1. **Question 1. What Is Oops?**

**Answer :**

OOPS is abbreviated as Object Oriented Programming system in which programs are considered as a collection of objects. Each object is nothing but an instance of a class.

1. **Question 2. Write Basic Concepts Of Oops?**

**Answer :**

Following are the concepts of OOPS and are as follows:.  
1.Abstraction.  
2.Encapsulation.  
3.Inheritance.  
4.Polymorphism.

1. **Question 3. What Is A Class?**

**Answer :**

A class is simply a representation of a type of object. It is the blueprint/ plan/ template that describe the details of an object.

1. **Question 4. What Is An Object?**

**Answer :**

Object is termed as an instance of a class, and it has its own state, behavior and identity.

1. **Question 5. What Is Encapsulation?**

**Answer :**

Encapsulation is an attribute of an object, and it contains all data which is hidden. That hidden data can be restricted to the members of that class.  
Levels are Public,Protected, Private, Internal and Protected Internal.

1. **Question 6. What Is Polymorphism?**

**Answer :**

Polymorphism is nothing butassigning behavior or value in a subclass to something that was already declared in the main class. Simply, polymorphism takes more than one form.

1. **Question 7. What Is Inheritance?**

**Answer :**

Inheritance is a concept where one class shares the structure and behavior defined in another class. Ifinheritance applied on one class is called Single Inheritance, and if it depends on multiple classes, then it is called multiple Inheritance.

1. **Question 8. What Are Manipulators?**

**Answer :**

Manipulators are the functions which can be used in conjunction with the insertion (<<) and extraction (>>) operators on an object. Examples are endl and setw.

1. **Question 9. Define A Constructor?**

**Answer :**

Constructor is a method used to initialize the state of an object, and it gets invoked at the time of object creation. Rules forconstructor are:.  
•Constructor Name should be same asclass name.  
•Constructor must have no return type.

[JavaServer Faces (JSF) Interview Questions](https://www.wisdomjobs.com/e-university/javaserver-faces-jsf-interview-questions.html)

1. **Question 10. Define Destructor?**

**Answer :**

Destructor is a method which is automatically called when the object ismade ofscope or destroyed. Destructor name is also same asclass name but with the tilde symbol before the name.

[Android Tutorial](https://www.wisdomjobs.com/e-university/android-tutorial-288.html)

1. **Question 11. What Is Inline Function?**

**Answer :**

Inline function is a technique used by the compilers and instructs to insert complete body of the function wherever that function is used in the program source code.

[JSON (JavaScript Object Notation) Interview Questions](https://www.wisdomjobs.com/e-university/json-javascript-object-notation-interview-questions-answers.html)

1. **Question 12. What Is A Virtual Function?**

**Answer :**

Virtual function is a member function ofclass and its functionality can be overridden in its derived class. This function can be implemented by using a keyword called virtual, and it can be given during function declaration.  
Virtual function can be achieved in C++, and it can be achieved in C Languageby using function pointers or pointers to function.

[Adv Java Interview Questions](https://www.wisdomjobs.com/e-university/adv-java-practice-tests-227-327247)

1. **Question 13. What Is Friend Function?**

**Answer :**

Friend function is a friend of a class that is allowed to access to Public, private or protected data in that same class. If the function is defined outside the class cannot access such information.  
Friend can be declared anywhere in the class declaration, and it cannot be affected by access control keywords like private, public or protected.

[JavaServer Faces (JSF) Tutorial](https://www.wisdomjobs.com/e-university/javaserver-faces-jsf-tutorial-1064.html)

1. **Question 14. What Is Function Overloading?**

**Answer :**

Function overloading is defined as a normal function, but it has the ability to perform different tasks. It allowscreation of several methods with the same name which differ from each other by type of input and output of the function.  
**Example**  
void add(int& a, int& b);  
void add(double& a, double& b);  
void add(struct bob& a, struct bob& b);

1. **Question 15. What Is Operator Overloading?**

**Answer :**

Operator overloading is a function where different operators are applied and depends on the arguments. Operator,-,\* can be used to pass through the function , and it has their own precedence to execute.  
**Example:**class complex {   
double real,   
imag; public: complex(double r, double i) : real(r),   
imag(i) {} complex operator+(complex a, complex b);   
complex operator\*(complex a, complex b);   
complex& operator=(complex a, complex b);  
}  
a=1.2, b=6

[Advanced C# Interview Questions](https://www.wisdomjobs.com/e-university/advanced-c-hash-interview-questions.html)

1. **Question 16. What Is An Abstract Class?**

**Answer :**

An abstract class is a class which cannot be instantiated. Creation of an object is not possible with abstract class , but it can be inherited. An abstract class can contain only Abstract method. Java allows only abstract method in abstract class while for other language it allows non-abstract method as well.

[JSON (JavaScript Object Notation) Tutorial](https://www.wisdomjobs.com/e-university/json-javascript-object-notation-tutorial-1662.html)

1. **Question 17. What Is A Ternary Operator?**

**Answer :**

Ternary operator is said to be an operator which takes three arguments. Arguments and results are of different data types , and it is depends on the function. Ternary operator is also called asconditional operator.

[Advanced C++ Interview Questions](https://www.wisdomjobs.com/e-university/advanced-c-plus-plus-interview-questions.html)

1. **Question 18. What Is The Use Of Finalize Method?**

**Answer :**

Finalize method helps to perform cleanup operations on the resources which are not currently used. Finalize method is protected , and it is accessible only through this class or by a derived class.

