

---

# AMPLIACIÓ DE BASES DE DADES I ENGINYERIA DEL PROGRAMARI

Pràctica 3: Primera activitat de patrons de disseny

---

Aaron Arenas Tomas - 78098697N

Ramon Escoda Semís - 49751728Z

Marc Cervera Rosell - 47980320C

18 de maig 2022

Grau en Enginyeria Informàtica

# Índex

<b>1</b>	<b>Disseny</b>	<b>2</b>
<b>2</b>	<b>Solució consensuada</b>	<b>1</b>
2.1	Classe abstracta <i>Product</i> . . . . .	1
2.2	Classe <i>Item</i> . . . . .	2
2.3	Classe <i>Pack</i> . . . . .	3
2.4	Classe <i>PriceChanged</i> . . . . .	5
2.5	Interfície <i>Observer</i> . . . . .	5
<b>3</b>	<b>Enllaç al directori GitHub</b>	<b>6</b>
<b>4</b>	<b>Annex</b>	<b>6</b>
4.1	Classe <i>Main</i> . . . . .	6
4.1.1	Codi de la classe <i>Main</i> . . . . .	6

## Índex de figures

1	Disseny UML . . . . .	2
2	Especificacions dels mètodes de la classe <i>Item</i> . . . . .	2
3	Especificacions dels mètodes de la classe <i>Pack</i> . . . . .	2
4	Especificacions dels mètodes de la classe <i>PriceChanged</i> . . . . .	1

# 1 Disseny

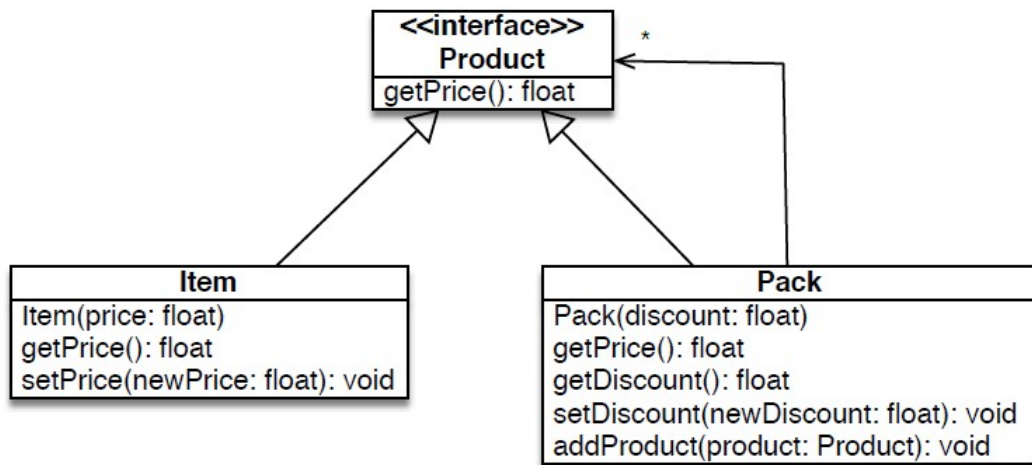


Figura 1: Disseny UML

Especificacions dels mètodes de les classes:

On la classe *Item*:

- representa un producte individual que, al seu constructor, rep el valor del preu el qual, per simplificar, serà un `float`.
- té el mètode `setPrice` per canviar el preu del producte
- té el mètode `getPrice` que retorna el preu del producte

Figura 2: Especificacions dels mètodes de la classe *Item*

I la classe *Pack*

- representa un grup de productes (que poden ser tant *Items* com *Packs*) que, al seu constructor rep el valor del tant per ú de descompte a aplicar
- té el mètode `setDiscount` per canviar el tant per ú de descompte
- té el mètode `addProduct` per afegir un nou producte al pack
- té el mètode `getDiscount` que retorna el tant per ú de descompte
- té el mètode `getPrice` que retorna el preu del pack: suma dels preus dels productes descomptada segons el tant per ú de descompte. Per exemple: si els productes tenen valors 10.0 i 20.0 i el tant per ú és 0.2, el preu és  $(1.0 - 0.2) * (10.0 + 20.0) = 24.0$

Figura 3: Especificacions dels mètodes de la classe *Pack*

```

1 public final class PriceChanged {
2     private final float oldPrice;
3     private final float newPrice;
4     public PriceChanged(float oldPrice, float newPrice) {
5         this.oldPrice = oldPrice; this.newPrice = newPrice;
6     }
7     public float getOldPrice() { return oldPrice; }
8     public float getNewPrice() { return newPrice; }
9 }

```

Figura 4: Especificacions dels mètodes de la classe *PriceChanged*

## 2 Solució consensuada

Un cop comparades totes les resolucions individuals, les quals s'adjunten juntament amb aquest document, s'ha decidit, per unanimitat, entregar com a solució vàlida (segons el criteri dels membres del grup) i puntuable, la solució que s'observa a continuació.

Com es podrà observar, en el codi de la solució final hi ha mètodes que no formen part de la solució al problema demanat, però s'han inclòs atès que s'ha usat l'IDE per programar la totalitat del codi i s'ha fet ús d'una classe anomenada *Main* que realitza diversos jocs de proves per comprovar la correctesa de la solució. Per poder veure la classe *Main*, hi ha dues opcions; veure l'annex d'aquest document (Secció 4 a la pàgina 6) o seguir l'enllaç al directori de GitHub que hi ha més andavant (Secció 3 a la pàgina 6).

