

Instituto Tecnológico de Oaxaca
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES



**Diseño E Implementacion De Software Con
Patrones**

Profesora: Espinosa Pérez Jacob

P R E S E N T A:

**Luisa Michel Mendez Mendoza
Fernando Girón Pacheco
Mario Alberto Barbosa Santiago
Lourdes Gloria López García
Samuel Pérez Carrasco**

ÍNDICE

INTRODUCCION.....	2
Documentación de todos los patrones en el sistema de ventas.....	3
UML de la implementacion de los patrones.....	3
Diagrama de los patrones que trabajan en conjunto.....	3
Diagrama de los patrones que trabajan en conjunto.....	5
Paquetes y estructura del sistema.....	6
Implementacion de patrones	6
Patrones que trabajan en conjunto.....	6
Patrones de Comportamiento.....	15
1. Mediator:.....	15
2. Observer:.....	15
3. Responsabilidad (Chain of Responsibility):	17
4. Iterator:	17
5. Memento:.....	18
6. Prototype:	20
7. Command:	20
Patrones Creacionales	23
1. Singleton:.....	23
2. Builder:	25
3. Factory:.....	26
Patrones Estructurales	28
1. Proxy:	28
2. Flyweith:	26.
3. Adapter:	29
4. Facade:.....	30
5. Composite:	32
CONCLUSION.....	36

INTRODUCCION

Este documento describe la implementación de varios patrones de diseño clave dentro de un sistema de ventas, categorizados en patrones de creación, estructurales y de comportamiento. Estos patrones mejoran colectivamente la flexibilidad, la capacidad de mantenimiento y la organización del sistema..

Los patrones de creación se centran en la creación de objetos, con Builder utilizado para construir objetos "Comprobante" complejos y Singleton garantizando instancias únicas para conexiones de bases de datos y sesiones de usuario. Los patrones estructurales, incluidos Adapter, Facade y Proxy, gestionan la arquitectura del sistema: Adapter facilita la generación de PDF, Facade simplifica los procesos de venta y Proxy controla el acceso de los usuarios a los formularios. Los patrones de comportamiento rigen el comportamiento del sistema, empleando Comando para encapsular acciones, Iterador para recorrer estructuras de datos, Memento para la restauración del estado, Observador para actualizaciones automáticas, Cadena de Responsabilidad para la validación secuencial y Estrategia para la aplicación flexible de descuentos.

DOCUMENTACIÓN DE TODOS LOS PATRONES EN EL SISTEMA DE VENTAS

UML DE LA IMPLEMENTACION DE LOS PATRONES

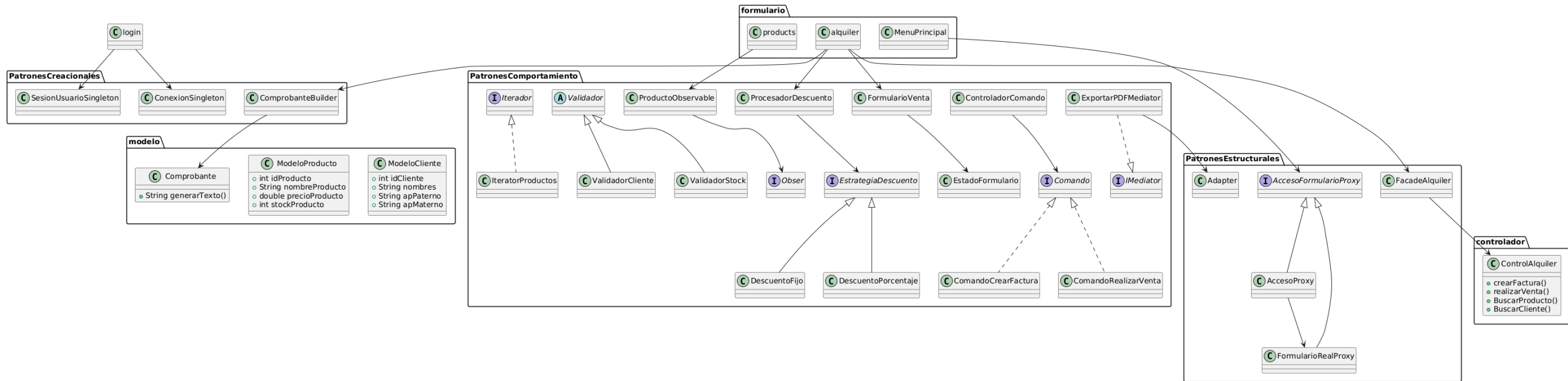


DIAGRAMA DE LOS PATRONES QUE TRABAJAN EN CONJUNTO

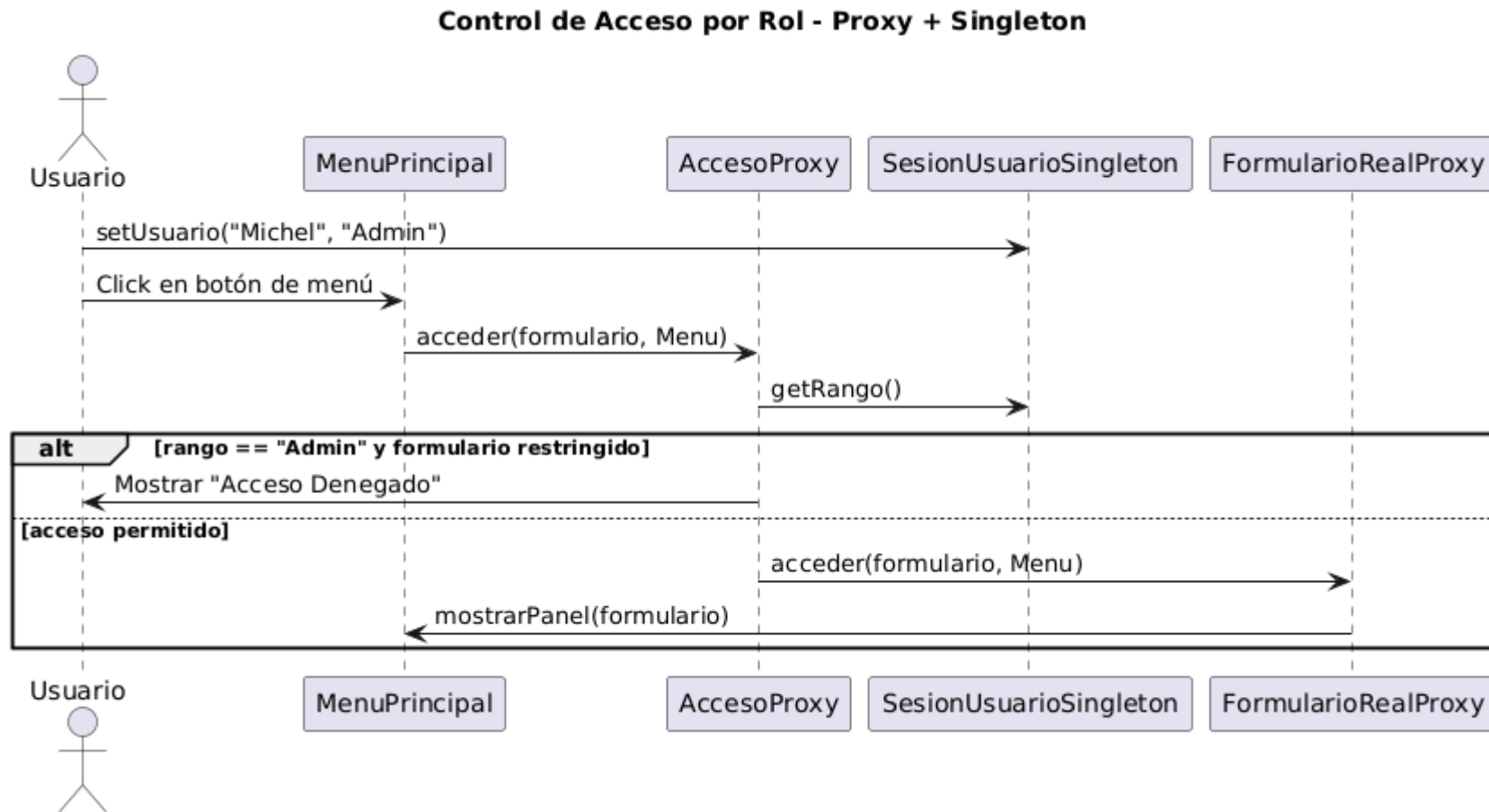
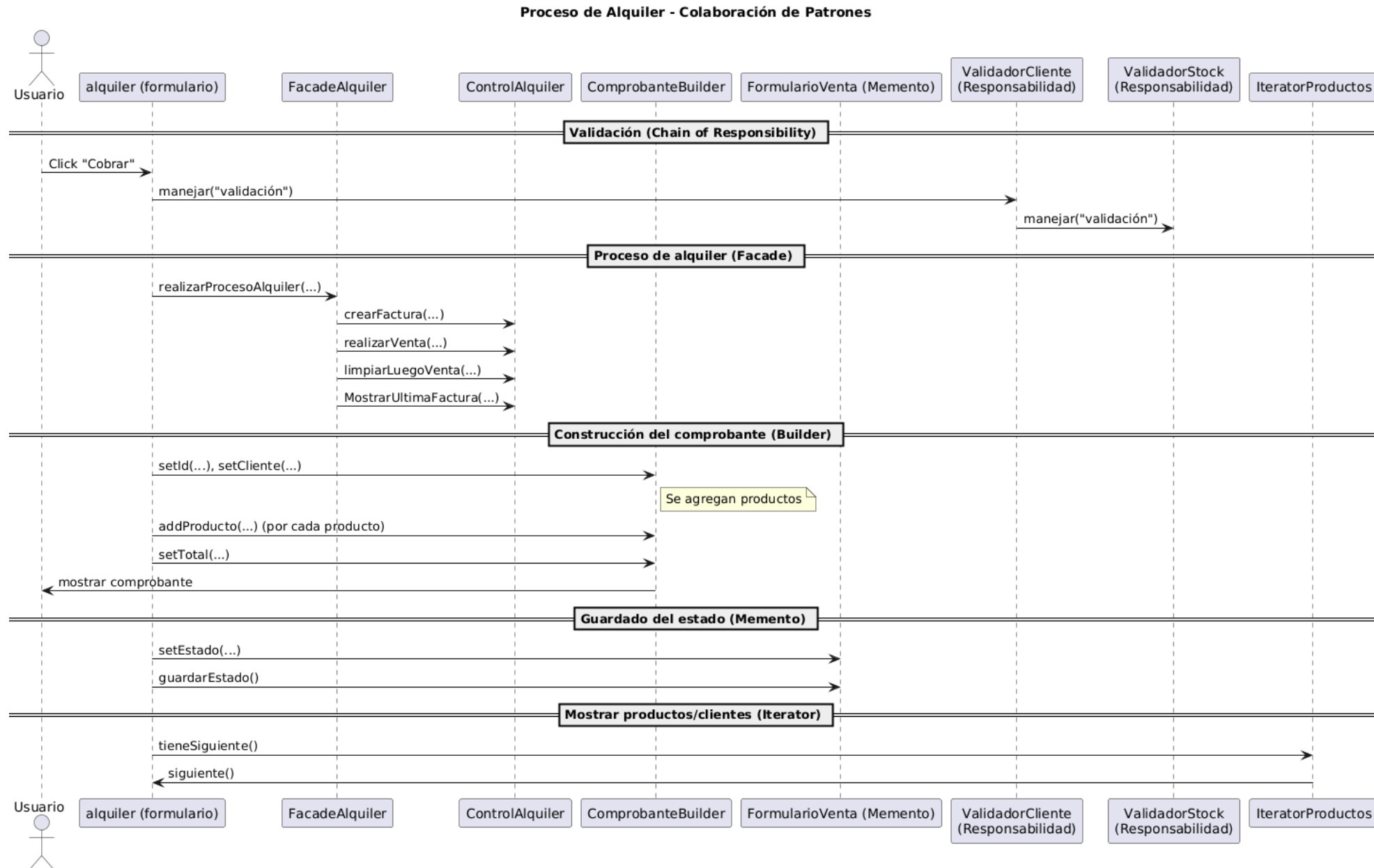




