UNIVERSIDAD
Blas Pascal

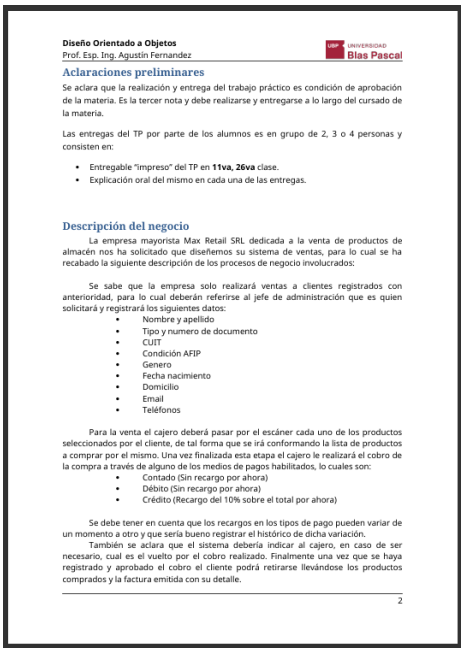
Diseño Orientado a Objetos: TP Integrador - 2025

- **Asignatura:** Diseño Orientado a Objetos.
- **Profesor:** Esp. Ing. Agustín Fernández.
- **Alumnos:** Gastón Molina, Tomas Molina, Facundo Gomez, y Edgar Karpowicz.
- **Tema:** Trabajo Práctico Integrador – 2025 – 2da Parte
- **Fecha:** 06/06/25

Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

Trabajo Práctico Integrador – 2025
Segunda Parte

- ## Consignas:



Diseño Orientado a Objetos
Prof. Esp. Ing. Agustín Fernandez

Aclaraciones preliminares

Se aclara que la realización y entrega del trabajo práctico es condición de aprobación de la materia. Es la tercer nota y debe realizarse y entregarse a lo largo del cursado de la materia.

Las entregas del TP por parte de los alumnos es en grupo de 2, 3 o 4 personas y consisten en:

- Entregable "impreso" del TP en **11va, 26va** clase.
- Explicación oral del mismo en cada una de las entregas.

Descripción del negocio

La empresa mayorista Max Retail SRL dedicada a la venta de productos de almacén nos ha solicitado que diseñemos su sistema de ventas, para lo cual se ha recabado la siguiente descripción de los procesos de negocio involucrados:

Se sabe que la empresa solo realizará ventas a clientes registrados con anterioridad, para lo cual deberán referirse al jefe de administración que es quien solicitará y registrará los siguientes datos:

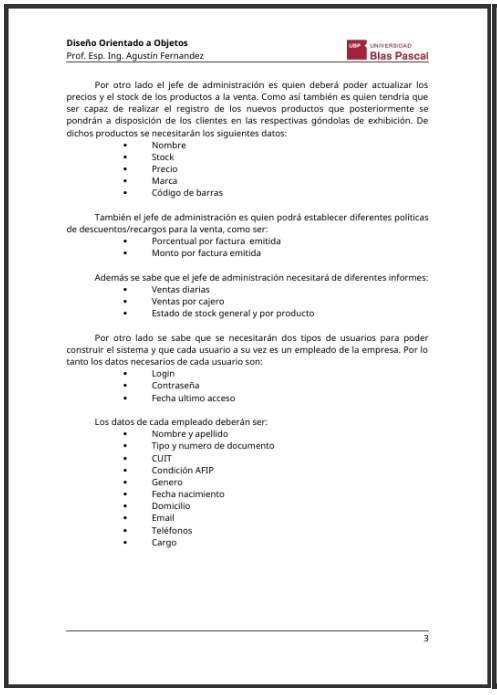
- Nombre y apellido
- Tipo y numero de documento
- CUIT
- Condición AFIP
- Genero
- Fecha nacimiento
- Domicilio
- Email
- Teléfonos

Para la venta el cajero deberá pasar por el escáner cada uno de los productos seleccionados por el cliente, de tal forma que se irá conformando la lista de productos a comprar por el mismo. Una vez finalizada esta etapa el cajero le realizará el cobro de la compra a través de alguno de los medios de pagos habilitados, lo cuales son:

