**The Facade pattern abstracts** details away from the developer and makes a certain portion of code easier to use.

**The Builder pattern seperates** the construction of an object from its representation. That makes it possible to use the same construction process across multiple types.

**The Factory and Abstract Factory** both deal with instanciating a related set of classes based on certain parameters used when the call to the Factory is made.

**PROXY PATTTERN:**

In Proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

In Proxy pattern, we create object having original object to interface its functionality to outer world.

**Difference between Proxy and Decorator pattern is that :**

The difference is that, with the Proxy pattern, the relationship between a proxy and the real subject is typically set at compile time, whereas decorators can be recursively constructed at runtime. But I'm getting ahead of myself.

**Decorator design pattern** is used to enhance the functionality of a particular object at run-time or dynamically.

<http://java67.blogspot.com/2012/09/top-10-java-design-pattern-interview-question-answer.html>

Designe pattern:

**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 follow open closed design principle and quite useful in Java. One example of Strategypattern 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 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).

**Observer**

This is one of the most common Java design pattern interview question. 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 below for more details.

**Observer design pattern** **in Java**is a fundamental core Java pattern where Observe watch for any change in state or property of Subject. For Example Company updates all its shareholders for any decision they make here Company is Subject and Shareholders are Observers, any change in policy of company and Company notifies all its Shareholders or Observer. This was simple real world explanation of Observer pattern. In this article we will in detail *what is Observer Design pattern*, what is *benefit of Observer design Pattern*, Example or Observer pattern in Java and few other points. Just like [Decorator design Pattern](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html) and [Factory Pattern in Java](http://javarevisited.blogspot.com/2011/12/factory-design-pattern-java-example.html), Observer pattern is also used in JDK

**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 same structure. If you look at UML class diagram for both pattern they look exactly same, but there intent is totally different. State design patterns used to define and mange 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 there state itself.

**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 good example of decorator pattern in Java.

<http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html>

**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 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.

Composite pattern means that there is some thing1 that does action1, and also there is some other thing2 that does action2. Then, there should be a third thing3 entirely that does action3, and it does that action as some combination of action1 and action2. Do achieve that, thing3 has a thing1 and thing2 as part of itself. To rephrase, thing3 is **composed** out of thing1 and thing2.

**What is Singleton pattern in Java?**  
Singleton pattern in Java is a pattern which allows only one instance of Singleton class available in whole application. java.lang.Runtime is good example of Singleton pattern in Java. There are lot's of follow up questions on Singleton pattern

**When to use Template method design Pattern in Java?**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 can not override and change steps of algorithm, but same time individual step should be abstract, so that child classes can implement them.

**What is Factory pattern in Java? What is advantage of using static factory method to create object?**  
Factory pattern in Java is a creation Java design pattern and favorite on many Java interviews.Factory pattern used to create object by providing static factory methods. There are many advantage 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.

<http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html>

**Factory design pattern in Java** one of the core design pattern which is used heavily not only in JDK but also in various Open Source framework such as Spring, Struts and Apache along with [decorator design pattern in Java](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html). Factory Design pattern is based on [**Encapsulation**](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html)object oriented concept. Factory method is used to create different object from factory often refereed as Item and it encapsulate the creation code. **So instead of having object creation code on client side we encapsulate inside Factory method in Java**. One of the best examples of factory pattern in Java is BorderFactory Class of Swing API. In this Design pattern tutorial we will see **What is Factory method design pattern in Java**, What are main *advantages of factory pattern in Java* , Code example of Factory design pattern and What problem **Factory pattern** solves in Java or when to use Factory design pattern.   
This article is in continuation of my design pattern article as [10 OOPS and SOLID design principles java programmer should know](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html)and [How to use Observer pattern in Java](http://javarevisited.blogspot.com/2011/12/observer-design-pattern-java-example.html)

Example of static factory method in JDK

 Best Example of Factory method design pattern is valueOf() method which is there in String and wrapper classes like Integer and Boolean and used for type conversion i.e. from converting String to Integer or String to double in java..

