**Design pattern interview questions**

**Q. What is design patterns ? Have you used any design pattern in your code ?**

**Ans.** Design patterns are tried and tested way to solve particular design issues by various programmers in the world. Design patterns are extension of code reuse.

**Q. Can you name few design patterns used in standard JDK library?**

**Ans.** Decorator design pattern which is used in various Java IO classes, Singleton pattern which is used in Runtime , Calendar and various other classes, Factory pattern which is used along with various Immutable classes likes Boolean e.g. Boolean.valueOf and Observer pattern which is used in Swing and many event listener frameworks.

**Q. What is Singleton design pattern in Java ?**

**Ans.** Singleton pattern focus on sharing of expensive object in whole system. Only one instance of a particular class is maintained in whole application which is shared by all modules. Java.lang.Runtime is a classical example of Singleton design pattern. From Java 5 onwards you can use enum to thread-safe singleton.

**Q. What is singleton class?**

**Ans.** Singleton class means that any given time only one instance of the class is present, in one JVM.

**Q. What is main benefit of using factory pattern ? Where do you use it?**

**Ans.** Factory pattern’s main benefit is increased level of encapsulation while creating objects. If you use Factory to create object you can later replace original implementation of Products or classes with more advanced and high performance implementation without any change on client layer.

**Q. What is observer design pattern in Java**

**Ans.** Observer design pattern is based on communicating changes in state of object to observers so that they can take their action. Simple example is a weather system where change in weather must be reflected in Views to show to public. Here weather object is Subject while different views are Observers.

**Q. Give example of decorator design pattern in Java ? Does it operate on object level or class level ?**

**Ans.** Decorator pattern enhances capability of individual object. Java IO uses decorator pattern extensively and classical example is Buffered classes like BufferedReader and BufferedWriter which enhances Reader and Writer objects to perform Buffer level reading and writing for improved performance.

**Q. When to use Strategy Design Pattern in Java?**

**Ans.** 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 follow open closed design principle and quite useful in Java. One example of Strategy pattern from JDK itself is a Collections.sort() method and Comparator interface, which is a strategy interface and defines strategy for comparing objects. Because of this pattern, we don't need to modify sort() method (closed for modification) to compare any object, at same time we can implement Comparator interface to define new comparing strategy (open for extension).

**Q. Difference between Strategy and State design Pattern in Java?**

**Ans.** If you look at UML class diagram for both pattern they look exactly same, but there intent is totally different. State design pattern is used to define and manage state of object, while Strategy pattern is used to define a set of interchangeable algorithm and let's client to choose one of them. So Strategy pattern is a client driven pattern while Object can manage their state itself.

**Q. When to use Composite design Pattern in Java? Have you used previously in your project?**

**Ans.** This design pattern question is asked on Java interview not just to check familiarity with Composite pattern but also, whether candidate has real life experience or not. Composite pattern is also a core Java design pattern, which allows you to treat both whole and part object to treat in similar way. Client code, which deals with Composite or individual object doesn't differentiate on them, it is possible because Composite class also implement same interface as there individual part. One of the good example of Composite pattern from JDK is JPanel class, which is both Component and Container. When paint() method is called on JPanel, it internally called paint() method of individual components and let them draw themselves. On second part of this design pattern interview question, be truthful, if you have used then say yes, otherwise say that you are familiar with concept and used it by your own. By the way always remember, giving an example from your project creates better impression.

**Q. Can you write thread-safe Singleton in Java? is it better to make the whole method synchronized or only critical section synchronized?**

**Ans.** 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. By the way using Java enum to create thread-safe singleton is most simple way. See Why Enum singleton is better in Java for more details.

Singleton in Java is a class with just one instance in whole Java application, for example, java.lang.Runtime is a Singleton class. Creating Singleton was tricky prior Java 4 but once Java 5 introduced Enum it's very easy. see my article How to create thread-safe Singleton in Java for more details on writing Singleton using the enum and double checked locking which is the purpose of this Java interview question.

**Q. When to use Template method design Pattern in Java?**

**Ans.** 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 let subclass implement individual steps. Key point to mention, while answering this question is that template method should be final, so that subclass cannot override and change steps of algorithm, but same time individual step should be abstract, so that child classes can implement them.

**Q. Difference between Decorator and Proxy pattern in Java?**

**Ans.** Both Decorator and Proxy implements 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 already created object, while Proxy pattern is used for controlling access to object. One more difference between Decorator and Proxy design pattern is that, Decorator doesn't create object, instead it get object in it's constructor, while Proxy actually creates objects.

**Q. When to use Adapter pattern in Java? Have you used it before in your project?**

**Ans.** 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 actual scenario.

**Q. Can you write code to implement producer consumer design pattern in Java?**

**Ans.** Producer consumer design pattern is a concurrency design pattern in Java which can be implemented using multiple way. 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. Here is a good example of implementing producer consumer problem using BlockingQueue in Java.

**Q. What is Open closed design principle in Java?**

**Ans.** Open closed design principle is one of the SOLID principle defined by Robert C. Martin, popularly known as Uncle Bob. 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 power of polymorphism, you will start finding patterns which can provide stability and flexibility of this principle. One of the key example 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 new state of strategy. See this article to know more about Open closed principle.

**Q. What is Builder design pattern in Java? When do you use Builder pattern ?**

**Ans.** 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 for more details

**Q. What is double checked locking in Singleton?**

**Ans.** One of the most hyped question on Singleton and really demands complete understanding to get it right because of Java Memory model caveat prior to Java 5. If a guy comes up with a solution of using volatile instance of Singleton then it really shows it has in depth knowledge of Java memory model and he is constantly updating his Java knowledge.

**Q. How do you prevent for creating another instance of Singleton using clone() method?**

**Ans.** This type of questions generally comes some time by asking how to break singleton or when Singleton is not Singleton in Java.

**Q. How do you prevent for creating another instance of Singleton using reflection?**

**Ans.** Open to all. In my opinion throwing exception from constructor is an option.

**Q. How do you prevent for creating another instance of Singleton during serialization?**

**Ans.** Another great question which requires knowledge of Serialization in Java and how to use it for persisting Singleton classes. This is open to you all but in my opinion use of readResolve() method can sort this out for you.

**Q. When is Singleton not a Singleton in Java?**

**Ans.** There is a very good article present in Sun's Java site which discusses various scenarios when a Singleton is not really remains Singleton and multiple instance of Singleton is possible. Here is the link of that article http://java.sun.com/developer/technicalArticles/Programming/singletons/