DIAGRAMA DE LOS PATRONES QUE TRABAJAN EN CONJUNTO



Nuestro proyecto es una aplicación de alquiler de productos, construida en Java con interfaz gráfica Swing. El sistema permite:

- Iniciar sesión con control de acceso (admin/empleado).
- Gestionar productos y clientes.
- Realizar procesos de facturación y ventas.
- Aplicar descuentos.
- Generar reportes en PDF.
- Restaurar estados anteriores de formularios.

La arquitectura está modularizada en capas y patrones de diseño para lograr bajo acoplamiento, cohesión alta y escalabilidad.

PAQUETES Y ESTRUCTURA DEL SISTEMA

1. controlador

Contiene la lógica de negocio central (ControlAlquiler), como creación de facturas, ventas, búsqueda y cálculo de totales.

2. formulario

Interfaz gráfica (alquiler, products, MenuPrincipal). Los formularios invocan servicios y patrones.

3. modelo

Representa las entidades del dominio: ModeloCliente, ModeloProducto, Comprobante.

4. PatronesComportamiento

Implementa patrones de comportamiento como Command, Observer, Strategy, Memento, Iterator, Chain of Responsibility, Mediator.

5. PatronesCreacionales

Incluye patrones como Singleton (para conexión y sesión) y Builder (para Comprobante).

6. PatronesEstructurales

Implementa Adapter (exportación PDF), Facade (orquestración de venta), y Proxy (control de acceso a formularios).

IMPLEMENTACION DE PATRONES

PATRONES QUE TRABAJAN EN CONJUNTO

La colaboración de patrones ocurre en el proceso de alquiler de productos. En ese flujo trabajan juntos los siguientes patrones:

Patrón	Categoría	Rol en la colaboración
Facade	Estructural	Coordina todo el proceso de alquiler
Composite	Comportamiento	El patron Composite y que con ayuda al implementar el patron Mediator nos ayuda a Exportar en PDF
Mediator	Comportamiento	: Centraliza la comunicación entre el formulario y el adaptador PDF, evitando que se acoplen directamente.
Builder	Creacional	Construye el comprobante de alquiler
Memento	Comportamiento	Guarda y restaura el estado del formulario
Responsabilidad	Comportamiento	Valida cliente y stock de productos
Iterator	Comportamiento	Recorre productos y clientes mostrados

FacadeAlquiler (Patrón Facade)

Objetivo:

Simplificar el proceso de alquiler agrupando múltiples llamadas internas en un solo método.

Código:

```

12
13  ✓ public class FacadeAlquiler {
14      private final ControlAlquiler control;
15
16  ✓ public FacadeAlquiler() {
17      // en lugar de new ControlAlquiler():
18      IControlador ctrl = new ControladorFactory().crear("alquiler");
19      // casteamos al tipo concreto para seguir usando sus métodos
20      this.control = (ControlAlquiler) ctrl;
21  }
22
23  /**
24   * Realiza el proceso completo de crear factura y registrar la venta.
25   *
26   * @param cliente Campo de texto con el ID del cliente
27   * @param productos Tabla resumen de productos seleccionados
28   */
29  ✓ public void realizarAlquiler(JTextField cliente, JTable productos) {
30      if (cliente.getText().isEmpty() || productos.getRowCount() == 0) {
31          JOptionPane.showMessageDialog(null,
32              "Debe seleccionar un cliente y agregar productos antes de alquilar.");
33          return;
34      }
35
36      control.crearFactura(cliente);
37      control.realizarVenta(productos);
38
39      JOptionPane.showMessageDialog(null, "¡Alquiler completado exitosamente!");
40  }
41  }

```


Mediator y Composite

```
package PatronesComportamiento;

import javax.swing.*;

✓ public interface IMediator {
    void exportarPDF(JLabel lblFactura, JLabel lblFechaFactura, JLabel lblNombreCliente,
                    JLabel lblAppaterno, JLabel lblApmaterno,
                    JTable tablaProductos, JLabel lblIVA, JLabel lblTotal);
}

/* los pasos concretos, delegando a tu Adapter para el volcado real.
*/
✓ public class ExportarPDFMediator extends ExportadorPDFTemplate {

    private final Adapter adaptadorPDF;

    public ExportarPDFMediator() {
        this.adaptadorPDF = new Adapter();
    }

    @Override
    protected void generarCabecera(JLabel lblFactura,
                                    JLabel lblFechaFactura,
                                    JLabel lblNombreCliente,
                                    JLabel lblAppaterno,
                                    JLabel lblApmaterno) {
        // Aquí "arma" la cabecera usando tu adapter
        adaptadorPDF.agregarEncabezado(
            lblFactura, lblFechaFactura,
            lblNombreCliente, lblAppaterno, lblApmaterno
        );
    }

    @Override
    protected void generarContenido(JTable tablaProductos) {
        // Volcar la tabla al PDF
        adaptadorPDF.agregarTabla(tablaProductos);
    }

    @Override
    protected void generarPie(JLabel lblIVA, JLabel lblTotal) {
        // Agregar líneas finales (IVA, totales)
        adaptadorPDF.agregarTotales(lblIVA, lblTotal);
    }
}
```

Estructura para el PDF

```

public abstract class ExportadorPDFTemplate implements IMediator {

    @Override
    public final void exportarPDF(JLabel lblFactura,
                                JLabel lblFechaFactura,
                                JLabel lblNombreCliente,
                                JLabel lblAppaterno,
                                JLabel lblApmaterno,
                                JTable tablaProductos,
                                JLabel lblIVA,
                                JLabel lblTotal) {

        iniciarDocumento();
        generarCabecera(lblFactura, lblFechaFactura, lblNombreCliente, lblAppaterno, lblApmaterno);
        generarContenido(tablaProductos);
        generarPie(lblIVA, lblTotal);
        finalizarDocumento();
    }

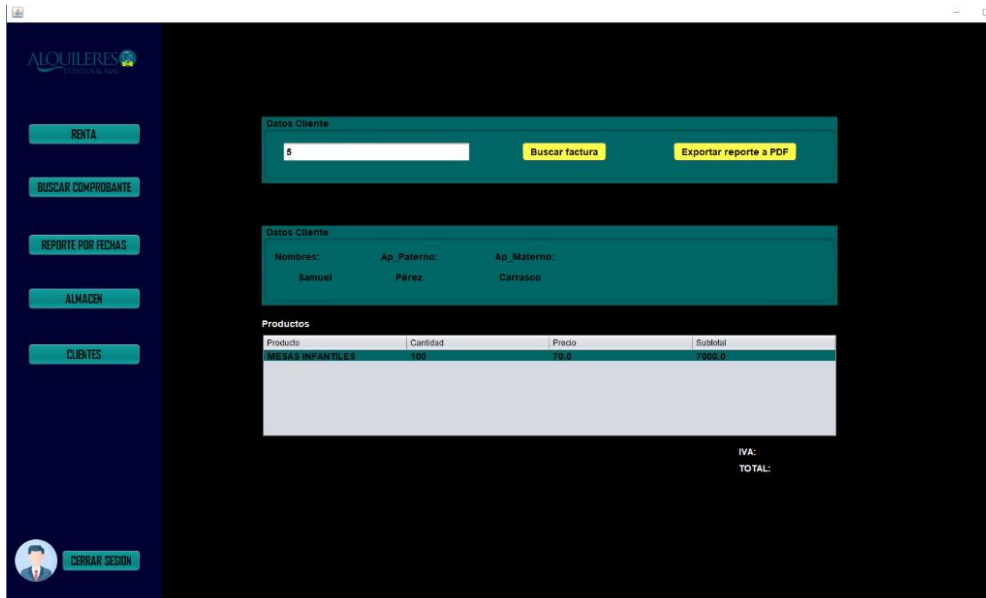
    /** Paso fijo (hook): configuración previa al PDF */
    protected void iniciarDocumento() {
        System.out.println("🔧 Iniciando exportación de PDF...");
        // Por ejemplo: crear carpeta, abrir stream, etc.
    }

    /** Paso variable: la subclase define cómo crea la cabecera */
    protected abstract void generarCabecera(JLabel lblFactura,
                                            JLabel lblFechaFactura,
                                            JLabel lblNombreCliente,
                                            JLabel lblAppaterno,
                                            JLabel lblApmaterno);

    /** Paso variable: la subclase define cómo vuelca la tabla */
    protected abstract void generarContenido(JTable tablaProductos);
}

```

EJECUCION:



The screenshot shows the 'ALQUILERES' application interface. On the left is a sidebar with navigation buttons: RENTA, BUSCAR COMPROBANTE, REPORTE POR FECHAS, ALMACEN, and CLIENTES. The main area displays the 'Datos Cliente' section with a search bar containing 'S', a 'Buscar factura' button, and an 'Exportar reporte a PDF' button. Below this, the 'Productos' section shows a table with the following data:

Producto	Cantidad	Precio	Subtotal
MESES INFANTILES	100	70.0	7000.0

At the bottom right, there are labels for 'IVA:' and 'TOTAL:'.

=====

COMPROBANTE DE ALQUILER

=====

Factura N°: 6

Fecha de Venta: 2025-05-22

Cliente: Samuel Pérez Carrasco

Producto	Cantidad	Precio Venta	Subtotal
SILLAS MADERA	100	15.0	1500.0

IVA: 270

Total a Pagar: 1500

```
1  * @author Michel Mendez
2  */
3  public class ComprobanteBuilder {
4      private int id;
5      private String cliente;
6      private List<String> productos = new ArrayList<>();
7      private double total;
8
9      public ComprobanteBuilder setId(int id) {
10         this.id = id;
11         return this;
12     }
13     public ComprobanteBuilder setCliente(String cliente) {
14         this.cliente = cliente;
15         return this;
16     }
17     public ComprobanteBuilder addProducto(String producto) {
18         productos.add(producto);
19         return this;
20     }
21     public ComprobanteBuilder setTotal(double total) {
22         this.total = total;
23         return this;
24     }
25     public Comprobante build() {
26         return new Comprobante(id, cliente, productos, total);
27     }
28 }
```

ComprobanteBuilder (Patrón Builder)

Objetivo:

Construye un objeto Comprobante paso a paso, añadiendo productos y configurando propiedades.

