Visitor Design Pattern in Java

APRIL 2, 2018 BY PANKAJ — 29 COMMENTS

Visitor Design Pattern is one of the behavioral design pattern.

Table of Contents [hide]

Visitor Design Pattern

Visitor pattern is used when we have to perform an operation on a group of similar kind of Objects. With the help of visitor pattern, we can move the operational logic from the objects to another class.

For example, think of a Shopping cart where we can add different type of items (Elements). When we click on checkout button, it calculates the total amount to be paid. Now we can have the calculation logic in item classes or we can move out this logic to another class using visitor pattern. Let's implement this in our example of visitor pattern.

Visitor Design Pattern Java Example

To implement visitor pattern, first of all we will create different type of items (Elements) to be used in shopping cart.

`ItemElement.java`

```
package com.journaldev.design.visitor;

public interface ItemElement {

	public int accept(ShoppingCartVisitor visitor);
}
```

Notice that accept method takes Visitor argument. We can have some other methods also specific for items but for simplicity I am not going into that much detail and focusing on visitor pattern only.

Let's create some concrete classes for different types of items.

`Book.java`

```
package com.journaldev.design.visitor;

public class Book implements ItemElement {

	private int price;
	private String isbnNumber;

	public Book(int cost, String isbn){
		this.price=cost;
		this.isbnNumber=isbn;
	}

	public int getPrice() {
		return price;
	}

	public String getIsbnNumber() {
		return isbnNumber;
	}

	@Override
```

Fruit.java

```
package com.journaldev.design.visitor;

public class Fruit implements ItemElement {

        private int pricePerKg;
        private int weight;
        private String name;

        public Fruit(int priceKg, int wt, String nm){
                this.pricePerKg=priceKg;
                this.weight=wt;
                this.name = nm;
        }

        public int getPricePerKg() {
                return pricePerKg;
        }


        public int getWeight() {
                return weight;
        }
```

Notice the implementation of accept() method in concrete classes, its calling visit() method of Visitor and passing itself as argument.

We have visit() method for different type of items in Visitor interface that will be implemented by concrete visitor class.

ShoppingCartVisitor.java

```
package com.journaldev.design.visitor;

public interface ShoppingCartVisitor {

        int visit(Book book);
        int visit(Fruit fruit);
}
```

Now we will implement visitor interface and every item will have it's own logic to calculate the cost.

`ShoppingCartVisitorImpl.java`

```
package com.journaldev.design.visitor;

public class ShoppingCartVisitorImpl implements ShoppingCartVisitor {

        @Override
        public int visit(Book book) {
                int cost=0;
                //apply 5$ discount if book price is greater than 50
                if(book.getPrice() > 50){
                        cost = book.getPrice()-5;
                }else cost = book.getPrice();
                System.out.println("Book ISBN::"+book.getIsbnNumber() + " cost
="+cost);
                return cost;
        }

        @Override
        public int visit(Fruit fruit) {
                int cost = fruit.getPricePerKg()*fruit.getWeight();
                System.out.println(fruit.getName() + " cost = "+cost);
                return cost;
        }
```

Lets see how we can use visitor pattern example in client applications.

`ShoppingCartClient.java`

```
package com.journaldev.design.visitor;

public class ShoppingCartClient {

        public static void main(String[] args) {
                ItemElement[] items = new ItemElement[]{new Book(20, "1234"),new
Book(100, "5678"),
                                new Fruit(10, 2, "Banana"), new Fruit(5, 5,
"Apple")};

                int total = calculatePrice(items);
                System.out.println("Total Cost = "+total);
        }

        private static int calculatePrice(ItemElement[] items) {
                ShoppingCartVisitor visitor = new ShoppingCartVisitorImpl();
                int sum=0;
                for(ItemElement item : items){
                        sum = sum + item.accept(visitor);
                }
                return sum;
        }
```

