

Universitat de Lleida Actividad del Patrón Visitante

Ampliación de BBDD e Ingeniería del Software Grado en Ingeniería Informática

> Luis Carrasquer Sampietro Dídac Cayuela Dolcet Aleix Drudis Mola

25 de mayo de 2024

Índex

Introducció	3
Implementació del Patró Visitant per Afegir Noves Operacions (Apartat a)	4
Item.java	4
Pack.java	5
Visitant per Modificar Preus per sota d'un MinPrice (Apartat b)	6
Visitant per Comptar Items per sota d'un MinPrice (Apartat c)	8
Observacions	10
ltem.java	10
Pack.java	11
Conclusió	13
Solució Dídac	14
Solució Luis	19
Solució Aleix	24

Introducció

Aquest informe descrivim les decisions de disseny preses per implementar el patró visitant en una jerarquia de productes que inclouen Items i Packs. El problema a resoldre consisteix en afegir operacions als productes sense modificar les seves classes, facilitant així l'extensibilitat i mantenibilitat del codi. Els objectius específics inclouen l'actualització dels preus dels Items que estan per sota d'un preu mínim i el còmput del nombre d'Items que es troben en aquesta situació, tot implementant aquestes funcionalitats mitjançant el patró visitant.

Implementació del Patró Visitant per Afegir Noves Operacions (Apartat a)

El primer pas per implementar el patró visitant ha consistit en definir una interfície ProductVisitor. Aquesta interfície declara mètodes de visita per cada tipus de producte (Item i Pack). Aquest enfocament permet que qualsevol visitant pugui operar tant en Items com en Packs de manera transparent. La interfície ProductVisitor es defineix a continuació:

```
public interface ProductVisitor {
    void visit(Item item);
    void visit(Pack pack);
}
```

També hem fet unes modificacions a la interfície Product, hem afegit el mètode Accept que implementarem a continuació.

```
import java.math.BigDecimal;
import java.util.HashSet;
import java.util.Set;

public interface Product {
    public BigDecimal getPrice();
    public void checkMinItemPrice(BigDecimal minItemPrice);
    long countItemsBelowMinPrice(BigDecimal minPrice, Set<Product> visitedProducts);

    default long countItemsBelowMinPrice(BigDecimal minPrice) {
        Set<Product> visited = new HashSet<>();
        return countItemsBelowMinPrice(minPrice, visited);
    }

    void accept(ProductVisitor visitor);
}
```

Les classes Item i Pack implementen el mètode accept per acceptar un visitant i permetre que aquest executi l'operació corresponent. Afegint el següent mètode a cada classe, assegurem que aquestes poden rebre visitants:

```
Item.java
@Override
public void accept(ProductVisitor visitor) {
    visitor.visit(this);
}
```

```
Pack.java
@Override
public void accept(ProductVisitor visitor) {
    visitor.visit(this);
}
```

Aquest enfocament permet afegir noves operacions simplement implementant nous visitants sense haver de modificar les classes Item i Pack, mantenint així el codi obert per a l'extensió però tancat per a la modificació, seguint el principi d'OCP (Open/Closed Principle).

Visitant per Modificar Preus per sota d'un MinPrice (Apartat b)

Per implementar un visitant que actualitzi els preus dels Items per sota d'un preu mínim (minPrice), hem dissenyat la classe MinPriceUpdaterVisitor. Aquest visitant encapsula la lògica d'actualització de preus i garanteix que aquesta es realitzi de manera coherent en tota la jerarquia de productes.

