Façade vs. Mediator

Asked 15 years, 11 months ago Modified 1 year, 11 months ago

Viewed 28k times

96

I've been researching the difference between these two patterns.

I understand that facade encapsulates access to a sub system and mediator encapsulates the interactions between components.

I understand that sub system components are not aware of the facade, where as components are obviously aware of the mediator.

I'm currently using a facade for encapsulating the method of retrieving configuration information, e.g. App.Config, user setting stored in SQL, Assembly info, etc, and a mediator for navigation between different windows forms.

However, most sites point out that the mediator “adds functionality”. What do they mean by this? How does mediator add functionality?

design-patterns **facade** **mediator**

Share

Improve this question

edited Jan 20, 2014 at 13:13

Sam R.

16.4k ● 14 ● 72 ● 125

Follow

asked Jan 27, 2009 at 0:52

Tired

8 Answers

Sorted by: Highest score (default)

118

> *...most sites point out that the mediator “adds functionality”...*

The **facade** only exposes the existing functionality from a different perspective.

The **mediator** "adds" functionality because it combines different existing functionality to create a new one.

Take the following example:

You have a logging system. From that logging system, you can either log to a file, to a socket, or to a database.

Using the facade design pattern you would "hide" all the relationships from existing functionality behind a single "interface" the one that the facade exposes.

Client code:

```
Logger logger = new Logger();
logger.initLogger("someLogger");
logger.debug("message");
```

The implementation may involve the interaction of many objects. But at the end, the functionality already exists. Probably the "debug" method is implemented as follows:

Implementation:

```
 class Logger {

      private LoggerImpl internalLogger;
      private LoggerManager manager;

      public void initLogger( String loggerName )
{
          this.internalLogger = manager.getLogger(
loggerName );
      }

      public void debug( String message ) {
          this.internalLogger.debug( message );
      }
 }
```

The functionality already exists. The facade only hides it. In this hypothetical case, the LoggerManager handles the creation of the correct logger, and the LoggerImpl is a package-private object that has the "debug" method. This way the Facade is not adding functionality it is just delegating to some existing objects.

On the other hand the mediator adds new functionality by combining different objects.

Same Client code:

```
 Logger logger = new Logger();
 logger.initLogger("someLogger");
```

```
 logger.debug("message");
```

Implementation:

```
 class Logger {

      private java.io.PrintStream out;
      private java.net.Socket client;
      private java.sql.Connection dbConnection;
      private String loggerName;


      public void initLogger( String loggerName )
{
               this.loggerName = loggerName;
               if ( loggerName == "someLogger" ) {
                    out = new PrintStream( new
File("app.log"));
               } else if ( loggerName ==
"serverLog" ) {
                    client = new
Socket("127.0.0.1", 1234 );
               } else if( loggerName == "dblog") {
                    dbConnection =
Class.forName()... .
               }

      }

      public void debug( String message ) {

               if ( loggerName == "someLogger" ) {
                    out.println( message );
               } else if ( loggerName ==
"serverLog" ) {
                    ObjectOutputStrewam oos =
                           new
ObjectOutputStrewam( client.getOutputStream());
                    oos.writeObject( message );
               } else if( loggerName == "dblog") {
                    Pstmt pstmt =
dbConnection.prepareStatment( LOG_SQL );
```

In this code, the mediator is the one that contains the business logic to create an appropriate "channel" to log and also to make the log into that channel. The mediator is "creating" the functionality.

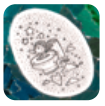
Of course, there are better ways to implement this using polymorphism, but the point here is to show how the mediator "adds" new functionality by combining existing functionality (in my sample didn't show very much sorry) but imagine the mediator, read from the database the remote host where to log, then creates a client and finally write to that client print stream the log message. This way the mediator would be "mediating" between the different objects.

Finally, the **facade** is a structural pattern, that is it describes the composition of the objects, while the **mediator** is a behavioral, that is, it describes the way the objects interact.

Share Improve this answer

Follow

edited Dec 24, 2022 at 4:18



starball

48.3k ● 28 ● 187 ● 848

answered Jan 27, 2009 at 4:53

OscarRyz

199k ● 119 ● 396 ● 573

1 Great explanation..I have a question related to this. The way ReentrantLock and AbstractQueueSynchronizer(AQS) are composed, is that fits into example of Facade pattern? I mean ReentrantLock only exposes the functionality of AQS

which is present inside it as subsystem. – AKS Sep 30, 2013 at 14:17

1 Does @RayTayek answer contradict to your answer? Your mediator's protocol is unidirectional, right? – Narek Nov 18, 2015 at 6:56

I could not find any site (including wikipedia) stating that a Mediator adds new functionality. Can you point some references ? – Vasu Sep 2, 2016 at 5:39

@developer here is a reference, look at the bottom. – Dario Fumagalli Mar 4, 2018 at 8:59

So, a Mediator is a Facade with combined functionality! – Rui Monteiro Jun 3 at 13:37

16

under related patterns, gof says: Facade (185) differs from Mediator in that it abstracts a subsystem of objects to provide a more convenient interface. Its protocol is unidirectional; that is, Facade objects make requests of the subsystem classes but not vice versa. In contrast, Mediator enables cooperative behavior that colleague objects don't or can't provide, and the protocol is multidirectional.

