2 (DOS)

Programación 2
Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas
FINAL 29-02-2024

Una carpintería especializada en trabajos con hierro y madera desea informatizar el sistema de pedidos. En esta carpintería, cuando se ordena un pedido en una determinado local, se registra el conjunto de elementos, el número del local y el nombre del vendedor que atendió al cliente. Un elemento puede ser algo tan simple como una tabla de picar, como algo tan complejo como un juego de comedor completo. Cada elemento posee un nombre, un tipo (Mueble, adorno, cocina, juguete, etc), un modo de presentación (crudo, combinado, laqueado, esmaltado, etc), el peso, y un precio base. Asimismo, para elementos complejos, puede ser que se requiera de otros elementos para su preparación. Estos elementos complejos, tienen nombre, tipo y modo de presentación. Por ejemplo, para poder realizar un "juego de asado completo", cuyo tipo es "cocina" y su presentación es "crudo", es necesario contar con elementos específicos (plato, cuchillo, tenedor), y con los elementos complejos "Juego de utensilios asador" y "juego tablas de carne y picada" (ambos a su vez compuestos de sus respectivos elementos, simples o complejos). En este caso el peso del elemento complejo se calcula como la suma de los pesos de cada una de sus partes, al igual que el precio base.

La carpintería está organizada en talleres que serán los encargados de la elaboración de los elementos encargados. Dado que cada taller tiene especialidades diferentes, cuando un pedido arriba a la carpintería, los elementos que componen el mismo, se van asignando a los diferentes talleres, hasta que alguno pueda realizarlo. Un taller aceptará preparar un elemento, si su especialidad lo permite según los siguientes criterios:

- Solo aceptar elementos cuyo modo de presentación sea "laqueado".
- Solo se aceptan elementos que sean del tipo "cocina".
- Solo se aceptan elementos cuyo precio base sea menor a $50.000.
- También pueden existir combinaciones de especialidades, por ejemplo aceptar solo elementos del tipo "juguete" y que el precio base sea menor a $10.000; o aceptar elementos que se presenten laqueados y cuyo precio base no sea menor $30.000.

La especialidad de cada taller puede variar en cualquier momento (en tiempo de ejecución). Si no hay ningún taller que pueda elaborar el elemento, éste no se realiza (no se asigna a ningún taller).

El costo total de un pedido es la suma de los precios base de todos sus elementos más un adicional que aplica la carpintería, **a cada elemento que compone el pedido,** en base a algún criterio. Ejemplos de posibles formas de calcular el adicional por cada elemento son:

- Si su presentación es "laqueado" y el precio base es menor a $20.000 se suman $500; en otro caso se suman $1300.
- Si es un Juguete se suma $1000; en otro caso $5000.
- Sumar $1000 si el peso es mayor a 500, sino nada.
- Sumar el 10% del precio base del elemento en calidad de propina para el empleado que tomó el pedido.

NOTA: La forma de calcular el costo de un pedido es la suma de los costos de sus elementos considerando siempre el cálculo del adicional de cada uno de ellos, el cual lo establece la carpintería y es el mismo para todos los pedidos. Sin embargo, la carpintería puede establecer una forma diferente de calcular el adicional y la misma se debe poder cambiar en tiempo de ejecución.

M. Victoria Caballero

```
public class Carpintera {
    private String nombre;
    private ArrayList<Pedido> pedidos.
    public Carpintera (String nombre) {
        this.nombre = nombre;
        this.pedidos = new ArrayList<>();
    }
    public boolean tallerAceptaPedido (Taller t) {
        for (Pedido p : pedidos) {
            if (p.buscar (t)) {
                return true;
            }
        }
        return false;
    }
    public double montoTotal (Calculador c) {
        double total = 0.0;
        for (Pedido p : pedidos) {
            total += p.costoTotal () + p.costoTotal (c);
        }
        return total;
    }
}
```

deberia tener los talleres

Clase Taller es el Criterio o Filtro

El buscar retorna un arrayList y no un boolean

buscar?
El pedido deberia ser el parametro y el taller en su listado

El boolean dice si un taller acepta un PEDIDO y no elemento a elemento

costo con y sin calculador

```
public abstract class Calculador {
    public double calcular (ElementoSimple e);
}
```

solo simple? no funciona el pedido?