El MinPriceUpdaterVisitor conté una variable minPrice que s'inicialitza al seu constructor. Aquest visitant actualitza els preus dels Items si estan per sota del minPrice. El codi del visitant és el següent:

```
import java.math.BigDecimal;
import java.util.HashSet;
import java.util.Set;
public class CountItemsBelowMinPriceVisitor implements ProductVisitor
    private BigDecimal minPrice;
    private int count;
   private Set<Product> visited;
    private CountItemsBelowMinPriceVisitor(BigDecimal minPrice) {
        if (minPrice.compareTo(BigDecimal.ZERO) <= 0) {</pre>
            throw new IllegalArgumentException ("Minimum price must be
greater than zero");
        this.minPrice = minPrice;
        this.count = 0;
        this.visited = new HashSet<>();
    }
    @Override
    public void visit(Item item) {
        if (!visited.contains(item)) {
            visited.add(item);
            if (item.getPrice().compareTo(minPrice) < 0) {</pre>
                count++;
            }
        }
    }
    @Override
    public void visit(Pack pack) {
        if (!visited.contains(pack)) {
            visited.add(pack);
            for (Product product : pack.getProducts()) {
                product.accept(this);
        }
    }
    public int getCount() {
       return count;
    }
```

Es defineix un mètode estàtic per iniciar el procés de visita, validant prèviament el valor de minPrice per assegurar que és positiu. Si el minPrice no és positiu, es llença una excepció IllegalArgumentException. Aquest mètode estàtic s'utilitza per iniciar la visita sobre un producte donat:

```
public static void updatePrices(Product product, BigDecimal minPrice)
{
    MinPriceUpdaterVisitor visitor = new
MinPriceUpdaterVisitor(minPrice);
    product.accept(visitor);
}
```

Aquest enfocament permet garantir que tots els Items dins d'un Pack (i de qualsevol estructura recursiva de Packs) s'actualitzin correctament si estan per sota del minPrice.

Ens agradaria deixar clar que el mètode com a tal no fa la comprovació que llança l'excepció ja que hem preferit implementar aquesta en el constructor per tal de mantenir una estructura mes coherent.

Visitant per Comptar Items per sota d'un MinPrice (Apartat c)

Per comptar el nombre d'Items que estan per sota d'un preu mínim, es va dissenyar el CountItemsBelowMinPriceVisitor. Aquest visitant encapsula la lògica de còmput i permet evitar cicles i comptar duplicats en estructures de productes potencialment complexes.

El CountItemsBelowMinPriceVisitor manté un conjunt (Set) de productes visitats per prevenir cicles i duplicats. La variable count manté el recompte d'Items que estan per sota del minPrice. El codi del visitant és el següent:

```
import java.math.BigDecimal;
import java.util.HashSet;
import java.util.Set;
public class CountItemsBelowMinPriceVisitor implements ProductVisitor
   private BigDecimal minPrice;
   private int count;
   private Set<Product> visited;
   private CountItemsBelowMinPriceVisitor(BigDecimal minPrice) {
        if (minPrice.compareTo(BigDecimal.ZERO) <= 0) {</pre>
            throw new IllegalArgumentException("Minimum price must be
greater than zero");
        }
       this.minPrice = minPrice;
       this.count = 0;
        this.visited = new HashSet<>();
    }
    @Override
    public void visit(Item item) {
        if (!visited.contains(item)) {
            visited.add(item);
            if (item.getPrice().compareTo(minPrice) < 0) {</pre>
                count++;
        }
    }
    @Override
    public void visit(Pack pack) {
        if (!visited.contains(pack)) {
            visited.add(pack);
            for (Product product : pack.getProducts()) {
                product.accept(this);
        }
    }
```

```
public int getCount() {
        return count;
}

public static int countItemsBelowMinPrice(Product product,
BigDecimal minPrice) {
        CountItemsBelowMinPriceVisitor visitor = new
CountItemsBelowMinPriceVisitor(minPrice);
        product.accept(visitor);
        return visitor.getCount();
}
```

Es defineix un mètode estàtic per iniciar el procés de visita i obtenir el resultat, assegurant-se que cada producte només es compta una vegada, fins i tot si es troba en cicles dins de l'estructura de Packs:

```
public static int countItemsBelowMinPrice(Product product, BigDecimal
minPrice) {
    CountItemsBelowMinPriceVisitor visitor = new
CountItemsBelowMinPriceVisitor(minPrice);
    product.accept(visitor);
    return visitor.getCount();
}
```

Aquest enfocament assegura que el recompte d'Items és precís i no es veurà afectat per estructures no arborescents o cicles dins dels Packs.