Some more examples of factory method design pattern from JDK is :

valueOf() method which returns object created by factory equivalent to value of parameter passed.

getInstance() method which creates instance of Singleton class.

newInstance() method which is used to create and return new instance from factory method every time called.

getType() and newType() equivalent of getInstance() and newInstance() factory method but used when factory method resides in separate class.

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

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.

When to use Static Class in place of Singleton in Java

Indeed there are some situations, where static classes makes sense than Singleton. Prime example of this is java.lang.Math which is not Singleton, instead a class with all static methods. Here are few situation where I think using static class over Singleton pattern make sense:

1) If your Singleton is not maintaining any state, and just providing global access to methods, than consider using static class, as static methods are much faster than Singleton, because of [static binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) during compile time. But remember its not advised to maintain state inside static class, especially in concurrent environment, where it could lead subtle [race conditions](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html) when modified parallel by multiple threads without adequate synchronization

Explain DecoratorPattern:

## Java Decorator Design Pattern

### What is decorator design pattern in Java?

          Decorator design pattern is used to **enhance the functionality of a particular object at run-time** or dynamically.

          At the same time **other instance of same class will not be affected by this** so individual object gets the new behavior.

          Basically we wrap the original object through decorator object.

          Decorator design pattern is based on abstract classes and we derive concrete implementation from that classes,

          It’s a structural design pattern and most widely used.

Should be possible to cite one for most of the 23 patterns in GoF:

1. Abstract Factory: java.sql interfaces all get their concrete implementations from JDBC JAR when driver is registered.
2. Builder: java.lang.StringBuilder.
3. Factory Method: XML factories, among others.
4. Prototype: Maybe clone(), but I'm not sure I'm buying that.
5. Singleton: java.lang.System, java.lang.Runtime.
6. Adapter: Adapter classes in java.awt.event, e.g., WindowAdapter.
7. Bridge: Collection classes in java.util. List implemented by ArrayList.
8. Composite: java.awt. java.awt.Component + java.awt.Container
9. Decorator: All over the java.io package.
10. Facade: [ExternalContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/ExternalContext.html) behaves as a facade for performing cookie, session scope and similar operations.
11. Flyweight: Integer, Character, etc.
12. Proxy: java.rmi package
13. Chain of Responsibility: Servlet filters
14. Command: Swing menu items
15. Interpreter: No directly in JDK, but JavaCC certainly uses this.
16. Iterator: java.util.Iterator interface; can't be clearer than that.
17. Mediator: JMS?
18. Memento:
19. Observer: java.util.Observer/Observable (badly done, though)
20. State:
21. Strategy:
22. Template:
23. Visitor:

In java design pattern examples:

