

A **Java interface** is a bit like a class, except a **Java interface** can only contain method signatures and fields. An **Java interface** cannot contain an implementation of the methods, only the signature (name, parameters and exceptions) of the method. You can use **interfaces** in **Java** as a way to achieve polymorphism.

The word '**polymorphism**' literally means 'a state of having many shapes' or 'the capacity to take on different forms'. When applied to object oriented programming languages like **Java**, it describes a language's ability to process objects of various types and classes through a single, uniform interface

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

**Example:**

Let us look at an example.

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

Now, the Deer class is considered to be polymorphic since this has multiple inheritance. Following are true for the above example:

* A Deer IS-A Animal
* A Deer IS-A Vegetarian
* A Deer IS-A Deer
* A Deer IS-A Object

When we apply the reference variable facts to a Deer object reference, the following declarations are legal:

Deer d = new Deer();

Animal a = d;

Vegetarian v = d;

Object o = d;

All the reference variables d,a,v,o refer to the same Deer object in the heap.

**Virtual Methods:**

In this section, I will show you how the behaviour of overridden methods in Java allows you to take advantage of polymorphism when designing your classes.

We already have discussed method overriding, where a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

/\* File name : Employee.java \*/

public class Employee

{

private String name;

private String address;

private int number;

public Employee(String name, String address, int number)

{

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public void mailCheck()

{

System.out.println("Mailing a check to " + this.name

+ " " + this.address);

}

public String toString()

{

return name + " " + address + " " + number;

}

public String getName()

{

return name;

}

public String getAddress()

{

return address;

}

public void setAddress(String newAddress)

{

address = newAddress;

}

public int getNumber()

{

return number;

}

}

Now suppose we extend Employee class as follows:

/\* File name : Salary.java \*/

public class Salary extends Employee

{

private double salary; //Annual salary

public Salary(String name, String address, int number, double

salary)

{

super(name, address, number);

setSalary(salary);

}

public void mailCheck()

{

System.out.println("Within mailCheck of Salary class ");

System.out.println("Mailing check to " + getName()

+ " with salary " + salary);

}

public double getSalary()

{

return salary;

}

public void setSalary(double newSalary)

{

if(newSalary >= 0.0)

{

salary = newSalary;

}

}

public double computePay()

{

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

}

Now, you study the following program carefully and try to determine its output:

/\* File name : VirtualDemo.java \*/

public class VirtualDemo

{

public static void main(String [] args)

{

Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Call mailCheck using Salary reference --");

s.mailCheck();

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

This would produce the following result:

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

ailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

ailing check to John Adams with salary 2400.0

Here, we instantiate two Salary objects . one using a Salary reference s, and the other using an Employee reference e.

While invoking *s.mailCheck()* the compiler sees mailCheck() in the Salary class at compile time, and the JVM invokes mailCheck() in the Salary class at run time.

Invoking mailCheck() on e is quite different because e is an Employee reference. When the compiler sees *e.mailCheck()*, the compiler sees the mailCheck() method in the Employee class.

Here, at compile time, the compiler used mailCheck() in Employee to validate this statement. At run time, however, the JVM invokes mailCheck() in the Salary class.

This behavior is referred to as virtual method invocation, and the methods are referred to as virtual methods. All methods in Java behave in this manner, whereby an overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

# Recursive Programming

## Introduction

When we write a method for solving a particular problem, one of the basic design techniques is to break the task into smaller subtasks. For example, the problem of adding (or multiplying) n consecutive integers can be reduced to a problem of adding (or multiplying) n-1consecutive integers:

1 + 2 + 3 +... + n = n + [1 + 2 + 3 + .. + (n-1)]

1 \* 2 \* 3 \*... \* n = n \* [1 \* 2 \* 3 \* .. \* (n-1)]

Therefore, if we introduce a method sumR(n) (or timesR(n)) that adds (or multiplies) integers from 1 to n, then the above arithmetics can be rewritten as

sumR(n) = n + sumR(n-1)

timesR(n) = n \* timesR(n-1)

Such functional definition is called a **recursive** definition, since the definition contains a call to itself. On each recursive call the argument of sumR(n) (or timesR(n)) gets smaller by one. It takes n-1 calls until we reach the **base case** - this is a part of a definition that does not make a call to itself. Each recursive definition requires base cases in order to prevent infinite recursion.

In the following example we provide iterative and recursive implementations for the addition and multiplication of n natural numbers.

public int sum(int n) public int sumR(int n)

{ {

int res = 0; if(n == 1)

for(int i = 1; i = n; i++) return 1;

res = res + i; else

return n + sumR(n-1);

return res; }

}

To solve a problem recursively means that you have to first redefine the problem in terms of a smaller subproblem of the same type as the original problem. In the above summation problem, to sum-up n integers we have to know how to sum-up n-1 integers. Next, you have to figure out how the solution to smaller subproblems will give you a solution to the problem as a whole. This step is often called as a *recursive leap of faith*. Before using a recursive call, you must be convinced that the recursive call will do what it is supposed to do. You do not need to think how recursive calls works, just assume that it returns the correct result.

