**Q #1) What is JAVA?**

**Answer:** Java is a high-level programming language and is platform-independent.

Java is a collection of objects. It was developed by Sun Microsystems. There are a lot of applications, websites, and games that are developed using Java.

**Q #2) What are the features of JAVA?**

**Answer: Features of Java are as follows:**

* **OOP concepts**
  + Object-oriented
  + Inheritance
  + Encapsulation
  + Polymorphism
  + Abstraction
* **Platform independent:** A single program works on different platforms without any modification.
* **High Performance:** JIT (Just In Time compiler) enables high performance in Java. JIT converts the bytecode into machine language and then JVM starts the execution.
* **Multi-threaded:** A flow of execution is known as a Thread. JVM creates a thread which is called the main thread. The user can create multiple threads by extending the thread class or by implementing the Runnable interface.

**Q #3) How does Java enable high performance?**

**Answer:** Java uses Just In Time compiler to enable high performance. It is used to convert the instructions into bytecodes.

**Q #4) Name the Java IDE’s?**

**Answer:** Eclipse and NetBeans are the IDE’s of JAVA.

**Q #5) What do you mean by Constructor?**

**Answer: Constructor can be explained in detail with enlisted points:**

* When a new object is created in a program a constructor gets invoked corresponding to the class.
* The constructor is a method which has the same name as the class name.
* If a user doesn’t create a constructor implicitly a default constructor will be created.
* The constructor can be overloaded.
* If the user created a constructor with a parameter then he should create another constructor explicitly without a parameter.
* **Q #6) What is meant by the Local variable and the Instance variable?**
* **Answer:**
* **Local variables** are defined in the method and scope of the variables that exist inside the method itself.
* **Instance variable** is defined inside the class and outside the method and the scope of the variables exists throughout the class.
* **Q #7) What is a Class?**
* **Answer:** All Java codes are defined in a Class. It has variables and methods.
* **Variables**are attributes which define the state of a class.
* **Methods** are the place where the exact business logic has to be done. It contains a set of statements (or) instructions to satisfy the particular requirement.
* **Example:**

|  |
| --- |
| **public** **class** Addition{ //Class name declaration  **int** a = 5; //Variable declaration  **int** b= 5;  **public** **void** add(){ //Method declaration  **int** c = a+b;  }  } |

**Q #8) What is an Object?**

**Answer:** An instance of a class is called an object. The object has state and behavior.

Whenever the JVM reads the “new()” keyword then it will create an instance of that class.

**Example:**

|  |
| --- |
| **public** **class** Addition{  **public** **static** **void** main(String[] args){  Addion add = **new** Addition();//Object creation  }  } |

The above code creates the object for the Addition class.

**Q #9)What are the OOPs concepts?**

**Answer: OOPs concepts include:**

* Inheritance
* Encapsulation
* Polymorphism
* Abstraction
* Interface

***Suggested Read =>>***[***Top OOPs Interview Questions***](https://www.softwaretestinghelp.com/oops-interview-questions-and-answers/)

**Q #10) What is Inheritance?**

**Answer:** Inheritance means one class can extend to another class. So that the codes can be reused from one class to another class. The existing class is known as the Super class whereas the derived class is known as a sub class.

**Example:**

|  |
| --- |
| Super **class**:  **public** **class** Manupulation(){  }  Sub **class**:  **public** **class** Addition **extends** Manipulation(){  } |

Inheritance is only applicable to the public and protected members only. Private members can’t be inherited.

**Q #11) What is Encapsulation?**

**Answer: Purpose of Encapsulation:**

* Protects the code from others.
* Code maintainability.

**Example:**

We are declaring ‘a’ as an integer variable and it should not be negative.

|  |
| --- |
| **public** **class** Addition(){  **int** a=5;  } |

If someone changes the exact variable as “***a = -5”***then it is bad.

**In order to overcome the problem we need to follow the steps below:**

* We can make the variable private or protected.
* Use public accessor methods such as set<property> and get<property>.

**So that the above code can be modified as:**

|  |
| --- |
| **public** **class** Addition(){  **private** **int** a = 5; //Here the variable is marked as private  } |

**The code below shows the getter and setter.**

Conditions can be provided while setting the variable.

|  |
| --- |
| get A(){  }  set A(**int** a){  **if**(a&gt;0){// Here condition is applied  .........  }  } |

For encapsulation, we need to make all the instance variables private and create setter and getter for those variables. Which in turn will force others to call the setters rather than access the data directly.

**Q #12) What is Polymorphism?**

**Answer:** Polymorphism means many forms.

A single object can refer to the super-class or sub-class depending on the reference type which is called polymorphism.

**Example:**

|  |
| --- |
| Public **class** Manipulation(){ //Super class  **public** **void** add(){  }  }  **public** **class** Addition **extends** Manipulation(){ // Sub class  **public** **void** add(){  }  **public** **static** **void** main(String args[]){  Manipulation addition = **new** Addition();//Manipulation is reference type and Addition is reference type  addition.add();  }  } |

Using the Manipulation reference type we can call the Addition class “add()” method. This ability is known as Polymorphism. Polymorphism is applicable for **overriding**and not for **overloading**.

**Q #13) What is meant by Method Overriding?**

**Answer: Method overriding happens if the sub-class method satisfies the below conditions with the Super-class method:**

* Method name should be the same
* The argument should be the same
* Return type should also be the same

The key benefit of overriding is that the Sub-class can provide some specific information about that sub-class type than the super-class.

**Example:**

|  |
| --- |
| **public** **class** Manipulation{ //Super class  **public** **void** add(){  ………………  }  }    Public **class** Addition **extends** Manipulation(){  Public **void** add(){  ………..  }  Public **static** **void** main(String args[]){  Manipulation addition = **new** Addition(); //Polimorphism is applied  addition.add(); // It calls the Sub class add() method  }  } |

**addition.add()**method calls the add() method in the Sub-class and not the parent class. So it overrides the Super-class method and is known as Method Overriding.

**Q #14) What is meant by Overloading?**

**Answer:** Method overloading happens for different classes or within the same class.

**For method overloading, sub-class method should satisfy the below conditions with the Super-class method (or) methods in the same class itself:**

* Same method name
* Different argument types
* There may be different return types

**Example:**

|  |
| --- |
| **public** **class** Manipulation{ //Super class  **public** **void** add(String name){ //String parameter  ………………  }  }    Public **class** Addition **extends** Manipulation(){  Public **void** add(){//No Parameter  ………..  }  Public **void** add(**int** a){ //integer parameter    }  Public **static** **void** main(String args[]){  Addition addition = **new** Addition();  addition.add();  }  } |

Here the add() method has different parameters in the Addition class is overloaded in the same class as with the super-class.