|  |  |
| --- | --- |
| down vote accepted  +250 | You can find an overview of a lot of design patterns in [Wikipedia](http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29#Classification_and_list). It also mentions which patterns are mentioned by GoF. I'll sum them up here and try to assign as much as possible pattern implementations found in both the Java SE and Java EE API's.  [**Creational patterns**](http://en.wikipedia.org/wiki/Creational_pattern)  [**Abstract factory**](http://en.wikipedia.org/wiki/Abstract_factory_pattern) **(recognizeable by creational methods returning the factory itself which in turn can be used to create another abstract/interface type)**   * [javax.xml.parsers.DocumentBuilderFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#newInstance%28%29) * [javax.xml.transform.TransformerFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/transform/TransformerFactory.html#newInstance%28%29) * [javax.xml.xpath.XPathFactory#newInstance()](http://docs.oracle.com/javase/6/docs/api/javax/xml/xpath/XPathFactory.html#newInstance%28%29)   [**Builder**](http://en.wikipedia.org/wiki/Builder_pattern) **(recognizeable by creational methods returning the instance itself)**   * [java.lang.StringBuilder#append()](http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuilder.html#append%28boolean%29) (unsynchronized) * [java.lang.StringBuffer#append()](http://docs.oracle.com/javase/6/docs/api/java/lang/StringBuffer.html#append%28boolean%29) (synchronized) * [java.nio.ByteBuffer#put()](http://docs.oracle.com/javase/6/docs/api/java/nio/ByteBuffer.html#put%28byte%29) (also on [CharBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/CharBuffer.html#put%28char%29), [ShortBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/ShortBuffer.html#put%28short%29), [IntBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/IntBuffer.html#put%28int%29), [LongBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/LongBuffer.html#put%28long%29), [FloatBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/FloatBuffer.html#put%28float%29) and [DoubleBuffer](http://docs.oracle.com/javase/6/docs/api/java/nio/DoubleBuffer.html#put%28double%29)) * [javax.swing.GroupLayout.Group#addComponent()](http://docs.oracle.com/javase/6/docs/api/javax/swing/GroupLayout.Group.html#addComponent%28java.awt.Component%29) * All implementations of [java.lang.Appendable](http://docs.oracle.com/javase/6/docs/api/java/lang/Appendable.html)   [**Factory method**](http://en.wikipedia.org/wiki/Factory_method_pattern) **(recognizeable by creational methods returning an implementation of an abstract/interface type)**   * [java.util.Calendar#getInstance()](http://docs.oracle.com/javase/6/docs/api/java/util/Calendar.html#getInstance%28%29) * [java.util.ResourceBundle#getBundle()](http://docs.oracle.com/javase/6/docs/api/java/util/ResourceBundle.html#getBundle%28java.lang.String%29) * [java.text.NumberFormat#getInstance()](http://docs.oracle.com/javase/6/docs/api/java/text/NumberFormat.html#getInstance%28%29) * [java.nio.charset.Charset#forName()](http://docs.oracle.com/javase/6/docs/api/java/nio/charset/Charset.html#forName%28java.lang.String%29) * [java.net.URLStreamHandlerFactory#createURLStreamHandler(String)](http://docs.oracle.com/javase/6/docs/api/java/net/URLStreamHandlerFactory.html) (Returns singleton object per protocol)   [**Prototype**](http://en.wikipedia.org/wiki/Prototype_pattern) **(recognizeable by creational methods returning a different instance of itself with the same properties)**   * [java.lang.Object#clone()](http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#clone%28%29) (the class has to implement [java.lang.Cloneable](http://docs.oracle.com/javase/6/docs/api/java/lang/Cloneable.html))   [**Singleton**](http://en.wikipedia.org/wiki/Singleton_pattern) **(recognizeable by creational methods returning the same instance (usually of itself) everytime)**   * [java.lang.Runtime#getRuntime()](http://docs.oracle.com/javase/6/docs/api/java/lang/Runtime.html#getRuntime%28%29) * [java.awt.Desktop#getDesktop()](http://docs.oracle.com/javase/6/docs/api/java/awt/Desktop.html#getDesktop%28%29)   [**Structural patterns**](http://en.wikipedia.org/wiki/Structural_pattern)  [**Adapter**](http://en.wikipedia.org/wiki/Adapter_pattern) **(recognizeable by creational methods taking an instance of different abstract/interface type and returning an implementation of own/another abstract/interface type which decorates/overrides the given instance)**   * [java.util.Arrays#asList()](http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html#asList%28T...