### Towers of Hanoi

In the great temple of Brahma in Benares group of spiritually advanced monks have to move 64 golden disks from one diamond needle to another. And, there is only one other location in the temple (besides the original and destination locations) sacred enough that a pile of disks can be placed there. The 64 disks have different sizes, and the monks must obey two rules:

1. only one disk can be moved at a time
2. a bigger disk can never be placed on a top of a smaller disk.

The legend is that, before the monks make the final move to complete the new pile in the new location, the next Maha Pralaya will begin and the temple will turn to dust and the world will end. Is there any truth to this legend?

See the simulation applet at <http://www.mazeworks.com/hanoi/index.htm>.

The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. The puzzle is well known to students of Computer Science since it appears in virtually any introductory text on data structures or algorithms.

*Recursive solution:* first we move the top n - 1 discs to an empty pole, then we move the largest disc to the other empty pole, then complete the job by moving the n - 1 discs onto the largest disc. Let T(n) represent the number of steps needed to move n discs. Then T(n) can be counted as follows

T(n) = T(n-1) + 1 + T(n-1)

### Bookkeeping

One might wonder how the runtime system handles recursive functions. There is a lot of bookkeeping information that one has to keep track of: for each call one has to record who made the call and what arguments are to be handed over. Most importantly, though, one has to keep track of all the pending calls, which may be very deeply nested inside each other. As it turns out, all that is needed is a single stack. Whenever a function call is made (recursive or not), all the necessary bookkeeping information is pushed onto the stack. When the execution of the function terminates, the return value is handed over to whoever made the call (pop from the stack). Consider the following call sumR(5). Here is the bookkeeping information

sumR(5)

sumR(4)

sumR(3)

sumR(2)

sumR(1)

return 1

return 2 + 1

return 3 + 2 + 1

return 4 + 3 + 2 + 1

return 5 + 4 + 3 + 2 + 1

Comparing recursive implementation against iterative implementation, we can say that the former is at least twice slower, since, first, we unfold recursive calls (pushing them on a stack) until we reach the base case and ,second, we traverse the stack and retrieve all recursive calls. Note, actual computation happends when we pop recursive calls from that system stack.

### Tail and Head recursions

If the recursive call occurs at the end of a method, it is called a tail recursion. The tail recursion is similar to a loop. The method executes all the statements before jumping into the next recursive call.

If the recursive call occurs at the beginning of a method, it is called a head recursion. The method saves the state before jumping into the next recursive call. Compare these:

public void tail(int n) public void head(int n)

{ {

if(n == 1) if(n == 0)

return; return;

else else

System.out.println(n); head(n-1);

tail(n-1); System.out.println(n);

}

### Mathematical Induction

Recursive programming is directly related to mathematical induction

The **base case** is to prove the statement true for some specific value or values of N.

The **induction step** -- assume that a statement is true for all positive integers less than N,then prove it true for N.

## Binary Search

Locate the element x in a sorted array by first comparing x with the middle element and then (if they are not equal) dividing the array into two subarrays and repeat the whole procedure in one of them. If x is less than the middle element you search in the left subarray, otherwise - in the right subarray.

Let T(n) denote the number of comparisons required to find a key in a sorted array of size n. Then we have the following recurrent equation for T(n);

T(n) = T(n/2) + 1

This directly translates into the following recursive code:

public int searchR(int[] a, int key) {

return helper(a, key, 0, a.length-1);

}

private int helper(int[] a, int key, int left, int right) {

if (left > right) return -1;

int mid=(left+right)/2;

if (key == a[mid]) return mid;

else

if (key > a[mid])

return helper(a, key, mid + 1, right);

else

return helper(a, key, left, mid - 1);

}

## The Mandelbrot Set

|  |  |
| --- | --- |
|  | The Mandelbrot set is the set of all complex numbers **c** for which sequence defined by the iteration  f(n+1) = f(n)2 + c, f(0) = c  remains *bounded* or converges to a *fixed point* when n tends to infinity. In the picture the Mandelbrot set is that blue shape in the middle. The Mandelbrot set is named after Benoit Mandelbrot who constructed the first images of this set in 1978.  Applets to explore the Mandelbrot set, and other fractals, can be found at [Dynamical Systems and Technology Project](http://math.bu.edu/DYSYS/) website.  The Mandelbrot set is a famous example of a *fractal* - fragmented geometric shape that can be split into parts, each of which is a copy of the whole. |

Here are two examples of bunded and unbounded sequences:

## Fibonacci Numbers

Fibonacci was born 1170 in Pisa, Italy and died in 1250. His real name is Leonardo Pisano. In 1202 he wrote a book: Liber Abbaci, meaning "Book of Calculating".