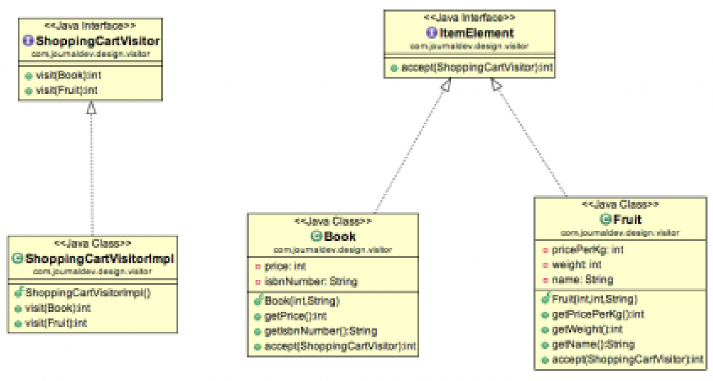
When we run above visitor pattern client program, we get following output.

```
Book ISBN::1234 cost =20
Book ISBN::5678 cost =95
Banana cost = 20
Apple cost = 25
Total Cost = 160
```

Notice that implementation if accept() method in all the items are same but it can be different, for example there can be logic to check if item is free then don't call the visit() method at all.

Visitor Design Pattern Class Diagram

Class diagram for our visitor design pattern implementation is:



Visitor Pattern Benefits

The benefit of this pattern is that if the logic of operation changes, then we need to make change only in the visitor implementation rather than doing it in all the item classes.

Another benefit is that adding a new item to the system is easy, it will require change only in visitor interface and implementation and existing item classes will not be affected.

Visitor Pattern Limitations

The drawback of visitor pattern is that we should know the return type of visit() methods at the time of designing otherwise we will have to change the interface and all of its implementations. Another drawback is that if there are too many implementations of visitor interface, it makes it hard to extend.

Thats all for visitor design pattern, let me know if I have missed anything. Please share it with others also if you liked it.

« PREVIOUS
Template Method Design Pattern in Java

NEXT »
Java Design Patterns – Example Tutorial

FILED UNDER: DESIGN PATTERNS

Comments

shashi says

DECEMBER 22, 2016 AT 7:47 AM

One can write a generic visit() method …that way you don;t have to change the interface each time a new ItemElement is created.

```
@Override
public int visit(ItemElement item) {
if (item instanceof Book) {
Book book = (Book) item;
int cost = 0;
// apply 5$ discount if book price is greater than 50
if (book.getPrice() > 50) {
cost = book.getPrice() – 5;
} else
cost = book.getPrice();
System.out.println("Book ISBN::" + book.getIsbnNumber()
+ " cost = " + cost);
return cost;
} else if (item instanceof Fruit) {
Fruit fruit = (Fruit) item;
int cost = fruit.getPricePerKg() * fruit.getWeight();
System.out.println(fruit.getName() + " cost = " + cost);
return cost;
}
// default return nothing
return 0;
}
```

Reply

robothy says

OCTOBER 25, 2016 AT 1:03 AM

It's a nice example of Visitor Design Pattern, I hava some personal opinions about "Visitor Pattern Benefits". In visitor design pattern, adding a new operation is easy, not adding an item. If we want add a operation such as get summary weight(if possible), we just need to add an ShoppingCartVisitor's implementation. However, adding an item will add a file and modify two files, I think it's not easy .

Reply

yami says

SEPTEMBER 28, 2016 AT 1:58 AM

i'm not sure how :

"`

return visitor.visit(this);

"`

gives us ShoppingCartVisitorImpl method.

Does it automaticly find implementation of interface and uses it ?

Reply

just me says

JULY 11, 2018 AT 11:37 PM

In runtime the jvm knows it's implementation thanks to dynamic binding.

Reply

Parmod says

FEBRUARY 23, 2016 AT 9:10 PM

Thanks.. you explained with great details

Reply

vqa nguyen says

NOVEMBER 26, 2015 AT 3:53 AM

For me, just do simply like this

public interface ItemElement {

public int prix ();

}

public static class Fruit implements ItemElement {

….

@Override

public int prix() {

int cost = getPricePerKg()*getWeight();

System.out.println(getName() + " cost = "+cost);

return cost;

}

}

public static class Book implements ItemElement {

…..
@Override
public int prix() {
int cost=0;
//apply 5$ discount if book price is greater than 50
if(getPrice() > 50){
cost = getPrice()-5;
}else cost = getPrice();
System.out.println("Book ISBN::"+getIsbnNumber() + " cost ="+cost);
return cost;
}
}
The question is what is the advantage of using VisitorPattern ???
Reply