### 2.1 Classe abstracta *Product*

---

```

/*
    Marc Cervera Rossell
    Aaron Arenas Tomas
    Ramon Escoda Semis
*/

public abstract class Product extends Observable {
    public abstract float getPrice();
}

```

---

## 2.2 Classe *Item*

---

```
/*
    Marc Cervera Rossell
    Aaron Arenas Tomas
    Ramon Escoda Semis
*/

public class Item extends Product {
    private float price;

    public Item(float price) {
        this.price = price;
    }

    @Override
    public float getPrice() {
        return this.price;
    }

    public void setPrice(float price) {
        if (price < 0) {
            throw new IllegalArgumentException("Negative price");
        }
        if (this.price != price) {
            float oldP = this.price;
            this.price = price;
            notifyObservers(new PriceChanged(oldP, this.price));
        }
    }

    @Override
    public String toString() {
        return "Item {" +
            "price = " + price +
            '}';
    }
}
```

---

## 2.3 Classe *Pack*

---

```
/*
    Marc Cervera Rossell
    Aaron Arenas Tomas
    Ramon Escoda Semis
*/

public class Pack extends Product implements Observer {

    private float discount;

    private boolean hasChanged;

    private float price;

    private List<Product> products;

    public Pack(float discount) {
        this.discount = discount;
        this.products = new ArrayList<Product>();
        this.hasChanged = false;
    }

    public void removeProd(Product p) {
        this.products.remove(p);
    }

    @Override
    public float getPrice() {
        if (!this.hasChanged) {
            return this.price * (1f - getDiscount());
        }
        for (Product p : products) {
            this.price += p.getPrice();
        }
        this.hasChanged = false;
        return this.price * (1f - getDiscount());
    }

    public float getDiscount() {
        return discount;
    }
}
```

```

    }

    public void setDiscount(float discount) {
        if (discount < 0f || discount > 1f) {
            throw new IllegalArgumentException("Not valid discount");
        }
        if (this.discount != discount) {
            float oldD = this.discount;
            this.discount = discount;
        }
    }

    public void addProduct(Product product) {
        this.products.add(product);
        float oldP = this.price;
        product.addObserver((Observer) this);
        this.price += product.getPrice();
        notifyObservers(new PriceChanged(oldP, this.price));
    }

    @Override
    public String toString() {
        return "Pack {" +
            "discount = " + discount +
            ", products = " + products +
            '}';
    }

    @Override
    public void update(Observable o, Object arg) {
        PriceChanged pC = (PriceChanged) arg;
        float oldP = this.price;
        if (pC.getNewPrice() != pC.getOldPrice()) {
            this.hasChanged = true;
            notifyObservers(new PriceChanged(oldP, this.getPrice()));
        }
    }
}

```

---

## 2.4 Classe *PriceChanged*

---

```
/*
    Marc Cervera Rossell
    Aaron Arenas Tomas
    Ramon Escoda Semis
*/

public final class PriceChanged {
    private final float oldPrice;
    private final float newPrice;

    public PriceChanged(float oldPrice, float newPrice) {
        this.oldPrice = oldPrice;
        this.newPrice = newPrice;
    }

    public float getOldPrice() {
        return oldPrice;
    }

    public float getNewPrice() {
        return newPrice;
    }
}
```

---

## 2.5 Interfície *Observer*

---

```
public interface Observer {
    /*
        Marc Cervera Rossell
        Aaron Arenas Tomas
        Ramon Escoda Semis
    */

    void update(Observable o, Object arg);
}
```

---



## 3 Enllaç al directori GitHub

En aquesta secció és troba l'enllaç al directori de GitHub amb el codi de la solució final. Com es podrà observar, el projecte d'IntelliJ disposa d'una classe *Main*, on hi ha diferents exemples d'execució on es comprova la correctesa dels diferents patrons aplicats.

*Enllaç al directori GitHub*

## 4 Annex

### 4.1 Classe *Main*

Com s'ha esmentat en anterioritat en aquest document, aquesta classe ha servit a l'equip de desenvolupament per ratificar la correctesa de la solució que s'ha donat com a vàlida per ésser puntuable.

#### 4.1.1 Codi de la classe *Main*

---

```
public class Main {  
    /*  
        Marc Cervera Rossell  
        Aaron Arenas Tomas  
        Ramon Escoda Semis  
    */  
  
    public class Main {  
    public static void main(String[] args) {  
        /* Pack */  
        Pack p1 = new Pack(0.5f);  
  
        /* Items */  
        Item i1 = new Item(12.7f);  
        Item i2 = new Item(10.13f);  
        Item i3 = new Item(400.99f);  
  
        /* Including items inside the pack */  
        p1.addProduct(i1);  
        p1.addProduct(i2);  
        p1.addProduct(i3);  
  
        System.out.println("----- Composite pattern test part  
        -----");  
        System.out.println("\n");  
    }  
}
```

```

System.out.println("----- Pack with items -----");
System.out.println(p1);
System.out.println("Final price after discount: " + (p1.getPrice()));

/* Adding a pack inside a pack */
Pack p2 = new Pack(0.1f);
Item i4 = new Item(1.7f);
Item i5 = new Item(123.45f);
p2.addProduct(i4);
p2.addProduct(i5);
p1.addProduct(p2);

System.out.println("\n");

System.out.println("----- Adding a pack inside a pack
-----");
System.out.println(p1);
System.out.println("Final price after discount: " + (p1.getPrice()));

System.out.println("\n");

System.out.println("----- Observer test part -----");
/* Pack */
Pack p3 = new Pack(0.6f);

/* Items */
Item i6 = new Item(121.7f);
Item i7 = new Item(105.13f);
Item i8 = new Item(4000.99f);

p3.addProduct(i6);
p3.addProduct(i7);
p3.addProduct(i8);

System.out.println(p3);
i6.setPrice(120.45f);
System.out.println(i6.getPrice());
System.out.println(p3);
}
}

```

---