# INSTITUTO TECNOLÓGICO DE OAXACA

## Ingeniería en Sistemas Computacionales

## Diseño E Implementación De Software Con Patrones


**Alumnos:**

Méndez Mendoza Luisa Michel

Pérez Carrasco Samuel

López García Lourdes Gloria

Girón Pacheco Fernando

Barbosa Santiago Mario Alberto


## Unidad 4

Espinosa Pérez Jacob

**Horario:** 07:00 am- 08:00 am

## 8SC

# PATRON INTERPRETER

El patrón de diseño Intérprete es un patrón de diseño conductual que define una forma de interpretar y evaluar la gramática o las expresiones lingüísticas. Proporciona un mecanismo para evaluar oraciones en un idioma mediante la representación de su gramática como un conjunto de clases. Cada clase representa una regla o expresión en la gramática, y el patrón permite que estas clases se compongan jerárquicamente para interpretar expresiones complejas.

## Ventajas

**Flexibilidad:** Permite interpretar nuevos tipos de expresiones.

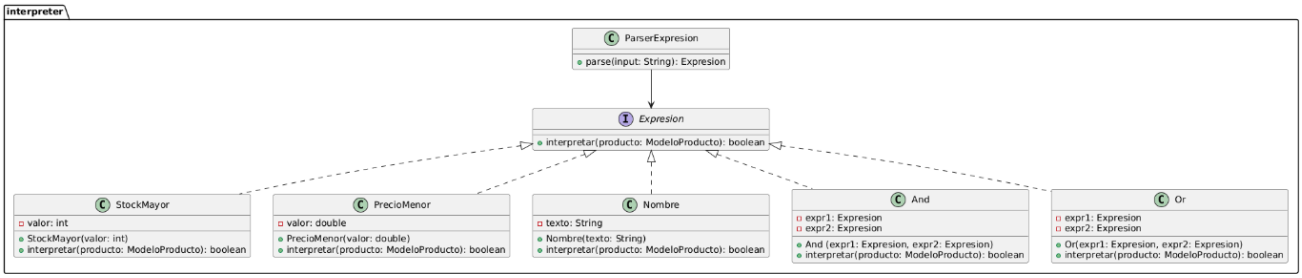
**Extensibilidad:** Fácil de ampliar y modificar el lenguaje o gramática.

## Desventajas

**Complejidad:** Puede volverse complejo si la gramática del lenguaje es complicada.

**Rendimiento:** Interpretar expresiones puede ser menos eficiente que otros métodos de procesamiento.

## Diagrama UML

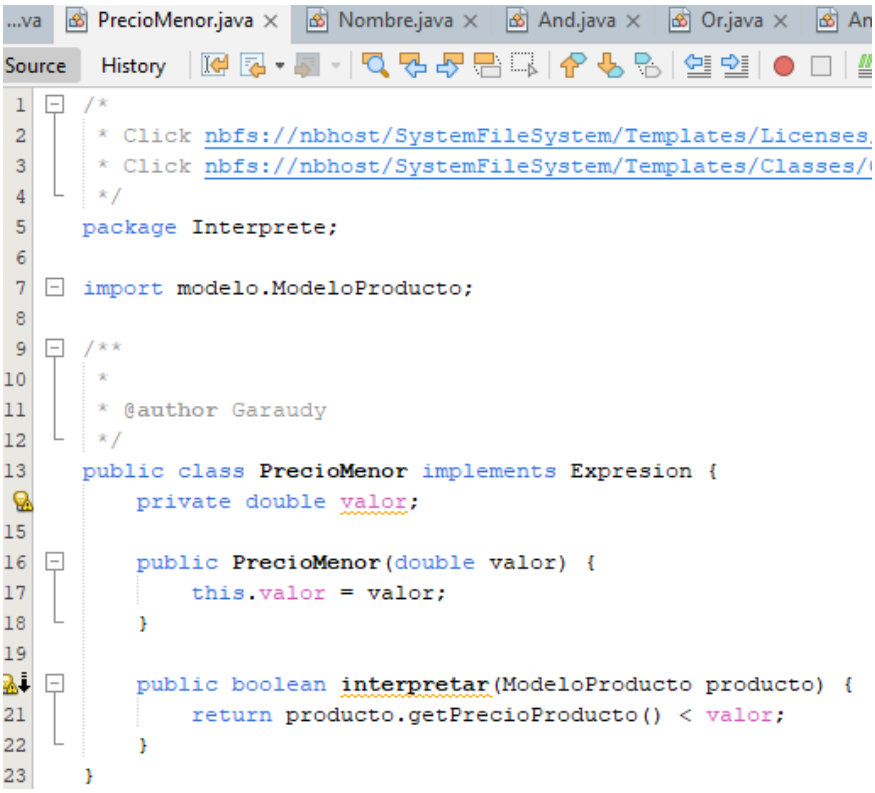



Primero creamos nuestra interfaz base, esta interfaz define el contrato es decir que cualquier clase que implemente esta interfaz debe interpretar o evaluar si un producto cumple una condición.

