Patró Visitor

Enginyeria del Software

David Moreno Oliveras

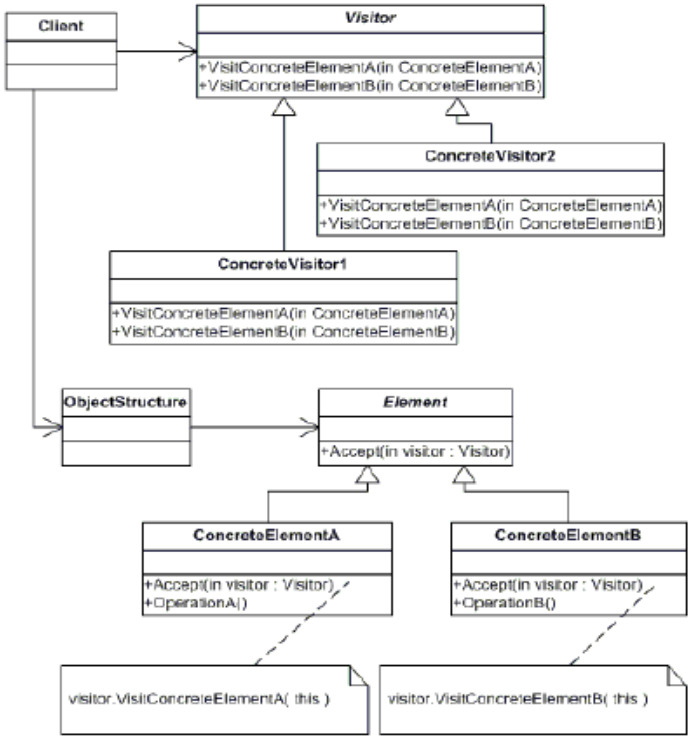
Definició

- Representa una operació que s'ha de fer sobre els elements d'una estructura d'objectes.
- Permet definir noves operacions sense modificar les classes dels elements sobre les que opera.
- Per a fer-ho, sugereix definir l'operació en una classe diferent, referida com a classe Visitor, separant així l'operació de la col.lecció d'objectes.

Visitor: Elements Bàsics

- **Visitor**: Declara una operació de visita per cada element concret de l'estructura d'objectes.
- **ConcreteVisitor**: Implementa cadasquna de les operacions declarades per Visitor.
- **Element**: Defineix l'operació que li permet acceptar la visita d'un Visitador
- **ConcreteElement**: Implementa l'operació d'accept que es limita a invocar el seu mètode corresponent del Visitor.
- **ObjectStructure**: Gestiona l'estructura d'objectes i pot oferir una interfície d'alt nivell per permetre als Visitor visitar els seus elements

Estructura General del Patró



Hi ha un conjunt de classes element que formen l'estructura d'un objecte. Cadasquna d'aquestes classes té un mètode ***Accept()*** que rep a l'objecte Visitor com argument.

El visitor és una interfície que té un mètode visitor diferent per cada classe element, de manera que tenim diferents implementacions de l'interficie visitor tals com:

VisitConcreteElementA,Visit ConcreteElementB, ect....tantes com elements hi hagi.

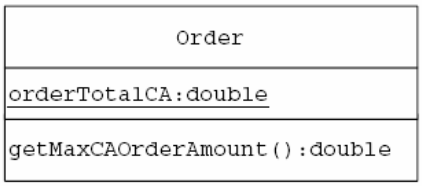
Exemple: Aplicació de diferents objectes Comanda

- Tenim diferents tipus de comandes de repartiment:
  - Comandes de fora d'Estats Units : "overseas order"
  - Comandes de California: "california order"
  - Comandes de fora de California: "non-california order"

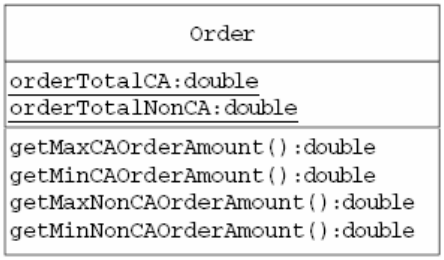
  Cadasquna de les quals, al fer repartiment, té diferents tipus de taxes que se li apliquen segons el tipus de transport utilitzat.