%29) * [java.io.InputStreamReader(InputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/InputStreamReader.html#InputStreamReader%28java.io.InputStream%29) (returns a Reader) * [java.io.OutputStreamWriter(OutputStream)](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStreamWriter.html#OutputStreamWriter%28java.io.OutputStream%29) (returns a Writer) * [javax.xml.bind.annotation.adapters.XmlAdapter#marshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#marshal%28BoundType%29) and [#unmarshal()](http://docs.oracle.com/javase/6/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#unmarshal%28ValueType%29)   [**Bridge**](http://en.wikipedia.org/wiki/Bridge_pattern) **(recognizeable by creational methods taking an instance of different abstract/interface type and returning an implementation of own abstract/interface type which delegates/uses the given instance)**   * None comes to mind yet. A fictive example would be new LinkedHashMap(LinkedHashSet<K>, List<V>) which returns an unmodifiable linked map which doesn't clone the items, but uses them. The [java.util.Collections#newSetFromMap()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#newSetFromMap%28java.util.Map%29) and [singletonXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#singleton%28T%29) methods however comes close.   [**Composite**](http://en.wikipedia.org/wiki/Composite_pattern) **(recognizeable by behavioral methods taking an instance of same abstract/interface type into a tree structure)**   * [java.awt.Container#add(Component)](http://docs.oracle.com/javase/6/docs/api/java/awt/Container.html#add%28java.awt.Component%29) (practically all over Swing thus) * [javax.faces.component.UIComponent#getChildren()](http://docs.oracle.com/javaee/6/api/javax/faces/component/UIComponent.html#getChildren%28%29) (practically all over JSF UI thus)   [**Decorator**](http://en.wikipedia.org/wiki/Decorator_pattern) **(recognizeable by creational methods taking an instance of same abstract/interface type which adds additional behaviour)**   * All subclasses of [java.io.InputStream](http://docs.oracle.com/javase/6/docs/api/java/io/InputStream.html), [OutputStream](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html), [Reader](http://docs.oracle.com/javase/6/docs/api/java/io/Reader.html) and [Writer](http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html) have a constructor taking an instance of same type. * [java.util.Collections](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html), the [checkedXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#checkedCollection%28java.util.Collection,%20java.lang.Class%29), [synchronizedXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#synchronizedCollection%28java.util.Collection%29) and [unmodifiableXXX()](http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#unmodifiableCollection%28java.util.Collection%29) methods. * [javax.servlet.http.HttpServletRequestWrapper](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequestWrapper.html) and [HttpServletResponseWrapper](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponseWrapper.html)   [**Facade**](http://en.wikipedia.org/wiki/Facade_pattern) **(recognizeable by behavioral methods which internally uses instances of different independent abstract/interface types)**   * [javax.faces.context.FacesContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/FacesContext.html), it internally uses among others the abstract/interface types [LifeCycle](http://docs.oracle.com/javaee/6/api/javax/faces/lifecycle/Lifecycle.html), [ViewHandler](http://docs.oracle.com/javaee/6/api/javax/faces/application/ViewHandler.html), [NavigationHandler](http://docs.oracle.com/javaee/6/api/javax/faces/application/NavigationHandler.html) and many more without that the end-user has to worry about it (which are however overrideable by injection). * [javax.faces.context.ExternalContext](http://docs.oracle.com/javaee/6/api/javax/faces/context/ExternalContext.html), which internally uses [ServletContext](http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html), [HttpSession](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSession.html), [HttpServletRequest](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html), [HttpServletResponse](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html), etc.   [**Flyweight**](http://en.wikipedia.org/wiki/Flyweight_pattern) **(recognizeable by creational methods returning a cached instance, a bit the "multiton" idea)**   * [java.lang.Integer#valueOf(int)](http://docs.oracle.com/javase/6/docs/api/java/lang/Integer.html#valueOf%28int%29) (also on [Boolean](http://docs.oracle.com/javase/6/docs/api/java/lang/Boolean.html#valueOf%28boolean%29), [Byte](http://docs.oracle.com/javase/6/docs/api/java/lang/Byte.html#valueOf%28byte%29), [Character](http://docs.oracle.com/javase/6/docs/api/java/lang/Character.html#valueOf%28char%29), [Short](http://docs.oracle.com/javase/6/docs/api/java/lang/Short.html#valueOf%28short%29) and [Long](http://docs.oracle.com/javase/6/docs/api/java/lang/Long.html#valueOf%28long%29))   [**Proxy**](http://en.wikipedia.org/wiki/Proxy_pattern) **(recognizeable by creational methods which returns an implementation of given abstract/interface type which in turn delegates/uses a different implementation of given abstract/interface type)**   * [java.lang.reflect.Proxy](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Proxy.html) * [java.rmi.