- Contado (Sin recargo por ahora)
- Débito (Sin recargo por ahora)
- Crédito (Recargo del 10% sobre el total por ahora)

Se debe tener en cuenta que los recargos en los tipos de pago pueden variar de un momento a otro y que sería bueno registrar el histórico de dicha variación.
También se aclara que el sistema debería indicar al cajero, en caso de ser necesario, cual es el vuelto por el cobro realizado. Finalmente una vez que se haya registrado y aprobado el cobro el cliente podrá retirarse llevándose los productos comprados y la factura emitida con su detalle.

2



Diseño Orientado a Objetos
Prof. Esp. Ing. Agustín Fernandez

Por otro lado el jefe de administración es quien deberá poder actualizar los precios y el stock de los productos a la venta. Como así también es quien tendría que ser capaz de realizar el registro de los nuevos productos que posteriormente se pondrán a disposición de los clientes en las respectivas góndolas de exhibición. De dichos productos se necesitarán los siguientes datos:

- Nombre
- Stock
- Precio
- Marca
- Código de barras

También el jefe de administración es quien podrá establecer diferentes políticas de descuentos/recargos para la venta, como ser:

- Porcentual por factura emitida
- Monto por factura emitida

Además se sabe que el jefe de administración necesitará de diferentes informes:

- Ventas diarias
- Ventas por cajero
- Estado de stock general y por producto

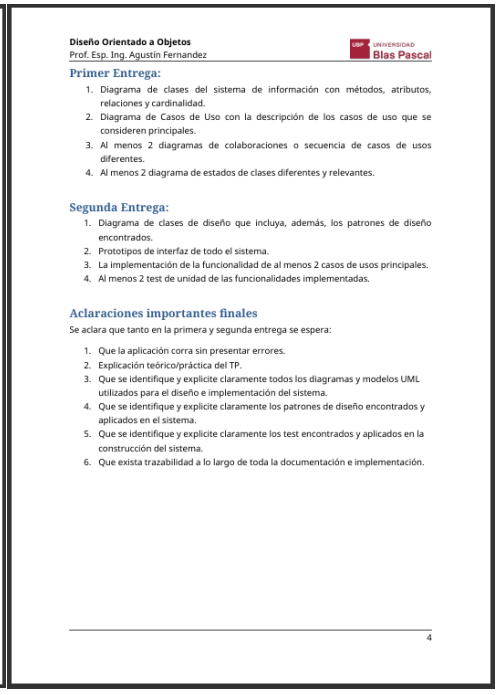
Por otro lado se sabe que se necesitarán dos tipos de usuarios para poder construir el sistema y que cada usuario a su vez es un empleado de la empresa. Por lo tanto los datos necesarios de cada usuario son:

- Login
- Contraseña
- Fecha ultimo acceso

Los datos de cada empleado deberán ser:

- Nombre y apellido
- Tipo y numero de documento
- CUIT
- Condición AFIP
- Genero
- Fecha nacimiento
- Domicilio
- Email
- Teléfonos
- Cargo

3



Diseño Orientado a Objetos
Prof. Esp. Ing. Agustín Fernandez

Primer Entrega:

1. Diagrama de clases del sistema de información con métodos, atributos, relaciones y cardinalidad.
2. Diagrama de Casos de Uso con la descripción de los casos de uso que se consideren principales.
3. Al menos 2 diagramas de colaboraciones o secuencia de casos de usos diferentes.
4. Al menos 2 diagrama de estados de clases diferentes y relevantes.

Segunda Entrega:

1. Diagrama de clases de diseño que incluya, además, los patrones de diseño encontrados.
2. Prototipos de interfaz de todo el sistema.
3. La implementación de la funcionalidad de al menos 2 casos de usos principales.
4. Al menos 2 test de unidad de las funcionalidades implementadas.

