



Mejoras realizadas

En la primera versión solo se tenía las clases ideadas y conexiones entre ciertas clases e interfaces, en la mejora se agregaron los atributos, métodos además que se organizó para cumplir con la mayoría de los principios Solid

Usuario

```

1 public class Usuario {
2     protected int id;
3     protected String nombre;
4     protected String email;
5
6     public Usuario(String nombre, String email) {
7         this.nombre = nombre;
8         this.email = email;
9     }
10
11    public void iniciarSesion() {
12        System.out.println(nombre + " inició sesión");
13    }
14
15
16
17
18
19
20
21
22
23

```

```

1 public class Usuario {
2     protected int id;
3     protected String nombre;
4     protected String email;
5
6     public Usuario(int id, String nombre, String email) {
7         this.id = id;
8         this.nombre = nombre;
9         this.email = email;
10    }
11
12    public void iniciarSesion() {
13        System.out.println(nombre + " inició sesión");
14    }
15
16    public void cerrarSesion() {
17        System.out.println(nombre + " cerró sesión");
18    }
19
20    public void mostrarDatos() {
21        System.out.println(id + " - " + nombre + " - " + email);
22    }
23

```

SRP

La clase Usuario tiene como única responsabilidad gestionar la información y acciones básicas de un usuario (id, nombre, email, iniciar/cerrar sesión).

Cliente

```
public class Cliente extends Usuario {
    private Carrito carrito;

    public Cliente(String nombre, String email) {
        super(nombre, email);
        carrito = new Carrito();
    }

    public Carrito getCarrito() {
        return carrito;
    }
}
```

```
public class Cliente extends Usuario {
    private Carrito carrito;
    private String direccion;
    private String telefono;

    public Cliente(int id, String nombre, String email, String direccion, String telefono) {
        super(id, nombre, email);
        this.direccion = direccion;
        this.telefono = telefono;
        this.carrito = new Carrito();
    }

    public Carrito getCarrito() {
        return carrito;
    }

    public void actualizarDireccion(String direccion) {
        this.direccion = direccion;
    }

    public void mostrarCliente() {
        System.out.println(nombre + " - " + direccion + " - " + telefono);
    }
}
```

SRP

Cliente extiende a Usuario y añade únicamente información y comportamiento específico del cliente (carrito, dirección, teléfono).

LSP

Cliente puede sustituir a Usuario sin alterar el comportamiento esperado, ya que respeta la estructura base.

Carrito

```
public class Carrito {
```

```
import java.util.ArrayList;
public class Carrito {
    private int id;
    private ArrayList<ItemCarrito> items;
    private double total;

    public Carrito() {
        this.id = 1;
        items = new ArrayList<>();
        total = 0;
    }

    public void agregarProducto(Producto p, int cantidad) {
        items.add(new ItemCarrito(p, cantidad));
    }

    public double calcularTotal() {
        total = 0;
        for (ItemCarrito item : items) {
            total += item.calcularSubtotal();
        }
        return total;
    }

    public void vaciarCarrito() {
        items.clear();
    }
}
```

SRP

Carrito administra los productos agregados y calcula el total.

No procesa pagos ni genera facturas.

OCP

Puede ampliarse agregando nuevos tipos de ItemCarrito sin modificar su estructura principal.

Producto

The image shows two side-by-side code editors. Both have a toolbar at the top with 'Compilar', 'Deshacer', 'Cortar', 'Copiar', 'Pegar', 'Buscar...', and 'Cerrar' buttons. The left editor has a dropdown menu 'Código Fuente'. The right editor also has a dropdown menu 'Código Fuente'.

Left Editor (Initial State):

```
1 public class Producto {  
2 }  
3  
4 }
```

Right Editor (Final State):

```
1 public class Producto {  
2     private int id;  
3     private String nombre;  
4     private double precio;  
5  
6     public Producto(int id, String nombre, double precio) {  
7         this.id = id;  
8         this.nombre = nombre;  
9         this.precio = precio;  
10    }  
11  
12    public double getPrecio() {  
13        return precio;  
14    }  
15  
16    public void actualizarPrecio(double precio) {  
17        this.precio = precio;  
18    }  
19  
20    public void mostrarProducto() {  
21        System.out.println(id + " - " + nombre + " - $" + precio);  
22    }  
23}  
24
```

A status bar at the bottom of the right editor says 'guardado'.