Disseny 1:

- Volem definir una operació "getMaxCAOrderAmount" per trobar la quantitat de diners total de les comandes de California, i definim una classe genèrica "Order"

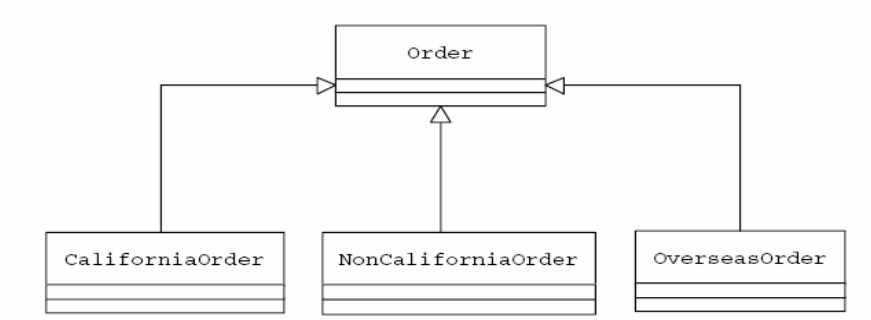


- orderTotalCA guarda aquesta quantitat i es va actualitzant si la quantitat entrada es major.
- Si volem afegir nous mètodes com "getCAOrderTotal", "getMinNonCAOrderAmount", ect...el codi de la classe s'ha de modificar per cada nova operació que hi afegim



Disseny 2: Subclasses

- Donat a que les comandes son de 3 tipus diferents podríem dissenyar 3 subclasses de la classe Comanda (Order),cadasquna de les quals representa un tipus diferent.
- Aquest nou disseny és millor i més pràctic però té 2 limitacions:
  - No és el lloc addïent per definir operacions en les que participen comandes de diferents tipus
  - Sempre que es defineixi una nova operació de la classe Comanda (Order), caldrà canviar el codi de la classe

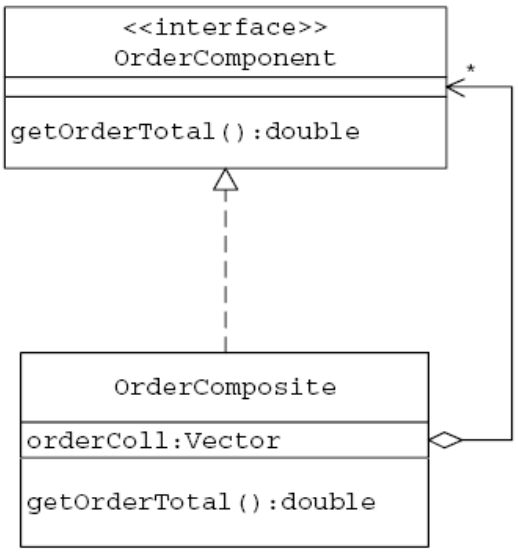


Disseny 3: Patró Composite

- Fent servir el patró composite, dissenyem una nova classe (*OrderComposite*) per mantenir la col.lecció i operar-hi a dins la col.lecció. Dins aquesta classe definim operacions dels diferents objectes comanda
- L'atribut *orderColl* serveix per

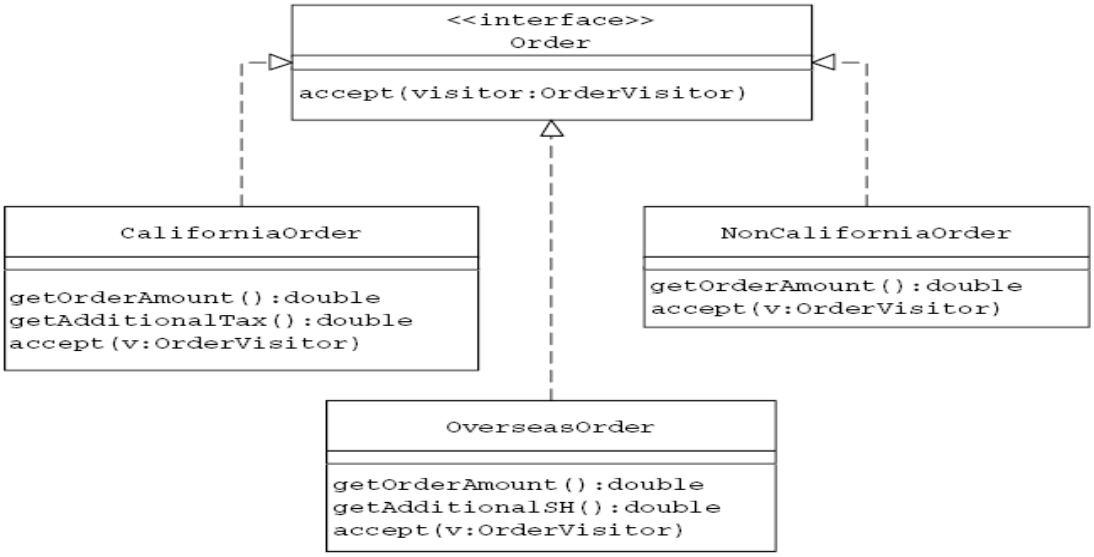
guardar diferents tipus de comandes i l'operació getOrderTotal itera per sumar els diferents imports.

-> Aquest disseny millora molt el 2 i permet definir i tractar operacions de diferents col.lecció d'objectes però requereix canviar la classe OrderComposite quan s'afegeix una nova operació.



