



Behavioral Design Patterns: Mediator

by Emmanouil Gkatzouras MVB · Nov. 29, 18 · Java Zone · Tutorial

Download Microservices for Java Developers: A hands-on introduction to frameworks and containers. Brought to you in partnership with Red Hat.

Previously, we had a look at the iterator pattern. The mediator pattern is very different in what it tries to achieve. It is one of the behavioral patterns and its purpose is to alter the way objects communicate with each other. Instead of the objects communicating with each other directly, the mediator will handle the objects' interactions.

For example, imagine the scenario of a financial exchange. You do want to trade and buy but you don't buy directly from the one that makes the offer. Instead, the exchange is in the middle, in order for you to make the transaction.

People would like to sell and buy. This shall be facilitated by the exchange. First, you have the `Order` object.

```
1
2 package com.gkatzouras.design.behavioural.mediator;
3
4 public class Order {
5
6     private String stock;
7     private Integer quantity;
8     private Double price;
9
10    public String getStock() {
11        return stock;
12    }
13
14    public void setStock(String stock) {
15        this.stock = stock;
16    }
17
18    public Integer getQuantity() {
19        return quantity;
20    }
21
22    public void setQuantity(Integer quantity) {
23        this.quantity = quantity;
24    }
25
```

```
26     public Double getPrice() {
27         return price;
28     }
29
30     public void setPrice(Double price) {
31         this.price = price;
32     }
33
34 }
```

The next object will be the `FinancialEntity` that sells the stocks.

```
1
2 package com.gkatzioura.design.behavioural.mediator;
3
4 public class FinancialEntity {
5
6     public boolean sell(Order order) {
7
8         /**
9          * Supposing the sale was successful return true
10          */
11         return true;
12     }
13
14 }
```

Then, we create the `Exchange` object. We won't get further into commissions but imagine that things can be way more complex. The `Exchange` is actually our mediator.

```
1
2 package com.gkatzioura.design.behavioural.mediator;
3
4 public class Exchange {
5
6     private FinancialEntity financialEntity;
7
8     public Exchange(FinancialEntity financialEntity) {
9         this.financialEntity = financialEntity;
10    }
11
12    public void serve(Order order) {
13
14        /**
15         * Choose the financial entity suitable for the order
16         */
17    }
```

```
16         */
17         financialEntity.sell(order);
18     }
19
20 }
```

And the last step is creating the `Trader` object.

```
1
2 package com.gkatzioura.design.behavioural.mediator;
3
4 public class Trader {
5
6     private Exchange exchange;
7
8     public Trader(Exchange exchange) {
9         this.exchange = exchange;
10    }
11
12    public void buy(String stock,Integer quantity,Double price) {
13        Order order = new Order();
14        order.setStock(stock);
15        order.setQuantity(quantity);
16        order.setPrice(price);
17        exchange.serve(order);
18    }
19
20 }
```

As you can see, the `Trader` object is not interacting directly with the financial entity that provides the stocks.

Let's put them all together in the main class.

```
1
2 package com.gkatzioura.design.behavioural.mediator;
3
4 public class Mediator {
5
6     public static void main(String[] args) {
7
8         final FinancialEntity financialEntity = new FinancialEntity();
9         final Exchange exchange = new Exchange(financialEntity);
10        Trader trader = new Trader(exchange);
11        trader.buy("stock_a",2,32.2d);
12    }
13 }
```

That's it, you just used the mediator pattern for an exchange application! You can also find the source code on [GitHub](#).

Download Building Reactive Microservices in Java: Asynchronous and Event-Based Application Design. Brought to you in partnership with Red Hat.

Like This Article? Read More From DZone



Singleton Design Pattern: Making Singleton More Effective in Java



Levels of Abstraction



Why InputStream Design Is Wrong



**Free DZone Refcard
Java Containerization**

Topics: [JAVA](#) , [BEHAVIOUR](#) , [DESIGN PATTERN](#) , [MEDIATOR](#) , [CODE](#) , [TUTORIAL](#)

Published at DZone with permission of Emmanouil Gkatzouras , DZone MVB. [See the original article here.](#)

Opinions expressed by DZone contributors are their own.

Java Partner Resources

Deep insight into your code with IntelliJ IDEA.

atBrains

|

The Build vs. Buy Challenge: Insight Into a Hybrid Approach

Elissa

|

Free eBook: SCA Maturity Model - A Framework for Open Source Security and Compliance

Exera

|

Migrating to Microservice Databases

Red Hat Developer Program

|

Spring Core Skills: 5000 Ways to Build and Wire Your Spring Beans [Video]

by Marco Behler · Dec 17, 18 · Java Zone · Tutorial

Verify, standardize, and correct the Big 4 + more– name, email, phone and global addresses –

try our Data Quality APIs now at Melissa Developer Portal!

Having your Spring application set up is only the first step. Then, another question pops up. How are you going to wire your beans? Setter injection? Field injection? Constructor injection? Why would you want to use one over the other? Does that even make sense?

Find out how to wire your beans with the help of a fun and practical example in this episode.

Spring Core: 5000 ways to build and wire your Spring Bean...



Developers! Quickly and easily gain access to the tools and information you need! Explore, test and combine our data quality APIs at Melissa Developer Portal – home to tools that save time and boost revenue.

Like This Article? Read More From DZone



Spring Core Skills: Your First Spring Application [Video]



Spring Tips: Programmatic Bean Registration in Spring 5 [Video]




An Interview Question on Spring Singletons



Free DZone Refcard Java Containerization

Topics: SPRING, SPRING BEANS, BEANS, JAVA, TUTORIAL, VIDEO, SPRING TUTORIAL, BEGINNER JAVA TUTORIAL, BEGINNER SPRING TUTORIAL

Published at DZone with permission of Marco Behler . [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.