Real time example of all design patterns:

# Creational

How do you want to create objects?

**Prototype :** A fully initialized instance to be copied or cloned  
Example : initial status of chess game

* java.lang.Object#clone()

**Builder** - Separates the construction of a complex object from its representation so that the same construction process can create different representations  
Example : Restaurant crew

* StringBuffer and StringBuilder

**Singleton** - A class of which only a single instance can exist  
Example : President of a country

* java.lang.Runtime#getRuntime()

**Factory Method** - Creates an instance of several derived classes.  
Example : In an organisation HR works as factory method. Here development team request type of resource need to HR. Based on request type, HR provide resource to Development team.

* java.util.Calendar#getInstance()

**Abstract Factory** - Creates an instance of several families of classes  
Example : HP, Samsung and Dell laptops are uses Intel and AMD processor.

* javax.xml.parsers.DocumentBuilderFactory#newInstance()

[Factory Method vs Abstract Factory](https://stackoverflow.com/a/5740020/1697099)

# Structural

This design patterns is all about Class and Object composition i.e. How do you want structure the software component.

**Proxy** - An object representing another object  
Example : check book leaf, credit card, debit card are proxy for Money and a customer representative to order a product.

* java.rmi.\*, the whole API actually.

**Decorator** - Add responsibilities to objects dynamically  
Example : 1) Adding discounts on an order 2) gun is a deadly weapon on it's own. But you can apply certain "decorations" to make it more accurate, silent and devastating.

* All subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor taking an instance of same type.

**Facade** - A single class that represents an entire subsystem  
Example : Customer care

* javax.faces.context.ExternalContext, which internally uses ServletContext, HttpSession, HttpServletRequest, HttpServletResponse, et

**Adapter** - Match interfaces of different classes  
Example : Power Adapters

* java.util.Arrays#asList()

**Flyweight** - A fine-grained instance used for efficient sharing  
Example : Dial tone

* java.lang.Integer#valueOf(int) (also on Boolean, Byte, Character, Short and Long)

# Behavioral

This design patterns is all about Class's objects communication i.e How do you want a behavior in software?

**Chain of Responsibility** - A way of passing a request between a chain of objects  
Example : Loan or Leave approval process

* javax.servlet.Filter#doFilter()

**Iterator** - Sequentially access the elements of a collection  
Example : Next/Previous buttons on TV

* All implementations of java.util.Iterator & java.util.Enumeration

**State** - Alter an object's behavior when its state changes  
Example : A fan wall control

**Observer** - A way of notifying change to a number of classes  
Example : Bidding or auction

* Publish/Subscribe JMS API

**Visitor** - Defines a new operation to a class without change  
Example : Taxi

**Template** - Defer the exact steps of an algorithm to a subclass  
Example : A blue print

* All non-abstract methods of java.io.InputStream, java.io.OutputStream, java.io.Reader and java.io.Writer.
* All non-abstract methods of java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap.
* javax.servlet.http.HttpServlet, 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.
  + JMSTemplate HibernateTemplate and JdbcTemplate in Spring

**Command** - Encapsulate a command request as an object  
Example : The "Guest Check" at a diner is an example of a Command pattern. The waiter or waitress takes an order or command from a customer and encapsulates that order by writing it on the check. The order is then queued for a short order cook. Note that the pad of "checks" used by each waiter is not dependent on the menu, and therefore they can support commands to cook many different items.

* All implementations of java.lang.Runnable

**Memento** - Capture and restore an object's internal state  
Example : save the state in a game & Undo/Redo operation in Windows

* All implementations of java.io.Serializable

**Mediator** - Defines simplified communication between classes  
Example : Air Traffic Controller(ATC)

**Strategy** - A Strategy defines a set of algorithms that can be used interchangeably.  
Example : Modes of transportation

* java.util.Comparator#compare(), executed by among others Collections#sort().
* javax.servlet.http.HttpServlet, 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()

## [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/8/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#newInstance--)
* [javax.xml.transform.TransformerFactory#newInstance()](http://docs.oracle.com/javase/8/docs/api/javax/xml/transform/TransformerFactory.html#newInstance--)
* [javax.xml.xpath.XPathFactory#newInstance()](http://docs.oracle.com/javase/8/docs/api/javax/xml/xpath/XPathFactory.html#newInstance--)