Código:

```
//metodo de cobrar usando Facade + Command + Strategy + Builder

private void btncobrarActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_btncobrarActionPerformed
    if (!validarAntesDeCobrar()) return;

    FacadeAlquiler fachada = new FacadeAlquiler();
    fachada.realizarAlquiler(txtSidcliente, tbresumenventa);

    ComprobanteBuilder builder = new ComprobanteBuilder()
        .setCliente(txtSnombrecliente.getText())
        .setId(Integer.parseInt(txtSidcliente.getText()));

    double total = 0;
    for (int i = 0; i < tbresumenventa.getRowCount(); i++) {
        builder.addProducto(tbresumenventa.getValueAt(i, 1).toString());
        total += Double.parseDouble(tbresumenventa.getValueAt(i, 4).toString());
    }
}
```

Memento (Guardar y restaurar estado)

Objetivo:

Permite restaurar el estado del formulario (cliente/producto) luego de una operación.

Código:

```
public static class FormularioVenta {
    private String cliente;
    private String producto;

    public void setEstado(String cliente, String producto) {
        this.cliente = cliente;
        this.producto = producto;
    }

    public EstadoFormulario guardarEstado() {
        return new EstadoFormulario(cliente, producto);
    }

    public void restaurarEstado(EstadoFormulario estado) {
        this.cliente = estado.getCliente();
        this.producto = estado.getProducto();
    }
}

}
```

```
* @author Michel Mendez
*/
public class Memento {
    public static class EstadoFormulario {
        private final String cliente;
        private final String producto;

        public EstadoFormulario(String cliente, String producto) {
            this.cliente = cliente;
            this.producto = producto;
        }

        public String getCliente() {
            return cliente;
        }

        public String getProducto() {
            return producto;
        }
    }
}
```

```
//metodo restaurar formulario usando memento

private void btnRestaurarFormularioActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_btnRestaurarFormularioActionPerformed
    if (mementoFormulario != null) {
        FormularioVenta formulario = new FormularioVenta();
        formulario.restaurarEstado(mementoFormulario);

        String[] datosCliente = mementoFormulario.getCliente().split(";");
        String[] datosProducto = mementoFormulario.getProducto().split(";");

        txtSidcliente.setText(datosCliente[0]);
        txtSnombrecliente.setText(datosCliente[1]);
        txtSidproducto.setText(datosProducto[0]);
        txtSnombreproducto.setText(datosProducto[1]);

        JOptionPane.showMessageDialog(null,
            "Formulario restaurado.\nCliente: " + datosCliente[1] +
            "\nProducto: " + datosProducto[1]);
    } else {
        JOptionPane.showMessageDialog(null, "No hay estado guardado para restaurar.");
    }
}

}
```

Responsabilidad (Chain of Responsibility)

Objetivo:

Validar que el cliente esté seleccionado y que haya stock disponible antes de alquilar.

Código:

```

/*
 * @author Michel Mendez
 */
public class Responsabilidad {

    public static abstract class Validador {
        protected Validador siguiente;
        public void setSiguiente(Validador siguiente) {
            this.siguiente = siguiente;
        }
        public void manejar(String dato) {
            if (siguiente != null) {
                siguiente.manejar(dato);
            }
        }
    }

    public static class ValidadorStock extends Validador {
        public void manejar(String producto) {
            System.out.println("Validando stock para " + producto);
            super.manejar(producto);
        }
    }

    public static class ValidadorCliente extends Validador {
        public void manejar(String producto) {
            System.out.println("Validando cliente para producto: " + producto);
            super.manejar(producto);
        }
    }
}

```

IteratorProductos (Patrón Iterator)

Objetivo:

Recorrer listas de productos o clientes y mostrarlos uno a uno de manera ordenada.

Código:

```

/*
 * @author Michel Mendez
 */
public class IteratorProductos implements Iterator {

    private List<String> productos;
    private int posicion = 0;

    public IteratorProductos(List<String> productos) {
        this.productos = productos;
    }

    public boolean tieneSiguiente() {
        return posicion < productos.size();
    }

    public Object siguiente() {
        return productos.get(posicion++);
    }
}

/*
 * @author Michel Mendez
 */
public class IteratorProductos implements Iterator {

    private List<String> productos;
    private int posicion = 0;

    public IteratorProductos(List<String> productos) {
        this.productos = productos;
    }

    public boolean tieneSiguiente() {
        return posicion < productos.size();
    }

    public Object siguiente() {
        return productos.get(posicion++);
    }
}

private void btnMostrarProductosIteratorActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_btnMostrarProductosIteratorActionPerformed
    // TODO add your handling code here:
    controlador.ControlAlquiler control = new controlador.ControlAlquiler();

    control.BuscarProducto(new JTextField(""), tbproductos);

    List<String> listaProductos = new ArrayList<>();
    for (int i = 0; i < tbproductos.getRowCount(); i++) {
        Object valor = tbproductos.getValueAt(i, 1); // columna "Nombre"
        if (valor != null) {
            listaProductos.add(valor.toString());
        }
    }

    // Usar Iterator
    PatronesComportamiento.Iterator iterador = new PatronesComportamiento.IteratorProductos(listaProductos);
    StringBuilder resumen = new StringBuilder("Productos disponibles:\n");
    while (iterador.tieneSiguiente()) {
        resumen.append("- ").append(iterador.siguiente()).append("\n");
    }
}

```

Patrón	Rol
Proxy	Intermediario que controla si un usuario puede acceder a un formulario.
Singleton	Almacena los datos de sesión del usuario: nombre y rango.

PROXY

```

10 public class Proxy {
11
12     public interface AccesoFormularioProxy {
13         void acceder(VistaFormulario vista, MenuPrincipal menu);
14     }
15
16     public static class AccesoProxy implements AccesoFormularioProxy {
17         private final FormularioRealProxy real = new FormularioRealProxy();
18         private final String boton;
19         // Nombres EXACTOS de los botones (en mayúsculas) que el rango "Empleado" puede abrir:
20         private static final List<String> permitidosEmpleado = Arrays.asList(
21             "CLIENTES",
22             "ALMACEN",
23             "REPORTE POR FECHAS",
24             "BUSCAR COMPROBANTE",
25             "RENTA"
26         );
27
28         public AccesoProxy(String boton) {
29             this.boton = boton;
30         }
31

```

```

@Override
public void acceder(VistaFormulario vista, MenuPrincipal menu) {
    String rango = SesionUsuarioSingleton.getInstance().getRango();
    System.out.println("Proxy.acceder(): boton=" + boton + ", rango=" + rango);

    boolean esAdmin = "ADMIN".equalsIgnoreCase(rango);
    boolean permitido = permitidosEmpleado.contains(boton.toUpperCase());

    if (esAdmin || permitido) {
        real.acceder(vista, menu);
    } else {
        JOptionPane.showMessageDialog(
            menu,
            "Acceso denegado a: " + boton +
            "\nRango actual: " + rango,
            "Acceso Denegado",
            JOptionPane.WARNING_MESSAGE
        );
    }
}

private static class FormularioRealProxy implements AccesoFormularioProxy {
    @Override
    public void acceder(VistaFormulario vista, MenuPrincipal menu) {
        vista.mostrar();
    }
}

```

Aquí AccesoProxy usa el **Singleton** para obtener el rango del usuario actual y decide si le da acceso o no.

SINGLETON

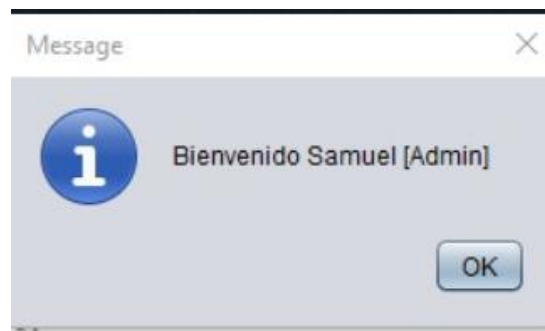
```

1 public class SesionUsuarioSingleton {
2     private static SesionUsuarioSingleton instancia;
3     private String nombreUsuario;
4     private String rango;
5     private SesionUsuarioSingleton() {}
6
7     public static SesionUsuarioSingleton getInstance() {
8         if (instancia == null) {
9             instancia = new SesionUsuarioSingleton();
10        }
11        return instancia;
12    }
13
14    public void setUsuario(String nombreUsuario, String rango) {
15        this.nombreUsuario = nombreUsuario;
16        this.rango = rango;
17    }
18
19    public String getNombreUsuario() {
20        return nombreUsuario;
21    }
22
23    public String getRango() {
24        return rango;
25    }
26
27 }

```

Este singleton guarda quién está logueado y cuál es su nivel de permiso (Admin, Empleado).

EJECUCION:



1. El usuario inicia sesión y se guarda en SesionUsuarioSingleton:

```
SesionUsuarioSingleton.getInstance().setUsuario("Michel", "Admin");
```

2. Al hacer clic en un botón de menú (clientes, renta, reportes, etc.):

```
AccesoFormularioProxy proxy = new Proxy().new AccesoProxy(" a rentas");  
proxy.acceder(new alquiler(), this); // this = MenuPrincipal
```

3. AccesoProxy:

- Llama a SesionUsuarioSingleton.getInstance().getRango()
- Verifica si el rol tiene permisos para acceder
- Si es válido, llama a FormularioRealProxy.acceder(), que muestra el panel