The Fibonacci number is defined as the sum of the two preceding numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

This recursive definition translates directly into code

public int fibonacci(int n)

{

if (n <= 0) return 0;

else if (n == 1) return 1

else return fibonacci(n-1) + fibonacci(n-2);

}

|  |  |
| --- | --- |
| https://www.cs.cmu.edu/%7Eadamchik/15-121/lectures/Recursions/pix/fib.bmp | This is a binary tree of recursive calls for fibonacci(5). The picture shows that the tree for fibonacci(5) has 5 levels, and thus, the total number of nodes is about 2^5. Based on this estimate we guess that the complexity of recursive implementation is exponential, namely O(2n). We can formally prove this statement by deriving a recursive equation for the number of calls: |

## Linked Lists Recursively

A linked list is a recursive data structure. A linked list is either empty or consistes of a node followed by a linked list. As an example, consider iterative and recursive implementations of the addLast() method

           iterative implementation                                               recursive implementation

public void addLast(Object item) public void addLast(Object item)

{ {

if( head == null) if( head == null)

addFirst(item); addFirst(item);

else else

{ addLast(head, item);

Node tmp = head; }

private void addLast(Node node,

while(tmp.next != null) Object item)

tmp = tmp.next; {

if(node.next != null)

tmp.next = new Node(item, null); addLast(node.next, item);

} else

} node.next = new Node(item, null);

}

As an exercise implement

public String toString()

public void insertAfter(Object key, Object toInsert)

public LinkedList clone()

Our next example is the insertBefore method - find the key and insert a new node before this node.

public void insertBefore(Object key, Object toInsert)

{

head = insertBefore(key, head, toInsert);

}

public Node insertBefore(Object key, Node curNode, Object toInsert)

{

if(curNode == null)

return null;

else

if(curNode.data.equals(key))

return new Node(toInsert, curNode);

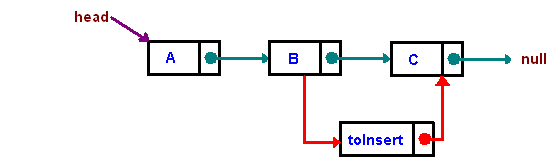
else

curNode.next = insertBefore(key, curNode.next, toInsert);

return curNode;

}

Suppose we want to insert before "C". Let us trace the above code by creating a system stack of calls



head = insertBefore(A, C, toInsert);

"A".next = insertBefore(B, C, toInsert);

"B".next = insertBefore(C, C, toInsert);

insertBefore(C, C, toInsert) returns new Node(toInsert, C)

As soon as we reach the base case, we pop calls from a system stack. The first two pops will insert a new node between "B" and "C"

"B".next = insertBefore(C, C, toInsert) = new Node(toInsert, C);

All following assignments

head = insertBefore(A, C, toInsert);

"A".next = insertBefore(B, C, toInsert);

are redundant, they do not add anything to the list. Another important sub-case of the above implementation is when we need to insert a new node before the head. The assignment head = insertBefore(head, key, toInsert); takes care of this case.

As an exercise implement

public void delete(Object key)

public void insertInOrder(Comparable key)

**1) What is difference between JDK,JRE and JVM?**

**JVM**

JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime environment in which java bytecode can be executed. It is a specification.

JVMs are available for many hardware and software platforms (so JVM is platform dependent).

**JRE**

JRE stands for Java Runtime Environment. It is the implementation of JVM.

**JDK**

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

[more details...](http://www.javatpoint.com/difference-between-jdk-jre-and-jvm)

**2) How many types of memory areas are allocated by JVM?**

Many types:

1. Class(Method) Area
2. Heap
3. Stack
4. Program Counter Register
5. Native Method Stack

[more details...](http://www.javatpoint.com/internal-details-of-jvm)

**3) What is JIT compiler?**

**Just-In-Time(JIT) compiler:**It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term “compiler” refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

**4) What is platform?**

A platform is basically the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform.

**5) What is the main difference between Java platform and other platforms?**

The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

**6) What gives Java its 'write once and run anywhere' nature?**

The bytecode. Java is compiled to be a byte code which is the intermediate language between source code and machine code. This byte code is not platform specific and hence can be fed to any platform.

**7) What is classloader?**

The classloader is a subsystem of JVM that is used to load classes and interfaces.There are many types of classloaders e.g. Bootstrap classloader, Extension classloader, System classloader, Plugin classloader etc.

**8) Is Empty .java file name a valid source file name?**

Yes, save your java file by .java only, compile it by **javac .java** and run by **java yourclassname** Let's take a simple example:

1. //save by .java only
2. class A{
3. public static void main(String args[]){
4. System.out.println("Hello java");
5. }
6. }
7. //compile by javac .java
8. //run by     java A

compile it by **javac .java**

run it by **java A**

**9) Is delete,next,main,exit or null keyword in java?**

No.

**10) If I don't provide any arguments on the command line, then the String array of Main method will be empty or null?**

It is empty. But not null.

**11) What if I write static public void instead of public static void?**

Program compiles and runs properly.

**12) What is the default value of the local variables?**

The local variables are not initialized to any default value, neither primitives nor object references.