**Note:** Polymorphism is not applicable for method overloading.

**Q #15) What is meant by Interface?**

**Answer:** Multiple inheritances cannot be achieved in java. To overcome this problem the Interface concept is introduced.

An interface is a template which has only method declarations and not the method implementation.

**Example:**

|  |
| --- |
| Public **abstract** **interface** IManupulation{ //Interface declaration  Public **abstract** **void** add();//method declaration  **public** **abstract** **void** subtract();  } |

* All the methods in the interface are internally **public abstract void**.
* All the variables in the interface are internally **public static final** that is constants.
* Classes can implement the interface and not extends.
* The class which implements the interface should provide an implementation for all the methods declared in the interface.

|  |
| --- |
| **public** **class** Manupulation **implements** IManupulation{ //Manupulation class uses the interface  Public **void** add(){  ……………  }  Public **void** subtract(){  …………….  }  } |

**Q #16) What is meant by Abstract class?**

**Answer:** We can create the Abstract class by using the “Abstract” keyword before the class name. An abstract class can have both “Abstract” methods and “Non-abstract” methods that are a concrete class.

**Abstract method:**

The method which has only the declaration and not the implementation is called the abstract method and it has the keyword called “abstract”. Declarations ends with a semicolon.

**Example:**

|  |
| --- |
| **public** **abstract** **class** Manupulation{  **public** **abstract** **void** add();//Abstract method declaration  Public **void** subtract(){  }  } |

* An abstract class may have a non- abstract method also.
* The concrete Subclass which extends the Abstract class should provide the implementation for abstract methods.

**Q #17) Difference between Array and Array List.**

**Answer:** **The Difference between Array and Array List can be understood from the table below:**

| **Array** | **Array List** |
| --- | --- |
| Size should be given at the time of array declaration.  String[] name = new String[2] | Size may not be required. It changes the size dynamically.  ArrayList name = new ArrayList |
| To put an object into array we need to specify the index.  name[1] = “book” | No index required.  name.add(“book”) |
| Array is not type parameterized | ArrayList in java 5.0 are parameterized.  Eg: This angle bracket is a type parameter which means a list of String. |

**#18) Difference between String, String Builder, and String Buffer.**

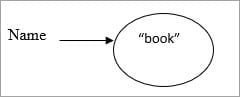
**Answer:**

**String:** String variables are stored in a “constant string pool”. Once the string reference changes the old value that exists in the “constant string pool”, it cannot be erased.

**Example:**

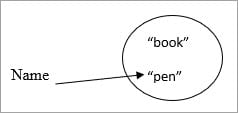
String name = “book”;

**Constant string pool**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Constant-string-pool.jpg).

If the name-value has changed from “book” to “pen”.

**Constant string pool**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Constant-string-pools.jpg)

Then the older value remains in the constant string pool.

**String Buffer:**

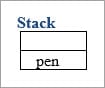
* Here string values are stored in a stack. If the values are changed then the new value replaces the older value.
* The string buffer is synchronized which is thread-safe.
* Performance is slower than the String Builder.

**Example:**

String Buffer name =”book”;

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Stack.jpg)

Once the name value has been changed to “pen” then the “book” is erased in the stack.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Stack1.jpg)

**String Builder:**

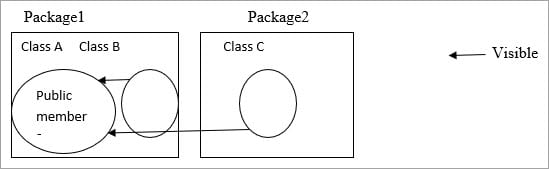
This is the same as String Buffer except for the String Builder which is not threaded safely that is not synchronized. So obviously the performance is fast.

**Q #19) Explain about Public and Private access specifiers.**

**Answer:** Methods and instance variables are known as members.

**Public:**

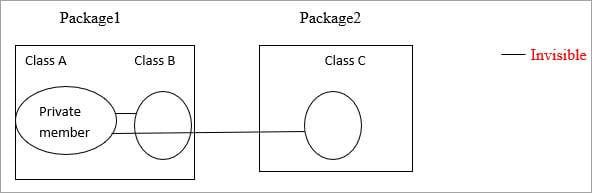
Public members are visible in the same package as well as the outside package that is for other packages.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Public.jpg)

Public members of Class A are visible to Class B (same package) as well as Class C (different packages).

**Private:**

Private members are visible in the same class only and not for the other classes in the same package as well as classes in the outside packages.

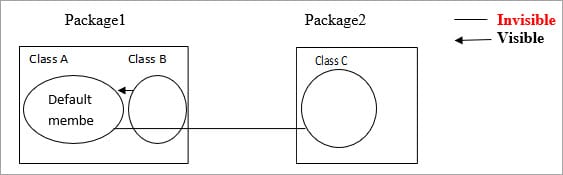
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Private.jpg)

Private members in class A are visible only in that class. It is invisible for class  B as well as class C.

**Q #20) Difference between Default and Protected access specifiers.**

**Answer:**

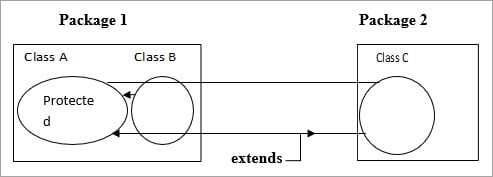
**Default:**Methods and variables declared in a class without any access specifiers are called default.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Default.jpg)

Default members in Class A are visible to the other classes which are inside the package and invisible to the classes which are outside the package.

So Class A members are visible to Class B and invisible to Class C.

**Protected:**

**[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Protected.jpg)             .**

Protected is the same as Default but if a class extends then it is visible even if it is outside the package.

Class A members are visible to Class B because it is inside the package. For Class C it is invisible but if Class C extends Class A then the members are visible to Class C even if it is outside the package.

**Q #21) Difference between HashMap and HashTable.**

**Answer:** **The difference between HashMap and HashTable can be seen below:**

| **HashMap** | **HashTable** |
| --- | --- |
| Methods are not synchronized | Key methods are synchronized |
| Not thread safety | Thread safety |
| Iterator is used to iterate the values | Enumerator is used to iterate the values |
| Allows one null key and multiple null values | Doesn’t allow anything that is null |
| Performance is high than HashTable | Performance is slow |

**Q #22) Difference between HashSet and TreeSet.**

**Answer:** **The difference between HashSet and TreeSet can be seen below:**

| **HashSet** | **TreeSet** |
| --- | --- |
| Inserted elements are in random order | Maintains the elements in the sorted order |
| Can able to store null objects | Couldn’t store null objects |
| Performance is fast | Performance is slow |

**Q #23) Difference between Abstract class and Interface.**

**Answer: The differences between Abstract Class and Interface are as follows:**

**Abstract Class:**

* Abstract classes have a default constructor and it is called whenever the concrete subclass is instantiated.
* It contains Abstract methods as well as Non-Abstract methods.
* The class which extends the Abstract class shouldn’t require the implementation of all the methods, only Abstract methods need to be implemented in the concrete sub-class.
* Abstract class contains instance variables.