[J2EE Interview Questions](https://www.wisdomjobs.com/e-university/j2ee-interview-questions.html)

1. **Question 19. What Are Different Types Of Arguments?**

**Answer :**

A parameter is a variable used during the declaration of the function or subroutine and arguments are passed to the function , and it should match with the parameter defined. There are two types of Arguments.  
•Call by Value – Value passed will get modified only inside the function , and it returns the same value whatever it is passed it into the function.  
•Call by Reference – Value passed will get modified in both inside and outside the functions and it returns the same or different value.

[Object Oriented Analysis and Design Tutorial](https://www.wisdomjobs.com/e-university/object-oriented-analysis-and-design-tutorial-2107.html)

1. **Question 20. What Is Super Keyword?**

**Answer :**

Super keyword is used to invoke overridden method which overrides one of its superclass methods. This keyword allows to access overridden methods and also to access hidden members of the superclass.  
It also forwards a call from a constructor to a constructor in the superclass.

[Basic C Interview Questions](https://www.wisdomjobs.com/e-university/basic-c-interview-questions.html)

1. **Question 21. What Is Method Overriding?**

**Answer :**

Method overriding is a feature that allows sub class to provide implementation of a method that is already defined in the main class. This will overrides the implementation in the superclass by providing the same method name, same parameter and same return type.

1. **Question 22. What Is An Interface?**

**Answer :**

An interface is a collection of abstract method. If the class implements an inheritance, and then thereby inherits all the abstract methods of an interface.

1. **Question 23. What Is Exception Handling?**

**Answer :**

Exception is an event that occurs during the execution of a program. Exceptions can be of any type – Run time exception, Error exceptions. Those exceptions are handled properly through exception handling mechanism like try, catch and throw keywords.

[C# OOPS Interview Questions](https://www.wisdomjobs.com/e-university/c-oops-interview-questions.html)

1. **Question 24. What Are Tokens?**

**Answer :**

Token is recognized by a compiler and it cannot be broken down into component elements. Keywords, identifiers, constants, string literals and operators are examples of tokens.  
Even punctuation characters are also considered as tokens – Brackets, Commas, Braces and Parentheses.

[Android Interview Questions](https://www.wisdomjobs.com/e-university/android-interview-questions.html)

1. **Question 25. Difference Between Overloading And Overriding?**

**Answer :**

**Overloading** is static binding whereas Overriding is dynamic binding. Overloading is nothing but the same method with different arguments , and it may or may not return the same value in the same class itself.  
**Overriding** is the same method names with same arguments and return types associates with the class and its child class.

1. **Question 26. Difference Between Class And An Object?**

**Answer :**

An object is an instance of a class. Objects hold any information , but classes don’t have any information. Definition of properties and functions can be done at class and can be used by the object.Class can have sub-classes, and an object doesn’t have sub-objects.

[Asp Dot Net Mvc 4 Interview Questions](https://www.wisdomjobs.com/e-university/asp-dot-net-mvc-4-interview-questions.html)

1. **Question 27. What Is An Abstraction?**

**Answer :**

Abstraction is a good feature of OOPS , and it shows only the necessary details to the client of an object. Means, it shows only necessary details for an object, not the inner details of an object. Example – When you want to switch On television, it not necessary to show all the functions of TV. Whatever is required to switch on TV will be showed by using abstract class.

[JavaServer Faces (JSF) Interview Questions](https://www.wisdomjobs.com/e-university/javaserver-faces-jsf-interview-questions.html)

1. **Question 28. What Are Access Modifiers?**

**Answer :**

Access modifiers determine the scope of the method or variables that can be accessed from other various objects or classes. There are 5 types of access modifiers , and they are as follows:.  
•Private.  
•Protected.  
•Public.  
•Friend.  
•Protected Friend.

1. **Question 29. What Is Sealed Modifiers?**

**Answer :**

Sealed modifiers are the access modifiers where it cannot be inherited by the methods. Sealed modifiers can also be applied to properties, events and methods. This modifier cannot be applied to static members.

[Object Oriented Analysis and Design Interview Questions](https://www.wisdomjobs.com/e-university/object-oriented-analysis-and-design-interview-questions.html)

1. **Question 30. How Can We Call The Base Method Without Creating An Instance?**

**Answer :**

Yes, it is possible to call the base method without creating an instance. And that method should be,.Static method.Doing inheritance from that class.-Use Base Keyword from derived class.

1. **Question 31. What Is The Difference Between New And Override?**

**Answer :**

The new modifier instructs the compiler to use the new implementation instead of the base class function. Whereas, Override modifier helps to override the base class function.

1. **Question 32. What Are The Various Types Of Constructors?**

**Answer :**

There are three various types of constructors , and they are as follows:.

* + **Default Constructor** – With no parameters.
  + **Parametric Constructor** – With Parameters. Create a new instance of a class and also passing arguments simultaneously.
  + **Copy Constructor** – Which creates a new object as a copy of an existing object.

[Asp Dot Net Database Interview Questions](https://www.wisdomjobs.com/e-university/asp-dot-net-database-interview-questions.html)

1. **Question 33. What Is Early And Late Binding?**

**Answer :**

Early binding refers to assignment of values to variables during design time whereas late binding refers to assignment of values to variables during run time.

[JSON (JavaScript Object Notation) Interview Questions](https://www.wisdomjobs.com/e-university/json-javascript-object-notation-interview-questions-answers.html)

1. **Question 34. What Is ‘this’ Pointer?**

**Answer :**

THIS pointer refers to the current object of a class. THIS keyword is used as a pointer which differentiates between the current object with the global object. Basically, it refers to the current object.