PATRONES DE COMPORTAMIENTO

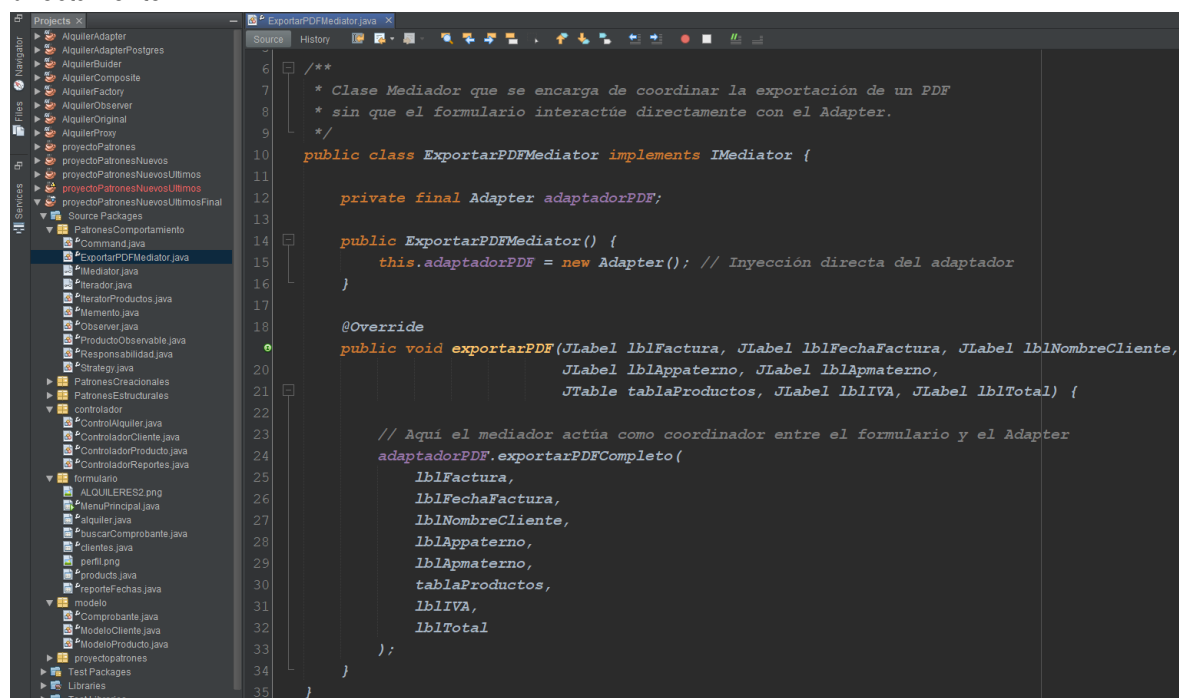
El paquete PatronesComportamiento de nuestro proyecto implementa varios patrones de comportamiento del diseño de software. A continuación, explicaremos cada uno con su propósito, uso dentro del proyecto y el código clave para entenderlo.

1. **MEDIATOR** – Coordina la comunicación entre objetos sin que se conozcan entre ellos

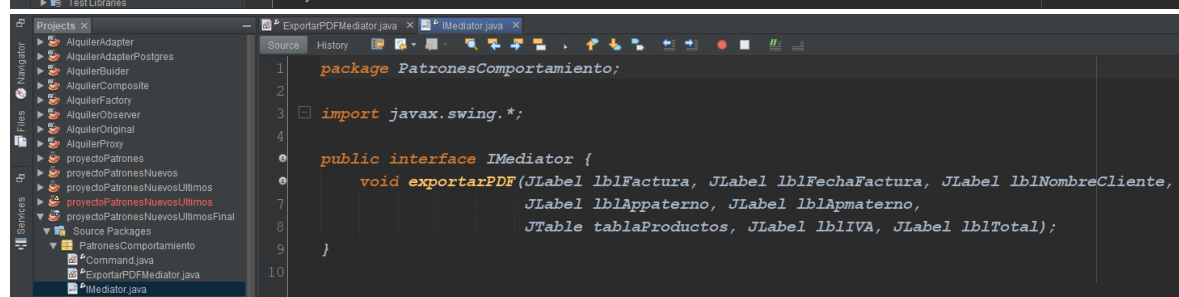
Clases:

- ExportarPDFMediator
- IMediator

Propósito: Centraliza la comunicación entre el formulario y el adaptador PDF, evitando que se acoplen directamente.



```
6  /**
7   * Clase Mediator que se encarga de coordinar la exportación de un PDF
8   * sin que el formulario interactúe directamente con el Adapter.
9   */
10 public class ExportarPDFMediator implements IMediator {
11
12     private final Adapter adaptadorPDF;
13
14     public ExportarPDFMediator() {
15         this.adaptadorPDF = new Adapter(); // Inyección directa del adaptador
16     }
17
18     @Override
19     public void exportarPDF(JLabel lblFactura, JLabel lblFechaFactura, JLabel lblNombreCliente,
20                           JLabel lblAppaterno, JLabel lblApmaterno,
21                           JTable tablaProductos, JLabel lblIVA, JLabel lblTotal) {
22
23         // Aquí el mediator actúa como coordinador entre el formulario y el Adapter
24         adaptadorPDF.exportarPDFCompleto(
25             lblFactura,
26             lblFechaFactura,
27             lblNombreCliente,
28             lblAppaterno,
29             lblApmaterno,
30             tablaProductos,
31             lblIVA,
32             lblTotal
33         );
34     }
35 }
```



```
1 package PatronesComportamiento;
2
3 import javax.swing.*;
4
5 public interface IMediator {
6     void exportarPDF(JLabel lblFactura, JLabel lblFechaFactura, JLabel lblNombreCliente,
7                     JLabel lblAppaterno, JLabel lblApmaterno,
8                     JTable tablaProductos, JLabel lblIVA, JLabel lblTotal);
9 }
10
```

Se usa para exportar la factura a PDF desde el formulario sin que este conozca los detalles del adaptador.

2 **OBSERVER** – Notificar a múltiples objetos cuando hay cambios

Clases:

- Observer.Observable
- Observer.Observer
- ProductoObservable

Propósito: Cuando se modifica un producto (agrega, edita, elimina), se notifica automáticamente a todos los observadores (como interfaces gráficas u otros módulos).

Código:

```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/
4   */
5  package PatronesComportamiento;
6
7  import controlador.ControladorProducto;
8  import java.util.ArrayList;
9  import java.util.List;
10
11  /**
12   *
13   * @author Michel Mendez
14   */
15
16  public class Observer {
17
18      public interface Obser {
19          void actualizar();
20      }
21
22      public interface Observable {
23          void agregarObservador(Obser o);
24          void eliminarObservador(Obser o);
25          void notificarObservadores();
26      }
27  }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594

```

3. CHAIN OF RESPONSIBILITY – Pasar una solicitud por una cadena de manejadores

Clases:

- Responsabilidad.Validador
- ValidadorStock
- ValidadorCliente

Propósito: Permite encadenar múltiples validadores (stock, cliente, etc.) y pasar una solicitud hasta que uno de ellos la procese.

Código:

```
Source  History  [Icons]
7  /**
8   *
9   * @author Michel Mendez
10  */
11  public class Responsabilidad {
12
13      public static abstract class Validador {
14          protected Validador siguiente;
15          public void setSiguiente(Validador siguiente) {
16              this.siguiente = siguiente;
17          }
18          public void manejar(String dato) {
19              if (siguiente != null) {
20                  siguiente.manejar(dato);
21              }
22          }
23      }
24
25      public static class ValidadorStock extends Validador {
26          public void manejar(String producto) {
27              System.out.println("Validando stock para " + producto);
28              super.manejar(dato: producto);
29          }
30      }
31
32      public static class ValidadorCliente extends Validador {
33          public void manejar(String producto) {
34              System.out.println("Validando cliente para producto: " + producto);
35              super.manejar(dato: producto);
36          }
37      }
38  }
```

Antes de realizar una venta o modificar un producto, se valida el stock, cliente, etc., en secuencia.

4. ITERATOR – Recorrer colecciones sin exponer su estructura interna

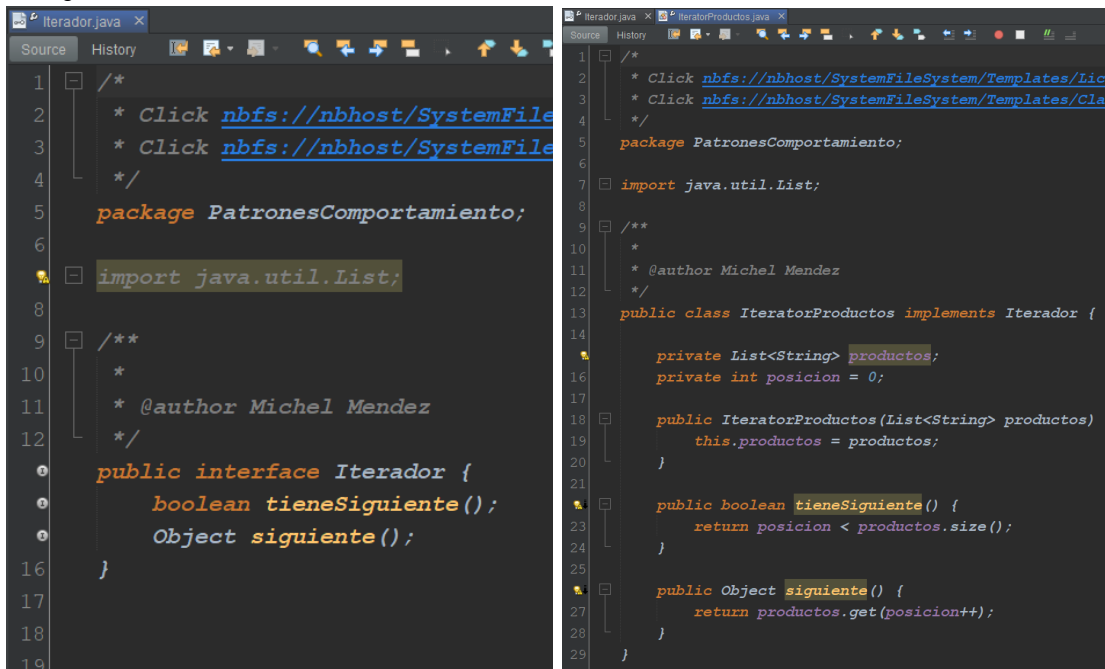
Clases:

- Iterador
- IteratorProductos

Propósito: Permite recorrer una lista de productos de manera uniforme sin exponer su estructura

interna.