**Core Java - OOPs Concepts: Initial OOPs Interview Questions**

There is given more than 50 OOPs (Object-Oriented Programming and System) interview questions. But they have been categorized in many sections such as constructor interview questions, static interview questions, Inheritance Interview questions, Abstraction interview question, Polymorphism interview questions etc. for better understanding.

**13) What is difference between object oriented programming language and object based programming language?**

Object based programming languages follow all the features of OOPs except Inheritance. Examples of object based programming languages are JavaScript, VBScript etc.

**14) What will be the initial value of an object reference which is defined as an instance variable?**

The object references are all initialized to null in Java.

**Core Java - OOPs Concepts: Constructor Interview Questions**

**15) What is constructor?**

* Constructor is just like a method that is used to initialize the state of an object. It is invoked at the time of object creation.

[more details...](http://www.javatpoint.com/constructor)

**16) What is the purpose of default constructor?**

* The default constructor provides the default values to the objects. The java compiler creates a default constructor only if there is no constructor in the class.[more details...](http://www.javatpoint.com/constructor)

**17) Does constructor return any value?**

**Ans:**yes, that is current instance (You cannot use return type yet it returns a value).[more details...](http://www.javatpoint.com/constructor)

**18)Is constructor inherited?**

No, constructor is not inherited.

**19) Can you make a constructor final?**

No, constructor can't be final.

**Core Java - OOPs Concepts: static keyword Interview Questions**

**20) What is static variable?**

* static variable is used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
* static variable gets memory only once in class area at the time of class loading.

[more details...](http://www.javatpoint.com/static-keyword-in-java)

**21) What is static method?**

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

[more details...](http://www.javatpoint.com/static-keyword-in-java)

**22) Why main method is static?**

because object is not required to call static method if It were non-static method,jvm creats object first then call main() method that will lead to the problem of extra memory allocation.[more details...](http://www.javatpoint.com/static-keyword-in-java)

**23) What is static block?**

* Is used to initialize the static data member.
* It is excuted before main method at the time of classloading.

[more details...](http://www.javatpoint.com/static-keyword-in-java)

**24) Can we execute a program without main() method?**

Ans) Yes, one of the way is static block.[more details...](http://www.javatpoint.com/static-keyword-in-java)

**25) What if the static modifier is removed from the signature of the main method?**

Program compiles. But at runtime throws an error "NoSuchMethodError".

**26) What is difference between static (class) method and instance method?**

|  |  |
| --- | --- |
| **static or class method** | **instance method** |
| 1)A method i.e. declared as static is known as static method. | A method i.e. not declared as static is known as instance method. |
| 2)Object is not required to call static method. | Object is required to call instance methods. |
| 3)Non-static (instance) members cannot be accessed in static context (static method, static block and static nested class) directly. | static and non-static variables both can be accessed in instance methods. |
| 4)For example: public static int cube(int n){ return n\*n\*n;} | For example: public void msg(){...}. |

**Core Java - OOPs Concepts: Inheritance Interview Questions**

**27) What is this in java?**

It is a keyword that that refers to the current object.[more details...](http://www.javatpoint.com/this-keyword)

**28)What is Inheritance?**

Inheritance is a mechanism in which one object acquires all the properties and behaviour of another object of another class. It represents IS-A relationship. It is used for Code Resusability and Method Overriding.

[more details...](http://www.javatpoint.com/inheritance-in-java)

**29) Which class is the superclass for every class.**

Object class.

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

* To reduce the complexity and simplify the language, multiple inheritance is not supported in java in case of class.[more details...](http://www.javatpoint.com/inheritance-in-java)

**31) What is composition?**

Holding the reference of the other class within some other class is known as composition.

**32) What is difference between aggregation and composition?**

Aggregation represents weak relationship whereas composition represents strong relationship. For example: bike has an indicator (aggregation) but bike has an engine (compostion).

**33) Why Java does not support pointers?**

Pointer is a variable that refers to the memory address. They are not used in java because they are unsafe(unsecured) and complex to understand.

**34) What is super in java?**

It is a keyword that refers to the immediate parent class object.[more details...](http://www.javatpoint.com/super-keyword)

**35) Can you use this() and super() both in a constructor?**

No. Because super() or this() must be the first statement.

**36)What is object cloning?**

The object cloning is used to create the exact copy of an object. [more details...](http://www.javatpoint.com/object-cloning)

**Core Java - OOPs Concepts: Method Overloading Interview Questions**

**37) What is method overloading?**

If a class have multiple methods by same name but different parameters, it is known as Method Overloading. It increases the readability of the program.[more details...](http://www.javatpoint.com/method-overloading-in-java)

**38) Why method overloading is not possible by changing the return type in java?**

Becauseof ambiguity.[more details...](http://www.javatpoint.com/method-overloading-in-java)

**39) Can we overload main() method?**

Yes, You can have many main() methods in a class by overloading the main method.

[more details...](http://www.javatpoint.com/method-overloading-in-java)

**Core Java - OOPs Concepts: Method Overriding Interview Questions**

**40) What is method overriding:**

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to provide the specific implementation of the method.[more details...](http://www.javatpoint.com/method-overriding-in-java)

**41) Can we override static method?**

No, you can't override the static method because they are the part of class not object.