\*](http://docs.oracle.com/javase/6/docs/api/java/rmi/package-summary.html), the whole API actually.   The Wikipedia example is IMHO a bit poor, lazy loading has actually completely nothing to do with the proxy pattern at all.  [**Behavioral patterns**](http://en.wikipedia.org/wiki/Behavioral_pattern)  [**Chain of responsibility**](http://en.wikipedia.org/wiki/Chain_of_responsibility_pattern) **(recognizeable by behavioral methods which (indirectly) invokes the same method in another implementation of same abstract/interface type in a queue)**   * [java.util.logging.Logger#log()](http://docs.oracle.com/javase/6/docs/api/java/util/logging/Logger.html#log%28java.util.logging.Level,%20java.lang.String%29) * [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html#doFilter%28javax.servlet.ServletRequest,%20javax.servlet.ServletResponse,%20javax.servlet.FilterChain%29)   [**Command**](http://en.wikipedia.org/wiki/Command_pattern) **(recognizeable by behavioral methods in an abstract/interface type which invokes a method in an implementation of a different abstract/interface type which has been encapsulated by the command implementation during its creation)**   * All implementations of [java.lang.Runnable](http://docs.oracle.com/javase/6/docs/api/java/lang/Runnable.html) * All implementations of [javax.swing.Action](http://docs.oracle.com/javase/6/docs/api/javax/swing/Action.html)   [**Interpreter**](http://en.wikipedia.org/wiki/Interpreter_pattern) **(recognizeable by behavioral methods returning a structurally different instance/type of the given instance/type; note that parsing/formatting is not part of the pattern, determining the pattern and how to apply it is)**   * [java.util.Pattern](http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html) * [java.text.Normalizer](http://docs.oracle.com/javase/6/docs/api/java/text/Normalizer.html) * All subclasses of [java.text.Format](http://docs.oracle.com/javase/6/docs/api/java/text/Format.html) * All subclasses of [javax.el.ELResolver](http://docs.oracle.com/javaee/6/api/javax/el/ELResolver.html)   [**Iterator**](http://en.wikipedia.org/wiki/Iterator_pattern) **(recognizeable by behavioral methods sequentially returning instances of a different type from a queue)**   * All implementations of [java.util.Iterator](http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html) (thus among others also [java.util.Scanner](http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html)!). * All implementations of [java.util.Enumeration](http://docs.oracle.com/javase/6/docs/api/java/util/Enumeration.html)   [**Mediator**](http://en.wikipedia.org/wiki/Mediator_pattern) **(recognizeable by behavioral methods taking an instance of different abstract/interface type (usually using the command pattern) which delegates/uses the given instance)**   * [java.util.Timer](http://docs.oracle.com/javase/6/docs/api/java/util/Timer.html) (all scheduleXXX() methods) * [java.util.concurrent.Executor#execute()](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/Executor.html#execute%28java.lang.Runnable%29) * [java.util.concurrent.ExecutorService](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ExecutorService.html) (the invokeXXX() and submit() methods) * [java.util.concurrent.ScheduledExecutorService](http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html) (all scheduleXXX() methods) * [java.lang.reflect.Method#invoke()](http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Method.html#invoke%28java.lang.Object,%20java.lang.Object...%29)   [**Memento**](http://en.wikipedia.org/wiki/Memento_pattern) **(recognizeable by behavioral methods which internally changes the state of the whole instance)**   * [java.util.Date](http://docs.oracle.com/javase/6/docs/api/java/util/Date.html) (the setter methods do that, Date is internally represented by a long value) * All implementations of [java.io.Serializable](http://docs.oracle.com/javase/6/docs/api/java/io/Serializable.html) * All implementations of [javax.faces.component.StateHolder](http://docs.oracle.com/javaee/6/api/javax/faces/component/StateHolder.html)   [**Observer (or Publish/Subscribe)**](http://en.wikipedia.org/wiki/Observer_pattern) **(recognizeable by behavioral methods which invokes a method on an instance of another abstract/interface type, depending on own state)**   * [java.util.Observer](http://docs.oracle.com/javase/6/docs/api/java/util/Observer.html)/[java.util.Observable](http://docs.oracle.com/javase/6/docs/api/java/util/Observable.html) (rarely