**Interface:**

* It doesn’t have any constructor and couldn’t be instantiated.
* The abstract method alone should be declared.
* Classes that implement the interface should provide the implementation for all the methods.
* The interface contains only constants.

**Q** **#24)  What is the meaning of Collections in Java?**

**Answer:** Collection is a framework that is designed to store the objects and manipulate the design to store the objects.

**Collections are used to perform the following operations:**

* Searching
* Sorting
* Manipulation
* Insertion
* Deletion

A group of objects is known as collections. All the classes and interfaces for collecting are available in Java util package.

**Further Reading =>**[**Most Popular Java Development Companies To Look For**](https://www.softwaretestinghelp.com/java-development-companies/)

**Q #25) What are all the Classes and Interfaces that are available in the collections?**

**Answer:** **Given below are the Classes and Interfaces that are available in Collections:**

**Interfaces:**

* Collection
* List
* Set
* Map
* Sorted Set
* Sorted Map
* Queue

**Classes:**

* Lists:
* Array List
* Vector
* Linked List

**Sets:**

* Hash set
* Linked Hash Set
* Tree Set

**Maps:**

* Hash Map
* Hash Table
* TreeMap
* Linked Hashed Map

**Queue:**

* Priority Queue

**Q #26) What is meant by Ordered and Sorted in collections?**

**Answer:**

**Ordered:**It means the values that are stored in a collection is based on the values that are added to the collection. So we can iterate the values from the collection in a specific order.

**Sorted:**Sorting mechanisms can be applied internally or externally so that the group of objects sorted in a particular collection is based on the properties of the objects.

**Q #27) Explain the different lists available in the collection.**

**Answer:**Values added to the list are based on the index position and it is ordered by index position. Duplicates are allowed.

**The types of Lists are:**

**a) Array List:**

* Fast iteration and fast Random Access.
* It is an ordered collection (by index) and not sorted.
* It implements the Random Access Interface.

**Example:**

|  |
| --- |
| **public** **class** Fruits{  **public** **static** **void** main (String [ ] args){  ArrayList &lt;String&gt;names=**new** ArrayList &lt;String&gt;();  names.add (“apple”);  names.add (“cherry”);  names.add (“kiwi”);  names.add (“banana”);  names.add (“cherry”);  System.out.println (names);  }  } |

**Output:**

[Apple, cherry, kiwi, banana, cherry]

From the output, Array List maintains the insertion order and it accepts the duplicates. But it’s not sorted.

**b) Vector:**

It is the same as Array List.

* Vector methods are synchronized.
* Thread safety.
* It also implements Random Access.
* Thread safety usually causes a performance hit.

**Example:**

|  |
| --- |
| **public** **class** Fruit {  **public** **static** **void** main (String [ ] args){  Vector &lt;String&gt; names = **new** Vector &lt;String&gt; ( );   names.add (“cherry”);  names.add (“apple”);  names.add (“banana”);  names.add (“kiwi”);  names.add (“apple”);  System.out.println (“names”);  }  } |

**Output:**

[cherry,apple,banana,kiwi,apple]

Vector also maintains the insertion order and accepts the duplicates.

**c) Linked List:**

* Elements are doubly linked to one another.
* Performance is slower than the Array list.
* Good choice for insertion and deletion.
* In Java 5.0 it supports common queue methods peek( ), Pool ( ), Offer ( ) etc.

**Example:**

|  |
| --- |
| **public** **class** Fruit {  **public** **static** **void** main (String [ ] args){  Linkedlist &lt;String&gt; names = **new** linkedlist &lt;String&gt; ( ) ;  names.add(“banana”);  names.add(“cherry”);  names.add(“apple”);  names.add(“kiwi”);  names.add(“banana”);  System.out.println (names);  }  } |

**Output:**

[ banana,cherry,apple,kiwi,banana]

Maintains the insertion order and accepts the duplicates.

**Q #28) Explain about Set and their types in a collection.**

**Answer:** Set cares about uniqueness. It doesn’t allow duplications. Here “equals ( )” method is used to determine whether two objects are identical or not.

**a) Hash Set:**

* Unordered and unsorted.
* Uses the hash code of the object to insert the values.
* Use this when the requirement is “no duplicates and don’t care about the order”.

**Example:**

|  |
| --- |
| **public** **class** Fruit {  **public** **static** **void** main (String[ ] args){  HashSet&lt;String&gt; names = **new** HashSet &lt;=String&gt;( ) ;  names.add(“banana”);  names.add(“cherry”);  names.add(“apple”);  names.add(“kiwi”);  names.add(“banana”);  System.out.println (names);  }  } |

**Output:**

[banana, cherry, kiwi, apple]

It doesn’t follow any insertion order. Duplicates are not allowed.

**b) Linked Hash set:**

* An ordered version of the hash set is known as Linked Hash Set.
* Maintains a doubly-Linked list of all the elements.
* Use this when an iteration order is required.

**Example:**

|  |
| --- |
| **public** **class** Fruit {  **public** **static** **void** main (String[ ] args){  LinkedHashSet&lt;String&gt;; names = **new** LinkedHashSet &lt;String&gt;( ) ;   names.add(“banana”);   names.add(“cherry”);   names.add(“apple”);   names.add(“kiwi”);   names.add(“banana”);   System.out.println (names);   }  } |

**Output:**

[banana, cherry, apple, kiwi]

It maintains the insertion order in which they have been added to the Set. Duplicates are not allowed.

**c) Tree Set:**

* It is one of the two sorted collections.
* Uses the “Read-Black” tree structure and guarantees that the elements will be in ascending order.
* We can construct a tree set with the constructor by using a comparable (or) comparator.

**Example:**

|  |
| --- |
| **public** **class** Fruits{  **public** **static** **void** main (String[ ]args) {  Treeset&lt;String&gt; names= **new** TreeSet&lt;String&gt;( ) ;  names.add(“cherry”);  names.add(“banana”);  names.add(“apple”);  names.add(“kiwi”);  names.add(“cherry”);  System.out.println(names);  }  } |

**Output:**

[apple, banana, cherry, kiwi]

TreeSet sorts the elements in ascending order. And duplicates are not allowed.

**Q #29) Explain about Map and its types.**

**Answer: Map** cares about the unique identifier. We can map a unique key to a specific value. It is a key/value pair. We can search a value, based on the key. Like the set, the map also uses the “equals ( )” method to determine whether two keys are the same or different.