**42) Why we cannot override static method?**

It is because the static method is the part of class and it is bound with class whereas instance method is bound with object and static gets memory in class area and instance gets memory in heap.

**43) Can we override the overloaded method?**

Yes.

**44) Difference between method Overloading and Overriding.**

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| 1) Method overloading increases the readability of the program. | Method overriding provides the specific implementation of the method that is already provided by its super class. |
| 2) method overlaoding is occurs within the class. | Method overriding occurs in two classes that have IS-A relationship. |
| 3) In this case, parameter must be different. | In this case, parameter must be same. |

**45) Can you have virtual functions in Java?**

Yes, all functions in Java are virtual by default.

**46) What is covariant return type?**

Now, since java5, it is possible to override any method by changing the return type if the return type of the subclass overriding method is subclass type. It is known as covariant return type. [more details...](http://www.javatpoint.com/covariant-return-type)

**Core Java - OOPs Concepts: final keyword Interview Questions**

**47) What is final variable?**

If you make any variable as final, you cannot change the value of final variable(It will be constant).[more details...](http://www.javatpoint.com/final-keyword)

**48) What is final method?**

Final methods can't be overriden.[more details...](http://www.javatpoint.com/final-keyword)

**49) What is final class?**

Final class can't be inherited. [more details...](http://www.javatpoint.com/final-keyword)

**50) What is blank final variable?**

A final variable, not initalized at the time of declaration, is known as blank final variable.[more details...](http://www.javatpoint.com/final-keyword)

**51) Can we intialize blank final variable?**

Yes, only in constructor if it is non-static. If it is static blank final variable, it can be initialized only in the static block.[more details...](http://www.javatpoint.com/final-keyword)

**52) Can you declare the main method as final?**

Yes, such as, public static final void main(String[] args){}.

### 53) What is Runtime Polymorphism?

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time.

In this process, an overridden method is called through the reference variable of a super class. The determination of the method to be called is based on the object being referred to by the reference variable.

[more details...](http://www.javatpoint.com/runtime-polymorphism-in-java)

### 54) Can you achieve Runtime Polymorphism by data members?

No.

[more details...](http://www.javatpoint.com/runtime-polymorphism-in-java)

### 55) What is the difference between static binding and dynamic binding?

In case of static binding type of object is determined at compile time whereas in dynamic binding type of object is determined at runtime.

[more details...](http://www.javatpoint.com/static-binding-and-dynamic-binding)

## Core Java - OOPs Concepts : Abstraction Interview Questions

### 56) What is abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

[more details...](http://www.javatpoint.com/abstract-class-in-java)

Abstraction lets you focus on what the object does instead of how it does it.

### 57) What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

[more details...](http://www.javatpoint.com/abstract-class-in-java)

### 58) What is abstract class?

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

[more details...](http://www.javatpoint.com/abstract-class-in-java)

### 59) Can there be any abstract method without abstract class?

No, if there is any abstract method in a class, that class must be abstract.

### 60) Can you use abstract and final both with a method?

No, because abstract method needs to be overridden whereas you can't override final method.

### 61) Is it possible to instantiate the abstract class?

No, abstract class can never be instantiated.

### 62) What is interface?

Interface is a blueprint of a class that have static constants and abstract methods.It can be used to achieve fully abstraction and multiple inheritance.

[more details...](http://www.javatpoint.com/interface-in-java)

### 63) Can you declare an interface method static?

No, because methods of an interface is abstract by default, and static and abstract keywords can't be used together.

### 64) Can an Interface be final?

No, because its implementation is provided by another class.

### 65) What is marker interface?

An interface that have no data member and method is known as a marker interface.For example Serializable, Cloneable etc.

### 66) What is difference between abstract class and interface?

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1)An abstract class can have method body (non-abstract methods). | Interface have only abstract methods. |
| 2)An abstract class can have instance variables. | An interface cannot have instance variables. |
| 3)An abstract class can have constructor. | Interface cannot have constructor. |
| 4)An abstract class can have static methods. | Interface cannot have static methods. |
| 5)You can extends one abstract class. | You can implement multiple interfaces. |

### 67) Can we define private and protected modifiers for variables in interfaces?

No, they are implicitly public.

### 68) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

## Core Java - OOPs Concepts : Package Interview Questions

### 69) What is package?

A package is a group of similar type of classes interfaces and sub-packages. It provides access protection and removes naming collision.

[more details...](http://www.javatpoint.com/package)

### 70) Do I need to import java.lang package any time? Why ?

No. It is by default loaded internally by the JVM.

### 71) Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or same class multiple times. Neither compiler nor JVM complains about it.But the JVM will internally load the class only once no matter how many times you import the same class.

### 72) What is static import ?

By static import, we can access the static members of a class directly, there is no to qualify it with the class name.

There is given a list of exception handling interview questions with answers. If you know any exception handling interview question, kindly post it in the comment section.