Código:



```

1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java
4   */
5   package PatronesComportamiento;
6
7   import java.util.List;
8
9   /**
10    *
11    * @author Michel Mendez
12    */
13   public interface Iterador {
14       boolean tieneSiguiente();
15       Object siguiente();
16   }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
262
```

EJECUCION




RENTA

BUSCAR COMPROBANTE

REPORTE POR FECHAS

ALMACEN

CLIENTES


CERRAR SESION

Clientes Disponibles

Buscador: 1 Click para Seleccionar Mostrar Clientes

id	Nombres	Ap.Paterno	Ap.Materno
1	Samuel	Pérez	Carrasco

Productos Disponibles

Id.Cliente: 1

Nombres: Samuel

Ap.Paterno: Pérez

Ap.Materno: Carrasco

Id.Producto: 3

Nombre: SILLAS MADERA

Precio: 15.0

Stock: 1000

Productos Disponibles

Precio de Venta: 15.0 Cantidad de Venta: 100

Agregar Producto

Productos Disponibles

Ultima Factura Creada: Seleccionar para Eliminar: Eliminar


Elegir tipo de descuento: Fijo Aplicar descuento

idProducto	N.Producto	Precio Producto	Cantidad Producto	Sub Total
3	SILLAS MADERA	15.0	100	1500.0

IVA (18%): 270.00

TOTAL: 1500.00 COBRAR

RESTAURAR FORMULARIO




RENTA

BUSCAR COMPROBANTE

REPORTE POR FECHAS

ALMACEN

CLIENTES


CERRAR SESION

Clientes Disponibles

Buscador: Click para Seleccionar Mostrar Clientes

id	Nombres	Ap.Paterno	Ap.Materno
----	---------	------------	------------

Productos Disponibles

Id.Cliente:

Nombres:

Ap.Paterno:

Ap.Materno:

Id.Producto:

Nombre:

Precio:

Stock:

Productos Disponibles

Precio de Venta: Cantidad de Venta:

Agregar Producto

Productos Disponibles

Ultima Factura Creada: Seleccionar para Eliminar: Eliminar

Elegir tipo de descuento: Ninguno Aplicar descuento

idProducto	N.Producto	Precio Producto	Cantidad Producto	Sub Total
------------	------------	-----------------	-------------------	-----------

IVA (18%):

TOTAL: COBRAR

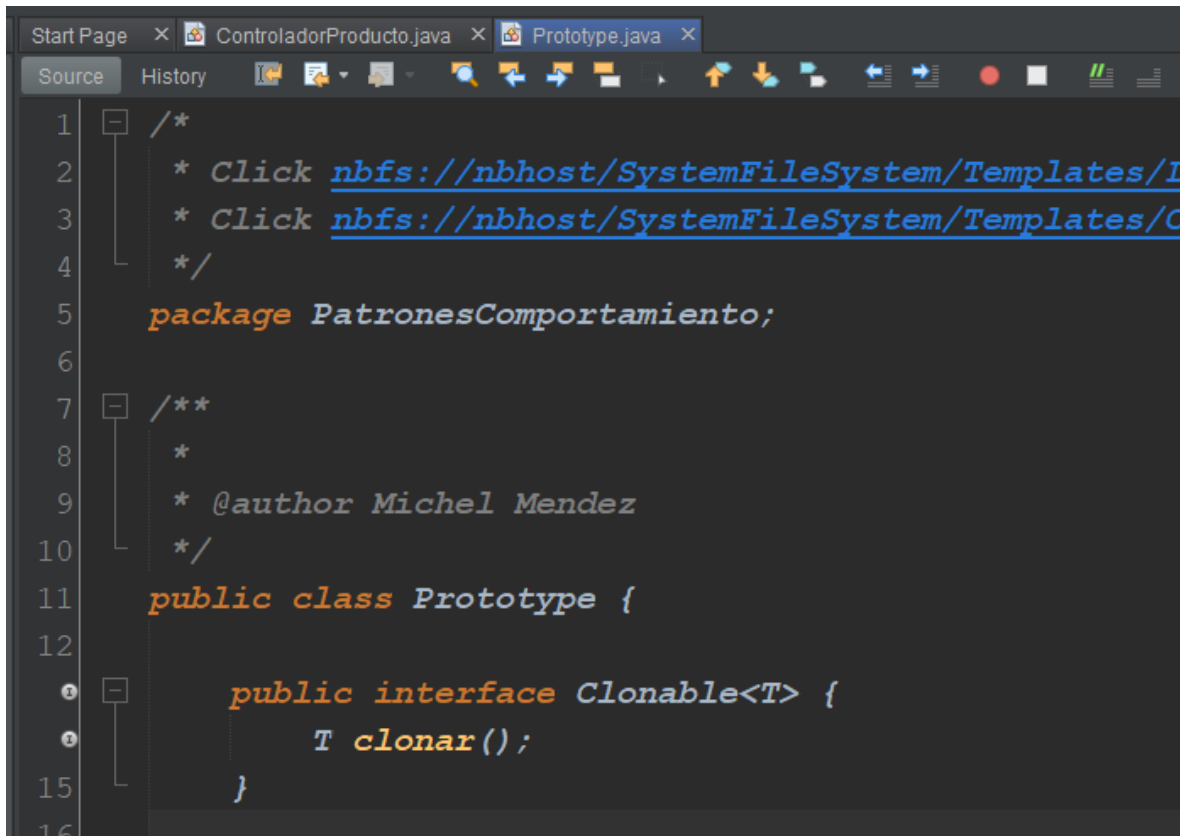
RESTAURAR FORMULARIO

6. PROTOTYPE

- Clase: ProductoPrototype (abstracta)
- Subclase: ProductoConcreto

Objetivo

Permitir **duplicar productos** para agregarlos a listas como comprobantes sin reescribir los datos ni usar new.



```
1  /*
2      * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses
3      * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/
4      */
5  package PatronesComportamiento;
6
7  /**
8      *
9      * @author Michel Mendez
10     */
11  public class Prototype {
12
13     public interface Clonable<T> {
14         T clonar();
15     }
16 }
```

Lo usamos cuando estamos trabajando en un alquiler, podemos clonar productos rápidamente sin instanciar uno nuevo

7. COMMAND

Objetivo:

Encapsular operaciones como **agregar**, **eliminar** o **limpiar productos** como comandos reutilizables y desacoplados.

```

Start Page x ControladorProducto.java x Prototype.java x Command.java x
Source History
15  * @author Michel Mendez
16  */
17  public class Command {
18      public interface Comando {
19          void ejecutar();
20      }
21
22      public class ComandoCrearFactura extends ControlAlquiler implements Comando {
23          private JTextField cliente;
24          public ComandoCrearFactura(JTextField cliente) {
25              this.cliente = cliente;
26          }
27
28          public void ejecutar() {
29              crearFactura(codcliente: cliente);
30          }
31      }
32
33      public class ComandoRealizarVenta extends ControlAlquiler implements Comando {
34          private JTable resumen;
35          public ComandoRealizarVenta(JTable resumen) {
36              this.resumen = resumen;
37          }
38
39          public void ejecutar() {
40              realizarVenta(resumen);
41          }
42      }
43
44      public class ControladorComando {
45          private List<Comando> historial = new ArrayList<>();
46          public void ejecutarComando(Comando comando) {
47              comando.ejecutar();
48              historial.add(e: comando);
49          }
50      }
51  }

```

APLICACIÓN:

```
//metodo de cobrar usando Facade + Command + Strategy + Builder

private void btncobrarActionPerformed(java.awt.event.ActionEvent evt) {GEN-FIRST:event_btncobrarActionPerformed
    if (!validarAntesDeCobrar()) return;

    FacadeAlquiler fachada = new FacadeAlquiler();
    fachada.realizarAlquiler(txtSidcliente, tbresumenventa);

    ComprobanteBuilder builder = new ComprobanteBuilder()
        .setCliente(txtSnombrecliente.getText())
        .setId(Integer.parseInt(txtSidcliente.getText()));

    double total = 0;
    for (int i = 0; i < tbresumenventa.getRowCount(); i++) {
        builder.addProducto(tbresumenventa.getValueAt(i, 1).toString());
        total += Double.parseDouble(tbresumenventa.getValueAt(i, 4).toString());
    }

    PatronesComportamiento.Strategy.ProcesadorDescuento procesador = new PatronesComportamiento.Strategy().new ProcesadorDescuento();
    String seleccion = cmbDescuento.getSelectedItem().toString();
    switch (seleccion) {
        case "Fijo":
            procesador.setEstrategia(new PatronesComportamiento.Strategy().new DescuentoFijo());
            break;
        case "Porcentaje":
            procesador.setEstrategia(new PatronesComportamiento.Strategy().new DescuentoPorcentaje());
            break;
        default:
            procesador.setEstrategia(t -> t);
    }

    double totalConDescuento = procesador.calcularTotalConDescuento(total);
    builder.setTotal(totalConDescuento);

    JOptionPane.showMessageDialog(null, builder.build().generarTexto());

    FormularioVenta formulario = new FormularioVenta();
    formulario.setEstado(
        txtSidcliente.getText() + ";" + txtSnombrecliente.getText(),
        txtSidproducto.getText() + ";" + txtSnombreproducto.getText()
    );
    mementoFormulario = formulario.guardarEstado();

    ControlAlquiler objetoVenta = new ControlAlquiler();
    objetoVenta.limpiarLuegoVenta(txtbuscarcliente, tbclientes, txtbuscarproductos, tbproductos,
        txtSidcliente, txtSnombrecliente, txtSappaterno, txtSapmaterno,
        txtSidproducto, txtSnombreproducto, txtSprecioproducto, txtSstockproducto,
        txtSprecioventa, txtcantidadventa, tbresumenventa, lbliva, lbltotal);

    objetoVenta.MostrarUltimaFactura(lblUltimaFactura);
}GEN-LAST:event_btncobrarActionPerformed
```


PATRONES DE CREACIONALES

1. **SINGLETON** - Clase ConexionSingleton

Propósito:

Asegura que una clase tenga una única instancia y proporciona un punto de acceso global a ella, ideal para la conexión a bases de datos.

Código:

```

package PatronesCreacionales;

//PATRON SINGLETON
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.swing.JOptionPane;

/**
 *
 * @author Michel Mender
 */
public class ConexionSingleton {

    private static ConexionSingleton instancia;
    private Connection conexion;

    private final String usuario = "root";
    private final String contraseña = "25486271"; // o usa una variable segura
    private final String bd = "alquileres";
    private final String ip = "localhost";
    private final String puerto = "3306";
    private final String cadena = "jdbc:mysql://" + ip + ":" + puerto + "/" + bd;

    // Constructor privado: solo se crea una vez
    private ConexionSingleton() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conexion = DriverManager.getConnection(cadena, usuario, contraseña);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error en conexión: " + e.toString());
        }
    }

    // Metodo estático para obtener la única instancia
    public static ConexionSingleton getInstancia() {
        if (instancia == null) {
            instancia = new ConexionSingleton();
        }
        return instancia;
    }

    public Connection getConexion() {
        try {
            if (conexion == null || conexion.isClosed()) {
                conexion = DriverManager.getConnection(cadena, usuario, contraseña);
            }
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Error al reconectar: " + e.toString());
        }
        return conexion;
    }

    // Metodo para cerrar la conexión (opcional)
    public void cerrarConexion() {
        try {
            if (conexion != null && !conexion.isClosed()) {
                conexion.close();
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error al cerrar conexión: " + e.toString());
        }
    }
}

```

- Clase SesionUsuarioSingleton

Propósito:

Controla que solo exista una instancia activa del usuario autenticado durante la sesión.

La usamos para manejar la sesión del usuario logueado en todo el sistema (por ejemplo, mostrar su nombre o rol en diferentes vistas).


```

5 package PatronesCreacionales;
6 //aplicacion del patron singleton
7 /**
8  *
9  * @author Michel Mendez
10 */
11 public class SesionUsuarioSingleton {
12     private static SesionUsuarioSingleton instancia;
13     private String nombreUsuario;
14     private String rango;
15     private SesionUsuarioSingleton() {}
16
17     public static SesionUsuarioSingleton getInstance() {
18         if (instancia == null) {
19             instancia = new SesionUsuarioSingleton();
20         }
21         return instancia;
22     }
23     public void setUsuario(String nombreUsuario, String rango) {
24         this.nombreUsuario = nombreUsuario;
25         this.rango = rango;
26     }
27     public String getNombreUsuario() {
28         return nombreUsuario;
29     }
30     public String getRango() {
31         return rango;
32     }
33 }

```

EJECUCION:



Esta clase mantiene una única instancia de conexión a la base de datos.

