1. **What is Busy Spinning? Why Should You Use It in Java?**

   Busy spinning is a *waiting strategy*, in which a thread just waits in a loop, without releasing the CPU for going to sleep. This is a very advanced and specialized waiting strategy used in the high-frequency trading application when wait time between two messages is very minimal.

   By not releasing the CPU or suspending the thread, your thread retains all the cached data and instruction, which may be lost if the thread was suspended and resumed back in a different core of CPU.

2. **What is Immutable class in Java?**

   Immutable classes are those classes, whose object cannot be modified once created; it means any modification on immutable object will result in another immutable object. Best example of immutable objects is ***String*** and ***StringBuffer.*** Since ***String*** is immutable class, any change on existing string object will result in another string e.g. replacing a character into String, creating substring from String, all result in new objects. While in case of mutable object like ***StringBuffer***, any modification is done on object itself and no new objects are created.

3. **Why Should You Make an Object Immutable?**

   Well, Immutability offers several advantages including thread-safety, ability to cache and result in more readable multithreading code.

4. **Why String is Immutable or Final in Java?**

   The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, String was made final so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc. by extending and overriding behaviors. Another reason of why String class is immutable could die due to ***HashMap***.

5. **What is Thread in Java?**

   The thread is an independent path of execution. It's way to take advantage of multiple CPU available in a machine. By employing multiple threads you can speed up CPU bound task. For example, if one thread takes 100 milliseconds to do a job, you can use 10 threads to reduce that task into 10 milliseconds. Java provides excellent support for multithreading at the language level, and it's also one of the strong selling points.

6. **How do you implement Thread in Java?**

   At the language level, there are two ways to implement Thread in Java. An instance of java.lang.Thread represent a thread but it needs a task to execute, which is an instance of interface java.lang.Runnable. Since Thread class itself implement Runnable, you can override run() method either by extending Thread class or just implementing Runnable interface.

7. **When to use Runnable vs Thread in Java?**

   This is a follow-up of previous multi-threading interview question. As we know we can implement thread either by extending **Thread** class or implementing ***Runnable*** interface, the question arises, which one is better and when to use one? This question will be easy to answer if you know that Java programming language doesn't support multiple inheritances of class, but it allows you to implement multiple interfaces. Which means, it's better to implement Runnable then extends Thread if you also want to extend another class e.g. Canvas or CommandListener? For more points and discussion you can also refer this post.

8. **What is the difference between start() and run() method of Thread class?**

   One of trick Java question from early days, but still good enough to differentiate between shallow understanding of Java threading model start() method is used to start newly created thread, while start() internally calls run() method, there is difference calling run() method directly. When you invoke run() as normal method, it's called in the same thread, no new thread is started, which is the case when you call start() method. Read this answer for much more detailed discussion.

9. **What is the difference between Runnable and Callable in Java?**

   Both Runnable and Callable represent task which is intended to be executed in a separate thread. Runnable is there from JDK 1.0 while Callable was added on JDK 1.5. Main difference between these two is that Callable's call() method can return value and throw Exception, which was not possible with Runnable's run() method. Callable return Future object, which can hold the result of computation.

10. **What is method hiding in Java?**

    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.

11. **Is Java a pure object oriented language? If not why?**

    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.

12. **What are rules of method overloading and overriding in Java?**

    One of the most important rule of method overloading in Java is that 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 compiler will throw an error. On the other hand, method overriding has more rules e.g. name and return type must be same, method signature should also be same, the overloaded method cannot throw a higher exception etc.

13. **The difference between method overloading and overriding?**

    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 object to resolved overridden method calls.

14. **Can we overload a static method in Java?**

    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.

15. **Can we override static method in Java?**

    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 reference variable of type subclass then new method will be called, but if you use reference variable of superclass than old method will be called.

16. **Can we prevent overriding a method without using the final modifier?**

    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.

17. **Can we override a private method in Java?**

    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.

18. **What is covariant method overriding in Java?**

    In covariant method overriding, the overriding method can return the subclass of the object returned by original or overridden method. This concept was introduced in Java 1.5 (Tiger) version and it's very helpful in case original method is returning general type like Object class, because, then by using covariant method overriding you can return more suitable object and prevent client side type casting. One of the practical use of this concept is in when you override the clone() method in Java.

19. **Can we change the return type of method to subclass while overriding?**

    Yes, you can, but only from Java 5 onward. This feature is known as co-variant method overriding and it was introduced in JDK 5 release. This is immensely helpful if original method return super-class e.g. clone() method return java.lang.Object. By

using this, you can directly return the actual type, preventing client-side type casting of the result.

20. **Can we change the argument list of an overriding method?**

    No, you cannot. The argument list is part of the method signature and both overriding and overridden method must have the same signature.

21. **Can we override a method which throws runtime exception without throws clause?**

    Yes, there is no restriction on unchecked exception 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 original method is throwing IOException than overriding method cannot throw java.lang.Exception or java.lang.Throwable.

22. **How do you call superclass version of an overriding method in sub class?**

    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().

23. **Can we override a non-static method as static in Java?**

    Yes, you can override the non-static method in Java, no problem on them but it should not be private or final.

24. **Can we override the final method in Java?**

    No, you cannot override a final method in Java; final keyword with the method is to prevent method overriding. You use final when you don't want subclass changing the logic of your method by overriding it due to security reason. This is why String class is final in Java. This concept is also used in template design pattern where template method is made final to prevent overriding.