1. **Question 35. What Is The Difference Between Structure And A Class?**

**Answer :**

Structure default access type is public , but class access type is private. A structure is used for grouping data whereas class can be used for grouping data and methods. Structures are exclusively used for dataand it doesn’t require strict validation , but classes are used to encapsulates and inherit data which requires strict validation.

1. **Question 36. What Is The Default Access Modifier In A Class?**

**Answer :**

The default access modifier of a class is Private by default.

[Advanced C# Interview Questions](https://www.wisdomjobs.com/e-university/advanced-c-hash-interview-questions.html)

1. **Question 37. What Is Pure Virtual Function?**

**Answer :**

A pure virtual function is a function which can be overridden in the derived classbut cannot be defined. A virtual function can be declared as Pure by using the operator =0.  
  
**Example:**Virtual void function1() // Virtual, Not pure  
Virtual void function2() = 0 //Pure virtual

1. **Question 38. What Are All The Operators That Cannot Be Overloaded?**

**Answer :**

Following are the operators that cannot be overloaded -.  
1.Scope Resolution (:: )  
2.Member Selection (.)  
3.Member selection through a pointer to function (.\*)

1. **Question 39. What Is Dynamic Or Run Time Polymorphism?**

**Answer :**

Dynamic or Run time polymorphism is also known as method overriding in which call to an overridden function is resolved during run time, not at the compile time. It means having two or more methods with the same name,same signature but with different implementation.

1. **Question 40. Do We Require Parameter For Constructors?**

**Answer :**

No, we do not require parameter for constructors.

[Advanced C++ Interview Questions](https://www.wisdomjobs.com/e-university/advanced-c-plus-plus-interview-questions.html)

1. **Question 41. What Is A Copy Constructor?**

**Answer :**

This is a special constructor for creating a new object as a copy of an existing object. There will be always only on copy constructor that can be either defined by the user or the system.

1. **Question 42. What Does The Keyword Virtual Represented In The Method Definition?**

**Answer :**

It means, we can override the method.

[Basic C Interview Questions](https://www.wisdomjobs.com/e-university/basic-c-interview-questions.html)

1. **Question 43. What Are Base Class, Sub Class And Super Class?**

**Answer :**

* + Base class is the most generalized class , and it is said to be a root class.
  + Sub class is a class that inherits from one or more base classes.
  + Super class is the parent class from which another class inherits.

1. **Question 44. What Is Static And Dynamic Binding?**

**Answer :**

Binding is nothing but the association of a name with the class. Static binding is a binding in which name can be associated with the class during compilation time , and it is also called as early Binding.  
Dynamic binding is a binding in which name can be associated with the class during execution time , and it is also called as Late Binding.

1. **Question 45. How Many Instances Can Be Created For An Abstract Class?**

**Answer :**

Zero instances will be created for an abstract class.

1. **Question 46. Which Keyword Can Be Used For Overloading?**

**Answer :**

Operator keyword is used for overloading.

1. **Question 47. What Is The Default Access Specifier In A Class Definition?**

**Answer :**

Private access specifier is used in a class definition.

1. **Question 48. Which Oops Concept Is Used As Reuse Mechanism?**

**Answer :**

Inheritance is the OOPS concept that can be used as reuse mechanism.

1. **Question 49. Which Oops Concept Exposes Only Necessary Information To The Calling Functions?**

**Answer :**

Data Hiding / Abstraction

1. **Question 50. What Are The Types Of Constructors?**

**Answer :**

Basically constructors are 5 types those are

* + Default Constructor
  + Parameterized Constructor
  + Copy Constructor
  + Static Constructor
  + Private Constructor

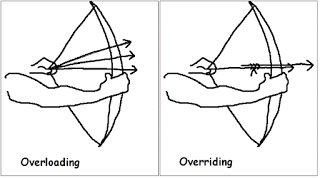
**Top 30 OOP Concept Interview Questions Answers in Java - Object Oriented Programming**

Java is an object-oriented programming language and you will see a lot of object-oriented programming concept questions on [Java interviews](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html). The classic questions like the difference between an interface and abstract class are always there but from the last couple of years more sophisticated questions based upon advanced design principles and patterns are also asked to check OOP knowledge of the candidate. Though, Object-oriented programming questions are more popular on [Java interviews for 1 to 3 years experienced](http://java67.blogspot.com/2014/07/21-frequently-asked-java-interview-questions-answers.html)programmers. It makes sense as well, as these are the programmers who must know the OOP basics like Abstraction, Inheritance, Composition, Class, Object, Interface, Encapsulation, etc.  
  
If you look for Java interview questions for 2 to 4 years experienced programmer, you will find lots of questions based upon OOP fundamentals like Inheritance and Encapsulation but as you gain more experience, you will see questions based on object-oriented analysis and design e.g. code a vending design machine or implement a coffeemaker in Java.  
  
These questions are more difficult and require not only a true understanding of OOP fundamentals but also about [SOLID design principles and patterns](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fprinciples-oo-design).

**OOPS Concept Interview Questions in Java**

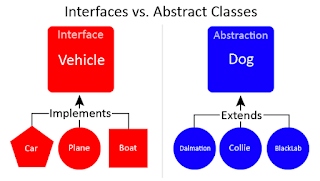
In this article, I am going to share with you some *OOPS concept based Java interview questions* that I have collected from friends and colleagues and they have seen in various Java interviews on different companies. They are mostly asked at first a few rounds like on screening round or on the [telephonic round](http://java67.blogspot.com/2015/03/top-40-core-java-interview-questions-answers-telephonic-round.html).  
  