### [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/8/docs/api/java/lang/StringBuilder.html#append-boolean-) (unsynchronized)
* [java.lang.StringBuffer#append()](http://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html#append-boolean-) (synchronized)
* [java.nio.ByteBuffer#put()](http://docs.oracle.com/javase/8/docs/api/java/nio/ByteBuffer.html#put-byte-) (also on [CharBuffer](http://docs.oracle.com/javase/8/docs/api/java/nio/CharBuffer.html#put-char-), [ShortBuffer](http://docs.oracle.com/javase/8/docs/api/java/nio/ShortBuffer.html#put-short-), [IntBuffer](http://docs.oracle.com/javase/8/docs/api/java/nio/IntBuffer.html#put-int-), [LongBuffer](http://docs.oracle.com/javase/8/docs/api/java/nio/LongBuffer.html#put-long-), [FloatBuffer](http://docs.oracle.com/javase/8/docs/api/java/nio/FloatBuffer.html#put-float-) and [DoubleBuffer](http://docs.oracle.com/javase/8/docs/api/java/nio/DoubleBuffer.html#put-double-))
* [javax.swing.GroupLayout.Group#addComponent()](http://docs.oracle.com/javase/8/docs/api/javax/swing/GroupLayout.Group.html#addComponent-java.awt.Component-)
* All implementations of [java.lang.Appendable](http://docs.oracle.com/javase/8/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/8/docs/api/java/util/Calendar.html#getInstance--)
* [java.util.ResourceBundle#getBundle()](http://docs.oracle.com/javase/8/docs/api/java/util/ResourceBundle.html#getBundle-java.lang.String-)
* [java.text.NumberFormat#getInstance()](http://docs.oracle.com/javase/8/docs/api/java/text/NumberFormat.html#getInstance--)
* [java.nio.charset.Charset#forName()](http://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html#forName-java.lang.String-)
* [java.net.URLStreamHandlerFactory#createURLStreamHandler(String)](http://docs.oracle.com/javase/8/docs/api/java/net/URLStreamHandlerFactory.html) (Returns singleton object per protocol)
* [java.util.EnumSet#of()](https://docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html#of(E))
* [javax.xml.bind.JAXBContext#createMarshaller()](https://docs.oracle.com/javase/8/docs/api/javax/xml/bind/JAXBContext.html#createMarshaller--) and other similar methods

### [Prototype](http://en.wikipedia.org/wiki/Prototype_pattern) (recognizeable by creational methods returning adifferentinstance of itself with the same properties)

* [java.lang.Object#clone()](http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#clone--) (the class has to implement [java.lang.Cloneable](http://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html))

### [Singleton](http://en.wikipedia.org/wiki/Singleton_pattern) (recognizeable by creational methods returning thesameinstance (usually of itself) everytime)

* [java.lang.Runtime#getRuntime()](http://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#getRuntime--)
* [java.awt.Desktop#getDesktop()](http://docs.oracle.com/javase/8/docs/api/java/awt/Desktop.html#getDesktop--)
* [java.lang.System#getSecurityManager()](http://docs.oracle.com/javase/8/docs/api/java/lang/System.html#getSecurityManager--)

## [Structural patterns](http://en.wikipedia.org/wiki/Structural_pattern)

### [Adapter](http://en.wikipedia.org/wiki/Adapter_pattern) (recognizeable by creational methods taking an instance ofdifferentabstract/interface type and returning an implementation of own/another abstract/interface type whichdecorates/overridesthe given instance)

* [java.util.Arrays#asList()](http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#asList-T...-)
* [java.util.Collections#list()](https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#list-java.util.Enumeration-)
* [java.util.Collections#enumeration()](https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#enumeration-java.util.Collection-)
* [java.io.InputStreamReader(InputStream)](http://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html#InputStreamReader-java.io.InputStream-) (returns a Reader)
* [java.io.OutputStreamWriter(OutputStream)](http://docs.oracle.com/javase/8/docs/api/java/io/OutputStreamWriter.html#OutputStreamWriter-java.io.OutputStream-) (returns a Writer)
* [javax.xml.bind.annotation.adapters.XmlAdapter#marshal()](http://docs.oracle.com/javase/8/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#marshal-BoundType-) and [#unmarshal()](http://docs.oracle.com/javase/8/docs/api/javax/xml/bind/annotation/adapters/XmlAdapter.html#unmarshal-ValueType-)