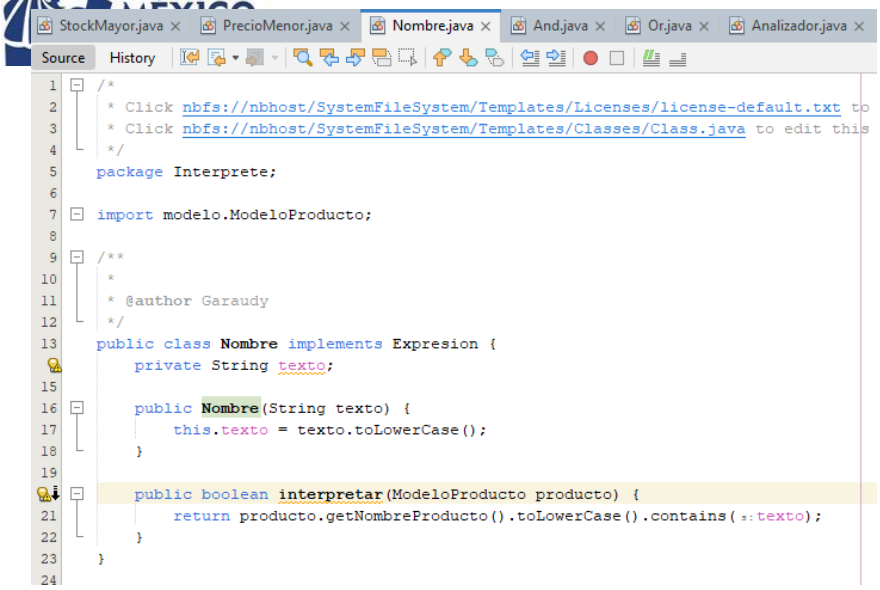
Evalúa si el stock de un producto es mayor al valor indicado

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/licen
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.
 */
package Interprete;

import modelo.ModeloProducto;

/**
 *
 * @author Garaudy
 */
public class StockMayor implements Expresion {
    private int valor;