NO HACE CALCULADOR CONDICIONAL

```
public class CalculadorSuma {
    private Calculador op1, op2;
```

no hereda de nadie?

NOTA

```
    public CalculadorSuma ( Calculador op1, Calculador op2){
        this.op1 = op1;
        this.op2 = op2;
    }
    @Override
public double calcular (ElementoSimple e){
        return op1.calcular(e) + op2.calcular(e)
    }
}
```

```
public class MontoFijo extends Calculador {
    private double monto;

    public MontoFijo (double monto){
        this.monto = monto;
    }
    @Override
    public double calcular(Elemento Simple e){
        return monto;
    }
}
```

ok

```
public class PorcentajeAdicional {
    private double porcentaje;
    public PorcentajeAdicional (double porcentaje){
        this.porcentaje = porcentaje;
    }
    @Override
    public double calcular ( ElementoSimple e){
        return e.getPrecio() + porcentaje;
    }
}
```

FALTA HERENCIA

ok, pero deberia ser solo el procentaje por la forma en que se usa luego
costo() + costo(Claculador)
El porcentaje ademas deberia multiplicar el precio

M. Victoria Cabello Molina

```
public class Pedido {
    private ArrayList<Elemento> elementos;
    private int numeroDelLocal;
    private String vendedor;

    public Pedido(int numeroDelLocal, String vendedor){
        this.elementos = new ArrayList<>();
        this.numeroDelLocal = numeroDelLocal;
        this.vendedor = vendedor;
    }
    public int numeroDelLocal(){
        return numeroDelLocal;
    }
    public void setNumeroDelLocal(int numeroDelLocal){
        this.numeroDelLocal = numeroDelLocal;
    }
    public String vendedor(){
        return vendedor;
    }
    public void setVendedor(String vendedor){
        this.vendedor = vendedor;
    }
    public addElemento(Elemento e){
        this.elementos.add(e);
    }
    public double costoTotal(){
        double total = 0.0;
        for(Elemento e : elementos){
            total += e.getPrecio()
        }
        return total;
    }
```

Por como se hizo es igual al compuesto,

ok

NOTA

```
public ArrayList<Elemento> buscar (Taller t){
    ArrayList<Elemento> encontrado = new ArrayList<>();
    for(Elemento e : elementos){
        encontrado.addAll(e.buscar(t));
    }
    return encontrado;
}

public double costoTotal(Calcular c){
    double total=0.0;
    for(Elemento e : elementos){
        total = e.costoTotal(c);
    }
    return total;
}
```

delega al buscar con el taller como parametro

elemento deberia tener un buscar

ok

```
public abstract class Elemento {
    private String nombre;                    ok
    private String tipo;
    private String presentacion;
    public Elemento (String nombre, String tipo, String presentacion) {
    this.nombre = nombre;
    this.tipo = tipo;
    this.presentacion = presentacion;
    }
    public String getNombre () {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getTipo () {
        return tipo;
    }
    public void setTipo (String tipo) {
        this.tipo = tipo;
    }
    private String getPresentacion () {
        return presentacion;
    }
    public void setPresentacion (String presentacion) {
        this.presentacion = presentacion;
    }
    public abstract double getPeso ();        ok
    public abstract double getPrecio ();      ok
```

NOTA

```
public abstract double costeTotal (CalidadBarco c);
public abstract ArrayList<ElementoSimple> buscar (Talleres t);
}
```

ok

ok

simple

HOJA N° 4

M. Victoria Cabello R

FECHA

```
public class ElementoSimple {        falta la herencia
    private double peso;
    private double precio;
    public ElementoSimple(String nombre, String tipo, String presentacion, double peso
                                                                    double precio ) {
        super(nombre, tipo, presentacion);
        this.peso = peso;
        this.precio = precio;
    }
    @Override
    public double getPeso(){
        return peso;                                    ok
    }
    public void setPeso(double peso){
        this.peso = peso;
    }
    @Override                                           ok
    public double getPrecio (){
        return precio;
    }

    public boolean equals (Object obj) {
        try {
            ElementoSimple otro = (Elemento).obj;
            return this.getNombre().equals(otro.getNombre()) &&
                   this.getPresentacion.equals(otro.getPresentacion()) &&
                   this.getPeso() == otro.getPeso(1);
        }catch (Exception e){
            return false;
        }
}
```