SRP

Producto solo gestiona información relacionada con un producto (id, nombre, precio) y su modificación.

ItemCarrito

The image shows two side-by-side code editors. Both have a toolbar at the top with 'Compilar', 'Deshacer', 'Cortar', 'Copiar', 'Pegar', 'Buscar...', and 'Cerrar' buttons. The left editor has a dropdown menu 'Código Fuente'. The right editor also has a dropdown menu 'Código Fuente'.

Left Editor (Initial State):

```
1 public class ItemCarrito {  
2 }  
3  
4 }
```

Right Editor (Final State):

```
1 public class ItemCarrito {  
2     private Producto producto;  
3     private int cantidad;  
4     private double subtotal;  
5  
6     public ItemCarrito(Producto producto, int cantidad) {  
7         this.producto = producto;  
8         this.cantidad = cantidad;  
9         this.subtotal = producto.getPrecio() * cantidad;  
10    }  
11  
12    public double calcularSubtotal() {  
13        subtotal = producto.getPrecio() * cantidad;  
14        return subtotal;  
15    }  
16  
17    public void aumentarCantidad(int cantidad) {  
18        this.cantidad += cantidad;  
19    }  
20  
21    public void mostrarItem() {  
22        System.out.println(producto + " x" + cantidad);  
23    }  
24}
```

A status bar at the bottom of the right editor says 'Guardado'.

SRP

Se encarga únicamente de representar un producto dentro del carrito junto con su cantidad y subtotal.

Pedido

The screenshot shows two side-by-side code editors. Both have a toolbar at the top with 'Compilar', 'Deshacer', 'Cortar', 'Copiar', 'Pegar', 'Buscar...', 'Cerrar', and 'Código Fuente'. The left editor contains a very simple implementation of the Pedido class:1 public class Pedido {
2 }
3 }The right editor contains a more complex implementation of the Pedido class, showing methods for setting the ID, calculating the total, applying discounts, processing payments, and displaying the order details:1 public class Pedido {
2 private int id;
3 private double total;
4 private String estado;
5 private Factura factura; // + ahora sí está asociada
6
7 public Pedido(int id, Carrrito carrito) {
8 this.id = id;
9 this.total = carrito.calcularTotal();
10 this.estado = "Pendiente";
11 }
12
13 public void aplicarDescuento(Descuento descuento) {
14 total = descuento.aplicar(total);
15 }
16
17 public void procesarPago(Pago pago) {
18 pago.pagar(total);
19 estado = "Pagado";
20 factura = new Factura(id, "24/02/2026", total);
21 factura.generarFactura();
22 }
23
24 public void mostrarPedido() {
25 System.out.println("Pedido #"+ id + " - \$" + total + " - " + estado);
26 }
27}

SRP

Pedido se encarga exclusivamente de gestionar la compra: calcular total, aplicar descuento y procesar pago.

DIP

Pedido no depende directamente de PagoTarjeta ni PagoEfectivo.

Depende de la abstracción Pago (interfaz).

Factura

The screenshot shows two side-by-side code editors. Both have a toolbar at the top with 'Compilar', 'Deshacer', 'Cortar', 'Copiar', 'Pegar', 'Buscar...', 'Cerrar', and 'Código Fuente'. The left editor contains a very simple implementation of the Factura class:1 public class Factura {
2 }
3
4 }The right editor contains a more complex implementation of the Factura class, showing methods for generating the invoice, sending it, and displaying its details:1 public class Factura {
2 private int numero;
3 private String fecha;
4 private double total;
5
6 public Factura(int numero, String fecha, double total) {
7 this.numero = numero;
8 this.fecha = fecha;
9 this.total = total;
10 }
11
12 public void generarFactura() {
13 System.out.println("Factura #"+ numero + " - \$" + total);
14 }
15
16 public void enviarFactura() {
17 System.out.println("Factura enviada");
18 }
19
20 public void mostrarFactura() {
21 System.out.println(numero + " - " + fecha + " - \$" + total);
22 }
23}A small 'guardado' message is visible in the bottom right corner of the right editor.