**73) What is Exception Handling?**

Exception Handling is a mechanism to handle runtime errors.It is mainly used to handle checked exceptions.

[more details...](http://www.javatpoint.com/exception-handling-and-checked-and-unchecked-exception)

**74) What is difference between Checked Exception and Unchecked Exception?**

**1)Checked Exception**

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException,SQLException etc. Checked exceptions are checked at compile-time.

**2)Unchecked Exception**

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException,NullPointerException etc. Unchecked exceptions are not checked at compile-time.

[more details...](http://www.javatpoint.com/exception-handling-and-checked-and-unchecked-exception)

**75) What is the base class for Error and Exception?**

Throwable.

**76) Is it necessary that each try block must be followed by a catch block?**

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. And whatever exceptions are likely to be thrown should be declared in the throws clause of the method.

**77) What is finally block?**

* finally block is a block that is always executed.[more details...](http://www.javatpoint.com/finally-block-in-exception-handling)

**78) Can finally block be used without catch?**

* Yes, by try block. finally must be followed by either try or catch.[more details...](http://www.javatpoint.com/finally-block-in-exception-handling)

**79) Is there any case when finally will not be executed?**

finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).[more details...](http://www.javatpoint.com/finally-block-in-exception-handling)

**80) What is difference between throw and throws?**

|  |  |
| --- | --- |
| **throw keyword** | **throws keyword** |
| 1)throw is used to explicitly throw an exception. | throws is used to declare an exception. |
| 2)checked exceptions can not be propagated with throw only. | checked exception can be propagated with throws. |
| 3)throw is followed by an instance. | throws is followed by class. |
| 4)throw is used within the method. | throws is used with the method signature. |
| 5)You cannot throw multiple exception | You can declare multiple exception e.g. public void method()throws IOException,SQLException. |

[more details...](http://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)

**81) Can an exception be rethrown?**

Yes.

**82) Can subclass overriding method declare an exception if parent class method doesn't throw an exception ?**

Yes but only unchecked exception not checked.

[more details...](http://www.javatpoint.com/exception-handling-with-method-overriding)

**83) What is exception propagation ?**

Forwarding the exception object to the invoking method is known as exception propagation.

[more details...](http://www.javatpoint.com/exception-propagation)

**Core Java: String Handling Interview Questions**

There is given a list of string handling interview questions with short and pointed answers. If you know any string handling interview question, kindly post it in the comment section.

**84) What is the meaning of immutable in terms of String?**

The simple meaning of immutable is unmodifiable or unchangeable. Once string object has been created, its value can't be changed.

[more details...](http://www.javatpoint.com/immutable-string)

**85) Why string objects are immutable in java?**

Because java uses the concept of string literal. Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

[more details...](http://www.javatpoint.com/immutable-string)

**86) How many ways we can create the string object?**

There are two ways to create the string object, by string literal and by new keyword.

[more details...](http://www.javatpoint.com/string-handling-in-java)

**87) How many objects will be created in the following code?**

1. String s1="Welcome";
2. String s2="Welcome";
3. String s3="Welcome";

Only one object.

[more details...](http://www.javatpoint.com/string-handling-in-java)

**88) Why java uses the concept of string literal?**

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

[more details...](http://www.javatpoint.com/string-handling-in-java)

**89)How many objects will be created in the following code?**

1. String s = new String("Welcome");

Two objects, one in string constant pool and other in non-pool(heap).

[more details...](http://www.javatpoint.com/string-handling-in-java)

**90) What is the basic difference between string and stringbuffer object?**

String is an immutable object. StringBuffer is a mutable object.

**91) What is the difference between StringBuffer and StringBuilder ?**

StringBuffer is synchronized whereas StringBuilder is not synchronized.

**92) How can we create immutable class in java ?**

We can create immutable class as the String class by defining final class and

[more details...](http://www.javatpoint.com/how-to-create-immutable-class)

**93) What is the purpose of toString() method in java ?**

The toString() method returns the string representation of any object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.

[more details...](http://www.javatpoint.com/understanding-toString%28%29-method)

**Core Java : Nested classes and Interfaces Interview Questions**

**94)What is nested class?**

A class which is declared inside another class is known as nested class. There are 4 types of nested class member inner class, local inner class, annonymous inner class and static nested class.

[more details...](http://www.javatpoint.com/difference-between-nested-classes-and-inner-classes)

**95) Is there any difference between nested classes and inner classes?**

Yes, inner classes are non-static nested classes i.e. inner classes are the part of nested classes.

[more details...](http://www.javatpoint.com/difference-between-nested-classes-and-inner-classes)

**96) Can we access the non-final local variable, inside the local inner class?**

No, local variable must be constant if you want to access it in local inner class.

[more details...](http://www.javatpoint.com/local-inner-class)

**97) What is nested interface ?**

Any interface i.e. declared inside the interface or class, is known as nested interface. It is static by default.

[more details...](http://www.javatpoint.com/nested-interface)

**98) Can a class have an interface?**

Yes, it is known as nested interface.