### [Bridge](http://en.wikipedia.org/wiki/Bridge_pattern) (recognizeable by creational methods taking an instance ofdifferentabstract/interface type and returning an implementation of own abstract/interface type whichdelegates/usesthe 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 usesthem. The [java.util.Collections#newSetFromMap()](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#newSetFromMap-java.util.Map-) and [singletonXXX()](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#singleton-T-) methods however comes close.

### [Composite](http://en.wikipedia.org/wiki/Composite_pattern) (recognizeable by behavioral methods taking an instance ofsameabstract/interface type into a tree structure)

* [java.awt.Container#add(Component)](http://docs.oracle.com/javase/8/docs/api/java/awt/Container.html#add-java.awt.Component-) (practically all over Swing thus)
* [javax.faces.component.UIComponent#getChildren()](http://docs.oracle.com/javaee/7/api/javax/faces/component/UIComponent.html#getChildren--) (practically all over JSF UI thus)

### [Decorator](http://en.wikipedia.org/wiki/Decorator_pattern) (recognizeable by creational methods taking an instance ofsameabstract/interface type which adds additional behaviour)

* All subclasses of [java.io.InputStream](http://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html), [OutputStream](http://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html), [Reader](http://docs.oracle.com/javase/8/docs/api/java/io/Reader.html) and [Writer](http://docs.oracle.com/javase/8/docs/api/java/io/Writer.html) have a constructor taking an instance of same type.
* [java.util.Collections](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html), the [checkedXXX()](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#checkedCollection-java.util.Collection-java.lang.Class-), [synchronizedXXX()](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#synchronizedCollection-java.util.Collection-) and [unmodifiableXXX()](http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#unmodifiableCollection-java.util.Collection-)methods.
* [javax.servlet.http.HttpServletRequestWrapper](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequestWrapper.html) and [HttpServletResponseWrapper](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponseWrapper.html)

### [Facade](http://en.wikipedia.org/wiki/Facade_pattern) (recognizeable by behavioral methods which internally uses instances ofdifferentindependent abstract/interface types)

* [javax.faces.context.FacesContext](http://docs.oracle.com/javaee/7/api/javax/faces/context/FacesContext.html), it internally uses among others the abstract/interface types [LifeCycle](http://docs.oracle.com/javaee/7/api/javax/faces/lifecycle/Lifecycle.html), [ViewHandler](http://docs.oracle.com/javaee/7/api/javax/faces/application/ViewHandler.html), [NavigationHandler](http://docs.oracle.com/javaee/7/api/javax/faces/application/NavigationHandler.html) and many more without that the enduser has to worry about it (which are however overrideable by injection).
* [javax.faces.context.ExternalContext](http://docs.oracle.com/javaee/7/api/javax/faces/context/ExternalContext.html), which internally uses [ServletContext](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html), [HttpSession](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html), [HttpServletRequest](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html), [HttpServletResponse](http://docs.oracle.com/javaee/7/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/8/docs/api/java/lang/Integer.html#valueOf-int-) (also on [Boolean](http://docs.oracle.com/javase/8/docs/api/java/lang/Boolean.html#valueOf-boolean-), [Byte](http://docs.oracle.com/javase/8/docs/api/java/lang/Byte.html#valueOf-byte-), [Character](http://docs.oracle.com/javase/8/docs/api/java/lang/Character.html#valueOf-char-), [Short](http://docs.oracle.com/javase/8/docs/api/java/lang/Short.html#valueOf-short-), [Long](http://docs.oracle.com/javase/8/docs/api/java/lang/Long.html#valueOf-long-) and [BigDecimal](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html#valueOf-long-int-))

### [Proxy](http://en.wikipedia.org/wiki/Proxy_pattern) (recognizeable by creational methods which returns an implementation of given abstract/interface type which in turndelegates/usesadifferentimplementation of given abstract/interface type)