used in real world though) * All implementations of [java.util.EventListener](http://docs.oracle.com/javase/6/docs/api/java/util/EventListener.html) (practically all over Swing thus) * [javax.servlet.http.HttpSessionBindingListener](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSessionBindingListener.html) * [javax.servlet.http.HttpSessionAttributeListener](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpSessionAttributeListener.html) * [javax.faces.event.PhaseListener](http://docs.oracle.com/javaee/6/api/javax/faces/event/PhaseListener.html)   [**State**](http://en.wikipedia.org/wiki/State_pattern) **(recognizeable by behavioral methods which changes its behaviour depending on the instance's state which can be controlled externally)**   * [javax.faces.lifecycle.LifeCycle#execute()](http://docs.oracle.com/javaee/6/api/javax/faces/lifecycle/Lifecycle.html#execute%28javax.faces.context.FacesContext%29) (controlled by [FacesServlet](http://docs.oracle.com/javaee/6/api/javax/faces/webapp/FacesServlet.html), the behaviour is dependent on current phase (state) of JSF lifecycle)   [**Strategy**](http://en.wikipedia.org/wiki/Strategy_pattern) **(recognizeable by behavioral methods in an abstract/interface type which invokes a method in an implementation of a different abstract/interface type which has been passed-in as method argument into the strategy implementation)**   * [java.util.Comparator#compare()](http://docs.oracle.com/javase/6/docs/api/java/util/Comparator.html#compare%28T,%20T%29), executed by among others Collections#sort(). * [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html), the service() and all doXXX() methods take HttpServletRequest and HttpServletResponse and the implementor has to process them (and not to get hold of them as instance variables!). * [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/6/api/javax/servlet/Filter.html#doFilter%28javax.servlet.ServletRequest,%20javax.servlet.ServletResponse,%20javax.servlet.FilterChain%29)   [**Template method**](http://en.wikipedia.org/wiki/Template_method_pattern) **(recognizeable by behavioral methods which already have a "default" behaviour definied by an abstract type)**   * All non-abstract methods of [java.io.InputStream](http://docs.oracle.com/javase/6/docs/api/java/io/InputStream.html), [java.io.OutputStream](http://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html), [java.io.Reader](http://docs.oracle.com/javase/6/docs/api/java/io/Reader.html) and [java.io.Writer](http://docs.oracle.com/javase/6/docs/api/java/io/Writer.html). * All non-abstract methods of [java.util.AbstractList](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractList.html), [java.util.AbstractSet](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractSet.html) and [java.util.AbstractMap](http://docs.oracle.com/javase/6/docs/api/java/util/AbstractMap.html). * [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServlet.html), all the doXXX() methods by default sends a HTTP 405 "Method Not Allowed" error to the response. You're free to implement none or any of them.   [**Visitor**](http://en.wikipedia.org/wiki/Visitor_pattern) **(recognizeable by two different abstract/interface types which has methods definied which takes each the other abstract/interface type; the one actually calls the method of the other and the other executes the desired strategy on it)**   * [javax.lang.model.element.AnnotationValue](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/AnnotationValue.html) and [AnnotationValueVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/AnnotationValueVisitor.html) * [javax.lang.model.element.Element](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/Element.html) and [ElementVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/element/ElementVisitor.html) * [javax.lang.model.type.TypeMirror](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/type/TypeMirror.html) and [TypeVisitor](http://docs.oracle.com/javase/6/docs/api/javax/lang/model/type/TypeVisitor.html) |

<http://ageekview.wordpress.com/2011/05/15/a-dive-into-design-patterns-used-in-jdk-%E2%80%93-ii/>

# [Difference between Bridge pattern and adapter pattern](http://stackoverflow.com/questions/1425171/difference-between-bridge-pattern-and-adapter-pattern)

Adapter makes things work after they're designed; Bridge makes them work before they are. [GoF, p219]"

Effectively, the Adapter pattern is useful when you have existing code, be it third party, or in-house, but out of your control, or otherwise not changeable to quite meet the interface you need it to. For instance, we have a SuperWeaponsArray which can control a fine array of doomsday devices.

### Top 18 Java Design Pattern Interview Questions Answers for Experienced

<http://java67.blogspot.com/2012/09/top-10-java-design-pattern-interview-question-answer.html>