[more details...](http://www.javatpoint.com/nested-interface)

**99) Can an Interface have a class?**

Yes, they are static implicitely.

**117) What is Garbage Collection?**

Garbage collection is a process of reclaiming the runtime unused objects.It is performed for memory management.

[more details...](http://www.javatpoint.com/Garbage-Collection)

**118) What is gc()?**

gc() is a daemon thread.gc() method is defined in System class that is used to send request to JVM to perform garbage collection.

**119) What is the purpose of finalize() method?**

finalize() method is invoked just before the object is garbage collected.It is used to perform cleanup processing.

**120) Can an unrefrenced objects be refrenced again?**

Yes.

**121)What kind of thread is the Garbage collector thread?**

Daemon thread.

**122)What is difference between final, finally and finalize?**

|  |
| --- |
| **final:** final is a keyword, final can be variable, method or class.You, can't change the value of final variable, can't override final method, can't inherit final class. |
| **finally:** finally block is used in exception handling. finally block is always executed. |
| **finalize():**finalize() method is used in garbage collection.finalize() method is invoked just before the object is garbage collected.The finalize() method can be used to perform any cleanup processing. |

**123)What is the purpose of the Runtime class?**

The purpose of the Runtime class is to provide access to the Java runtime system.

**124)How will you invoke any external process in Java?**

By Runtime.getRuntime().exec(?) method.

**I/O Interview Questions**

**125)What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?**

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.

**126)What an I/O filter?**

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.

**Serialization Interview Questions**

**127) What is serialization?**

Serialization is a process of writing the state of an object into a byte stream.It is mainly used to travel object's state on the network.

[more details...](http://www.javatpoint.com/serialization)

**128) What is Deserialization?**

Deserialization is the process of reconstructing the object from the serialized state.It is the reverse operation of serialization.

**129) What is transient keyword?**

If you define any data member as transient,it will not be serialized.[more details...](http://www.javatpoint.com/serialization)

**130)What is Externalizable?**

Externalizable interface is used to write the state of an object into a byte stream in compressed format.It is not a marker interface.

**131)What is the difference between Serializalble and Externalizable interface?**

Serializable is a marker interface but Externalizable is not a marker interface.When you use Serializable interface, your class is serialized automatically by default. But you can override writeObject() and readObject() two methods to control more complex object serailization process. When you use Externalizable interface, you have a complete control over your class's serialization process.

**Networking Interview Questions**

**132)How do I convert a numeric IP address like 192.18.97.39 into a hostname like java.sun.com?**

By InetAddress.getByName("192.18.97.39").getHostName() where 192.18.97.39 is the IP address.

**Reflection Interview Questions**

**133) What is reflection?**

Reflection is the process of examining or modifying the runtime behaviour of a class at runtime.It is used in:

* IDE (Integreted Development Environment) e.g. Eclipse, MyEclipse, NetBeans.
* Debugger
* Test Tools etc.

**134) Can you access the private method from outside the class?**

Yes, by changing the runtime behaviour of a class if the class is not secured.

# Java Multithreading Interview Questions

Multithreading and Synchronization is considered as the typical chapter in java programming. In game development company, mulithreading related interview questions are asked mostly. A list of frequently asked java multithreading interview questions are given below.

### 1) What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Its main advantage is:

* Threads share the same address space.
* Thread is lightweight.
* Cost of communication between process is low.

[more details...](http://www.javatpoint.com/multithreading)

### 2) What is thread?

A thread is a lightweight subprocess.It is a separate path of execution.It is called separate path of execution because each thread runs in a separate stack frame.

[more details...](http://www.javatpoint.com/multithreading)

### 3)What is the difference between preemptive scheduling and time slicing?

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

### 4) What does join() method?

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

[more details...](http://www.javatpoint.com/join%28%29-method)

### 5) What is difference between wait() and sleep() method?

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| 1) The wait() method is defined in Object class. | The sleep() method is defined in Thread class. |
| 2) wait() method releases the lock. | The sleep() method doesn't releases the lock. |

### 6) Is it possible to start a thread twice?

No, there is no possibility to start a thread twice. If we does, it throws an exception.

[more details...](http://www.javatpoint.com/can-we-start-a-thread-twice)

### 7) Can we call the run() method instead of start()?

yes, but it will not work as a thread rather it will work as a normal object so there will not be context-switching between the threads.

[more details...](http://www.javatpoint.com/what-if-we-call-run%28%29-method-directly)

### 8) What about the daemon threads?

The daemon threads are basically the low priority threads that provides the background support to the user threads. It provides services to the user threads.

[more details...](http://www.javatpoint.com/daemon-thread)

### 9)Can we make the user thread as daemon thread if thread is started?

No, if you do so, it will throw IllegalThreadStateException

[more details...](http://www.javatpoint.com/daemon-thread)

### 10)What is shutdown hook?

The shutdown hook is basically a thread i.e. invoked implicitely before JVM shuts down. So we can use it perform clean up resource.