* [java.lang.reflect.Proxy](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Proxy.html)
* [java.rmi.\*](http://docs.oracle.com/javase/8/docs/api/java/rmi/package-summary.html)
* [javax.ejb.EJB](http://docs.oracle.com/javaee/7/api/javax/ejb/EJB.html) ([explanation here](https://stackoverflow.com/questions/25514361/when-using-ejb-does-each-managed-bean-get-its-own-ejb-instance))
* [javax.inject.Inject](http://docs.oracle.com/javaee/7/api/javax/inject/Inject.html) ([explanation here](https://stackoverflow.com/questions/29651008/field-getobj-returns-all-nulls-on-injected-cdi-managed-beans-while-manually-i/29672591#29672591))
* [javax.persistence.PersistenceContext](http://docs.oracle.com/javaee/7/api/javax/persistence/PersistenceContext.html)

## [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 inanotherimplementation ofsameabstract/interface type in a queue)

* [java.util.logging.Logger#log()](http://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html#log-java.util.logging.Level-java.lang.String-)
* [javax.servlet.Filter#doFilter()](http://docs.oracle.com/javaee/7/api/javax/servlet/Filter.html#doFilter-javax.servlet.ServletRequest-javax.servlet.ServletResponse-javax.servlet.FilterChain-)

### [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 adifferentabstract/interface type which has beenencapsulatedby the command implementation during its creation)

* All implementations of [java.lang.Runnable](http://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
* All implementations of [javax.swing.Action](http://docs.oracle.com/javase/8/docs/api/javax/swing/Action.html)

### [Interpreter](http://en.wikipedia.org/wiki/Interpreter_pattern) (recognizeable by behavioral methods returning astructurallydifferent 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/8/docs/api/java/util/regex/Pattern.html)
* [java.text.Normalizer](http://docs.oracle.com/javase/8/docs/api/java/text/Normalizer.html)
* All subclasses of [java.text.Format](http://docs.oracle.com/javase/8/docs/api/java/text/Format.html)
* All subclasses of [javax.el.ELResolver](http://docs.oracle.com/javaee/7/api/javax/el/ELResolver.html)

### [Iterator](http://en.wikipedia.org/wiki/Iterator_pattern) (recognizeable by behavioral methods sequentially returning instances of adifferenttype from a queue)

* All implementations of [java.util.Iterator](http://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html) (thus among others also [java.util.Scanner](http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html)!).
* All implementations of [java.util.Enumeration](http://docs.oracle.com/javase/8/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/8/docs/api/java/util/Timer.html) (all scheduleXXX() methods)
* [java.util.concurrent.Executor#execute()](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute-java.lang.Runnable-)
* [java.util.concurrent.ExecutorService](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html) (the invokeXXX() and submit() methods)
* [java.util.concurrent.ScheduledExecutorService](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledExecutorService.html) (all scheduleXXX() methods)
* [java.lang.reflect.Method#invoke()](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Method.html#invoke-java.lang.Object-java.lang.Object...-)

### [Memento](http://en.wikipedia.org/wiki/Memento_pattern) (recognizeable by behavioral methods which internally changes the state of thewholeinstance)

* [java.util.Date](http://docs.oracle.com/javase/8/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/8/docs/api/java/io/Serializable.html)
* All implementations of [javax.faces.component.StateHolder](http://docs.oracle.com/javaee/7/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 ofanotherabstract/interface type, depending on own state)

* [java.util.Observer](http://docs.oracle.com/javase/8/docs/api/java/util/Observer.html)/[java.util.Observable](http://docs.oracle.com/javase/8/docs/api/java/util/Observable.html) (rarely used in real world though)
* All implementations of [java.util.EventListener](http://docs.oracle.com/javase/8/docs/api/java/util/EventListener.html) (practically all over Swing thus)
* [javax.servlet.http.HttpSessionBindingListener](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSessionBindingListener.html)
* [javax.servlet.http.HttpSessionAttributeListener](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSessionAttributeListener.html)
* [javax.faces.event.PhaseListener](http://docs.oracle.com/javaee/7/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/7/api/javax/faces/lifecycle/Lifecycle.html#execute-javax.faces.context.FacesContext-) (controlled by [FacesServlet](http://docs.oracle.com/javaee/7/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 adifferentabstract/interface type which has beenpassed-inas method argument into the strategy implementation)