**Map is of following types:**

**a) Hash Map:**

* Unordered and unsorted map.
* Hashmap is a good choice when we don’t care about the order.
* It allows one null key and multiple null values.

**Example:**

|  |
| --- |
| Public **class** Fruit{  Public **static** **void** main(String[ ] args){  HashMap&lt;Sting,String&gt; names =**new** HashMap&lt;String,String&gt;( );  names.put(“key1”,“cherry”);  names.put (“key2”,“banana”);  names.put (“key3”,“apple”);  names.put (“key4”,“kiwi”);  names.put (“key1”,“cherry”);  System.out.println(names);  }   } |

**Output:**

{key2 =banana, key1=cherry, key4 =kiwi, key3= apple}

Duplicate keys are not allowed in Map.

It doesn’t maintain any insertion order and is unsorted.

**b) Hash Table:**

* Like the vector key, methods of the class are synchronized.
* Thread safety and therefore slows the performance.
* It doesn’t allow anything that is null.

**Example:**

|  |
| --- |
| **public** **class** Fruit{  **public** **static** **void** main(String[ ]args){  Hashtable&lt;Sting,String&gt; names =**new** Hashtable&lt;String,String&gt;( );  names.put(“key1”,“cherry”);  names.put(“key2”,“apple”);  names.put(“key3”,“banana”);  names.put(“key4”,“kiwi”);  names.put(“key2”,“orange”);  System.out.println(names);  }   } |

**Output:**

{key2=apple, key1=cherry,key4=kiwi, key3=banana}

Duplicate keys are not allowed.

**c) Linked Hash Map:**

* Maintains insertion order.
* Slower than Hash map.
* I can expect a faster iteration.

**Example:**

|  |
| --- |
| **public** **class** Fruit{  **public** **static** **void** main(String[ ] args){  LinkedHashMap&lt;Sting,String&gt; names =**new** LinkedHashMap&lt;String,String&gt;( );   names.put(“key1”,“cherry”);   names.put(“key2”,“apple”);   names.put(“key3”,“banana”);   names.put(“key4”,“kiwi”);   names.put(“key2”,“orange”);   System.out.println(names);   }   } |

**Output:**

{key2=apple, key1=cherry,key4=kiwi, key3=banana}

Duplicate keys are not allowed.

**d) TreeMap:**

* Sorted Map.
* Like Tree set, we can construct a sort order with the constructor.

**Example:**

|  |
| --- |
| **public** **class** Fruit{  **public** **static** **void** main(String[ ]args){  TreeMap&lt;Sting,String&gt; names =**new** TreeMap&lt;String,String&gt;( );  names.put(“key1”,“cherry”);  names.put(“key2”,“banana”);  names.put(“key3”,“apple”);  names.put(“key4”,“kiwi”);  names.put(“key2”,“orange”);  System.out.println(names);  }  } |

**Output:**

{key1=cherry, key2=banana, key3 =apple, key4=kiwi}

It is sorted in ascending order based on the key. Duplicate keys are not allowed.

**Q #30) Explain the Priority Queue.**

**Answer: Queue Interface**

**Priority Queue:**Linked list class has been enhanced to implement the queue interface. Queues can be handled with a linked list. The purpose of a queue is “Priority-in, Priority-out”.

Hence elements are ordered either naturally or according to the comparator. The elements ordering represents their relative priority.

**Q #31) What is meant by Exception?**

**Answer:** An Exception is a problem that can occur during the normal flow of execution. A method can throw an exception when something wails at runtime. If that exception couldn’t be handled, then the execution gets terminated before it completes the task.

If we handled the exception, then the normal flow gets continued. Exceptions are a subclass of java.lang.Exception.

**Example for handling Exception:**

|  |
| --- |
| **try**{  //Risky codes are surrounded by this block  }**catch**(Exception e){  //Exceptions are caught in catch block  } |

**Q #32) What are the types of Exceptions?**

**Answer:** There are two types of Exceptions. They are explained below in detail.

**a) Checked Exception:**

These exceptions are checked by the compiler at the time of compilation. Classes that extend Throwable class except Runtime exception and Error are called checked Exception.

Checked Exceptions must either declare the exception using throws keyword (or) surrounded by appropriate try/catch.

**For Example,** ClassNotFound Exception

**b) Unchecked Exception:**

These exceptions are not checked during the compile time by the compiler.  The compiler doesn’t force to handle these exceptions. **It includes:**

* Arithmetic Exception
* ArrayIndexOutOfBounds Exception

**Q #33) What are the different ways to handle exceptions?**

**Answer:** **Two different ways to handle exceptions are explained below:**

**a) Using try/catch:**

The risky code is surrounded by try block. If an exception occurs, then it is caught by the catch block which is followed by the try block.

**Example:**

|  |
| --- |
| **class** Manipulation{  **public** **static** **void** main(String[] args){  add();  }  Public **void** add(){  **try**{  addition();  }**catch**(Exception e){  e.printStacktrace();  }  }  } |

**b) By declaring throws keyword:**

At the end of the method, we can declare the exception using throws keyword.

**Example:**

|  |
| --- |
| **class** Manipulation{  **public** **static** **void** main(String[] args){  add();  }  **public** **void** add() **throws** Exception{  addition();  }  } |

**Q #34) What are the advantages of Exception handling?**

**Answer: The advantages are as follows:**

* The normal flow of the execution won’t be terminated if an exception gets handled
* We can identify the problem by using catch declaration

**Q #35) What are the Exception handling keywords in Java?**

**Answer: Enlisted below are the two Exception Handling Keywords:**

**a) try:**

When a risky code is surrounded by a try block. An exception occurring in the try block is caught by a catch block. Try can be followed either by catch (or) finally (or) both. But any one of the blocks is mandatory.

**b) catch:**

This is followed by a try block. Exceptions are caught here.

**c) finally:**

This is followed either by try block (or) catch block. This block gets executed regardless of an exception. So generally clean up codes are provided here.

**Q #36) Explain about Exception Propagation.**

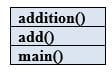
**Answer:** Exception is first thrown from the method which is at the top of the stack. If it doesn’t catch, then it pops up the method and moves to the previous method and so on until they are got.

This is called Exception propagation.

**Example:**

|  |
| --- |
| **public** **class** Manipulation{  **public** **static** **void** main(String[] args){  add();  }  **public** **void** add(){  addition();  } |

**From the above example, the stack looks like as shown below:**

**[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Stack-Example.jpg)**

If an exception occurs in the **addition()** method is not caught, then it moves to the method **add()**. Then it is moved to the **main()** method and then it will stop the flow of execution. It is called Exception Propagation.