  
If you are a senior Java developer then you already know answers to this question and I suggest you practice more on object-oriented analysis and design skill i.e. how to do code against a specification. If you are fresher and junior Java developer with 2 to 3 years experience then you must revise these questions, learn if you don't know to do well on your Java Job interviews.

**What is method overloading in OOP or Java?**([answer](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html))  
It's one of the oldest OOPS concept questions, I have seen it 10 years ago and still sees it now. When we have multiple methods with the same name but different functionality then it's called method overloading. For example. System.out.println() is overloaded as we have a 6 or 7 println() method each accepting a different type of parameter.  
  
  
**What is the method overriding in OOP or Java?** ([answer](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html))  
It's one of the magic of object-oriented programming where the method is chose based upon an object at runtime. In order for method overriding, we need Inheritance and Polymorphism, as we need a method with the same signature in both superclass and subclass. A call to such a method is resolved at runtime depending upon the actual object and not the type o variable. See the answer for a more detailed discussion.  
  
  
**What is the method of hiding in Java?**(answer)  
When you declare two static methods with same name and signature in both superclass and subclass then they hide each other i.e. a call to the method in the subclass will call the static method declared in that class and a call to the same method is superclass is resolved to the static method declared in the super-class.  
  
  
**Is Java a pure object-oriented language? if not why?** ([answer](http://java67.blogspot.com/2014/03/is-java-pure-object-oriented-programming-language.html))  
Java is not a pure object-oriented programming language e.g. there are many things you can do without objects e.g. static methods. Also, primitive variables are not objects in Java. See the answer for a more detailed explanation.  
  
  
  
**What are the rules of method overloading and overriding in Java?**([answer](http://java67.blogspot.sg/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html))  
One of the most important rules of method overloading in Java is that the method signature should be different i.e. either the number of arguments or the type of arguments. Simply changing the return type of two methods will not result in overloading, instead, the compiler will throw an error. On the other hand, method overriding has more rules e.g. name and return type must be the same, method signature should also be the same, the overloaded method cannot throw a higher exception, etc. See the answer for a full list of rules related to method overloading and overriding in Java.  
  
  
**The difference between method overloading and overriding?** ([answer](http://java67.blogspot.sg/2012/09/difference-between-overloading-vs-overriding-in-java.html))  
Several differences but the most important one is that method overloading is resolved at compile-time and method overriding is resolved at runtime. The compiler only used the class information for method overloading, but it needs to know the object to resolved overridden method calls. This diagram explains the difference quite well, though:

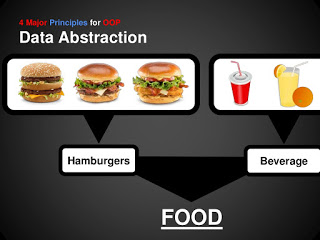
[](https://click.linksynergy.com/fs-bin/click?id=JVFxdTr9V80&subid=0&offerid=323058.1&type=10&tmpid=14538&RD_PARM1=https%3A%2F%2Fwww.udemy.com%2Fjava-the-complete-java-developer-course%2F)

**Can we overload a static method in Java?** ([answer](http://java67.blogspot.sg/2012/08/can-we-overload-static-method-in-java.html))  
Yes, you can overload a static method in Java. You can declare as many static methods of the same name as you wish provided all of them have different method signatures. See the answer for a more detailed explanation and code example.  
  
  
**Can we override the static method in Java?**([answer](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html))  
No, you cannot override a static method because it's not bounded to an object. Instead, static methods belong to a class and resolved at compile time using the type of reference variable. But, Yes, you can declare the same static method in a subclass, that will result in method hiding i.e. if you use the reference variable of type subclass then new method will be called, but if you use the reference variable of superclass than old method will be called.  
  
  
**Can we prevent overriding a method without using the final modifier?** (answer)  
Yes, you can prevent the method overriding in Java without using the final modifier. In fact, there are several ways to accomplish it e.g. you can mark the method private or static, those cannot be overridden.  
  
  
**Can we override a private method in Java?**([answer](http://java67.blogspot.sg/2013/08/can-we-override-private-method-in-java-inner-class.html))  
No, you cannot. Since the private method is only accessible and visible inside the class they are declared, it's not possible to override them in subclasses. Though, you can override them inside the inner class as they are accessible there.

**What is the covariant method overriding in Java?**([answer](http://javarevisited.blogspot.com/2014/03/covariant-method-overriding-of-java-5.html))  
In the covariant method overriding, the overriding method can return the subclass of the object returned by the original or overridden method. This concept was introduced in Java 1.5 (Tiger) version and it's very helpful in case the original method is returning general type like Object class, because, then by using the covariant method overriding you can return a more suitable object and prevent client-side typecasting. One of the practical use of this concept is when you override the clone() method in Java.  
  
  
  
**Can we change the return type of method to subclass while overriding?**(answer)  
Yes, you can, but only from Java 5 onward. This feature is known as covariant method overriding and it was introduced in JDK 5 release. This is immensely helpful if the original method return super-class like clone() method return java.lang.Object. By using this, you can directly return the actual type, preventing client-side type-casting of the result.  
  
  
**Can we change the argument list of an overriding method?**([answer](http://javarevisited.blogspot.com/2011/08/what-is-polymorphism-in-java-example.html))  
No, you cannot. The argument list is part of the method signature and both overriding and overridden methods must have the same signature.  
  