Disseny 4: Patró Visitor

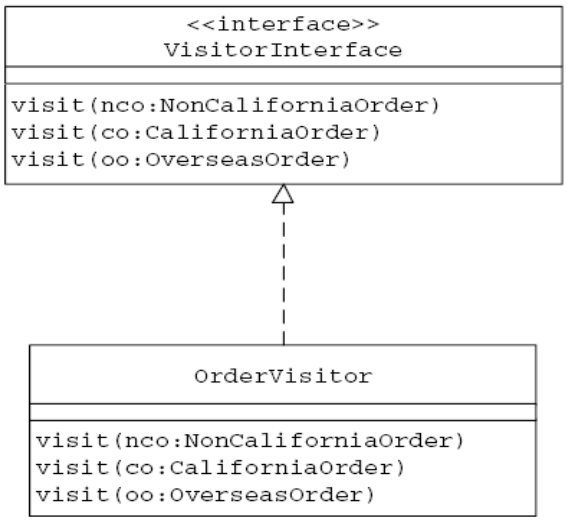
- Element: definim una classe interfície que tindrà 3 implementacions, una per cada tipus de comanda. Aquesta interfície té declarat un nou mètode *Accept(OrderVisitor).*
- Cada classe de tipus de comanda té implementat el seu mètode Accept(OrderVisitor)



Patró Visitor

- Visitor: Tenim la classe *VisitorInterface* que és la classe visitadora i que té declarats els mètodes visitadors visit(), un per cada classe.

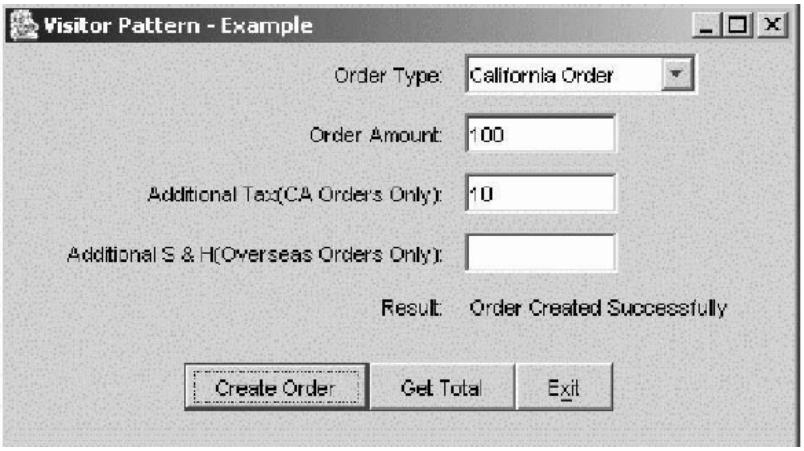
```
public interface VisitorInterface {
  public void visit(NonCaliforniaOrder nco);
  public void visit(CaliforniaOrder co);
  public void visit(OverseasOrder oo);
}
```



- Es defineix la classe *OrderVisitor per calcular la suma de totes les comandes*

Patró Visitor

- Client: hi ha la classe *OrderManager* que crea una instància OrderVisitor i permet a l'usuari crear comandes de diferents tipus.



Patró Visitor

- ***Fer una comanda:*** (Botó Get Order)
  - Es crea un objecte Comanda
  - S'invoca el mètode *Accept()* de l'objecte Comanda passant-li el tipus de visitador OrderVisitor (per exemple, comanda california)
  - Internament l'objecte Comanda crida el mètode *Visit()* de la seva classe corresponent OrderVisitor. Aquí podem calcular els imports necessaris.

- ***Obtenir Total:*** (Botó Get Total)
  - El client OrderManager invoca el mètode getOrderTotal del OrderVisitor corresponent (California Order en aquest cas).
  - La classe OrderVisitor retorna el valor guardat a la una variable anomenada *orderTotal*

Patró Visitor

```
class OrderVisitor implements VisitorInterface {
  private Vector orderObjList;
  private double orderTotal;
  public OrderVisitor() {
    orderObjList = new Vector();
  }
  public void visit(NonCaliforniaOrder inp_order) {
    orderTotal = orderTotal + inp_order.getOrderAmount();
  }
  public void visit(CaliforniaOrder inp_order) {
    orderTotal = orderTotal + inp_order.getOrderAmount() +
                 inp_order.getAdditionalTax();
  }
  public void visit(OverseasOrder inp_order) {
    orderTotal = orderTotal + inp_order.getOrderAmount() +
                 inp_order.getAdditionalSH();
  }
  public double getOrderTotal() {
    return orderTotal;
  }
}
```


Patró Visitor

Implementació Comanda de California

```
public class CaliforniaOrder implements Order {
  private double orderAmount;
  private double additionalTax;
  public CaliforniaOrder() {
  }
  public CaliforniaOrder(double inp_orderAmount,
      double inp_additionalTax) {
    orderAmount = inp_orderAmount;
    additionalTax = inp_additionalTax;
  }
  public double getOrderAmount() {
    return orderAmount;
  }
  public double getAdditionalTax() {
    return additionalTax;
  }
  public void accept(OrderVisitor v) {
    v.visit(this);
  }
}
```

Patró Visitor