**Q #37) What is the final keyword in Java?**

**Answer:**

**Final variable:**Once a variable is declared as final, then the value of the variable could not be changed. It is like a constant.

**Example:**

final int = 12;

**Final method:**A final keyword in a method, couldn’t be overridden. If a method is marked as a final, then it can’t be overridden by the subclass.

**Final class:**If a class is declared as final, then the class couldn’t be subclassed. No class can extend the final class.

**Q #38) What is a Thread?**

**Answer:**In Java, the flow of execution is called Thread. Every java program has at least one thread called the main thread, the main thread is created by JVM. The user can define their own threads by extending the Thread class (or) by implementing the Runnable interface. Threads are executed concurrently.

**Example:**

|  |
| --- |
| **public** **static** **void** main(String[] args){//main thread starts here  } |

**Q #39) How do you make a thread in Java?**

**Answer:**There are two ways available to make a thread.

**a) Extend Thread class:**Extending a Thread class and override the run method. The thread is available in java.lang.thread.

**Example:**

|  |
| --- |
| Public **class** Addition **extends** Thread {  **public** **void** run () {  }  } |

The disadvantage of using a thread class is that we cannot extend any other classes because we have already extended the thread class. We can overload the run () method in our class.

**b) Implement Runnable interface:**Another way is by implementing the runnable interface. For that, we should provide the implementation for the run () method which is defined in the interface.

**Example:**

|  |
| --- |
| Public **class** Addition **implements** Runnable {  **public** **void** run () {  }  } |

**Q #40) Explain about join () method.**

**Answer:** Join () method is used to join one thread with the end of the currently running thread.

**Example:**

|  |
| --- |
| **public** **static** **void** main (String[] args){  Thread t = **new** Thread ();  t.start ();  t.join ();  } |

Based on the above code, the main thread has started the execution. When it reaches the code ***t.start()*** then ‘thread t’ starts the own stack for the execution. JVM switches between the main thread and ‘thread t’.

Once it reaches the code ***t.join()*** then ‘thread t’ alone is executed and completes its task, then only the main thread starts the execution.

It is a non-static method. The Join () method has an overloaded version. So we can mention the time duration in join () method also “.s”.

**Q #41) What does the yield method of the Thread class do?**

**Answer:** A yield () method moves the currently running thread to a runnable state and allows the other threads for execution. So that equal priority threads have a chance to run. It is a static method. It doesn’t release any lock.

Yield () method moves the thread back to the Runnable state only, and not the thread to sleep (), wait () (or) block.

**Example:**

|  |
| --- |
| **public** **static** **void** main (String[] args){  Thread t = **new** Thread ();  t.start ();  }  **public** **void** run(){  Thread.yield();  }  } |

**Q #42) Explain about wait () method.**

**Answer: wait ()** method is used to make the thread to wait in the waiting pool. When the wait () method is executed during a thread execution then immediately the thread gives up the lock on the object and goes to the waiting pool. Wait () method tells the thread to wait for a given amount of time.

Then the thread will wake up after notify () (or) notify all () method is called.

Wait() and the other above-mentioned methods do not give the lock on the object immediately until the currently executing thread completes the synchronized code. It is mostly used in synchronization.

**Example:**

|  |
| --- |
| **public** **static** **void** main (String[] args){  Thread t = **new** Thread ();  t.start ();  Synchronized (t) {  Wait();  }  } |

**Q #43) Difference between notify() method and notifyAll() method in Java.**

**Answer: The differences between notify() method and notifyAll() method are enlisted below:**

| **notify()** | **notifyAll()** |
| --- | --- |
| This method is used to send a signal to wake up a single thread in the waiting pool. | This method sends the signal to wake up all the threads in a waiting spool. |

**Q #44) How to stop a thread in java? Explain about sleep () method in a thread?**

**Answer:** **We can stop a thread by using the following thread methods:**

* Sleeping
* Waiting
* Blocked

**Sleep:**Sleep () method is used to sleep the currently executing thread for the given amount of time. Once the thread is wake up it can move to the runnable state. So sleep () method is used to delay the execution for some period.

It is a static method.

**Example:**

**Thread. Sleep (2000)**

So it delays the thread to sleep 2 milliseconds. Sleep () method throws an uninterrupted exception, hence we need to surround the block with try/catch.

|  |
| --- |
| **public** **class** ExampleThread **implements** Runnable{  **public** **static** **void** main (String[] args){  Thread t = **new** Thread ();  t.start ();  }  **public** **void** run(){  **try**{  Thread.sleep(2000);  }**catch**(InterruptedException e){  }  } |

**Q #45) When to use the Runnable interface Vs Thread class in Java?**

**Answer:** If we need our class to extend some other classes other than the thread then we can go with the runnable interface because in java we can extend only one class.

If we are not going to extend any class then we can extend the thread class.

**Q #46) Difference between start() and run() method of thread class.**

**Answer:** Start() method creates a new thread and the code inside the run () method is executed in the new thread. If we directly called the run() method then a new thread is not created and the currently executing thread will continue to execute the run() method.

**Q #47) What is Multi-threading?**

**Answer:** Multiple threads are executed simultaneously. Each thread starts its own stack based on the flow (or) priority of the threads.

**Example Program:**

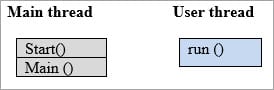
|  |
| --- |
| **public** **class** MultipleThreads **implements** Runnable  {  **public** **static** **void** main (String[] args){//Main thread starts here  Runnable r = **new** runnable ();  Thread t=**new** thread ();  t.start ();//User thread starts here  Addition add=**new** addition ();  }  **public** **void** run(){  go();  }//User thread ends here  } |

On the 1st line execution, JVM calls the main method and the main thread stack looks as shown below.

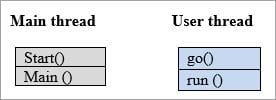
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Thread1.jpg)

Once the execution reaches, **t.start ()**line then a new thread is created and the new stack for the thread is also created. Now JVM switches to the new thread and the main thread are back to the runnable state.

The two stacks look as shown below.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Thread2.jpg)

Now, the user thread executed the code inside the run() method.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2017/10/Thread3.jpg)

Once the run() method has completed, then JVM switches back to the main thread and the user thread has completed the task and the stack was disappeared.

JVM switches between each thread until both the threads are completed. This is called Multi-threading.