  
**Can we override a method that throws runtime exception without throws clause?** ([answer](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html))  
Yes, there is no restriction on unchecked exceptions while overriding. On the other hand, in the case of checked exception, an overriding exception cannot throw a checked exception which comes higher in type hierarchy e.g. if the original method is throwing IOException than the overriding method cannot throw java.lang.Exception or java.lang.Throwable.  
  
  
**How do you call a superclass version of an overriding method in a subclass? (**answer**)**  
You can call a superclass version of an overriding method in the subclass by using super keyword. For example to call the toString() method from java.lang.Object class, you can call super.toString().  
  
  
**Can we override a non-static method as static in Java?**(answer)  
Yes, you can override the non-static method in Java, no problem on them but it should not be private or final :)  
  
  
**Can we override the final method in Java?** ([answer](http://javarevisited.blogspot.com/2013/12/when-to-make-method-final-in-java.html))  
No, you cannot override a final method in Java, the final keyword with the method is to prevent method overriding. You use the final when you don't want subclass changing the logic of your method by overriding it due to security reasons. This is [why the String class is final in Java](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html). This concept is also used in the template design patterns where the template method is made final to prevent overriding.  
  
  
  
**Can we have a non-abstract method inside an interface?** (answer)  
From Java 8 onward you can have a non-abstract method inside interface, prior to that it was not allowed as all method was implicitly public abstract. From JDK 8, you can add static and default methods inside an interface.  
  
  
**What is the default method of Java 8?** ([answer](http://javarevisited.blogspot.com/2014/07/default-defender-or-extension-method-of-Java8-example-tutorial.html))  
The default method, also known as the extension method is new types of the method which you can add on the interface now. These method has implementation and intended to be used by default. By using this method, JDK 8 managed to provide common functionality related to [lambda expression](http://javarevisited.blogspot.com/2014/02/10-example-of-lambda-expressions-in-java8.html) and [stream API](http://javarevisited.blogspot.com/2014/03/2-examples-of-streams-with-Java8-collections.html)without breaking all the clients which implement their interfaces. If you look at Java 8 API documentation you will find several useful default methods on key Java interface like Iterator, Map, etc.  
  
  
**What is an abstract class in Java? (**[answer](http://java67.blogspot.sg/2014/06/why-abstract-class-is-important-in-java.html)**)**  
An abstract class is a class that is incomplete. You cannot create an instance of an abstract class in Java. They are provided to define default behavior and ensured that client of that class should adore to those contract which is defined inside the abstract class. In order to use it, you must extend and implement their abstract methods. BTW, in Java, a class can be abstract without specifying any abstract method.  
  
  
**What is an interface in Java? What is the real user of an interface?**([answer](http://java67.blogspot.sg/2014/02/what-is-actual-use-of-interface-in-java.html))  
Like an abstract class, the interface is also there to specify the contract of an API. It supports the OOP abstraction concept as it defines only abstract behavior. It will tell that your program will give output but how is left to implementors. The real use of the interface to define types to leverage Polymorphism. See the answer for a more detailed explanation and discussion.  
  
  
**The difference between Abstract class and interface?** ([answer](http://java67.blogspot.sg/2012/09/what-is-difference-between-interface-abstract-class-java.html))  
In Java, the key difference is that abstract class can contain a non-abstract method but the interface cannot, but from Java 8 onward interface can also contain static and default methods that are non-abstract. See the answer for a more detailed discussion as I have described a lot of points there.

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fpatterns-library)

**Can we make a class abstract without an abstract method?** ([answer](http://javarevisited.blogspot.com/2013/04/10-abstract-class-and-interface-interview-question-java-answers.html))  
Yes, just add abstract keyword on the class definition and your class will become abstract.  
  
  
**Can we make a class both final and abstract at the same time?** ([answer](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html))  
No, you cannot apply both final and abstract keyword at the class at the same time because they are exactly opposite of each other. A final class in Java cannot be extended and you cannot use an abstract class without extending and make it a concrete class. As per Java specification, the compiler will throw an error if you try to make a class abstract and final at the same time.  
  
  
**Can we overload or override the main method in Java?** ([answer](http://java67.blogspot.com/2015/06/can-you-overload-or-override-main-in-java.html))  
No, since main() is a static method, you can only overload it, you cannot override it because the static method is resolved at compile time without needing object information hence we cannot override the main method in Java.  
  
  
  
**What is the difference between Polymorphism, Overloading, and Overriding?** ([answer](http://java67.blogspot.sg/2012/10/difference-between-polymorphism-overloading-overriding-java.html))  
This is a slight tricky OOP concept question because Polymorphism is the real concept behind on both Overloading and Overriding. Overloading is compiled time Polymorphism and Overriding are Runtime Polymorphism.  
  
  
**Can an interface extend more than one interface in Java?**  
Yes, an interface can extend more than one interface in Java, it's perfectly valid.  
  
  
**Can a class extend more than one class in Java?**  
No, a class can only extend another class because Java doesn't support multiple inheritances but yes, it can implement multiple interfaces.  
  
  
**What is the difference between abstraction and polymorphism in Java?** ([answer](http://java67.blogspot.sg/2015/05/difference-between-abstraction-and.html))  
Abstraction generalizes the concept and Polymorphism allows you to use different implementation without changing your code. This diagram explains the abstraction quite well, though:

[](http://www.amazon.com/dp/0596008678/?tag=javamysqlanta-20)

**Object-Oriented design principle and pattern Interview Questions**

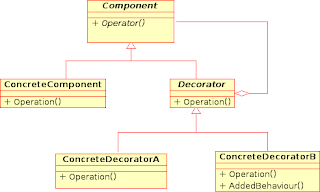
Now let's see some OOPS concept questions based on the SOLID design principles and GOF design patterns that take advantage of the OOPS concept discussed here.