Usas getInstance() en otras clases para obtener la conexión y evitar múltiples conexiones abiertas.

Es útil en sistemas que trabajan con MySQL y JDBC, tal y como nosotros lo estamos utilizando.

2. **BUILDER** - Clase ComprobanteBuilder

Propósito:

Permite construir objetos complejos paso a paso, separando la construcción del objeto de su representación final.

Código:

```
ComprobanteBuilder.java
Source History
import java.util.List;

/**
 * @author Michel Mendez
 */
public class ComprobanteBuilder {
    private int id;
    private String cliente;
    private List<String> productos = new ArrayList<>();
    private double total;

    public ComprobanteBuilder setId(int id) {
        this.id = id;
        return this;
    }

    public ComprobanteBuilder setCliente(String cliente) {
        this.cliente = cliente;
        return this;
    }

    public ComprobanteBuilder addProducto(String producto) {
        productos.add(producto);
        return this;
    }

    public ComprobanteBuilder setTotal(double total) {
        this.total = total;
        return this;
    }

    public Comprobante build() {
        return new Comprobante(id, cliente, productos, total);
    }
}
```

Se utiliza para crear instancias de la clase Comprobante de manera flexible.

Código:

```
ComprobanteBuilder.java Comprobante.java
Source History
/**
 * Click https://nhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click https://nhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package modelo;

import java.util.List;

/**
 * @author Michel Mendez
 */
public class Comprobante {
    private int id;
    private String cliente;
    private List<String> productos;
    private double total;

    public Comprobante(int id, String cliente, List<String> productos, double total) {
        this.id = id;
        this.cliente = cliente;
        this.productos = productos;
        this.total = total;
    }

    public String generarTexto() {
        return "Comprobante ID: " + id + "\nCliente: " + cliente + "\nProductos: " + productos + "\nTotal: $/ " + total;
    }
}
```

Puedes agregar productos uno por uno y configurar el objeto de forma fluida usando una interfaz encadenada (builder pattern).

Esto mejora la legibilidad y evita constructores largos con muchos parámetros.

3. FACTORY

En nuestra solución, se crea la clase iControler para poder crear la segunda clase llamado ControaldorFactory

```
1 // src/controlador/IControlador.java
2 package controlador;
3
4 /**
5  * Interfaz marca para todos los controladores.
6  * Aquí podrías añadir métodos comunes si los hay.
7  */
8 public interface IControlador {
9     // por ejemplo:
10    // void inicializar();
11 }
12
```

El cual nos ayudara para para gestionar los demás controladores y es por tal razón de tendremos que implementar la clase IControlador en los demás controladores

```
History
import controlador.ControladorReportes;

public class ControladorFactory {
    private final Map<String, Supplier<IControlador>> proveedores;

    public ControladorFactory() {
        proveedores = Map.of(
            "producto", ControladorProducto::new,
            "cliente", ControladorCliente::new,
            "alquiler", ControlAlquiler::new,
            "reportes", ControladorReportes::new
        );
    }
}
```

Aquí se puede observar la implementación de IControlador en ControladorProducto

```
public class ControladorProducto implements IControlador {


    public void MostrarProductos(JTable tablaTotalProductos) {
        DefaultTableModel modelo = new DefaultTableModel();
        modelo.addColumn("ID");
        modelo.addColumn("Nombre");
        modelo.addColumn("Precio");
        modelo.addColumn("Stock");
        modelo.addColumn("Categoría");
    }
}
```

Y como podremos observar ya podremos implementar el objeto controlador producto haciendo uso del factory en este caso se aplica en la clase facade

```
public class FacadeAlquiler {
    private final ControlAlquiler control;

    public FacadeAlquiler() {
        // en lugar de new ControlAlquiler():
        IControlador ctrl = new ControladorFactory().crear("alquiler");
        // casteamos al tipo concreto para seguir usando sus métodos
        this.control = (ControlAlquiler) ctrl;
    }
}
```

EJECUCION



The screenshot shows the 'ALQUILERES' application interface. On the left is a dark blue sidebar with buttons for 'RENTA', 'BUSCAR COMPROBANTE', 'REPORTE POR FECHAS', 'ALMACEN', 'CLIENTES', and 'CERRAR SESION'. The main area has a dark blue header with the 'ALQUILERES' logo. Below the header is a 'Datos Producto' form with input fields for 'id', 'Nombre', 'Precio', 'Stock', and a 'Sillas' dropdown menu. Below the form is a table with 5 columns: ID, Nombre, Precio, Stock, and Categoría. The table contains 15 rows of product data. To the right of the table are four buttons: 'Limpiar Campo', 'Guardar', 'Modificar', and 'Eliminar'.

ID	Nombre	Precio	Stock	Categoría
1	SILLAS METALICAS	5.0	950	Sin categoria
2	SILLAS PLASTICAS	5.0	0	Sin categoria
3	SILLAS MADERA	15.0	1000	Sin categoria
4	SILLAS TIFFANY	20.0	1000	Sin categoria
5	SILLAS ACOJINADAS	15.0	1000	Sin categoria
6	SILLAS INFANTILES	5.0	1000	Sin categoria
7	MESAS RECTANGUL...	70.0	100	Sin categoria
8	MESAS CIRCULARES	70.0	100	Sin categoria
9	MESAS CUADRADAS	70.0	100	Sin categoria
10	MESAS INFANTILES	70.0	0	Sin categoria
11	MANTELES MULTI C...	40.0	150	Sin categoria
12	CUBRE MANTELES	30.0	100	Sin categoria
13	PLATOS EXT/ TP GO...	5.0	200	Sin categoria
14	PLATOS / TP GSOPA	5.0	200	Sin categoria
15	PLATOS PI PASTEL	5.0	200	Sin categoria

PATRONES DE ESTRUCTURALES

1. PROXY

Propósito: Controlar el acceso a los formularios según el rango del usuario (Admin, Empleado).

Clases:

- AccesoFormularioProxy (interfaz): Define el método acceder.
- AccesoProxy: Implementa la lógica de verificación de permisos antes de permitir el acceso.
- FormularioRealProxy: Ejecuta el acceso real al formulario cuando está autorizado.

Código:

```
Proxy.java
Source History
5 package PatronesEstructurales;
6
7 import PatronesCreacionales.SesionUsuarioSingleton;
8 import formulario.MenuPrincipal;
9 import java.util.Arrays;
10 import java.util.List;
11 import javax.swing.JOptionPane;
12 import javax.swing.JPanel;
13 /**
14  *
15  * @author Michel Mendez
16  */
17 public class Proxy {
18
19     public interface AccesoFormularioProxy {
20
21         void acceder(JPanel panel, MenuPrincipal menu);
22     }
23
24     public class AccesoProxy implements AccesoFormularioProxy {
25         private FormularioRealProxy real = new FormularioRealProxy();
26         private String boton;
27
28         public AccesoProxy(String boton) {
29             this.boton = boton;
30         }
31
32         @Override
33         public void acceder(JPanel panel, MenuPrincipal menu) {
34             String rango = SesionUsuarioSingleton.getInstance().getRango();
35             List<String> permitidosEmpleado = Arrays.asList("rentas", "busqueda de comprobantes", "busqueda de reportes");
36
37             if (rango.equals("Admin") && permitidosEmpleado.contains(boton)) {
38                 JOptionPane.showMessageDialog(null, "Acceso denegado" + boton + "\nRango actual: " + rango);
39             } else {
40                 real.acceder(panel, menu);
41             }
42         }
43     }
44
45     public class FormularioRealProxy implements AccesoFormularioProxy {
46
47         @Override
48         public void acceder(JPanel panel, MenuPrincipal menu) {
49             JOptionPane.showMessageDialog(null, "Acceso autorizado al formulario.");
50             menu.mostrarPanel(panel);
51         }
52     }
53 }
```

Este nos ayuda a proteger el acceso a ciertos formularios del sistema dependiendo del rol actual del

usuario, accedido a través del singleton SesionUsuarioSingleton.

2. FLYWEIGHT

Propósito: Evitar la creación repetida de objetos de categoría al compartir instancias comunes (ahorra memoria).

Clases:

- Categoria: Representa una categoría de producto.
- CategoriaFactory: Administra y reutiliza instancias de Categoria para evitar duplicados.

Código:

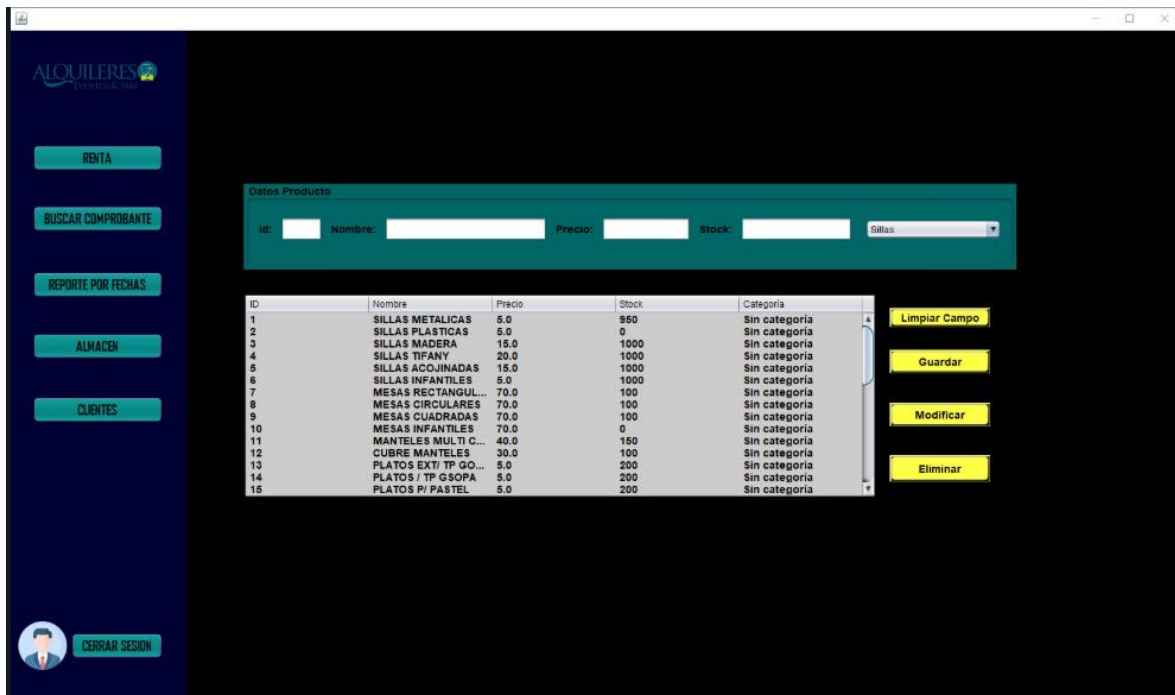
```
Source History
import java.util.HashMap;
import javax.swing.JOptionPane;

11  /**
12   *
13   * @author Michel Mendez
14   */
15  public class Flyweight {
16      public static class Categoria {
17          private String nombre;
18
19          public Categoria(String nombre) {
20              this.nombre = nombre;
21          }
22          public String getNombre() {
23              return nombre;
24          }
25      }
26
27      public static class CategoriaFactory {
28          private static final HashMap<String, Categoria> categorias = new HashMap<>();
29
30          public static Categoria getCategoria(String nombre) {
31              Categoria categoria = categorias.get(key: nombre);
32              if (categoria == null) {
33                  categoria = new Categoria(nombre);
34                  categorias.put(key: nombre, value: categoria);
35              }
36              return categoria;
37          }
38      }
39  }
```

Cuando se crea o se trabaja con categorías, se utiliza CategoriaFactory.getCategoria(...) para evitar crear nuevos objetos repetidos innecesariamente, reduciendo el uso de memoria y optimizando el rendimiento.