* [java.util.Comparator#compare()](http://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html#compare-T-T-), executed by among others Collections#sort().
* [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/7/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/7/api/javax/servlet/Filter.html#doFilter-javax.servlet.ServletRequest-javax.servlet.ServletResponse-javax.servlet.FilterChain-)

### [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/8/docs/api/java/io/InputStream.html), [java.io.OutputStream](http://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html), [java.io.Reader](http://docs.oracle.com/javase/8/docs/api/java/io/Reader.html)and [java.io.Writer](http://docs.oracle.com/javase/8/docs/api/java/io/Writer.html).
* All non-abstract methods of [java.util.AbstractList](http://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html), [java.util.AbstractSet](http://docs.oracle.com/javase/8/docs/api/java/util/AbstractSet.html) and [java.util.AbstractMap](http://docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.html).
* [javax.servlet.http.HttpServlet](http://docs.oracle.com/javaee/7/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 twodifferentabstract/interface types which has methods definied which takes each theotherabstract/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/8/docs/api/javax/lang/model/element/AnnotationValue.html) and [AnnotationValueVisitor](http://docs.oracle.com/javase/8/docs/api/javax/lang/model/element/AnnotationValueVisitor.html)
* [javax.lang.model.element.Element](http://docs.oracle.com/javase/8/docs/api/javax/lang/model/element/Element.html) and [ElementVisitor](http://docs.oracle.com/javase/8/docs/api/javax/lang/model/element/ElementVisitor.html)
* [javax.lang.model.type.TypeMirror](http://docs.oracle.com/javase/8/docs/api/javax/lang/model/type/TypeMirror.html) and [TypeVisitor](http://docs.oracle.com/javase/8/docs/api/javax/lang/model/type/TypeVisitor.html)
* [java.nio.file.FileVisitor](http://docs.oracle.com/javase/8/docs/api/java/nio/file/FileVisitor.html) and [SimpleFileVisitor](http://docs.oracle.com/javase/8/docs/api/java/nio/file/SimpleFileVisitor.html)
* [javax.faces.component.visit.VisitContext](http://docs.oracle.com/javaee/7/api/javax/faces/component/visit/VisitContext.html) and [VisitCallback](http://docs.oracle.com/javaee/7/api/javax/faces/component/visit/VisitCallback.html)

<https://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns-in-javas-core-libraries/2707195#2707195>

<https://stackoverflow.com/questions/11553804/design-patterns-with-real-time-example>

[**1. The observer pattern**](http://www.vogella.com/tutorials/DesignPatternObserver/article.html#observer)

[**1.1. Definition**](http://www.vogella.com/tutorials/DesignPatternObserver/article.html#observer_definition)

The *observer* pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

The object which is being watched is called the *subject*. The objects which are watching the state changes are called *observers* or *listeners*.

The observer pattern is very common in Java. For example, you can define a listener for a button in a user interface. If the button is selected, the listener is notified and performs a certain action.

But the observer pattern is not limited to single user interface components. For example, you could have a part A in your application which displays the current temperature.

Another part B displays a green light if the temperature is above 20 degree celsius. To react to changes in the temperature, part B registers itself as a listener to Part A.

If the temperature in part A is changed, an event is triggered. This event is sent to all registered listeners, as, for example, part B. Part B receives the changed data and can adjust its display.

**Advantage of Observer Design Pattern in Java:**

Main advantage is **loose coupling** between objects called observer and observable. The subject only know the list of observers it don’t care about how they have their implementation.All the observers are notified by subject in a single event call as **Broadcast communication**

**Disadvantage of Observer Design Pattern in Java:**

          The disadvantage is that the sometime if any problem comes, [debugging](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html) becomes very difficult because flow of control is implicitly between **observers** and **observable** we can predict that now observer is going to fire and if there is chain between observers then debugging become more complex.

          Another issue is Memory management because subject will hold all the reference of all the observers if we not unregister the object it can create the memory issue.

# Strategy Pattern