**What problem is solved by the Strategy pattern in Java?** ([answer](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html))

Strategy pattern allows you to introduce a new algorithm or new strategy without changing the code which uses that algorithm. For example, the Collections.sort() method which sorts the list of the object uses the Strategy pattern to compare objects. Since every object uses a different comparison strategy you can compare various objects differently without changing the sort method.

**Which OOP concept Decorator design Pattern is based upon?**([answer](http://java67.blogspot.com/2013/07/decorator-design-pattern-in-java-real-life-example-tutorial.html))

The decorator pattern takes advantage of Composition to provide new features without modifying the original class. A very good to-the-point question for the telephonic round. This is quite clear from the UML diagram of the Decorator pattern, as you can see the Component is associated with a Decorator.

[](https://2.bp.blogspot.com/-jnzC4Kx48Oc/Vmg-K8LVjqI/AAAAAAAAES8/sYAEghzm688/s1600/Decorator%2BDesign%2BPattern%2Bin%2BJava%2BUML.png)

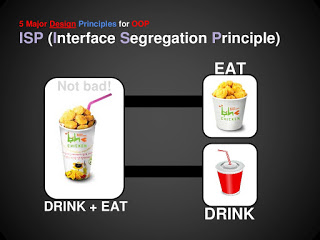
**When to use the Singleton design pattern in Java?** ([answer](http://java67.blogspot.com/2012/08/what-is-singleton-pattern-in-java.html))  
When you need just one instance of a class and want that to be globally available then you can use the [Singleton pattern](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html). It's not free of cost though because it increases the coupling between classes and makes them hard to test. This is one of the oldest design pattern questions from Java interviews. Please see the answer for a more detailed discussion.

**What is the difference between State and Strategy Patterns?**([answer](http://javarevisited.blogspot.com/2014/04/difference-between-state-and-strategy-design-pattern-java.html))  
Though the structure or class diagram of State and Strategy pattern is the same, their intent is completely different. The state pattern is used to do something specific depending upon state while [Strategy](http://java67.blogspot.com/2014/12/strategy-pattern-in-java-with-sample.html) allows you to switch between algorithms without changing the code which uses it.

**What is the difference between Association, Aggregation, and Composition in OOP?** ([answer](http://javarevisited.blogspot.com/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))  
When an object is related to another object is called association. It has two forms, aggregation, and composition. the former is the loose form of association where the related object can survive individually while later is a stronger form of association where a related object cannot survive individually. For example, the city is an aggregation of people but is the composition of body parts.

**What is the difference between Decorator, Proxy and Adapter pattern in Java?**([answer](http://javarevisited.blogspot.com/2015/01/adapter-vs-decorator-vs-facade-vs-proxy-pattern-java.html))  
Again they look similar because their structure or class diagram is very similar but their intent is quite different. The Decorator adds additional functionality without touching the class, Proxy provides access control and Adapter is used to make two incompatible interfaces work together.

**What is the 5 objects oriented design principle from SOLID?**([answer](http://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html))  
SOLID is the term given by Uncle Bob in his classic book, the [Clean Code](http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882?tag=javamysqlanta-20), one of the must-read books for programmers. In SOLID each character stands for one design principle:  
S for Single Responsibility Principle  
O for Open closed design principle  
L for Liskov substitution principle  
I for Interface segregation principle  
D for Dependency inversion principle

[](https://pluralsight.pxf.io/c/1193463/424552/7490?u=https%3A%2F%2Fwww.pluralsight.com%2Fcourses%2Fpatterns-library)

**What is the difference between Composition and Inheritance in OOP?**([answer](http://javarevisited.blogspot.sg/2015/06/difference-between-inheritance-and-Composition-in-Java-OOP.html))

This is another great OOPS concept question because it tests what matters, both of them are very important from a class design perspective. Though both Composition and Inheritance allows you to reuse code, formerly is more flexible than later. Composition allows the class to get an additional feature at runtime, but Inheritance is static. You can not change the feature at runtime by substitution of a new implementation. See the answer for a more detailed discussion.

That's all about in this list of **object-oriented programming or OOPS concept interview questions**. We have seen questions from various OOPS concepts like Abstraction, Encapsulation, Inheritance, Composition, Aggregation, Association, Class, Object, and Interface, etc.  
  
We have also seen questions from Object-oriented design principles also known as SOLID principles and GOF design patterns like Strategy, State, and Factory, which are based upon both the object-oriented programming concept and OOP design principle.  
  
But, as I said, if you are a senior Java developer then you focus more on object-oriented analysis and design and learn how to code against a requirement using all your OOP knowledge. You can also read [Cracking the Coding Interview, 6th Edition](http://www.amazon.com/dp/098478280X/?tag=javamysqlanta-20) to for more Object Oriented Programming questions.

Read more: <https://www.java67.com/2015/12/top-30-oops-concept-interview-questions-answers-java.html#ixzz6Z84oYmxN>

### **What is the super class of every class in Java?**

Every class in java is a sub class of the class Object. When we create a class we inherit all the methods and properties of Object class. Let’s look at a simple example:In the example below - toString, hashCode and clone methods for String class are inherited from Object class.