[more details...](http://www.javatpoint.com/ShutdownHook-thread)

### 11)When should we interrupt a thread?

We should interrupt a thread if we want to break out the sleep or wait state of a thread.

[more details...](http://www.javatpoint.com/interrupting-a-thread)

### 12) What is synchronization?

Synchronization is the capabilility of control the access of multiple threads to any shared resource.It is used:

1. To prevent thread interference.
2. To prevent consistency problem.

[more details...](http://www.javatpoint.com/synchronization)

### 13) What is the purpose of Synchronized block?

* Synchronized block is used to lock an object for any shared resource.
* Scope of synchronized block is smaller than the method.

[more details...](http://www.javatpoint.com/synchronized-block-example)

### 14)Can Java object be locked down for exclusive use by a given thread?

Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it.

### 15) What is static synchronization?

If you make any static method as synchronized, the lock will be on the class not on object. [more details...](http://www.javatpoint.com/static-synchronization-example)

### 16)What is the difference between notify() and notifyAll()?

The notify() is used to unblock one waiting thread whereas notifyAll() method is used to unblock all the threads in waiting state.

### 17)What is deadlock?

Deadlock is a situation when two threads are waiting on each other to release a resource. Each thread waiting for a resource which is held by the other waiting thread.

# 20 Java Collections Interview Questions

In java, collection interview questions are mostly asked by the interviewers. Here is the list of mostly asked collections interview questions with answers.

### 1) What is the difference between ArrayList and Vector?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **Vector** |
| 1) | ArrayList is not synchronized. | Vector is synchronized. |
| 2) | ArrayList is not a legacy class. | Vector is a legacy class. |
| 3) | ArrayList increases its size by 50% of the array size. | Vector increases its size by doubling the array size. |

### 2) What is the difference between ArrayList and LinkedList?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because a lot of shifting is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |

### 3) What is the difference between Iterator and ListIterator?

Iterator traverses the elements in forward direction only whereas ListIterator traverses the elements in forward and backward direction.

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | Iterator traverses the elements in forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | Iterator can be used in List, Set and Queue. | ListIterator can be used in List only. |

### 4) What is the difference between Iterator and Enumeration?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |

### 5) What is the difference between List and Set?

List can contain duplicate elements whereas Set contains only unique elements.

### 6) What is the difference between HashSet and TreeSet?

HashSet maintains **no order** whereas TreeSet maintains **ascending order**.

### 7) What is the difference between Set and Map?

Set contains values only whereas Map contains key and values both.

### 8) What is the difference between HashSet and HashMap?

HashSet contains only values whereas HashMap contains entry(key,value). HashSet can be iterated but HashMap need to convert into Set to be iterated.

### 9) What is the difference between HashMap and TreeMap?

HashMap maintains **no order** but TreeMap maintains **ascending order**.

### 10) What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |

### 11) What is the difference between Collection and Collections?

Collection is an interface whereas Collections is a class. Collection interface provides normal functionality of data structure to List, Set and Queue. But, Collections class is to sort and synchronize collection elements.

### 12) What is the difference between Comparable and Comparator?

|  |  |  |
| --- | --- | --- |
| **No.** | **Comparable** | **Comparator** |
| 1) | Comparable provides only one sort of sequence. | Comparator provides multiple sort of sequences. |
| 2) | It provides one method named compareTo(). | It provides one method named compare(). |
| 3) | It is found in java.lang package. | it is found in java.util package. |
| 4) | If we implement Comparable interface, actual class is modified. | Actual class is not modified. |

### 13) What is the advantage of Properties file?

If you change the value in properties file, you don't need to recompile the java class. So, it makes the application easy to manage.

### 14) What does the hashCode() method?

The hashCode() method returns a hash code value (an integer number).

The hashCode() method returns the same integer number, if two keys (by calling equals() method) are same.

But, it is possible that two hash code numbers can have different or same keys.

### 15) Why we override equals() method?

The equals method is used to check whether two objects are same or not. It needs to be overridden if we want to check the objects based on property.

For example, Employee is a class that has 3 data members: id, name and salary. But, we want to check the equality of employee object on the basis of salary. Then, we need to override the equals() method.

### 16) How to synchronize List, Set and Map elements?

Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

|  |
| --- |
| public static List synchronizedList(List l){} |
| public static Set synchronizedSet(Set s){} |
| public static SortedSet synchronizedSortedSet(SortedSet s){} |
| public static Map synchronizedMap(Map m){} |
| public static SortedMap synchronizedSortedMap(SortedMap m){} |

### 17) What is the advantage of generic collection?

If we use generic class, we don't need typecasting. It is typesafe and checked at compile time.

### 18) What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value is known as hash-collision. Two different entries will be kept in a single hash bucket to avoid the collision.

### 19) What is the Dictionary class?

The Dictionary class provides the capability to store key-value pairs.

### 20) What is the default size of load factor in hashing based collection?

The default size of load factor is **0.75**. The default capacity is computed as initial capacity \* load factor. For example, 16 \* 0.75 = 12. So, 12 is the default capacity of Map.