    public StockMayor(int valor) {
        this.valor = valor;
    }

    public boolean interpretar(ModeloProducto producto) {
        return producto.getStockProducto() > valor;
    }
}
```

Evalúa si el precio de un producto es menor



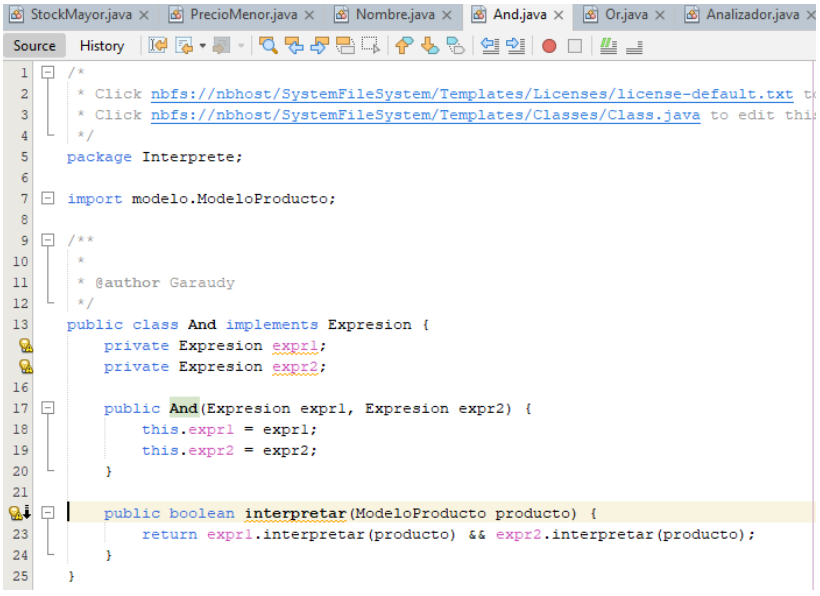

Este evalúa si el nombre de un producto contiene cierta palabra que se coloque en el buscador



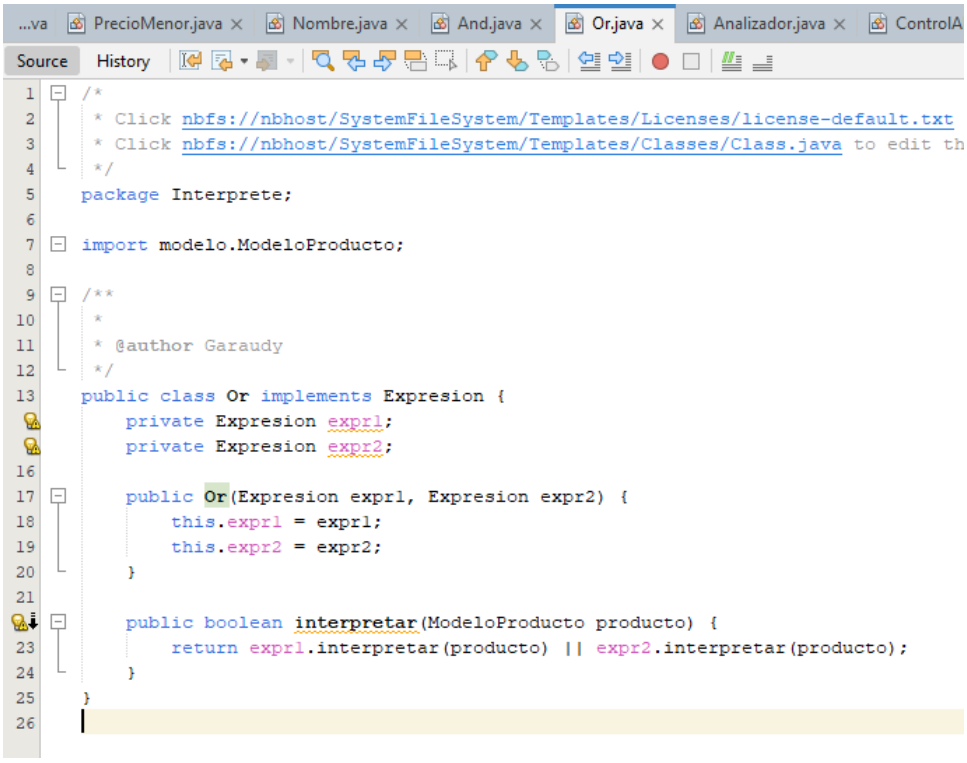

## CLASES CON EXPRESIONES COMBINADAS

Estas clases permites usar lógica compuesta (AND, OR)

Devuelve true si ambas condiciones son verdaderas

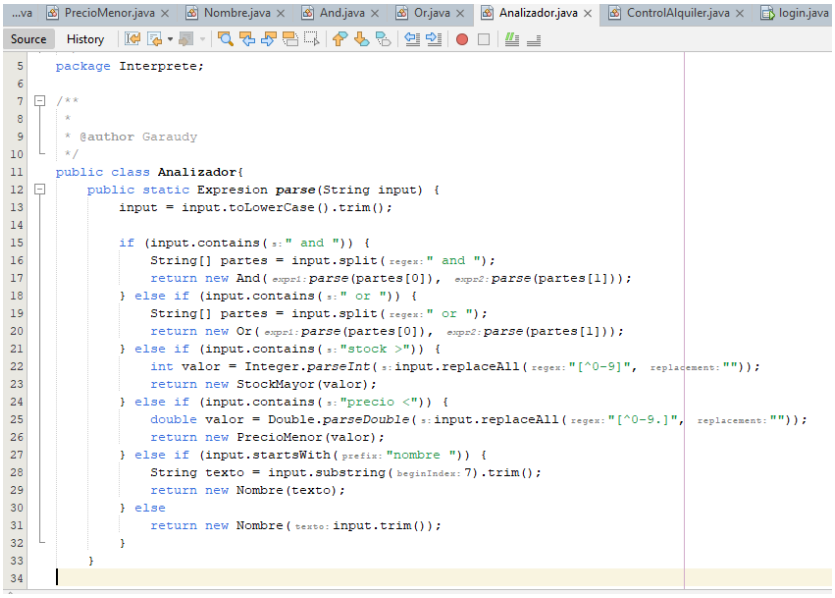


Devuelve true si al menos una condición es verdadera



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit th
 */
package Interprete;

import modelo.ModeloProducto;

/**
 *
 * @author Garaudy
 */
public class Or implements Expresion {
    private Expresion expr1;
    private Expresion expr2;

    public Or(Expresion expr1, Expresion expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }

    public boolean interpretar(ModeloProducto producto) {
        return expr1.interpretar(producto) || expr2.interpretar(producto);
    }
}
```

Esta clase convierte el texto ingresado por el usuario que implementan la interfaz Expresión.