EJECUCION:

En productos adaptamos una parte para seleccionar la categoría



ALQUILERES
Sistema de Alquileres

RENTA
BUSCAR COMPROBANTE
REPORTE POR FECHAS
ALMACEN
CLIENTES
CERRAR SESION

Datos Producto

ID: Nombre: Precio: Stock: Categorias:

ID	Nombre	Precio	Stock	Categoria
1	SILLAS METALICAS	5.0	950	Sin categoria
2	SILLAS PLASTICAS	5.0	0	Sin categoria
3	SILLAS MADERA	15.0	1000	Sin categoria
4	SILLAS TIFANY	20.0	1000	Sin categoria
5	SILLAS ACOJINADAS	15.0	1000	Sin categoria
6	SILLAS INFANTILES	5.0	1000	Sin categoria
7	MESAS RECTANGUL...	70.0	100	Sin categoria
8	MESAS CIRCULARES	70.0	100	Sin categoria
9	MESAS CUADRADAS	70.0	100	Sin categoria
10	MESAS INFANTILES	70.0	0	Sin categoria
11	MANTELES MULTI C...	40.0	150	Sin categoria
12	CUBRE MANTELES	30.0	100	Sin categoria
13	PLATOS EXTI TP GO...	5.0	200	Sin categoria
14	PLATOS / TP GSOPA	5.0	200	Sin categoria
15	PLATOS P/ PASTEL	5.0	200	Sin categoria

Limpiar Campo
Guardar
Modificar
Eliminar

3. ADAPTER

Clase:

- Adapter

Se utiliza para generar un PDF de forma estandarizada a partir de los datos del formulario.

Código:

```

1 package PatronesEstructurales;
2
3 import com.itextpdf.text.Document;
4 import com.itextpdf.text.Paragraph;
5 import com.itextpdf.text.pdf.PdfPTable;
6 import com.itextpdf.text.pdf.PdfWriter;
7
8 import javax.swing.JLabel;
9 import javax.swing.JTable;
10 import java.io.FileOutputStream;
11 import java.io.IOException;
12 import java.text.SimpleDateFormat;
13 import java.util.Date;
14
15 /**
16  * Adaptador para generar comprobantes de factura en PDF.
17  */
18 public class Adapter {
19
20     public void exportarPDFCompleto(JLabel lblFactura, JLabel lblFechaFactura, JLabel lblNombreCliente,
21                                     JLabel lblAppaterno, JLabel lblApmaterno,
22                                     JTable tablaProductos, JLabel lblIVA, JLabel lblTotal) {
23
24         Document documento = new Document();

```



```

25     try {
26         // Nombre del archivo con fecha y número de factura para evitar sobrescritura
27         String nombreArchivo = "reporte_comprobante_" + lblFactura.getText().replaceAll(regex: "\\s+", replacement: "_")
28             + "_" + new SimpleDateFormat(string: "yyyyMMdd_HHmmss").format(new Date()) + ".pdf";
29
30         PdfWriter.getInstance( document: documento, new FileOutputStream(string: nombreArchivo));
31         documento.open();
32
33         // Titulo
34         documento.add(new Paragraph(string: "====="));
35         documento.add(new Paragraph(string: "COMPROBANTE DE ALQUILER"));
36         documento.add(new Paragraph(string: "=====\\n"));
37
38         // Datos del cliente
39         documento.add(new Paragraph("Factura N°: " + lblFactura.getText()));
40         documento.add(new Paragraph("Fecha de Venta: " + lblFechaFactura.getText()));
41         documento.add(new Paragraph("Cliente: " + lblNombreCliente.getText() + " " +
42             lblApellidoPaterno.getText() + " " + lblApellidoMaterno.getText()));
43         documento.add(new Paragraph(string: " "));
44
45         // Tabla de productos
46         int columnas = tablaProductos.getColumnCount();
47         if (columnas < 4) {
48             throw new IllegalArgumentException(string: "La tabla debe tener al menos 4 columnas: Producto, Cantidad, Precio
49         }

```

```

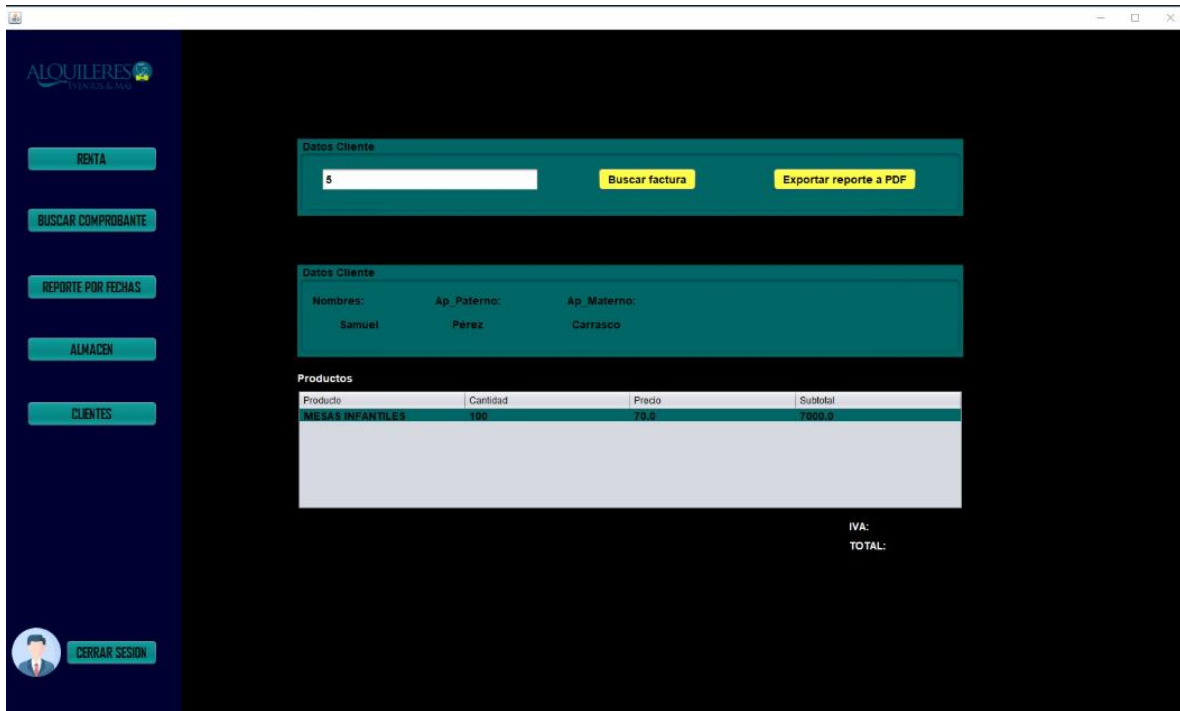
51         PdfPTable tabla = new PdfPTable(numColumnas: 4);
52         tabla.addCell(text: "Producto");
53         tabla.addCell(text: "Cantidad");
54         tabla.addCell(text: "Precio Venta");
55         tabla.addCell(text: "Subtotal");
56
57         for (int i = 0; i < tablaProductos.getRowCount(); i++) {
58             for (int j = 0; j < 4; j++) {
59                 Object valor = tablaProductos.getValueAt(row: i, column: j);
60                 tabla.addCell(valor != null ? valor.toString() : "");
61             }
62         }
63
64         documento.add(element: tabla);
65
66         // Totales
67         documento.add(new Paragraph(string: " "));
68         documento.add(new Paragraph("IVA: " + lblIVA.getText()));
69         documento.add(new Paragraph("Total a Pagar: $/ " + lblTotal.getText()));
70
71         documento.close();
72         System.out.println("PDF generado exitosamente: " + nombreArchivo);
73
74     } catch (Exception e) {
75         System.err.println(s: "Error al generar el PDF:");
76         e.printStackTrace();
77     } finally {
78         if (documento.isOpen()) {
79             documento.close();
80         }

```

El patrón Adapter permite que el formulario no tenga que interactuar directamente con la lógica compleja de generación de PDF. El ExportarPDFMediator coordina todo, y el Adapter se encarga de traducir la solicitud a un formato que la clase generadora de PDF pueda entender. Así se desacopla la interfaz del formulario del backend de generación de PDF.

EJECUCION.

Se exporta un pdf, con el boton Exportar reporte a PDF



The screenshot shows a web application for managing rentals. On the left is a dark blue sidebar with buttons: RENTA, BUSCAR COMPROBANTE, REPORTE POR FECHAS, ALMACEN, CLIENTES, and CERRAR SESION. The main area has a teal header with 'ALQUILERES' and a search bar containing '5'. Below the search bar are two buttons: 'Buscar factura' and 'Exportar reporte a PDF'. A section titled 'Datos Cliente' shows the following information:

Nombres:	Ap. Paterno:	Ap. Materno:
Samuel	Pérez	Carrasco

Below this is a table titled 'Productos' with the following data:

Producto	Cantidad	Precio	Subtotal
MESAS INFANTILES	100	15.0	1500.0

At the bottom right, it shows 'IVA: 270' and 'TOTAL: 1500'.

```
=====
COMPROBANTE DE ALQUILER
=====
Factura N°: 6
Fecha de Venta: 2025-05-22
Cliente: Samuel Pérez Carrasco



| Producto      | Cantidad | Precio Venta | Subtotal |
|---------------|----------|--------------|----------|
| SILLAS MADERA | 100      | 15.0         | 1500.0   |



IVA: 270
Total a Pagar: 1500
```

4. **FACADE**

Clase:

- FacadeAlquiler

Simplifica el proceso de alquiler de productos encapsulando varias llamadas al controlador ControlAlquiler.