Mark Valley says
FEBRUARY 26, 2016 AT 9:27 AM
This example is a little bit simple to show the pattern applicability.
Image you have to add the freight to the price calculation. Freight price will vary according to the region. These rules should not be calculated inside the Product classes as they don't cara about freight costs.
One possible solution would be the creation of multiple concrete visitors, one for each region. These visitors will hold the responsability to calculate the freight costs and return the total price.
public SouthRegionVisitor implements ShoppingCartVisitor {
public int visit (Book book) {
int freightCost = calculateFreightForSouthRegion(book.getWeight());
return book.price + freightCost;
}
}
public NorthRegionVisitor implements ShoppingCartVisitor { // and so on
Reply

Techguynarendra says
APRIL 1, 2018 AT 7:19 PM
Nice point
Reply

MaikoID says

AUGUST 29, 2015 AT 3:40 PM

Hi, I've a doubt why the accept methods from Book and Fruit need to have a ShoppingCartVisitor instead of a Visitor one? A ShoppingCartVisitor needs to implements the Visitor interface hence we can allow the polymorphism do its job. If we insist to only use the ShoppingCartVisitor we lose the hability to accept Visitor for more than one Class. Am I right?

BR

Reply

HieuDT says

DECEMBER 13, 2015 AT 7:35 PM

i total agree with you.

Reply

Jay says

MAY 28, 2015 AT 12:17 AM

I would like to visit() each of the below comments (in a loop) and accept() what they have said and collate it for you!!!!

Thanks.

Reply

Vijay says

MAY 23, 2015 AT 1:43 AM

The best explanation of Visitor Pattern I found in net. Great Work

Reply

Maurice Kingsley says

MAY 5, 2015 AT 8:07 AM

Thank you for this simple but all explaining example of a fairly neglected Java pattern. Good job!

Reply

esvet says

APRIL 30, 2015 AT 2:51 AM

I agree with my forecommenters – good Job and the best Visitor explanation I 've found found in the net so gar.. Cheers Pankaj!

Reply

shahnawaz says

JANUARY 27, 2015 AT 1:41 PM

Nice example..

Reply

jagdish says

JANUARY 26, 2015 AT 4:17 AM

I have been referring to this post, whenever I need to refresh my memory on patterns. And nowhere can I find a better suitable explanation and example then here. And the examples are real world which I can relate to a project than the abstract examples elsewhere.

Reply

maxat says

JANUARY 24, 2015 AT 9:11 PM

Better explanation than on Wikipedia. Good job.

Reply

Dhanasekar says

JANUARY 19, 2015 AT 2:50 AM

This example is very good and self explanatory.

Reply

ravi says

NOVEMBER 19, 2014 AT 9:14 AM

Good Job Pankaj.

Reply

Abhishek says

SEPTEMBER 16, 2014 AT 2:07 PM

Nice Explanation. I liked your work.

Good Job!!

Reply

Anjaneyulu says

SEPTEMBER 14, 2014 AT 5:22 AM

Thanks a lot. You really made us understand this clearly.

Reply

Vijay says

SEPTEMBER 3, 2014 AT 12:51 AM

Good clarity !

Reply

Nici says

JUNE 3, 2014 AT 12:48 PM

Thanks a lot for explaining all the design patterns! Your examples helped me understand it a whole lot better. Keep up the good work!

Reply

mehdi ghadimi says

APRIL 27, 2014 AT 11:23 PM

very good. tnx

Reply

Mbuku Ditutala says

MARCH 28, 2014 AT 3:19 PM

I've been reading lot's of stuffs from you, meaningly on C# Remoting, but never made any commnet. I just have to say thanks for all your efforts to make programming easy for many, like me. Keep Cool and may the almighty God bless you.

Reply

Peter Yan says

OCTOBER 21, 2014 AT 11:58 PM

Very nice post, I can't love you more.

Reply

Anand says

FEBRUARY 1, 2014 AT 3:40 PM

Very nice explanation

Reply

Chavda jaydeep says

SEPTEMBER 24, 2013 AT 5:19 PM

thnx….this is very help full for me bcz i am developing my final year project for online cloth store so….thnxxxxxxxxxxxxx

Reply

Anjaiah says

JULY 1, 2015 AT 8:55 AM

Visitor design pattern is applicable to any Java API related class or interface

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

DESIGN PATTERNS TUTORIAL

JAVA FILE APIs

- CONVERT
- PRINT
- CREATE
- COMBINE
- MODIFY

Files from your Code

100% Native Java

Try FREE Today!

ASPOSE File Format APIs