Observacions

Hem llegit amb deteniment les observacions proporcionades sobre l'exercici anterior, així doncs hem tingut en compte els errors de disseny que vam cometre i els hem intentat resoldre per tal d'intentar millorar i tenir una bona base a l'hora de fer aquest segon exercici de disseny. Ens sap greu no haver assolit de manera satisfactòria les expectatives en l'anterior entrega. A continuació mostrem els canvis implementats en les classes Item Item.java i Product.java per tal de corregir aquests errors.

<u>Item.java</u>

```
public class Item implements Product{
    private BigDecimal price;
    public Item(BigDecimal price) {
        checkGreaterThanZero(price);
        this.price = price.setScale(2, RoundingMode.HALF UP);
    private void checkGreaterThanZero(BigDecimal price) {
        if (price.compareTo(BigDecimal.ZERO) <= 0) {</pre>
            throw new IllegalArgumentException("Price must be greater
than zero");
        }
    }
    @Override
    public BigDecimal getPrice() {
        return this.price;
    public void setPrice(BigDecimal newPrice) {
        newPrice = newPrice.setScale(2, RoundingMode.HALF UP);
        checkGreaterThanZero(newPrice);
        this.price = newPrice;
    }
    @Override
    public void checkMinItemPrice(BigDecimal minItemPrice) {
        checkGreaterThanZero(minItemPrice);
        if(this.price.compareTo(minItemPrice) < 0){</pre>
            this.price = minItemPrice.setScale(2,
RoundingMode.HALF UP);
       }
    }
    @Override
    public long countItemsBelowMinPrice (BigDecimal minPrice,
Set<Product> visitedProducts) {
       return this.price.compareTo(minPrice) < 0 ? 1 : 0;</pre>
```

```
@Override
   public void accept(ProductVisitor visitor) {
        visitor.visit(this);
}
Pack.java
public class Pack implements Product{
    private List<Product> products;
    public Pack() {
        this.products = new ArrayList<>();
    public void addProduct(Product product) {
        if (product == null) {
            throw new NullPointerException("Product is null");
        products.add(product);
    }
    @Override
    public BigDecimal getPrice() {
        return products.stream()
                .map(Product::getPrice)
                .reduce(BigDecimal.ZERO, BigDecimal::add)
                .setScale(2, RoundingMode.HALF UP);
    }
    public List<Product> getProducts() {
        return Collections.unmodifiableList(products);
    @Override
    public void checkMinItemPrice(BigDecimal minItemPrice) {
       products.forEach(product ->
product.checkMinItemPrice(minItemPrice));
    }
    @Override
   public long countItemsBelowMinPrice(BigDecimal minPrice,
Set<Product> visited) {
        visited.add(this);
        long count = 0;
        for (Product product : products) {
            if (!visited.contains(product)){
                count += product.countItemsBelowMinPrice(minPrice,
visited);
        return count;
```

```
@Override
public void accept(ProductVisitor visitor) {
    visitor.visit(this);
}
```

Conclusió

La implementació del patró visitant permet afegir noves operacions als productes de manera elegant i escalable. Aquest enfocament millora la mantenibilitat del codi i facilita l'extensió de funcionalitats sense modificar les classes existents. La modularitat del codi es veu millorada, ja que les operacions es poden afegir mitjançant nous visitants en lloc de modificar les classes dels productes directament.

