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 elements(items) to be used in shopping cart

\*/

interface ItemElement{

public int accept(ShoppingCartVisitor visitor);

}

/\*

\* Notice that accept method takes visitor argument we can have some other methods also specific for items for simplicity

\* i am not going into that much detail and focusing on visitor pattern only.

\* Lets create some concrete classes for different types of items

\*/

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 this.price;

}

public String getIsbnNumber() {

return this.isbnNumber;

}

@Override

public int accept(ShoppingCartVisitor visitor) {

// TODO Auto-generated method stub

return visitor.visit(this);

}

}

class Fruit implements ItemElement{

private int pricePerKg;

private int weight;

private String name;

public Fruit(int pricePerKg, int weight, String name) {

this.pricePerKg = pricePerKg;

this.weight = weight;

this.name = name;

}

public int getPricePerKg() {

return pricePerKg;

}

public int getWeight() {

return weight;

}

public String getName() {

return name;

}

@Override

public int accept(ShoppingCartVisitor visitor) {

return visitor.visit(this);

}

}

/\*

\* notice that implementation of accept method in concrete classes , its called visit() method of visitor and passing itself as

\* argument

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

\*/

interface ShoppingCartVisitor{

int visit(Book book);

int visit(Fruit fruit);

}

/\*

\* Now we will implement visitor interface and every item will have its own logic to calculate the cost.

\*/

class Customer implements ShoppingCartVisitor {

//ShoppingCartVisitorImplementation = customer

@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;

}

}

//test pattern

public class TestVisitorPattern {

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[] elements) {

Customer visitor = new Customer();

int sum = 0;

for(ItemElement item : elements) {

sum = sum + item.accept(visitor);

}

return sum;

}

}

/\*

\* source : https://www.journaldev.com/1769/visitor-design-pattern-java

\*/

Output :

Book ISBN :: 1234 cost = 20

Book ISBN :: 5678 cost = 95

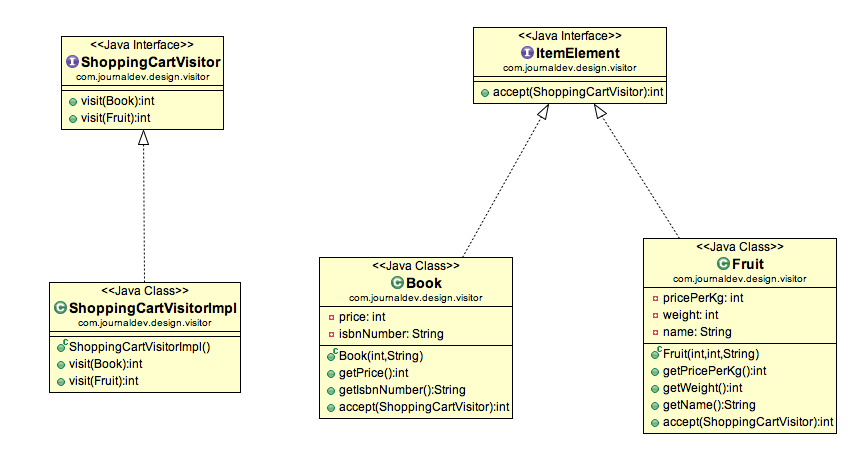
Banana cost = 20

Apple cost = 25

total cost = 160

### **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.