Aclaraciones importantes finales

Se aclara que tanto en la primera y segunda entrega se espera:

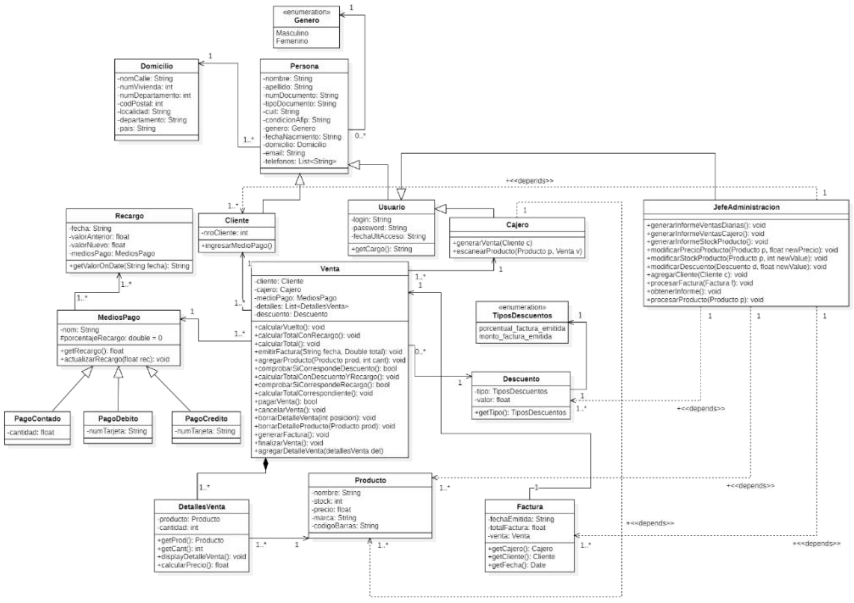
1. Que la aplicación corra sin presentar errores.
2. Explicación teórico/práctica del TP.
3. Que se identifique y explicite claramente todos los diagramas y modelos UML utilizados para el diseño e implementación del sistema.
4. Que se identifique y explicite claramente los patrones de diseño encontrados y aplicados en el sistema.
5. Que se identifique y explicite claramente los test encontrados y aplicados en la construcción del sistema.
6. Que exista trazabilidad a lo largo de toda la documentación e implementación.

4

Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

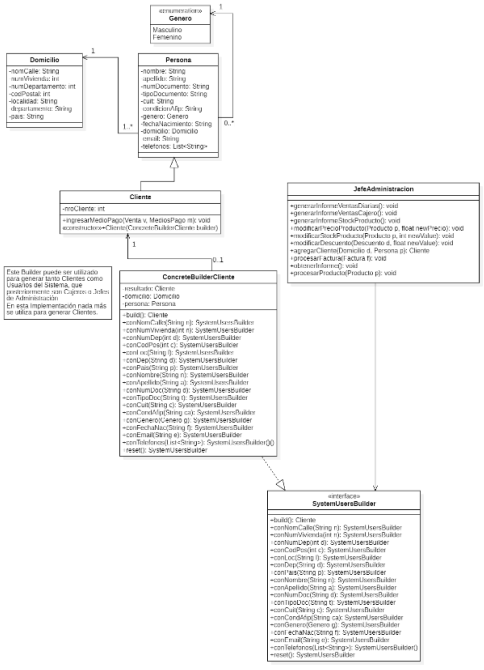
- # Diagramas de Clases de Diseño con Patrones:

  - ## Diagrama Original:



  - ## Patrón Builder - Cliente:

