) {  BufferedReader reader;  try {  reader = new BufferedReader(new FileReader(  "/Users/Rajkumar/Downloads/STM.txt"));  String line = reader.readLine();  while (line != null) {  System.out.println(line);  // read next line  line = reader.readLine();  }  reader.close();  } catch (IOException e) {  e.printStackTrace();  }  }  } |

**Using Scanner Class:**

Java Scanner class provides the nextLine() method to facilitates line by line of file’s content.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | package softwareTestingMaterial;    import java.io.File;  import java.io.FileNotFoundException;  import java.util.Scanner;    public class ReadLineByProgram {    public static void main(String[] args) {  try {  Scanner scanner = new Scanner(new File("/Users/Rajkumar/Downloads/STM.txt"));  while (scanner.hasNextLine()) {  System.out.println(scanner.nextLine());  }  scanner.close();  } catch (FileNotFoundException e) {  e.printStackTrace();  }  }  } |

### **46. Difference between static binding and dynamic binding?**

1. Static binding is also known as early binding whereas dynamic binding is also known as late binding.  
2. Determining the type of an object at compile time is Static binding whereas determining the type of an object at run time is dynamic binding  
3. Java uses static binding for overloaded methods and dynamic binding for overridden methods.

To know more about this you have to go through [Method Overloading](https://www.softwaretestingmaterial.com/method-overloading-in-java/) and [Method Overriding](https://www.softwaretestingmaterial.com/method-overriding-in-java/).

# Autoboxing and Unboxing in Java

In Java, primitive data types are treated differently so do there comes the introduction of [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) where two components play a role namely Autoboxing and Unboxing. [Autoboxing](https://www.geeksforgeeks.org/autoboxing-unboxing-java/) refers to the conversion of a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

* Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
* Assigned to a variable of the corresponding **wrapper class**.

**Unboxing**on the other hand refers to converting an object of a wrapper type to its corresponding primitive value. For example conversion of Integer to int. The Java compiler applies to unbox when an object of a wrapper class is:

* Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
* Assigned to a variable of the corresponding **primitive type**.

<https://www.geeksforgeeks.org/autoboxing-unboxing-java/>

# Java Reflection API

**Java Reflection** is a *process of examining or modifying the run time behavior of a class at run time*.

There are 3 ways to get the instance of Class class. They are as follows:

* forName() method of Class class
* getClass() method of Object class
* the .class syntax

https://www.geeksforgeeks.org/reflection-in-java/

# Java Lambda Expressions

It provides a clear and concise way to represent one method interface using an expression.

The Lambda expression is used to provide the implementation of an interface which has functional interface

## **Functional Interface**

Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface. Java provides an anotation @FunctionalInterface, which is used to declare an interface as functional interface.

## **Why use Lambda Expression**

1. To provide the implementation of Functional interface.

### **49. What are the differences between throw and throws in Java?**

**throw keyword**

* The throw keyword is used to explicitly throw an exception in the program inside a function or inside a block of code.
* The checked exceptions cannot be propagated with throw only.
* The throw keyword is followed by an instance.
* The throw keyword is used within the method.
* You cannot throw multiple exceptions.

**throws keyword**

* The throws keyword is used in the method signature to declare an exception which might get thrown by the function while executing the code.
* The checked exception can be propagated with throws
* The throws keyword is followed by class.
* The throws keyword is used with the method signature.
* You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

Error Handing in Java

 Error: refers to an illegal operation performed by the user

It is of three types:

* Compile-time
* Run-time
* Logical

Exceptions: Exceptions in java refer to an unwanted or unexpected event, which occurs during the execution of a program.

* Checked exceptions
* Unchecked exceptions

### **Differences between Checked Exception and Unchecked Exception:**

**Checked Exception**

1. Checked exceptions occur at compile time.
2. Here, the JVM needs the exception to catch and handle.

* File Not Found Exception
* No Such Field Exception
* Interrupted Exception
* No Such Method Exception
* Class Not Found Exception

**Un-Checked Exception**

1. Un-Checked exceptions occur at run time.
2. Here, the JVM does not require the exception to catch and handle

* No Such Element Exception
* Undeclared Throwable Exception
* Empty Stack Exception
* Arithmetic Exception
* Null Pointer Exception
* Array Index Out of Bounds Exception
* Security Exception

AOP(Aspect oriented programming)

# AOP (Aspect Oriented Programming)

## **AOP:** AOP **(Aspect-Oriented Programming)** is a programming pattern that increases modularity by allowing the separation of the **cross-cutting concern**. These cross-cutting concerns are different from the main business logic. We can add additional behavior to existing code without modification of the code itself.

Spring's AOP framework helps us to implement these cross-cutting concerns.

Using AOP, we define common functionality in one place. We are free to define how and where this functionality is applied without modifying the class to which we are applying the new feature. The cross-cutting concern can now be modularized into special classes, called **aspect**.

There are **two** benefits of aspects:

* First, the logic for each concern is now in one place instead of scattered all over the codebase.
* Second, the business modules only contain code for their primary concern. The secondary concern has been moved to the **aspect**.

The aspects have the responsibility that is to be implemented, called **advice**. We can implement an aspect's functionality into a program at one or more join points.