String str = "Testing";

System.out.println(str.toString());

System.out.println(str.hashCode());

System.out.println(str.clone());

if(str instanceof Object){

System.out.println("I extend Object");//Will be printed

}

### **Can super class reference variable can hold an object of sub class?**

Yes. Look at the example below:

Actor reference variables actor1, actor2 hold the reference of objects of sub classes of Animal, Comedian and Hero.

Since object is super class of all classes, an Object reference variable can also hold an instance of any class.

//Object is super class of all java classes

Object object = new Hero();

public class Actor {

public void act(){

System.out.println("Act");

};

}

//IS-A relationship. Hero is-a Actor

public class Hero extends Actor {

public void fight(){

System.out.println("fight");

};

}

//IS-A relationship. Comedian is-a Actor

public class Comedian extends Actor {

public void performComedy(){

System.out.println("Comedy");

};

}

Actor actor1 = new Comedian();

Actor actor2 = new Hero();

### **Is Multiple Inheritance allowed in Java?**

Multiple Inheritance results in a number of complexities. Java does not support Multiple Inheritance.

class Dog extends Animal, Pet { //COMPILER ERROR

}

However, we can create an Inheritance Chain

class Pet extends Animal {

}

class Dog extends Pet {

}

### **What is Polymorphism?**

Refer to this video(<https://www.youtube.com/watch?v=t8PTatUXtpI>) for a clear explanation of polymorphism.

Polymorphism is defined as “Same Code” giving “Different Behavior”. Let’s look at an example.

Let’s define an Animal class with a method shout.

public class Animal {

public String shout() {

return "Don't Know!";

}

}

Let’s create two new sub classes of Animal overriding the existing shout method in Animal.

class Cat extends Animal {

public String shout() {

return "Meow Meow";

}

}

class Dog extends Animal {

public String shout() {

return "BOW BOW";

}

public void run(){

}

}

Look at the code below. An instance of Animal class is created. shout method is called.

Animal animal1 = new Animal();

System.out.println(

animal1.shout()); //Don't Know!

Look at the code below. An instance of Dog class is created and store in a reference variable of type Animal.

Animal animal2 = new Dog();

//Reference variable type => Animal

//Object referred to => Dog

//Dog's bark method is called.

System.out.println(

animal2.shout()); //BOW BOW

When shout method is called on animal2, it invokes the shout method in Dog class (type of the object pointed to by reference variable animal2).

Even though dog has a method run, it cannot be invoked using super class reference variable.

//animal2.run();//COMPILE ERROR

### **What is the use of instanceof Operator in Java?**

instanceof operator checks if an object is of a particular type. Let us consider the following class and interface declarations:

class SuperClass {

};

class SubClass extends SuperClass {

};

interface Interface {

};

class SuperClassImplementingInteface implements Interface {

};

class SubClass2 extends SuperClassImplementingInteface {

};

class SomeOtherClass {

};

Let’s consider the code below. We create a few instances of the classes declared above.

SubClass subClass = new SubClass();

Object subClassObj = new SubClass();

SubClass2 subClass2 = new SubClass2();

SomeOtherClass someOtherClass = new SomeOtherClass();

Let’s now run instanceof operator on the different instances created earlier.

System.out.println(subClass instanceof SubClass);//true

System.out.println(subClass instanceof SuperClass);//true

System.out.println(subClassObj instanceof SuperClass);//true

System.out.println(subClass2

instanceof SuperClassImplementingInteface);//true

instanceof can be used with interfaces as well. Since Super Class implements the interface, below code prints true.

System.out.println(subClass2

instanceof Interface);//true

If the type compared is unrelated to the object, a compilation error occurs.

//System.out.println(subClass

// instanceof SomeOtherClass);//Compiler Error

Object referred by subClassObj(SubClass)- NOT of type SomeOtherClass

System.out.println(subClassObj instanceof SomeOtherClass);//false

### **What is an Abstract Class?**

An abstract class (Video Link - <https://www.youtube.com/watch?v=j3GLUcdlz1w>) is a class that cannot be instantiated, but must be inherited from. An abstract class may be fully implemented, but is more usually partially implemented or not implemented at all, thereby encapsulating common functionality for inherited classes.

In code below ”AbstractClassExample ex = new AbstractClassExample();” gives a compilation error because AbstractClassExample is declared with keyword abstract.

public abstract class AbstractClassExample {

public static void main(String[] args) {

//An abstract class cannot be instantiated

//Below line gives compilation error if uncommented

//AbstractClassExample ex = new AbstractClassExample();

}

}

### **How do you define an abstract method?**

An Abstract method does not contain body. An abstract method does not have any implementation. The implementation of an abstract method should be provided in an over-riding method in a sub class.

//Abstract Class can contain 0 or more abstract methods

//Abstract method does not have a body

abstract void abstractMethod1();

abstract void abstractMethod2();

Abstract method can be declared only in Abstract Class. In the example below, abstractMethod() gives a compiler error because NormalClass is not abstract.

class NormalClass{

abstract void abstractMethod();//COMPILER ERROR

}

### **What is Coupling?**

Coupling is a measure of how much a class is dependent on other classes. There should minimal dependencies between classes. So, we should always aim for low coupling between classes.

