javaTcenter

Search your favourite tutorials... Search

The largest *Interview Solution Library* on the web

Interview Questions

- Core Java Questions
- Spring Questions
- JSF Questions
- Hibernate Questions
- Struts Questions
- Jsp Questions
- Servlets Questions
- JDBC Questions
- EJB Questions
- XML Questions
- AJAX Questions
- Webservices Questions
- SOA Questions
- Android Questions
- JMS Questions
- webMethods Questions
- Tibco Questions
- BREW Questions
- Esb Questions
- Core Java interview Questions
- Spring interview Questions
- JSF interview Questions
- Hibernate interview Questions
- Struts interview Questions
- Jsp interview Questions
- Servlets interview Questions
- JDBC interview Questions
- EJB interview Questions
- XML interview Questions
- Mule ESB interview Questions
-
- Jasper Report interview Questions
- AJAX interview Questions

« Previous | **1** | 2 | 3 | 4 | 5 | Next »

1. What are the principle concepts of OOPS?

There are four principle concepts upon which object oriented design and programming rest. They are:

- Abstraction
- Polymorphism
- Inheritance
- Encapsulation

(i.e. easily remembered as A-PIE).

2. What is Abstraction?

Abstraction refers to the act of representing essential features without including the background details or explanations.

3. What is Encapsulation?

Encapsulation is a technique used for hiding the properties and behaviors of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

4. What is the difference between abstraction and encapsulation?

Abstraction focuses on the outside view of an object (i.e. the interface)
Encapsulation (information hiding) prevents clients from seeing it’s inside view, where the behavior of the abstraction is implemented.
Abstraction solves the problem in the design side while Encapsulation is the Implementation.
Encapsulation is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

5. What is Inheritance?

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- A class that is inherited is called a superclass.
- The class that does the inheriting is called a subclass.
- Inheritance is done by using the keyword extends.
- The two most common reasons to use inheritance are:
  - To promote code reuse
  - To use polymorphism

6. What is Polymorphism?

Polymorphism is briefly described as "one interface, many implementations." Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

7. How does Java implement polymorphism?

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

- In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
- In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

8. Explain the different forms of Polymorphism.

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. Runtime time polymorphism is done using inheritance and interface.

Note: From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:

- Method overloading
- Method overriding through inheritance
- Method overriding through the Java interface

9. What is runtime polymorphism or dynamic method dispatch?

In Java, 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 superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

10. What is Dynamic Binding?

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

11. What is method overloading?

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.

Note:

- Overloaded methods MUST change the argument list
- Overloaded methods CAN change the return type
- Overloaded methods CAN change the access modifier
- Overloaded methods CAN declare new or broader checked exceptions
- A method can be overloaded in the same class or in a subclass

12. What is method overriding?

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. The key benefit of overriding is the ability to define behavior that’s specific to a particular subclass type.

Note:

- The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can't override a method marked public and make it protected).

- You cannot override a method marked final
- You cannot override a method marked static

13. What are the differences between method overloading and method overriding?

	Overloaded Method	**Overridden Method**
Arguments	Must change	Must not change
Return type	Can change	Can't change except for covariant returns
Exceptions	Can change	Can reduce or eliminate. Must not throw new or broader checked exceptions
Access	Can change	Must not make more restrictive (can be less restrictive)
Invocation	Reference type determines which overloaded version is selected. Happens at compile time.	Object type determines which method is selected. Happens at runtime.

14. Can overloaded methods be override too?

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

15. Is it possible to override the main method?

NO, because main is a static method. A static method can't be overridden in Java.

16. How to invoke a superclass version of an Overridden method?

To invoke a superclass method that has been overridden in a subclass, you must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass' implementation of the method.

```
// From subclass


super.overriddenMethod();
```

17. What is super?

super is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.

Note:

- You can only go back one level.
- In the constructor, if you use super(), it must be the very first code, and you cannot access any this.xxx variables or methods to compute its parameters.

18. How do you prevent a method from being overridden?

To prevent a specific method from being overridden in a subclass, use the final modifier on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

```
public final void
exampleMethod() {
// Method statements
}
```

19. What is an Interface?

An interface is a description of a set of methods that conforming implementing classes must have.

Note:

- You can’t mark an interface as final.
- Interface variables must be static.
- An Interface cannot extend anything but another interfaces.

20. Can we instantiate an interface?

You can’t instantiate an interface directly, but you can instantiate a class that implements an interface.

21. Can we create an object for an interface?

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so you must write a class that implements the interface and fulfill all the methods defined in it.

22. Do interfaces have member variables?

Interfaces may have member variables, but these are implicitly public, static, and final- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

23. What modifiers are allowed for methods in an Interface?

Only public and abstract modifiers are allowed for methods in interfaces

24. What is a marker interface?

Marker interfaces are those which do not declare any required methods, but signify their compatibility with certain operations. The java.io.Serializable interface and Cloneable are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

25. What is an abstract class?

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.

Note:

- If even a single method is abstract, the whole class must be declared abstract.
- Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.
- You can’t mark a class as both abstract and final.

26. Can we instantiate an abstract class?

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

27. What are the differences between Interface and Abstract class?

Abstract Class	Interfaces
An abstract class can provide complete, default code and/or just the details that have to be overridden.	An interface cannot provide any code at all,just the signature.
In case of abstract class, a class may extend only one abstract class	A Class may implement several interfaces.
An abstract class can have non-abstract methods.	All methods of an Interface are abstract.
An abstract class can have instance variables.	An Interface cannot have instance variables.
An abstract class can have any visibility: public, private, protected.	An Interface visibility must be public (or) none.
If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly.	If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method.
An abstract class can contain constructors .	An Interface cannot contain constructors .
Abstract classes are fast.	Interfaces are slow as it requires extra indirection to find corresponding method in the actual class.

28. When should I use abstract classes and when should I use interfaces?

Use Interfaces when…

- You see that something in your design will change frequently.
- If various implementations only share method signatures then it is better to use Interfaces.
- you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface

Use Abstract Class when…

- If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
- When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
- Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

29. When you declare a method as abstract, can other nonabstract methods access it?

Yes, other nonabstract methods can access a method that you declare as abstract.

30. Can there be an abstract class with no abstract methods in it?

Yes, there can be an abstract class without abstract methods.

« Previous | 1 | 2 | 3 | 4 | 5 | Next »

1 Comment Sort by Oldest

Add a comment...

Aarif Mohammad
beat the heat of interview in java,j2ee........
Like · Reply · 2 · 4y

Facebook Comments Plugin