25. **Can we have a non-abstract method inside interface?**

    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 method inside an interface.

26. **What is the default method of Java 8?**

    Default method, also known as extension method is new types of the method which you can add on the interface now. These methods have implementation and intended to be used by default. By using this method, JDK 8 managed to provide common functionality related to lambda expression and stream API without breaking all the clients which implement their interfaces. If you look Java 8 API documentation you will find several useful default methods on key Java interface like Iterator, Map etc.

27. **What is an abstract class in Java?**

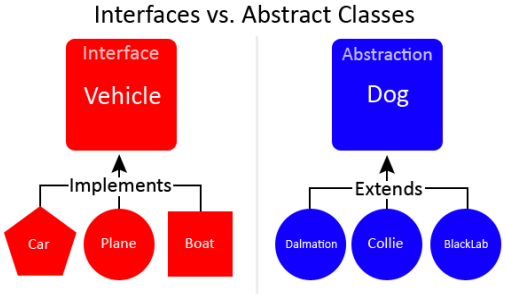
    An abstract class is a class which is incomplete. You cannot create an instance of an abstract class in Java. They are provided to define default behavior and ensured that client of that class should adore to those contract which are defined inside the abstract class. In order to use it, you must extend and implement their abstract methods. BTW, in Java a class can be abstract without specifying any abstract method.

28. **What is an interface in Java? What is the real user of an interface?**

    Like an abstract class, the interface is also there to specify the contract of an API. It supports OOP abstraction concept as it defines only abstract behavior. It will tell that your program will give output but how is left to implementers. The real use of the interface to define types to leverage Polymorphism. See the answer for more detailed explanation and discussion.

29. **The difference between Abstract class and interface?**

    In Java, the key difference is that abstract class can contain non-abstract method but the interface cannot, but from Java 8 onward interface can also contain static and default methods which are non-abstract.



30. **Can we make a class abstract without an abstract method?**
    Yes, just add abstract keyword on the class definition and your class will become abstract.

31. **Can we make a class both final and abstract at the same time?**
    No, you cannot apply both final and abstract keyword at the class 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 making 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.

32. **Can we overload or override the main method in Java?**
    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.

33. **What is the difference between Polymorphism, Overloading, and Overriding?**
    This is slight tricky OOP concept question because Polymorphism is the real concept behind on both Overloading and Overriding. Overloading is compile time Polymorphism and Overriding are runtime Polymorphism.

34. **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.

35. **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.

36. **What is the difference between abstraction and polymorphism in Java?**
    Abstraction generalizes the concept and Polymorphism allows you to use different implementation without changing your code. This diagram explains the abstraction quite well, though:

Object Oriented design principle and pattern Interview Questions

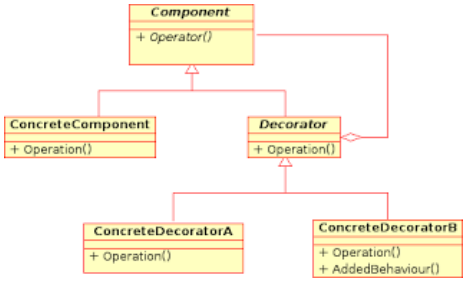
---

1. What problem is solved by Strategy pattern in Java?

Strategy pattern allows you to introduce 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 Strategy pattern to compare object. Since every object uses different comparison strategy you can compare various object differently without changing sort method.

2. Which OOP concept Decorator design Pattern is based upon?

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 UML diagram of Decorator pattern, as you can see the Component is associated with Decorator.



3. When to use Singleton design pattern in Java?

When you need just one instance of a class and wants that to be globally available then you can use Singleton pattern. It's not free of cost though because it increase coupling between classes and make them hard to test.

4. What is the difference between State and Strategy Pattern?

Though the structure or class diagram of State and Strategy pattern is same, their intent is completely different. State pattern is used to do something specific depending upon state while Strategy allows you to switch between algorithms without changing the code which uses it.

5. What is the difference between Association, Aggregation, and Composition in OOP?

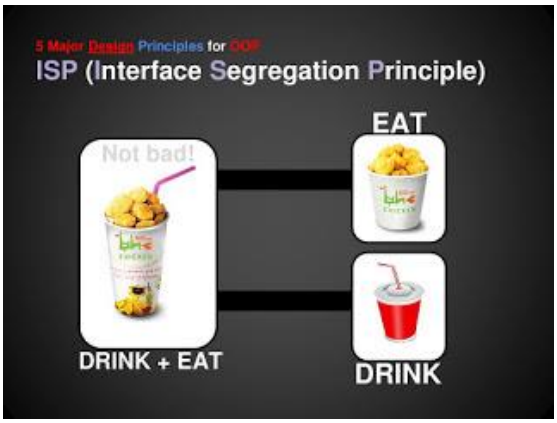
When an object is related to another object it called association. It has two forms, aggregation, and composition. The former is the loose form of association where the related object can survive individual while later is a stronger form of association where a related object cannot survive individually. For example, the city is an aggregation of people but is the composition of body parts.

6. What is the difference between Decorator, Proxy and Adapter pattern in Java?

Again they look similar because their structure or class diagram is very similar but their intent is quite different. Decorator adds additional functionality without touching the class, Proxy provides access control and Adapter is used to make two incompatible interfaces work together.

7. What is the 5 objects oriented design principle from SOLID?

SOLID is the term given by Uncle Bob in his classic book, the Clean Code, 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



8. What is the difference between Composition and Inheritance in OOP?

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