Solució Dídac

```
Didec Cayvela Dolcet
Exercici Patró Composite
                   (1) Product yava
  import java math. Big Decimal;
 public interface Product &
      BigDecimal getPrice();
       Void check Min I ten Price (Big Decimal min I tem Price);
             Court tens Below Hin Price (Big Decimol Min Price, set < Products united Products);
       default long count I tems Below Min Price (Big Decimal Min Price) {
             set < Andrets visited = New Hash Set <> ();
             return Count I temo Below Min Parce (min Parce, visited);
      Void accept (Product Visitor visitor);
                    ( Item. java
  Import java. math. Big Decimal;
  Import java moth. Rounding Mode;
 public class Item implements Product &
       private BigDecimal price;
     public Item (BigDecimal price) {
         check Greater Than Zero ( price );
         this price = price set Scale(2, Rouding Mode .HALF-UP);
     @ Overvide
     Public Big Decimal getPrice(){
         return this price;
     public void settince (BigDecimal newPrice) {
        newPrice = newPrice. set Scale (2, Routing Mode. HALF_UP);
        check Greater Than Zero ( now Price );
        the price = new Price;
```

```
3
 public void check Hin I tem Price (Big Decimal min I tem Price) {
        check Greater Than Zero (minItem Price);
       if (tho. price. compare To (miniten Price) 20) }
             this price = MinItemPrice setScale (2, RandingHode . HALF_UP);
  3
  private void check Greater Than Zero ( Bigliand min I tem Price) {
        if c price compare to (Bistain land) <= 0) {
             thow new Illegal Argument Exception ("Price most be greater than zero");
  3
                 Count Items Below Min Pine (Big Dermal min Price, Set & Product) visited Products) {
 1 Overvide
 public long
                 tho price . Compacto (min Purce) 2 0 ? 1:0;
         return
Governde
 Public wit accept (Protat Visitor visitor) {
      voltar. viol+ (ths);
                  (c) Pack java
public class Pack implants
                               Product
    private List ( Produt > products;
    public Pack () {
        this products = new Array List (SC);
   public void get Price () {
       BigDecimal totalPrice = BigDecimal. ZERO;
      for (Product P: products) {
         tolophic = totolpia add(p. getPriac());
```

```
return total Price. Set Scale (2, Rounding Hode. HALF. UP);
  Public void Add Product (Product product) ?
       if ( produt == NUI) {
          throw new Null Pointe Exception ("Froduct is Null");
     products. add (product);
Public Lot 2 Products get Products () {
     return Collections. Unmodifiable List (pusheds);
3
public void checktin I tem Price (Big Decimal min I tem Price) {
     for (Product p: product) {
          p. check Hin I tem Price (min I tem Price);
3
public long countItemoBelowMinPrice (BigDecimal MinPrice, Set ZPWodut) visited) {
       visited add (this);
       long count = 0;
       for (Product product: product) {
            If (! visited. Contains (product)) {
                Court += product. Court Item Bebw Min Price (min Price, visited);
       3
       betwo Count;
g overvide
 public void accept (Produttion visitor) }
       ( ant ) thoir rotal
       to (Product product: product))
```

```
for ( Product product: product) }

product except ( woster);
}
```

(I) Produt Vistor

```
public interface Product Visitor {

void visit (zten iten);

void visit (Pux pux);

7
```