NOTA

```
@Override
public double costoTotal(Calculer c){
  double total = 0.0;
  if(c.acepta(this)){
    total = getPrecio();
  }
  return total;
}
```

Mal
El calculador lo debe aceptar? el calculador modifica el valor

```
@Override
public ArrayList<ElementoSimple> buscar(Taller t){
  ArrayList<ElementoSimple> encontrado = new ArrayList<>();
  if(t.acepta(this)){
    encontrado.add(this);
  }
  return encontrado;
}
}
```

si el taller lo acepta, lo agrega
TAller es el criterio

NOTA

M. Victoria Cabello M

falta herencia

```
public class ElementoComplejo {
    private ArrayList<ElementoComplejo> elementos;

    public Elemento(String nombre, String tipo, String presentacion){
        super(nombre, tipo, presentacion);
        this.elementos = new ArrayList<>();
    }

    public boolean tieneElemento(ElementoComplejo e){
        return this.elementos.contains(e);
    }

    public void addElemento(ElementoComplejo e){
        if(!tieneElemento(e)){
            this.elementos.add(e);
        }
    }

    public void removeElemento(ElementoComplejo e){
        this.elementos.remove(e);
    }

    @Override
    public double getPeso(){
        double total = 0.0;
        for(ElementoComplejo e : elementos){
            total += e.getPeso();
        }
        return total;
    }
    @Override
    public double costoTotal(Calcular c){
    double total = 0.0;
        for(Elemento e : elementos){
            total += e.costoTotal(c);
        }
    return total;
    }
```

equals de elemento

ok

ok

ok

ok

```
@Override
public double getPrecio(){
    double total = 0.0;
    for(ElementoComplejo e: elementos){
        total += e.getPrecio();
    }
    return total;
}
@Override
public ArrayList<ElementoSimple> buscar (Taller t){
    ArrayList<ElementoSimple> encontrado = new ArrayList<>();
    for(ElementoComplejo e: elementos){
        encontrado.addAll(e.buscar(t));
    }
    return encontrado;
}
```

ok

ok

```
public abstract class Taller {
    public abstract boolean acepta(ElementoSimple e);
}
```

Taller es el filtro!!!

solo sobre simples!!!

```
public CriterioPresentacion extends Taller {
    private String presentacion;

    public CriterioPresentacion(String pre) {
        this.presentacion = pre
    }
    @Override
    public boolean acepta(ElementoSimple e) {
        return e.getPresentacion.equals(presentacion);
    }
}
```

ok

```
public CriterioTipo extends Taller {
    private String tipo;

    public CriterioTipo(String t) {
        this.tipo = t;
    }
    @Override
    public boolean acepta(ElementoSimple e) {
        return e.getTipo().equals(tipo);
    }
}
```

ok

```
public class CriterioMenorPrecio {
    private double precio;
    public CriterioMenorPrecio(double p) {
        this.precio = p;
    }
```

Falta Herencia

NOTA

```
@Override
public boolea acepta (ElementoSimple e){
    return e.getPrecio()< precio;
}
}
```

```
public class CriterioAnd extends Taller{
    private Taller condicion1, condicion2;

    public CriterioAnd (Taller condicion1, Taller condicion2){
        this.condicion1 = condicion2;
        this.condicion2 = condicion2;
    }
    @Override
    public boolean acepta(ElementoSimple e){
        return condicion1.acepta(e) && condicion2.acepta(e);
    }
}
```

```
public class CriterioOr extends Taller{
    private Taller condicion1, condicion2;
    public CriterioOr(Taller condicion1, Taller condicion2){
        this.condicion1 = condicion1;
        this.condicion2 = condicion2;
    }
    @Override
    public boolean acepta(ElementoSimple e){
        return condicion1.acepta(e) || condicion2.acepta(e);
    }
}
```

```
public class Not extend Taller{
    private Taller condicion;

    public Not (Taller condicion){
        this.condicion = condicion;
    }@Override
    public boolean acepta (ElementoSimple e){
        return !condicion.acepta(e);
    }
}
```

-Mal implementacion, fatan muchos extends
-Bien calculador, pero suma al precio y se usa sumando, con lo cual el precio se suma dos veces
-El elemento tiene un precio y precio con calculador
-Criterio o filtro se llama Taller, y se verifica sobre el simple

-No esta el asignar a taller, hay una especie de busqueda invertida