Principio aplicado: SRP

Factura se encarga únicamente de generar y mostrar la información de la factura.

No procesa pagos ni aplica descuentos.

Pago

Tarjeta

The screenshot shows two side-by-side code editors. Both have identical toolbars at the top with buttons for 'Compilar' (Compile), 'Deshacer' (Undo), 'Cortar' (Cut), 'Copiar' (Copy), 'Pegar' (Paste), 'Buscar...' (Search), and 'Cerrar' (Close). The left editor has a single line of code: '1 public class PagoTarjeta implements Pago {'. The right editor contains the full implementation of the PagoTarjeta class:

```
1 public class PagoTarjeta implements Pago {
2     private String numeroTarjeta;
3     private String titular;
4     private String banco;
5
6     public PagoTarjeta(String numeroTarjeta, String titular, String banco) {
7         this.numeroTarjeta = numeroTarjeta;
8         this.titular = titular;
9         this.banco = banco;
10    }
11
12    public void pagar(double monto) {
13        System.out.println("Pago con tarjeta por $" + monto);
14    }
15
16    public String obtenerTipo() {
17        return "Tarjeta";
18    }
19
20    public void generarComprobante() {
21        System.out.println("Comprobante generado - Tarjeta");
22    }
23}
```

LSP

Puede sustituir a Pago sin alterar el funcionamiento del sistema.

DIP

Es una implementación concreta de la abstracción Pago.

Efectivo

The screenshot shows two side-by-side code editors. Both have identical toolbars at the top with buttons for 'Compilar' (Compile), 'Deshacer' (Undo), 'Cortar' (Cut), 'Copiar' (Copy), 'Pegar' (Paste), 'Buscar...' (Search), and 'Cerrar' (Close). The left editor has a single line of code: '1 public class PagoEfectivo implements Pago {'. The right editor contains the full implementation of the PagoEfectivo class:

```
1 public class PagoEfectivo implements Pago {
2     private double montoRecibido;
3     private double cambio;
4     private String cajero;
5
6     public PagoEfectivo(double montoRecibido, String cajero) {
7         this.montoRecibido = montoRecibido;
8         this.cajero = cajero;
9     }
10
11    public void pagar(double monto) {
12        cambio = montoRecibido - monto;
13        System.out.println("Pago en efectivo. Cambio: $" + cambio);
14    }
15
16    public String obtenerTipo() {
17        return "Efectivo";
18    }
19
20    public void generarComprobante() {
21        System.out.println("Comprobante generado - Efectivo");
22    }
23}
```

LSP

Puede reemplazar a cualquier implementación de Pago.

DIP

Depende de la interfaz Pago, no del sistema directamente.

Descuento

DescuentoPorcentaje

```
1 public class DescuentoPorcentaje implements Descuento {  
2     private double porcentaje;  
3     private String motivo;  
4     private String codigo;  
5  
6     public DescuentoPorcentaje(double porcentaje, String motivo, String codigo) {  
7         this.porcentaje = porcentaje;  
8         this.motivo = motivo;  
9         this.codigo = codigo;  
10    }  
11  
12    public double aplicar(double total) {  
13        return total - (total * porcentaje / 100);  
14    }  
15  
16    public String tipoDescuento() {  
17        return "Porcentaje";  
18    }  
19  
20    public void mostrarDescuento() {  
21        System.out.println("Descuento " + porcentaje + "% - " + motivo);  
22    }  
23}
```

LSP

Puede sustituir a la interfaz Descuento sin alterar el comportamiento del sistema.

OCP

Se puede crear otro tipo de descuento (por ejemplo, DescuentoFijo) sin modificar el código existente.