a ae mile

public wid visit (Item Hern) {

MinPrice Updater Visitor

```
public class MinPrice Update Votor implements Product Vistor {

private Big Decimal minPrice;

private MinPrice Update Votor (BigDecimal minPrice) {

if (minPrice compareto (RigDecimal Zero) <= 0) }

thow new Illegal Argumet Exception ("Minimum Price must be great than Zero");

}

this minPrice = minPrice Set Scale (2, Rainting Kde. HALF UP);
```

```
public wid visit (Item Hem) {
       if (item set Price (). compacto (minPrice) LD) {
           iten. set Prie (min Price);
  3
 @ overvide
 Public will wroth (Pack Pock) {
      for (Product product: pack. getProducts()) {
     pudet accept (this);
 public static voil update Prices (Product product, Big Decimal min Price) ?
     MinPrice Update Visitor visitor = New MinPrice Update Visitor (minPrice);
     produl acapt (visita);
 3
               Court I tens Below Min Price Visitor
puble class Countiters Beburth Price Visita implements Product Visitar {
         private Big Decorro minfrice;
         private int Gunt,
         purate Set (Product) VIVLe2;
               Court Itens Reby Mr Pine Vistor (Dy Deamy infine) ?
        printe
            if (minPrice. Compueto (BigDecumbl. 3ER) C= 0) {
                 thum new illest Argument Exception ("Minjum pure mot be greate than Zow");
            3
            the marrice - marries
           the cont = 0;
```

```
two. 1111111 - .
         the cont = 0;
         this . visited = new Habottos ();
         void voit (Item item) {
   public
       if (! Vaster. Contains (ilem)) {
            violed add ( item);
           if (itch. gethine(). compacto (minhine) < 0) {
                Count ++;
        }
  Governide
  public and most (Pack pack) {
      If (I UDITED. CONTAINS (POWK)) }
            violet. att (park);
            for (Product product : pack set Products()) {
                product accept (this);
       3
  3
Public int set (sunt () {
       return Count;
public Static int count Items Below Min Purce (Product product, By Decimal min Price) ;
COUNT I tem Below Min Price Victor visite = New Count I tem Below Min Price Victor (mnPrice);
      produt. accept ( visite )
     betun voltor get Gurt ();
```

Solució Luis

```
Práctica 4
```

```
sábado, 25 de mayo de 2024 12:10
 public interface Product ?

public interface Product ?

public Big Decimal getPrice ();

public void check Min Item Price (Big Decimal min Item Price);

long count Items Below Min Price (Big Decimal min Item Price);
            default long count Items Below Minfrice (Big Decimal minfrice) &
                            Set cloducts visited = new Harbsetcs();
return count I tems Below Hindrice (min Price, mostru);
            void account (Product Visitor visitor);
  3
  public interface product Visitor & void visit (Item item); void visit (Item item);
  public class Pack implements Arabact &
             private List C Products products;
            public Pack () }
this, products = new Army List <>();
 public void of strotust (Product product) {

if (product == null) {

throw now NullPartir-Exception ("Graduet is null");
            products.add();
@ Override
public Righerinal gettrice() {
return products.stream()
unp(brokets:gettrica)
                    . of Lice (Rig Deciral - ZERO, Rig Deciral: add)
. sot Scale (2, Randing Mad. HALP-10);
 public List errobusts pat Products () }
3 return Collection. connocificable List (products);
  public void check (Min It gentrice (Bigleson) win Itsufrice);
products for Each (product -> product. check (Min Itsufrice);
2
                         count I tems Below Hinfrice (By Decinal minfrior, Set 2 froduits visited) &
  @Override
  public long county.
```

```
121149.400 C. ....
      long count = 0;
      for ( Product: products)

in ( Product : products)

in ( Product : product)) &

count += product : count I trans Bolove Min Price (min Price, visited);
       return count;
Public void accept (Product Visitor visitor) }
        visitar. visit (this);
for (product product: products) {
product accept (visitar);
public class Item implements brodut &
        private Big Decimal pria;
        public Item (Big Decimal Price) {
check Greater Theo Zero (price);
               this price = price sets al (2, louding the NMF-UP);
      private void check Grafter Than Zero (Big Decimal price) {
if (price. compare to (By Decimal ZERO) <=0) {
                     throw now Algolfrycast Exception " Are nest be grown than zero");
       3
       @Overrise
      public Righerinal gothin() {
return this. prior;
       public void sattria (Righerind unufria) ?
           wouthin = newtree .setScale (2, Rossing Make HOLF-UP);
double Grost or Then Zaro (newbries);
           this , pria c ma Pria;
              : void check that toutrice (Big Decom) min Thanking) {
check Greater than Zaro (win Tentrice);
      @ Override
              il(this, price. crapers To (aintentra) < 0) {
this. price = un Thulrice. setScale (2, Recording to be. HALF-00);
```

```
3
    3
                  countries Bolow Min Brice (Rights in a nichia, Sot a Booket > visital Bookets) }
    @Override
   public by country this price compare to (michia) 20?1:0;
   public void accort (Produt Visitor visitor) ?
visitor. visitflix);
   public class Minfrice Optistar Visitar indensity Product Visitor &
        private highernal
                                walting
       Private Minfrice Opadar Visitor (Rig Decina) winfria) ?
             : fluintrice. compare To (Big Decing) . ZERO) L= 0) {
                throw now Illegal Argent Exception (" prince orice must be grade than zon");
      this will like = will rig set Scale (2, Renders tech 1847-UP);
  public void visot (Don Han) {
if (item. getPrice (). compare To (noin Price) < 0) {
; town. setPrice (minfrice);
   @Overile
  3
    @ Overrile
   public vaid visit (pull pull) {
       for (Product product: padl. get Products ()) {

product: accept (this);
   public static void updatefricas (froduct product, Bylecimal minfrica) &
          Minfrice Up Later Victor Visitor = new Mxfrice Up Later Visitor (was co);
          product acept (visita);
public class CountItons Bolow Harfrice Visitor implements Product Visitor &
        private Light cian neutrice;
        private int cout;
        private Set < Andrets visited;
        prints Cout It ans Adom Men Prialister (Bigle cinal min Pria) &
              illinitria. oraporto (Egipa: mol-2ERO) 2=0) }
throw now They appendite xeglia (unimum price met le grater than sent)
```

```
Howard = 0;

this void visit (Hom : An) ?

goverible

public void visit (Hom)?

if (Ivistal add And);

for (Product product: padd got Rock ds (1))?

product account (Ha);

}

public int get (wat (2))?

product account (Ha);

for this Below Making (Product product, By Documed waters)?

public static int count thus Below Making (Product product, By Documed waters)?

product account (ristor);

product account (ristor);

rature visitor get (out (1);

}
```