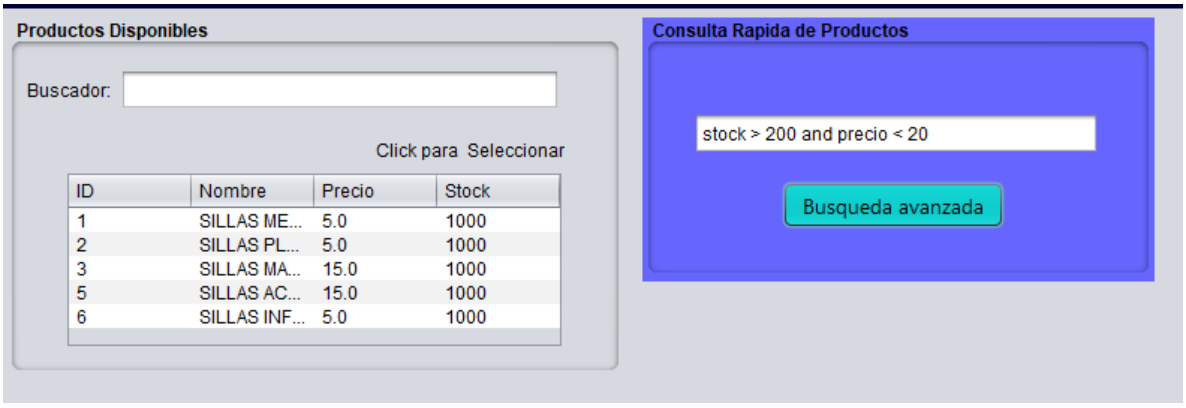
```java
package Interprete;

/**
 *
 * @author Garaudy
 */
public class Analizador{
    public static Expresion parse(String input) {
        input = input.toLowerCase().trim();

        if (input.contains(" and ")) {
            String[] partes = input.split(" and ");
            return new And(parse(partes[0]), parse(partes[1]));
        } else if (input.contains(" or ")) {
            String[] partes = input.split(" or ");
            return new Or(parse(partes[0]), parse(partes[1]));
        } else if (input.contains("stock >")) {
            int valor = Integer.parseInt(input.replaceAll("[^0-9]", ""));
            return new StockMayor(valor);
        } else if (input.contains("precio <")) {
            double valor = Double.parseDouble(input.replaceAll("[^0-9.]", ""));
            return new PrecioMenor(valor);
        } else if (input.startsWith("nombre ")) {
            String texto = input.substring(7).trim();
            return new Nombre(texto);
        } else
            return new Nombre(input.trim());
        }
    }
```

```java
public void BuscarProductoAvanzado(String consulta, JTable tablaProducto) {
configuracion.Conexion conexion = new configuracion.Conexion();
DefaultTableModel modelo = new DefaultTableModel();

modelo.addColumn(columnName: "ID");
modelo.addColumn(columnName: "Nombre");
modelo.addColumn(columnName: "Precio");
modelo.addColumn(columnName: "Stock");

tablaProducto.setModel(dataModel: modelo);

try {
    String sql = "SELECT * FROM producto";
    PreparedStatement ps = conexion.estableceConexion().prepareStatement(sql);
    ResultSet rs = ps.executeQuery();

    Interprete.Expresion expresion = Interprete.Analizador.parse(input: consulta);

    while (rs.next()) {
        modelo.ModeloProducto producto = new modelo.ModeloProducto();
        producto.setIdProducto(idProducto: rs.getInt(columnLabel: "idproducto"));
        producto.setNombreProducto(nombreProducto: rs.getString(columnLabel: "nombre"));
        producto.setPrecioProducto(precioProducto: rs.getDouble(columnLabel: "precioProducto"));
        producto.setStockProducto(stockProducto: rs.getInt(columnLabel: "stock"));

        if (expresion.interpretar(producto)) {
            modelo.addRow(new Object[]{
                producto.getIdProducto(),
                producto.getNombreProducto(),
```

Evalúa cada producto con la Expresión, solo los que devuelven true se muestran en la tabla de productos.