Código:

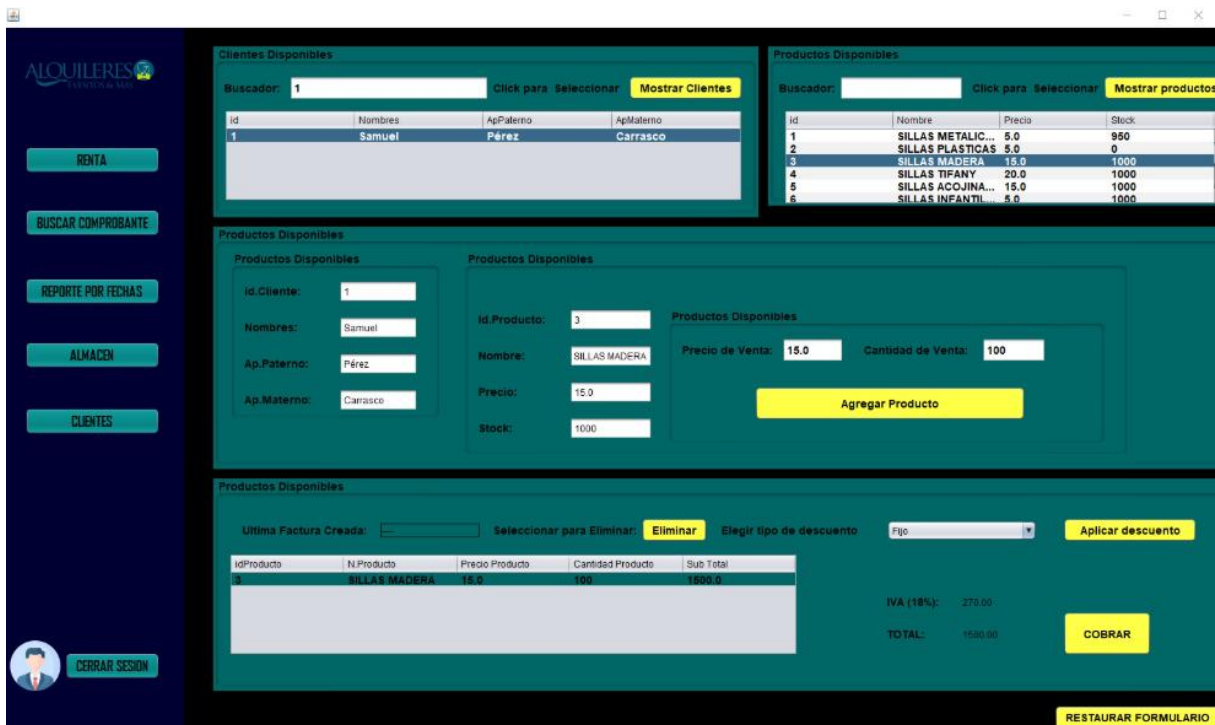
```

11
12 /**
13  *
14  * @author Michel Mendez
15  */
16 public class FacadeAlquiler {
17     private ControlAlquiler control;
18
19     public FacadeAlquiler() {
20         this.control = new ControlAlquiler();
21     }
22
23     /**
24      * Realiza el proceso completo de crear factura y registrar la venta.
25      *
26      * @param cliente Campo de texto con el ID del cliente
27      * @param productos Tabla resumen de productos seleccionados
28      */
29     public void realizarAlquiler(JTextField cliente, JTable productos) {
30         if (cliente.getText().isEmpty() || productos.getRowCount() == 0) {
31             JOptionPane.showMessageDialog(parentComponent, null, message: "Debe seleccionar un cliente y agregar productos antes de return;
32         }
33
34         control.crearFactura(ordenCliente: cliente);
35         control.realizarVenta(ventas: productos);
36
37         JOptionPane.showMessageDialog(parentComponent, null, message: "¡Alquiler completado exitosamente!");
38     }
39 }
40

```

El patrón Facade oculta la complejidad de múltiples operaciones (crear factura y realizar venta) bajo un solo método realizarAlquiler. Esto hace que el formulario cliente llame a un solo método en lugar de lidiar con múltiples clases directamente. Mejora la legibilidad y el mantenimiento del código.

EJECUCION:



The screenshot displays the ALQUILERES VENTURA S de RL application interface. It features a sidebar with navigation buttons: RENTA, BUSCAR COMPROBANTE, REPORTE POR FECHAS, ALMACEN, CLIENTES, and CERRAR SESION. The main content area is divided into several sections:

- Cientes Disponibles:** A table showing client information. The first row is highlighted:

id	Nombres	ApPaterno	ApMaterno
1	Samuel	Pérez	Carrasco
- Productos Disponibles:** A table showing product information. The first row is highlighted:

id	Nombre	Precio	Stock
1	SILLAS METALIC...	5.0	950
2	SILLAS PLASTICAS	5.0	0
3	SILLAS MADERA	15.0	1000
4	SILLAS TIFANY	20.0	1000
5	SILLAS ACOJINA...	15.0	1000
6	SILLAS INFANTIL	5.0	1000
- Formulario de Alquiler:** A section for adding a product to the rental. It includes fields for Id.Cliete (1), Nombres (Samuel), Ap.Paterno (Pérez), Ap.Materno (Carrasco), Id.Producto (3), Nombre (SILLAS MADERA), Precio (15.0), Stock (1000), Precio de Venta (15.0), and Cantidad de Venta (100). A yellow button labeled "Agregar Producto" is present.
- Factura y Resumen:** A section showing the last created invoice and a table of products. The table has columns: idProducto, N Producto, Precio Producto, Cantidad Producto, and Sub Total. The first row is highlighted:

idProducto	N Producto	Precio Producto	Cantidad Producto	Sub Total
3	SILLAS MADERA	15.0	100	1500.0

 Below the table, there is a summary: IVA (18%): 270.00, TOTAL: 1500.00, and a yellow button labeled "COBRAR".



5. COMPOSITE

En nuestra solución, se la clase MacroExport que es el patron Composite y que con ayuda al implementar el patron Mediator nos ayuda a Exportar en PDF

```
/**
 * Composite de IMediator: delega exportarPDF() a todos sus hijos.
 */
public class MacroExportPDF implements IMediator {
    private final List<IMediator> mediadores = new ArrayList<>();

    /** Agrega un exportador al composite */
    public void add(IMediator m) {
        mediadores.add(m);
    }
}
```

En este caso con la ayuda del patrón Composite La clase iMediator podrá delegar Exportar PDF a todos sus hijos

```

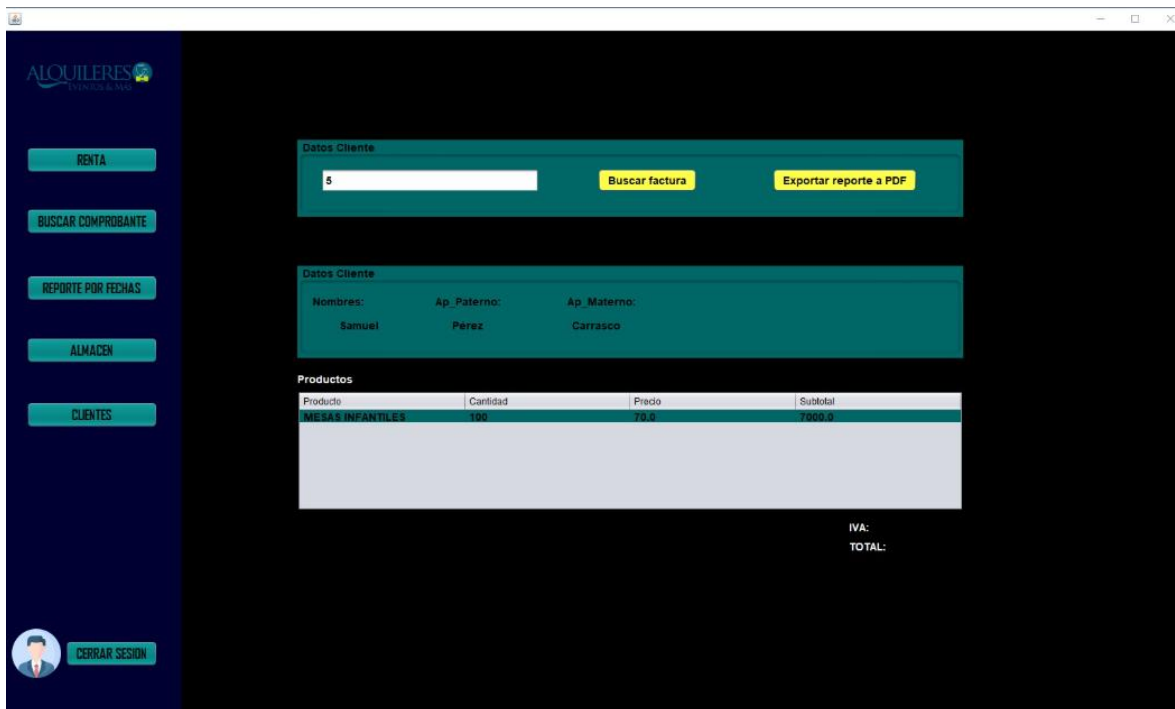
public void add(IMediator m) {
    mediadores.add(m);
}

/** Quita un exportador del composite */
public void remove(IMediator m) {
    mediadores.remove(m);
}

@Override
public void exportarPDF(JLabel lblFactura,
                        JLabel lblFechaFactura,
                        JLabel lblNombreCliente,
                        JLabel lblAppaterno,
                        JLabel lblApmaterno,
                        JTable tablaProductos,
                        JLabel lblIVA,
                        JLabel lblTotal) {
    for (IMediator m : mediadores) {
        m.exportarPDF(
            lblFactura, lblFechaFactura,
            lblNombreCliente, lblAppaterno, lblApmaterno,
            tablaProductos, lblIVA, lblTotal
        );
    }
}

```

EJECUCION:



ALQUILERES
SISTEMA DE ALQUILERES

RENTA
BUSCAR COMPROBANTE
REPORTE POR FECHAS
ALMACEN
CLIENTES

Datos Cliente

5 **Buscar factura** **Exportar reporte a PDF**

Datos Cliente

Nombres:	Ap. Paterno:	Ap. Materno:
Samuel	Perez	Carrasco

Productos

Producto	Cantidad	Precio	Subtotal
MESA INFANTILES	100	70.0	7000.0

IVA:
TOTAL:

CERRAR SESION

CONCLUSION

La aplicación de patrones de diseño creacionales, estructurales y conductuales en este sistema de ventas produce una arquitectura robusta y adaptable.

Los patrones de creación como Builder y Singleton agilizan la creación de objetos y garantizan la creación controlada de componentes clave.

Los patrones estructurales como Adaptador, Fachada y Proxy facilitan una integración perfecta, simplifican interacciones complejas y refuerzan los protocolos de seguridad.

Los patrones de comportamiento que incluyen Comando, Iterador, Recuerdo, Observador, Cadena de Responsabilidad y Estrategia mejoran el comportamiento dinámico del sistema, promueven un acoplamiento flexible entre los componentes y brindan flexibilidad en la implementación de varios algoritmos y procesos.

Al aprovechar estos patrones, el sistema logra una mejor modularidad, extensibilidad y capacidad de mantenimiento, lo que lo hace bien equipado para manejar requisitos cambiantes y crecimiento futuro.