**1. When to use Strategy Design Pattern in Java?**  
Strategy pattern in quite useful for implementing set of related algorithms e.g. compression algorithms, filtering strategies etc. Strategy design pattern allows you to create Context classes, which uses Strategy implementation classes for applying business rules. This pattern follows open closed design principle and quite useful in Java.  
One of a good example of Strategy pattern from JDK itself is a Collections.sort() method and [Comparator interface](http://java67.blogspot.sg/2012/10/how-to-sort-object-in-java-comparator-comparable-example.html), which is a strategy interface and defines a strategy for comparing objects. Because of this pattern, we don't need to modify sort() method (closed for modification) to compare any object, at the same time we can implement Comparator interface to define new comparing strategy (open for extension).

**2. What is Observer design pattern in Java? When do you use Observer pattern in Java?**  
This is one of the most common Java design pattern interview questions. Observer pattern is based upon notification, there are two kinds of object Subject and Observer. Whenever there is change on subject's state observer will receive notification. See [What is Observer design pattern in Java with real life example](http://javarevisited.blogspot.sg/2011/12/observer-design-pattern-java-example.html) for more details.  
  
**3. Difference between Strategy and State design Pattern in Java?**  
This is an interesting Java design pattern interview questions as both Strategy and State pattern has the same structure. If you look at UML class diagram for both patterns they look exactly same, but their intent is totally different.  
State design pattern is used to define and manage the state of an object, while Strategy pattern is used to define a set of an interchangeable algorithm and let's client choose one of them. So [Strategy pattern](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%2Ffrom-0-to-1-design-patterns%2F) is a client driven pattern while Object can manage their state itself.  
  
**4. What is decorator pattern in Java? Can you give an example of Decorator pattern?**  
Decorator pattern is another popular Java design pattern question which is common because of its heavy usage in java.io package. BufferedReader  and  BufferedWriter are a good example of decorator pattern in Java. See [How to use Decorator pattern in Java](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html) for more details.  
  
**5. When to use Composite design Pattern in Java? Have you used previously in your project?**  
This design pattern question is asked on Java interview not just to check familiarity with the Composite pattern but also, whether a candidate has the real life experience or not.The*Composite pattern* is also a core Java design pattern, which allows you to treat both whole and part object to treat in a similar way. Client code, which deals with a Composite or individual object doesn't differentiate between them, it is possible because Composite class also implement the same interface as their individual part.  
One of the good examples of the Composite pattern from JDK is JPanel class, which is both Component and Container.  When the paint() method is called on JPanel, it internally called paint() method of individual components and let them draw themselves.  
On the second part of this design pattern interview question, be truthful, if you have used then say yes, otherwise say that you are familiar with the concept and used it by your own. By the way, always remember, giving an example from your project creates a better impression.

**6. What is Singleton pattern in Java?**  
Singleton pattern in Java is a pattern which allows only one instance of Singleton class available in the whole application. java.lang.Runtime is a good example of Singleton pattern in Java. There are lot's of follow up questions on Singleton pattern see [10 Java singleton interview question answers](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) for those followups  
  
**7. Can you write thread-safe Singleton in Java?**  
There are multiple ways to write thread-safe singleton in Java e.g by writing singleton using double checked locking, by using static Singleton instance initialized during [class loading.](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html) By the way using Java enum to create thread-safe singleton is the most simple way. See [Why Enum singleton is better in Java](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) for more details.  
  
**8. When to use Template method design Pattern in Java?**  
The Template pattern is another popular core Java design pattern interview question. I have seen it appear many times in real life project itself. Template pattern outlines an algorithm in form of template method and lets subclass implement individual steps.  
The key point to mention, while answering this question is that template method should be final, so that subclass can not override and change steps of the algorithm, but same time individual step should be abstract, so that child classes can implement them.  
  
**9. What is Factory pattern in Java? What is the advantage of using a static factory method to create an object?**  
Factory pattern in Java is a creation Java design pattern and favorite on many Java interviews.Factory pattern used to create an object by providing static factory methods. There are many advantages of providing factory methods e.g. caching immutable objects, easy to introduce new objects etc. See [What is Factory pattern in Java and benefits](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html) for more details.  
  