## PRUEBAS

stock > 200 and precio < 20

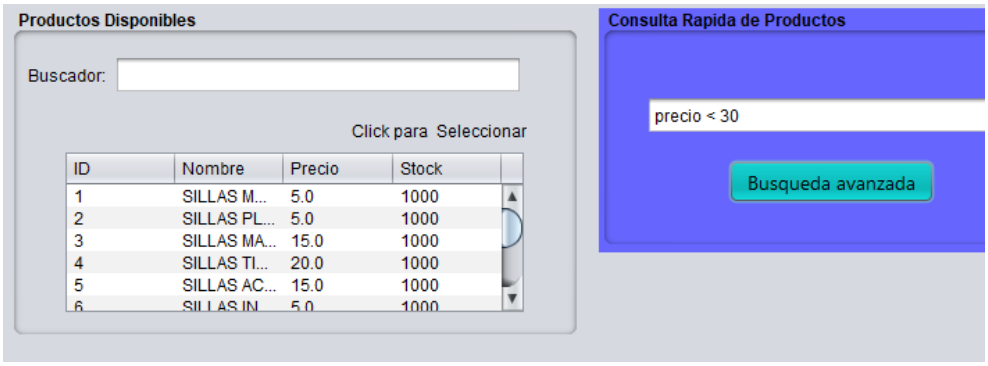


silla or stock < 100

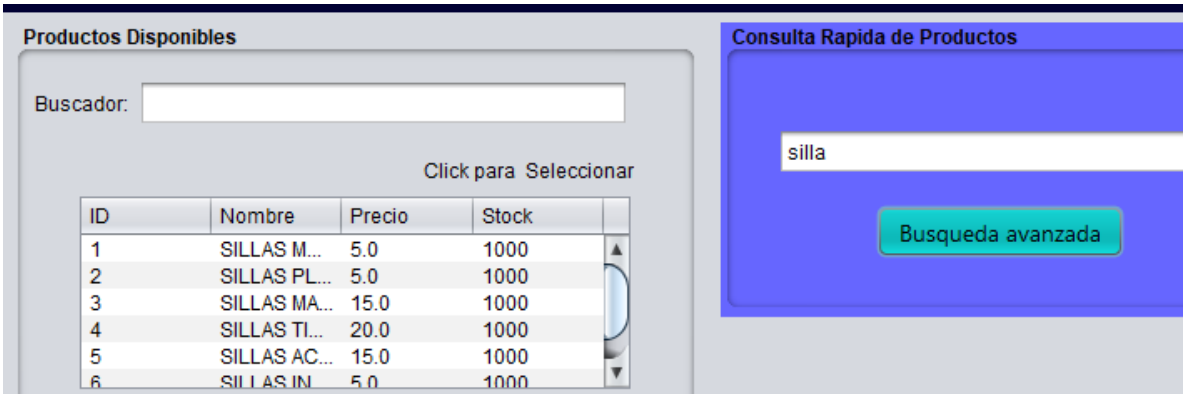

mesa


stock > 100  Productos con más de 100 unidades disponibles

precio < 30  Productos que cuesten menos de 30

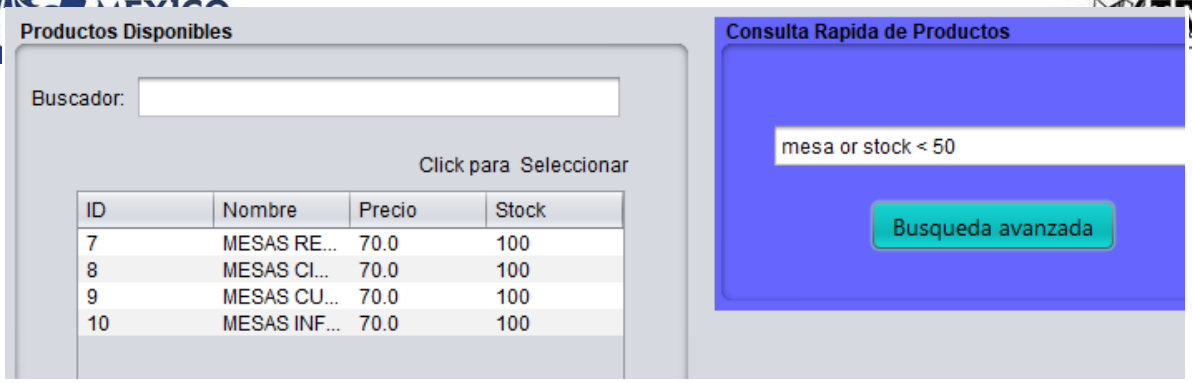



silla  Productos cuyo nombre contiene la palabra “silla”




stock > 100 and precio < 20  Productos con stock alto y precio bajo


mesa or stock < 50  Mesas o productos con poco stock




plato and precio < 10