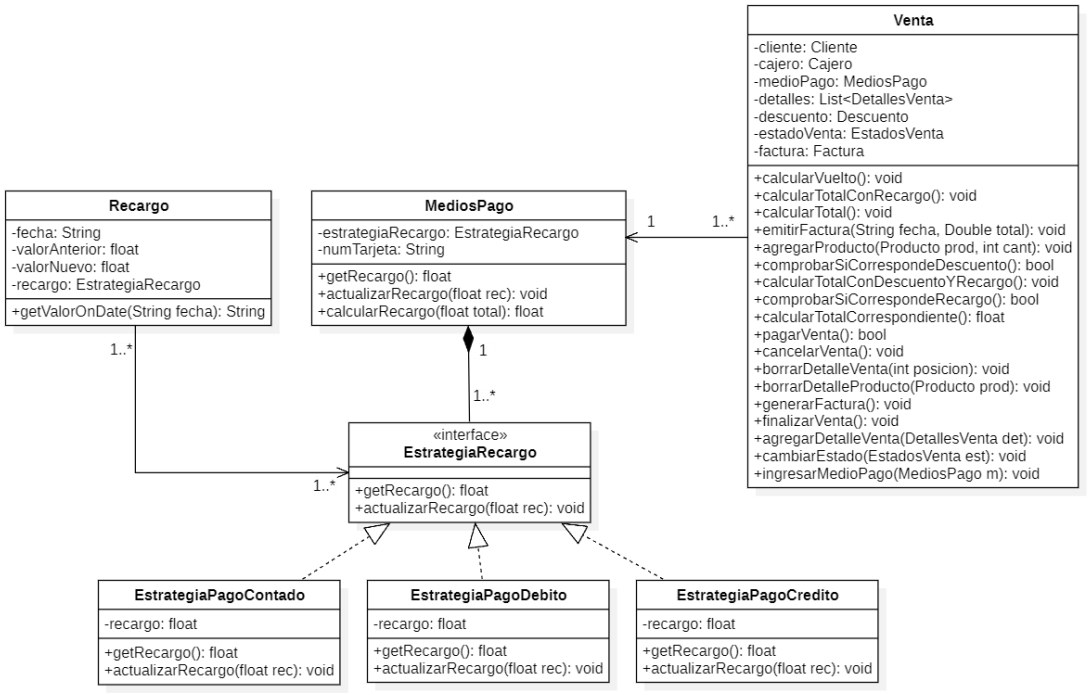
    Aplicado para la Construcción e Instanciación de Objetos Complejos de Tipo Cliente. El cual posee múltiples Objetos y Variables Asociadas debido a su Extensión de Persona.



Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

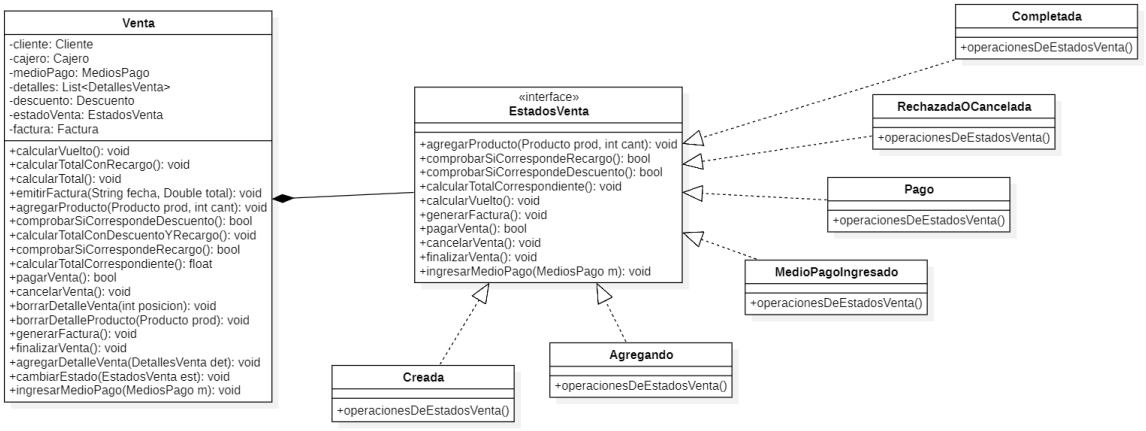
- ## Patrón Estrategia – Medios de Pago:

  Aplicado para los Tipos de Medios de Pago disponibles en la Aplicación y el Recargo que aplican. Debido a que cada uno de ellos implica un Algoritmo distinto en el cálculo del Total de la Venta.



- ## Patrón Estado – Venta:

  Aplicado a los posibles Estados de Ventas. Redefiniendo las diferentes Funciones de Venta según su Estado Interno, para asegurarse que el Objeto siga el Flujo adecuado y requerido de la Aplicación.