Share Improve this answer

Follow

answered Jun 13, 2015 at 7:25

Ray Tayek

10k ● 9 ● 54 ● 96

I'm using mediator to add log file functionality.

It works like this:

13

- Obj A tells the mediator it needs something done.
- The mediator sends the message to various client objects.
- Obj B does the thing Obj A needs, and sends an appropriate message back via the mediator.
- Meanwhile, Obj C is also sent both messages by the mediator, and logs the results. That way, we can get user statistics from the log files.
- Obj D could be an error checker as well, so that if Obj B responds that Obj A's request is impossible, Obj D could be the thing that reports that to the user. Errors can now be logged in a different file than regular activity, and could use some other means to behave (beeping, whatever) that Obj A shouldn't really concern itself with.

Share Improve this answer

Follow

answered Jan 27, 2009 at 1:03

mmr

14.9k ● 29 ● 97 ● 148

10

Take a simple analogy:

Facade: like a parking lot, when call

```
parkingLot.Out(car1);
```

mab be a simple chain works:

```
{
  car1.StartEngin();
  attendant.charge();
  car1.driverOut();
}
```

Mediator: like traffic light.

There are interactions between light and car,

and cars are controlled by it's state.

I though maybe this is the **mediator “adds functionality”**

And about the definition:

Facade's Type: **Structural**

Mediator's Type: **Behavioral**

facade more concerned about the **components** were **contained** in the **unified interface**,

and mediator concern **how** a set of objects **interact**.

Share Improve this answer Follow

edited May 31, 2013 at 0:58

answered May 31, 2013 at 0:38

yoyo

168 ● 2 ● 7

4

I thought the distinction was directional: facade is a one-way communication between client and facade; mediator can be a two-way conversation, with messages flowing back and forth between the client and mediator.

Share Improve this answer

Follow

answered Jan 27, 2009 at 2:07

duffymo

308k ● 46 ● 374 ● 565

> Sorry but that difference is actually wrong, mmr's answer is correct. Although I also believed the same as you when I first looked at them – Robert Gould Jan 27, 2009 at 2:59

3

From the "Design Patterns" book, the KEY of the Mediator pattern is described as follows: "It (a mediator) acts as a HUB of communication for widgets (i.e., 'a' group of interdependent objects)."

In other words, a mediator object is the sole superobject that knows all other objects in a group of collaborating objects and how they should interact with each other. All other objects should interact with the mediator object, instead of each other.

In contrast, a facade is a "unified interface" for a set of interfaces in a subsystem - for use by consumers of the subsystem - not among the components of the subsystem.

Share Improve this answer

Follow

answered Jan 12, 2011 at 4:09

Hwisung Im

31 ● 1

2

You can find details about Facade pattern in this SE question:

What is Facade Design Pattern?

`Facade` provides a simple and unified interface to complex system.

Real world example (**cleartrip flight+hotel booking**) is available in this post:

What is Facade Design Pattern?

Mediator pattern: Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

A real world example of Mesh network topology has been provided in below SE question:

Mediator Vs Observer Object-Oriented Design Patterns

Regarding your query on Mediator adds responsibility:

1. **Facade provides only interface to existing sub-systems**. Existing sub-systems are not aware of Facade class itself.

2. **Mediator knows about colleague objects**. It enables communication between different colleagues. In the example I have quoted in linked question, *ConcreteMediator*(*NetworkMediator*) sends notifications of register and unregister event of one colleague to all other colleagues.

Share Improve this answer Follow

edited Sep 25, 2017 at 18:25

answered Aug 8, 2016 at 16:09

Ravindra babu

38.9k ● 11 ● 256 ● 219

1

Both impose some kind of policy on another group of objects. **Facade** imposes the policy from above, and **Mediator** imposes the policy from below. The use of **Facade** is visible and constraining, while the use of **Mediator** is invisible and enabling.

The **Facade** pattern is used when you want to provide a simple and specific interface to a group of objects that has a complex and general interface.

The **Mediator** pattern also imposes policy. However, whereas Facade imposed its policy in a visible and constraining way, **Mediator** imposes its policies in a hidden and unconstrained way.

Agile Software Development, Principles, Patterns, and Practices Robert C. Martin.

Share Improve this answer Follow

edited Apr 7, 2019 at 5:05

answered May 16, 2018 at 12:07

Mohammad Akbari

4,746 ● 6 ● 49 ● 78