**Q #48) Explain the thread life cycle in Java.**

**Answer:** **Thread has the following states:**

* New
* Runnable
* Running
* Non-runnable (Blocked)
* Terminated

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/01/Thread-Life-Cycle-in-Java.jpg)

* **New:**In New state, a Thread instance has been created but start () method is not yet invoked. Now the thread is not considered alive.
* **Runnable**: The Thread is in the runnable state after the invocation of the start () method, but before the run () method is invoked. But a thread can also return to the runnable state from waiting/sleeping. In this state, the thread is considered alive.
* **Running**: The thread is in a running state after it calls the run () method. Now the thread begins the execution.
* **Non-Runnable**(Blocked): The thread is alive but it is not eligible to run. It is not in the runnable state but also, it will return to the runnable state after some time. **Example:** wait, sleep, block.
* **Terminated**: Once the run method is completed then it is terminated. Now the thread is not alive.

**Q #49) What is Synchronization?**

**Answer:** Synchronization makes only one thread to access a block of code at a time. If multiple threads accesses the block of code, then there is a chance for inaccurate results at the end. To avoid this issue, we can provide synchronization for the sensitive block of codes.

The synchronized keyword means that a thread needs a key in order to access the synchronized code.

Locks are per objects. Every Java object has a lock. A lock has only one key. A thread can access a synchronized method only if the thread can get the key to the objects to lock.

For this, we use the “Synchronized” keyword.

**Example:**

|  |
| --- |
| **public** **class** ExampleThread **implements** Runnable{  **public** **static** **void** main (String[] args){   Thread t = **new** Thread ();   t.start ();   }  **public** **void** run(){  **synchronized**(object){   {   }  } |

**Q #50) What is the disadvantage of Synchronization?**

**Ans:** Synchronization is not recommended to implement all the methods. Because if one thread accesses the synchronized code then the next thread should have to wait. So it makes a slow performance on the other end.

**Q #51) What is meant by Serialization?**

**Answer:** Converting a file into a byte stream is known as Serialization. The objects in the file are converted to bytes for security purposes. For this, we need to implement a java.io.Serializable interface. It has no method to define.

Variables that are marked as transient will not be a part of the serialization. So we can skip the serialization for the variables in the file by using a transient keyword.

***Learn More =>>***[***Serializable and Cloneable***](https://www.softwaretestinghelp.com/marker-interfaces-java/)

**Q #52) What is the purpose of a transient variable?**

**Answer:** Transient variables are not part of the serialization process. During deserialization, the values of the transient variables are set to the default value. It is not used with static variables.

**Example:**

transient int numbers;

**Q #53) Which methods are used during the Serialization and Deserialization process?**

**Answer:** ObjectOutputStream and ObjectInputStream classes are higher level java.io. package. We will use them with lower level classes FileOutputStream and FileInputStream.

ObjectOutputStream.writeObject**—->**Serialize the object and write the serialized object to a file.

ObjectInputStream.readObject **—>** Reads the file and deserializes the object.

To be serialized, an object must implement the serializable interface. If superclass implements Serializable, then the subclass will automatically be serializable.

**Q #54) What is the purpose of a Volatile Variable?**

**Answer:** Volatile variable values are always read from the main memory and not from thread’s cache memory. This is used mainly during synchronization. It is applicable only for variables.

**Example:**

volatile int number;

**Q #55) Difference between Serialization and Deserialization in Java.**

**Answer:** **These are the differences between serialization and deserialization in java:**

| **Serialization** | **Deserialization** |
| --- | --- |
| Serialization is the process which is used to convert the objects into byte stream | Deserialization is the opposite process of serialization where we can get the objects back from the byte stream. |
| An object is serialized by writing it an ObjectOutputStream. | An object is deserialized by reading it from an ObjectInputStream. |

**Q #56) What is SerialVersionUID?**

**Answer:** Whenever an object is Serialized, the object is stamped with a version ID number for the object class. This ID is called the  SerialVersionUID. This is used during deserialization to verify that the sender and receiver that are compatible with the Serialization.

**Conclusion**

These are some of the core JAVA interview questions that cover both the basic and advanced Java concepts for programming as well as developer interview, and these are ones which have been answered by our JAVA experts.

I hope that this tutorial will give you a great insight into JAVA core coding concepts in detail. The explanations given above will really enrich your knowledge and increase your understanding of JAVA programming.

***Get ready to crack a JAVA interview confidently.***

**Most Frequently Asked OOPS Interview Questions**

**Q #1) Explain in brief what do you mean by Object Oriented Programming in Java?**

**Answer:** OOP deals with objects, like real-life entities such as pen, mobile, bank account which has state (data) and behavior (methods).

With help of access, specifiers access to this data and methods is made secured.

Concepts of encapsulation and abstraction offer data hiding and access to essentials, inheritance, and polymorphism help code reuse and overloading/overriding of methods and constructors, making applications platform-independent, secured and robust using languages like Java.

**Q #2) Explain Is Java a pure Object Oriented language?**

**Answer:** Java is not an entirely pure object-oriented programming language. **The following are the reasons:**

* Java supports and uses primitive data types such as int, float, double, char, etc.
* Primitive data types are stored as variables or on the stack instead of the heap.
* In Java, static methods can access static variables without using an object, contrary to object-oriented concepts.

**Q #3) Describe class and object in Java?**

**Answer:** Class and object play an integral role in object-oriented programming languages like Java.

* Class is a prototype or a template that has state and behavior supported by an object and used in the creation of objects.
* The object is an instance of the class, **for example,** Human is a class with the state as having a vertebral system, brain, color, and height and has behavior such as canThink(), ableToSpeak(), etc.

**Q #4) What are the differences between class and objects in Java?**

**Answer:** **Following are a few major differences between class and objects in Java:**

| **Class** | **Object** |
| --- | --- |
| Class is a logical entity | Object is physical entity |
| Class is a template from which object can be created | Object is an instance of the class |
| Class is a prototype that has the state and behavior of similar objects | Objects are entities that exist in real life such as mobile, mouse, or intellectual objects such as bank account |
| Class is declared with class key word like class Classname { } | Object is created via new keyword as Employee emp = new Employee(); |
| During class creation, there is no allocation of memory | During object creation, memory is allocated to the object |
| There is only one-way class is defined using the class keyword | Object creation can be done many ways such as using new keyword, newInstance() method, clone() and factory method. |
| Real-life examples of Class can be a •A recipe to prepare food. •Blue prints for an automobile engine. | Real-life examples of Object can be •A food prepared from recipe. •Engine constructed as per blue-prints. |

**Q #5) Why is a need for Object-oriented programming?**

**Answer:** OOP provides access specifiers and data hiding features for more security and control data access, overloading can be achieved with function and operator overloading, Code Reuse is possible as already created objects in one program can be used in other programs.