## **Benefits of AOP**

* It is implemented in pure Java.
* There is no requirement for a special compilation process.
* It supports only method execution Join points.
* Only run time weaving is available.
* Two types of AOP proxy is available: **JDK dynamic proxy** and **CGLIB proxy.**

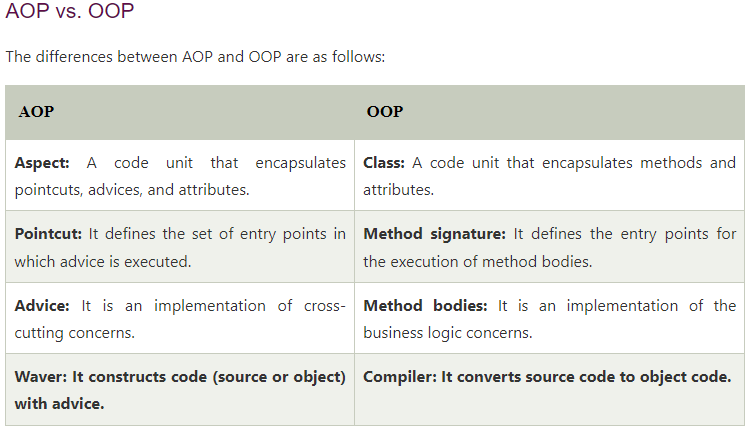
## **Cross-cutting concern**

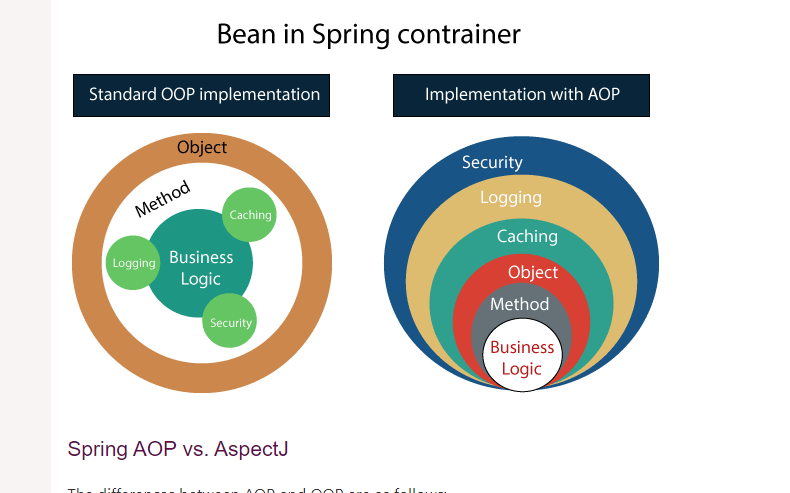
The cross-cutting concern is a concern that we want to implement in multiple places in an application. It affects the entire application. Ex **Logging, Security, validation, caching,** etc

## **AOP Terminology**

* **Aspect:** An aspect is a module that encapsulates **advice** and **pointcuts** and provides **cross-cutting** An application can have any number of aspects. We can implement an aspect using regular class annotated with **@Aspect** annotation.
* **Pointcut:** A pointcut is an expression that selects one or more join points where advice is executed. We can define pointcuts using **expressions** or **patterns**. It uses different kinds of expressions that matched with the join points. In Spring Framework, **AspectJ** pointcut expression language is used.
* **Join point:** A join point is a point in the application where we apply an **AOP aspect**. Or it is a specific execution instance of an advice. In AOP, join point can be a **method execution, exception handling, changing object variable value**, etc.
* **Advice:** The advice is an action that we take either **before** or **after** the method execution. The action is a piece of code that invokes during the program execution. There are **five** types of advices in the Spring AOP framework: **before, after, after-returning, after-throwing,**and **around advice.**Advices are taken for a particular **join point.**We will discuss these advices further in this section.
* **Target object:** An object on which advices are applied, is called the **target object**. Target objects are always a **proxied** It means a subclass is created at run time in which the target method is overridden, and advices are included based on their configuration.
* **Weaving:** It is a process of **linking aspects** with other application types. We can perform weaving at **run time, load time,** and **compile time**.

**Proxy:** It is an object that is created after applying advice to a target object is called **proxy**. The Spring AOP implements the **JDK dynamic proxy** to create the proxy classes with target classes and advice invocations. These are called AOP proxy classes





Link: <https://www.geeksforgeeks.org/what-is-is-a-relationship-in-java/#:~:text=In%20Java%2C%20we%20have%20two,is%20called%20HAS%2DA%20relationship.>

# What is Is-A-Relationship in Java?

**A relationship** in Java means different relations between two or more classes. For example, if a class Bulb inherits another class Device, then we can say that Bulb is having is-a relationship with Device, which implies Bulb is a device.

In Java, we have two types of relationship:

1. **Is-A relationship:** Whenever one class inherits another class, it is called an IS-A relationship.
2. **Has-A relationship:** Whenever an instance of one class is used in another class, it is called HAS-A relationship.

### Is-A relationship

**IS-A Relationship** is wholly related to [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/). For example – a kiwi is a fruit; a bulb is a device.

* IS-A relationship can simply be achieved by using [extends](https://www.geeksforgeeks.org/extends-vs-implements-in-java/) Keyword.
* IS-A relationship is additionally used for code reusability in Java and to avoid code redundancy.
* IS-A relationship is unidirectional, which means we can say that a bulb is a device, but vice versa; a device is a bulb is not possible since all the devices are not bulbs.
* IS-A relationship is tightly coupled, which means changing one entity will affect another entity.

### Advantage of IS-A relationship

* Code Reusability.
* Reduce redundancy

# What is Has-A-Relation in Java?

## Link: [https://www.geeksforgeeks.org/](https://www.geeksforgeeks.org/what-is-has-a-relation-in-java/)