##### **Coupling Example Problem**

Consider the example below:

class ShoppingCartEntry {

public float price;

public int quantity;

}

class ShoppingCart {

public ShoppingCartEntry[] items;

}

class Order {

private ShoppingCart cart;

private float salesTax;

public Order(ShoppingCart cart, float salesTax) {

this.cart = cart;

this.salesTax = salesTax;

}

// This method know the internal details of ShoppingCartEntry and

// ShoppingCart classes. If there is any change in any of those

// classes, this method also needs to change.

public float orderTotalPrice() {

float cartTotalPrice = 0;

for (int i = 0; i < cart.items.length; i++) {

cartTotalPrice += cart.items[i].price

\* cart.items[i].quantity;

}

cartTotalPrice += cartTotalPrice \* salesTax;

return cartTotalPrice;

}

}

Method orderTotalPrice in Order class is coupled heavily with ShoppingCartEntry and ShoppingCart classes. It uses different properties (items, price, quantity) from these classes. If any of these properties change, orderTotalPrice will also change. This is not good for Maintenance.

##### **Solution**

Consider a better implementation with lesser coupling between classes below: In this implementation, changes in ShoppingCartEntry or CartContents might not affect Order class at all.

class ShoppingCartEntry

{

float price;

int quantity;

public float getTotalPrice()

{

return price \* quantity;

}

}

class CartContents

{

ShoppingCartEntry[] items;

public float getTotalPrice()

{

float totalPrice = 0;

for (ShoppingCartEntry item:items)

{

totalPrice += item.getTotalPrice();

}

return totalPrice;

}

}

class Order

{

private CartContents cart;

private float salesTax;

public Order(CartContents cart, float salesTax)

{

this.cart = cart;

this.salesTax = salesTax;

}

public float totalPrice()

{

return cart.getTotalPrice() \* (1.0f + salesTax);

}

}

### **What is Cohesion?**

Cohesion (Video Link - <https://www.youtube.com/watch?v=BkcQWoF5124>) is a measure of how related the responsibilities of a class are. A class must be highly cohesive i.e. its responsibilities (methods) should be highly related to one another.

##### **Example Problem**

Example class below is downloading from internet, parsing data and storing data to database. The responsibilities of this class are not really related. This is not cohesive class.

class DownloadAndStore{

void downloadFromInternet(){

}

void parseData(){

}

void storeIntoDatabase(){

}

void doEverything(){

downloadFromInternet();

parseData();

storeIntoDatabase();

}

}

##### **Solution**

This is a better way of approaching the problem. Different classes have their own responsibilities.

class InternetDownloader {

public void downloadFromInternet() {

}

}

class DataParser {

public void parseData() {

}

}

class DatabaseStorer {

public void storeIntoDatabase() {

}

}

class DownloadAndStore {

void doEverything() {

new InternetDownloader().downloadFromInternet();

new DataParser().parseData();

new DatabaseStorer().storeIntoDatabase();

}

}

### **What is Encapsulation?**

Encapsulation is “hiding the implementation of a Class behind a well defined interface”. Encapsulation helps us to change implementation of a class without breaking other code.

##### **Approach 1**

In this approach we create a public variable score. The main method directly accesses the score variable, updates it.

public class CricketScorer {

public int score;

}

Let’s use the CricketScorer class.

public static void main(String[] args) {

CricketScorer scorer = new CricketScorer();

scorer.score = scorer.score + 4;

}

##### **Approach 2**

In this approach, we make score as private and access value through get and set methods. However, the logic of adding 4 to the score is performed in the main method.

public class CricketScorer {

private int score;

public int getScore() {

return score;

}

public void setScore(int score) {

this.score = score;

}

}

Let’s use the CricketScorer class.

public static void main(String[] args) {

CricketScorer scorer = new CricketScorer();

int score = scorer.getScore();

scorer.setScore(score + 4);

}

##### **Approach 3**

In this approach - For better encapsulation, the logic of doing the four operation also is moved to the CricketScorer class.

public class CricketScorer {

private int score;

public void four() {

score += 4;

}

}

Let’s use the CricketScorer class.

public static void main(String[] args) {

CricketScorer scorer = new CricketScorer();

scorer.four();

}

##### **Description**

In terms of encapsulation, Approach 3 is the best approach. In Approach 3, the user of scorer class does not even know that there is a variable called score. Implementation of Scorer can change without changing other classes using Scorer.

### **What is Method Overloading?**

A method having the same name as another method (in same class or a sub class) but having different parameters is called an Overloaded Method.

##### **Example 1**

doIt method is overloaded in the below example:

class Foo{

public void doIt(int number){

}

public void doIt(String string){

}

}

##### **Example 2**

Overloading can also be done from a sub class.

class Bar extends Foo{

public void doIt(float number){

}

}

### **What is Method Overriding?**

Creating a Sub Class Method with same signature as that of a method in SuperClass is called Method Overriding.

##### **Method Overriding Example 1:**

Let’s define an Animal class with a method shout.

public class Animal {

public String bark() {

return "Don't Know!";

}

}

Let’s create a sub class of Animal – Cat - overriding the existing shout method in Animal. bark method in Cat class is overriding the bark method in Animal class.

class Cat extends Animal {

public String bark() {

return "Meow Meow";

}

}

### **What is an Inner Class?**

Inner Classes are classes which are declared inside other classes. Consider the following example:

class OuterClass {

public class InnerClass {

}

public static class StaticNestedClass {

}

}

### **What is a Static Inner Class?**

A class declared directly inside another class and declared as static. In the example above, class name StaticNestedClass is a static inner class.

### **Can you create an inner class inside a method?**

Yes. An inner class can be declared directly inside a method. In the example below, class name MethodLocalInnerClass is a method inner class.

class OuterClass {

public void exampleMethod() {

class MethodLocalInnerClass {

};

}

}