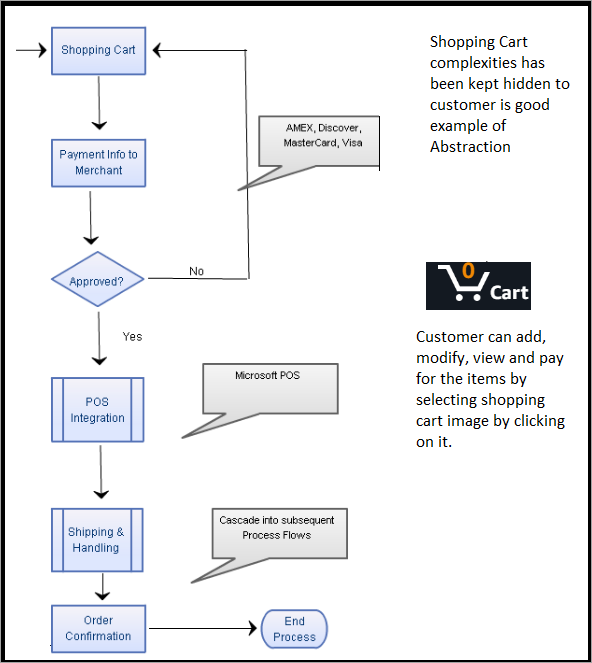
Data redundancy, code maintenance, data security, and advantage of concepts such as encapsulation, abstraction, polymorphism, and inheritance in object-oriented programming provide an advantage over previously used procedural programming languages.

**Q #6) Explain Abstraction with a real-time example.**

**Answer:** Abstraction in object-oriented programming means hiding complex internals but to expose only essential characteristics and behavior with respect to context.

In real life, an example of abstraction is an online shopping cart, say at any e-commerce site. Once you select a product and book order, you are just interested in receiving your product on time.

How things happen is not what you are interested in, as it is complex and kept hidden. This is known as abstraction. Similarly, take the example of ATM, the complexity of internals of how money is debited from your account is kept hidden, and you receive cash via a network. Similarly for cars, how petrol makes the engine run the automobile is extremely complex.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/09/abstraction.png)

**Q #7) Give some real-time examples and explain Inheritance.**

**Answer:** Inheritance means one class (sub class) acquiring properties of another class (super class) by inheritance.

In real life, take an example of inheritance of a normal bicycle where it is a parent class and a sports bike can be a child class, where sports bike has inherited properties and behavior of rotating wheels with pedals via gears that of a normal bike.

**Q #8) How polymorphism works in Java, explain with real-life examples?**

**Answer:** Polymorphism is an ability to have multiple forms or capability of the method to do different things.

In real life, the same person performing different duties behaves differently. In-Office he is an employee, at home, he is a father, during or in after school tuitions he is a student, on weekends he plays cricket and is a player in the playground.

In Java, there are two types of polymorphism

* **Compile-time** **polymorphism:** This is achieved by method overloading or operator overloading.
* **Runtime polymorphism:** This is achieved by method overriding.

**Q #9) How many types of inheritance are present?**

**Answer:** **Various types of inheritance are listed below:**

* **Single Inheritance:** Single child class inherits characteristics of the single-parent class.
* **Multiple Inheritance:** One class inherits features of more than one base class and is not supported in Java, but the class can implement more than one interface.
* **Multilevel Inheritance:** A class can inherit from a derived class making it a base class for a new class, **for example,** a Child inherits behavior from his father, and the father has inherited characteristics from his father.
* **Hierarchical Inheritance:** One class is inherited by multiple subclasses.
* **Hybrid Inheritance:**This is a combination of single and multiple inheritances.

**Q #10) What is Interface?**

**Answer:** Interface is similar to the class where it can have methods and variables, but its methods do not have a body, just a signature known as the abstract method. Variables declared in the interface can have public, static, and final by default. Interface is used in Java for abstraction and multiple inheritances, where the class can implement multiple interfaces.

**Q #11) Can you explain the advantages of Abstraction and Inheritance?**

**Answer:** Abstraction reveals only essential details to the user and ignores or hides irrelevant or complex details. In other words, data abstraction exposes the interface and hides implementation details. Java performs abstraction with the help of interfaces and abstract classes. Advantage of abstraction is that it makes simple in viewing things by reducing or hiding the complexity of implementation.

Duplication of code is avoided, and it increases code reusability. Only essential details are revealed to the user and improves the security of the application.

Inheritance is where child class inherits functionality (behavior) of the parent class. We need not write code once written in parent class for functionality again in the child class and thus making it easier to reuse the code. The code becomes readable as well. Inheritance is used where there “is a” relation. **Example:** Hyundai **is a** car OR MS Word **is a** software.

**Q #12) What is the difference between extends and implements?**

**Answer:** Both extends and implements keyword are used for inheritance but in different ways.

**The differences between Extends and Implements keywords in Java are explained below:**

| **Extends** | **Implements** |
| --- | --- |
| A class can extend another class (child extending parent by inheriting his characteristics). Interface as well inherit (using keyword extends) another interface. | A class can implement an interface |
| Sub class extending super class may not override all of the super class methods | Class implementing interface has to implement all the methods of the interface. |
| Class can only extend a single super class. | Class can implement any number of interfaces. |
| Interface can extend more than one interfaces. | Interface cannot implement any other interface. |
| **Syntax:** class Child extends class Parent | **Syntax:** class Hybrid implements Rose |

**Q #13) What are different access modifiers in Java?**

**Answer:** Access modifiers in Java controls access scope of class, constructor, variable, method, or data member.**Various types of access modifiers are as follows:**

* **Default access modifier** is without any access specifier data members, class and methods, and are accessible within the same package.
* **Private access modifiers** are marked with the keyword private, and are accessible only within class, and not even accessible by class from the same package.
* **Protected access modifiers** can be accessible within the same package or subclasses from different packages.
* **Public access modifiers** are accessible from everywhere.

**Q #14) Explain the difference between abstract class and method?**

**Answer:** **Following are some differences between abstract class and abstract method in Java:**

| **Abstract Class** | **Abstract Method** |
| --- | --- |
| Object cannot be created from the abstract class. | Abstract method has a signature but does not have a body. |
| Sub class created or inherit abstract class to access members of abstract class. | It is compulsory to override abstract methods of super class in their sub class. |
| Abstract class can contain abstract methods or non abstract methods. | Class containing abstract method should be made abstract class. |

**Q #15) What are the differences between method and constructor?**

**Answer:** **Following are the differences between constructors and methods in Java:**

| **Constructors** | **Methods** |
| --- | --- |
| Constructors name should match with that of Class. | Methods should not have same name as Class name. |
| They are used to create, initialize and allocate memory to the object. | Methods are used to execute certain statements written inside them. |
| Constructors are implicitly invoked by the system whenever objects are created. | Methods are invoked when it is called. |
| They are invoked using new keyword while creating an instance of the class (object). | Methods are invoked during program execution. |
| Constructor does not have return type. | Method has a return type. |
| Constructor cannot be inherited by the subclass. | Methods can be inherited by a sub class. |

**Q #16) What is a constructor in Java?**

**Answer:** Constructor is a method without a return type and has its name the same as the class name. When we create an object, a default constructor allocates memory for an object during the compilation of Java code. Constructors are used to initializing objects and set initial values for object attributes.