**10. What is the difference between Decorator and Proxy pattern in Java?**  
Another tricky Java design pattern question and trick here is that both Decorator and Proxy implements the interface of the object they decorate or encapsulate. As I said, many Java design pattern can have similar or exactly same structure but they differ in their intent.  
Decorator pattern is used to implement functionality on an already created object, while a Proxy pattern is used for controlling access to an object.  
One more difference between Decorator and the Proxy design pattern is that Decorator doesn't create an object, instead, it get the object in its constructor, while Proxy actually creates objects. You can also read [Head First Analysis and Design](http://www.amazon.com/dp/0596008678/?tag=javamysqlanta-20) to understand the difference between them.  
  
**11. When to use Setter and Constructor Injection in Dependency Injection pattern?**  
Use Setter injection to provide optional dependencies of an object, while use Constructor iInjection to provide a mandatory dependency of an object, without which it can not work. This question is related to [Dependency Injection design pattern](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html) and mostly asked in the context of Spring framework, which is now become a standard for developing Java application.  
Since Spring provides IOC container, it also gives you a way to specify dependencies either by using setter methods or constructors. You can also take a look my [previous post](http://javarevisited.blogspot.com/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html) on the same topic.

**12. What is difference between Factory and Abstract Factory in Java**  
I have already answered this question in detail with my article with the same title. The main difference is that Abstract Factory creates factory while Factory pattern creates objects. So both abstract the creation logic but one abstract is for factory and other for items. You can see [here](http://javarevisited.blogspot.sg/2013/01/difference-between-factory-and-abstract-factory-design-pattern-java.html) to answer this Java design pattern interview question.  
  
**13. When to use Adapter pattern in Java? Have you used it before in your project?**  
Use Adapter pattern when you need to make two class work with incompatible interfaces. Adapter pattern can also be used to encapsulate third party code so that your application only depends upon Adapter, which can adapt itself when third party code changes or you moved to a different third party library.  
By the way, this Java design pattern question can also be asked by providing an actual scenario. You can further read [Head First Design Pattern](http://www.amazon.com/dp/0596007124/?tag=javamysqlanta-20) to learn more about Adapter pattern and its real world usage. The book is updated for Java 8 as well so you will learn new, Java 8 way to implement these old design patterns.

**14. Can you write code to implement producer consumer design pattern in Java?**  
The Producer-consumer design pattern is a concurrency design pattern in Java which can be implemented using multiple ways. If you are working in Java 5 then its better to use Concurrency util to implement producer-consumer pattern instead of plain old [wait and notify in Java](http://javarevisited.blogspot.sg/2011/05/wait-notify-and-notifyall-in-java.html).  Here is a good example of implementing [producer consumer problem using BlockingQueue in Java](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html).  
  
**15. What is Open closed design principle in Java?**  
The Open closed design principle is one of the SOLID principle defined by Robert C. Martin, popularly known as Uncle Bob in his most popular book, [Clean Code](http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882?tag=javamysqlanta-20). This principle advises that a code should be open for extension but closed for modification.

At first, this may look conflicting but once you explore the power of polymorphism, you will start finding patterns which can provide stability and flexibility of this principle.  
One of the key examples of this is State and Strategy design pattern, where Context class is closed for modification and new functionality is provided by writing new code by implementing a new state of strategy. See [this](http://javarevisited.blogspot.com/2011/11/great-example-of-open-closed-design.html)article to know more about Open closed principle.  
  
**16. What is Builder design pattern in Java? When do you use Builder pattern?**  
Builder pattern in Java is another creational design pattern in Java and often asked in Java interviews because of its specific use when you need to build an object which requires multiple properties some optional and some mandatory. See [When to use Builder pattern in Java](http://javarevisited.blogspot.sg/2012/06/builder-design-pattern-in-java-example.html) for more details  
  
**17. Can you give an example of  SOLID design principles in Java?**  
There are lots of SOLID design pattern which forms acronym SOLID e.g.  
1. Single Responsibility Principle or SRP  
3. Open Closed Design Principle or OCD  
3. Liskov Substitution Principle  
4. Interface Segregation Principle  
5. Dependency Inversion Principle.  
You can further read this [list of SOLID design principles](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%2Ffrom-0-to-1-design-patterns%2F) for Java programmer to answer this Java interview question.  
 **18. What is the difference between Abstraction and Encapsulation in Java?**  
Even though both Abstraction and Encapsulation looks similar because both hide complexity and make the external interface simpler there is a subtle difference between them. Abstraction hides logical complexity while Encapsulation hides Physical Complexity.  
Btw, I have already covered answer of this Java interview question in my previous post as [Difference between encapsulation and abstraction in Java](http://java67.blogspot.sg/2012/08/difference-between-abstraction-and-encapsulation-java-oops.html), here are some more difference from that post.