Solució Aleix

```
ALEIX DRUDIS HOLA
  49696715 - M
   interface Visitar {
       noid visit (them item); raid visit (Pack jack);
    interfore Product {
       raid accept (Virtor mitar);
       DioPecinal gethrice ();
    class Istem implements Product {
        sinate BigDecimal price;
        public Item (Big Reciral price) {
           this . fice = fice;
         @ arenide
         public raid accest (Virta viritar) { winter. wirit (this); }
         @ arenide
         public Big Decimal gethice () { return since;}
      class Pack implements Product {
         private List & Product > products : new AnayList e> ();
         pullic vaid add Product (Product graduct) { products. add (raduct); }
         @ amenide
         pullic world accept (Visitor writer) (winter wisit (this); }
         pullic List chaduct > getProducts () { return products; }
         @ arenide
         public Biglecimal get Price () {
            return products. stream ()
. map (Product :: gethrice)
                  . reduce (Bighecimal . ZERO, Big Recimal : add);
         3
     3
```

```
class PriceAdjustVisita implements Visitar {
     private Big Decimal min Price;
     fullic Price Adjuster Viritor (Big Deciral min Price) { this. min Price : min Price ; }
     @ arenide
     jullic roid rivit ( Item item ) {
         if (item. get bice (), compare To (minhice) 60 {
             ten. rellie (Amir Rice);
         3
      3
     @ armide
      public raid rist (Pack fack) {
          for (Product raduct: rack. get Products ()) {
              radut. accept (this);
      3
 class Item Counter Victor implements Visitor {
     virate Diglecimal mintrice;
      welle Iten Court Vinta (Displement minhice) { this. milie = minhice; }
      @ arride
      public round nist (Iten item) {
         if (item. getPrice (). coyacTo (minhice) <0) { count ++ ; }
       3
      @ arride
      sullie roid ninit (Pack sock) {
           for (Roduct moderal: sack. get Products ()) {
              waded occept (this);
           3
      public int get Count () { return count; }
3
```