**Q #17) How many types of constructors can be used in Java? Please explain.**

**Answer:** There are basically three types of constructors in Java.

**These are:**

1. **Default constructor:** This constructor is without any parameter and invokes every time you create an instance of a class (object). If a class is an Employee, then the syntax of the default constructor will be Employee().
2. **No-arg constructor:** As the name implies, a constructor without any argument is called a no-arg constructor.
3. **Parameterized constructor:** Constructor with a number of parameters is called a parameterized constructor. You are required to provide arguments, i.e. initial values with respect to the data type of parameters in that constructor.

**Q #18) Why new keyword is used in Java?**

**Answer:** When we create an instance of class, i.e. objects, we use the Java keyword **new**. It allocates memory in the heap area where JVM reserve space for an object. Internally, it invokes the default constructor as well.

**Syntax:**

Class\_name obj = **new** Class\_name();

**Q #19) When do you use the super keyword?**

**Answer:** **Super** is a Java keyword used to identify or refer parent (base) class.

* We can use super to access super class constructor and call methods of the super class.
* When method names are the same in super class and sub class, to refer super class, the **super** keyword is used.
* To access the same name data members of parent class when they are present in parent and child class.
* **Super** can be used to make an explicit call to no-arg and parameterized constructors of the parent class.
* Parent class method access can be done using **super**, when child class has method overridden.

**Q #20) When do you use this keyword?**

**Answer:** **this** keyword in Java refers to the current object in the constructor or in the method.

* When class attributes and parameterized constructors both have the same name, **this** keyword is used.
* Keywords **this** invokes the current class constructor, method of the current class, return the object of the current class, pass an argument in the constructor, and method call.

**Q #21) What is the difference between Runtime and compile-time polymorphism?**

**Answer:** Both runtime and compile-time polymorphism are two different types of polymorphism. **Their differences are explained below:**

| **Compile Time Polymorphism** | **Runtime Polymorphism** |
| --- | --- |
| Call is resolved by a compiler in compile-time polymorphism. | Call is not resolved by the compiler in runtime polymorphism. |
| It is also known as static binding and method overloading. | It is also known as dynamic, late, and method overriding. |
| Same name methods with different parameters or methods with the same signature and different return types are compile-time polymorphism. | Same name method with the same parameters or signature associated in different classes are called method overriding. |
| It is achieved by function and operator overloading. | It can be achieved by pointers and virtual functions. |
| As all the things are executed at compile time. compile-time polymorphism is less flexible. | As things execute at run time, runtime polymorphism is more flexible. |

**Q #22) What object-oriented features are used in Java?**

**Answer:** A concept of using an object in Java programming language benefits by the use of object-oriented concepts like encapsulation for binding together the state and behavior of an object, secures data access with access specifiers, features like abstraction in information hiding, inheritance to extend state, and behavior of base classes to child classes, compile-time and runtime polymorphism for method overloading and method overriding, respectively.

**Q #23) What is method overloading?**

**Answer:** When two or more methods with the same name have either a different number of parameters or different types of parameters, these methods may have or may not have different return types, then they are overloaded methods, and the feature is method overloading. Method overloading is also called compile-time polymorphism.

**Q #24) What is method overriding?**

**Answer:** When a method of sub class (derived, child class) has the same name, parameters (signature), and same return type as the method in its super class (base, parent class) then the method in the subclass is said to be overridden the method in the superclass. This feature is also known as runtime polymorphism.

**Q #25) Explain constructor overloading.**

**Answer:** More than one constructor having different parameters so that different tasks can be carried out with each constructor is known as constructor overloading. With constructor overloading, objects can be created in different ways. Various Collection classes in Java API are examples of constructor overloading.

**Q #26) What types of arguments can be used in Java?**

**Answer:** For Java methods and functions, parameter data can be sent and received in different ways. If methodB() is called from methodA(), methodA() is a caller function and methodB() is called function, arguments sent by methodA() is actual arguments and parameters of methodB() is called formal arguments.

* **Call By Value:**Changes made to formal parameter (parameters of methodB()) do not get sent back to the caller (methodA()), This method is called **call by value**. Java supports the call by value.
* **Call by Reference:**Changes made to formal parameter (parameters of methodB()) are sent back to the caller (parameters of methodB()).
* Any changes in formal parameters (parameters of methodB()) are reflected in actual parameters (arguments sent by methodA()). This is called call by reference.

**Q #27) Differentiate between static and dynamic binding?**

**Answer:** The differences between Static and Dynamic binding are explained in the below table.

| **Static Binding** | **Dynamic Binding** |
| --- | --- |
| Static binding in Java use type of fields and class to as a resolution. | Dynamic binding in Java uses object for resolving binding. |
| Method Overloading is an example of static binding. | Method overriding is an example of dynamic binding. |
| Static binding gets resolved at compile time. | Dynamic binding gets resolved at run time. |
| Methods and variables using static binding are private, final and static types. | Virtual methods use dynamic binding. |

**Q #28) Can you explain base class, subclass, and superclass?**

**Answer: Base class, sub class, and super class in Java are explained as follows:**

* Base class or parent class is a super class and is a class from which sub class or child class is derived.
* Sub class is a class that inherits attributes (properties) and methods (behavior) from the base class.

**Q #29) Is Operator overloading supported in Java?**

**Answer:** Operator overloading is not supported by Java as,

* It makes the interpreter put more effort to understand the actual functionality of the operator making code complex and difficult to compile.
* Operator overloading makes programs more error-prone.
* However, the feature of operator overloading can be achieved in method overloading in a simple, clear, and error-free way.

**Q #30) When the finalize method is used?**

**Answer:** **finalize** method is called just before the object is about to be garbage collected. This method overrides to minimize memory leaks, undertake cleanup activities by removing system resources.

**Q #31) Explain about Tokens.**

**Answer:** Tokens in the Java program are the smallest elements that the compiler recognizes. Identifiers, keywords, literals, operators, and separators are examples of tokens.

## Conclusion

Object-oriented programming concepts are an integral part for developers, automation as well as manual testers who design automation testing framework to test an application or develop applications with Java programming language.

In-depth understanding is mandatory of all object-oriented features like class, object, abstraction, encapsulation, inheritance, polymorphism, and applying these concepts in a programming language like Java to achieve customer requirements.

We have tried to cover the most important object-oriented programming interview questions and gave appropriate answers with examples.

We wish you all the best for your upcoming interview!