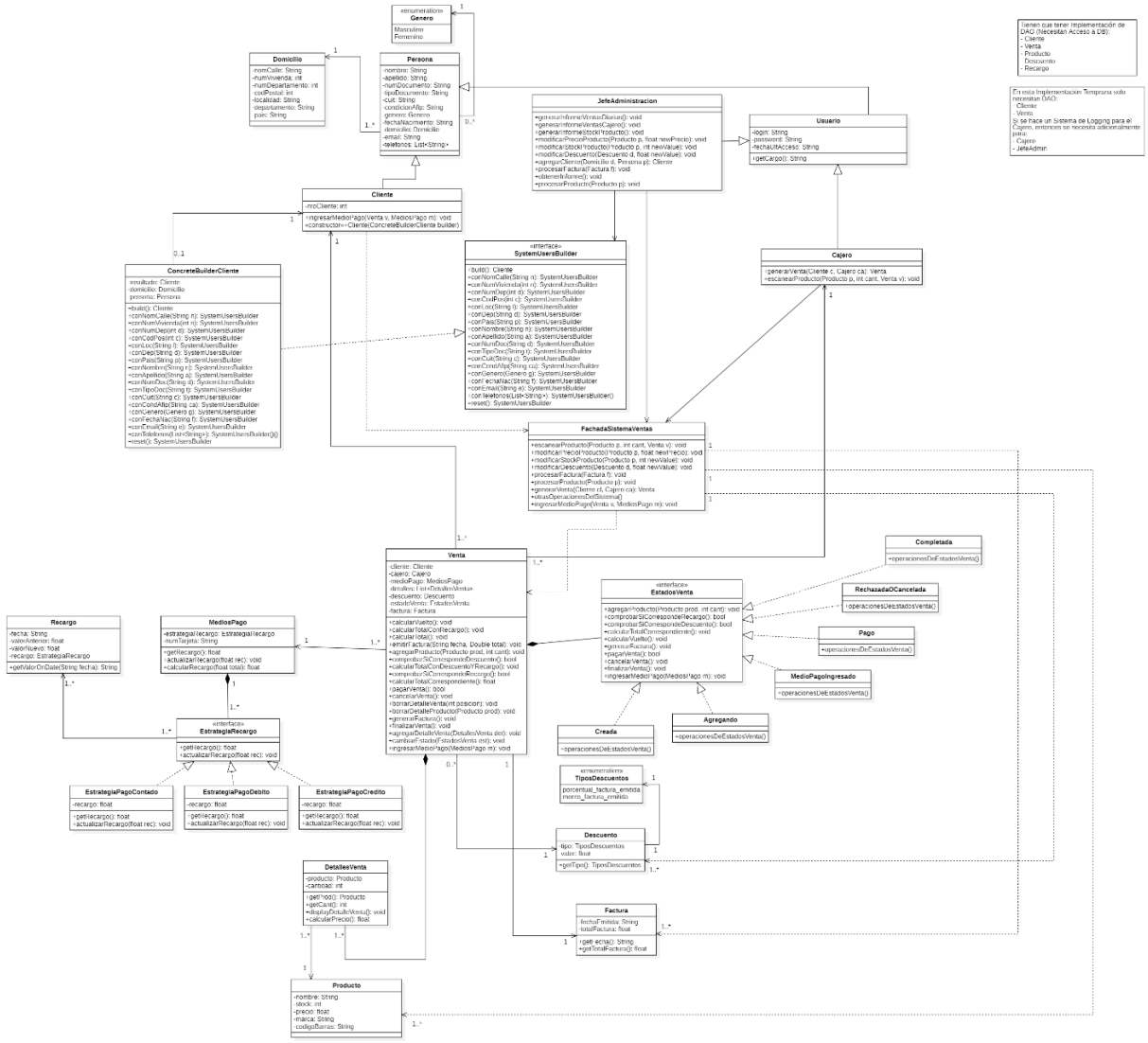


Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- # Patrón Fachada – Subsistema Ventas:

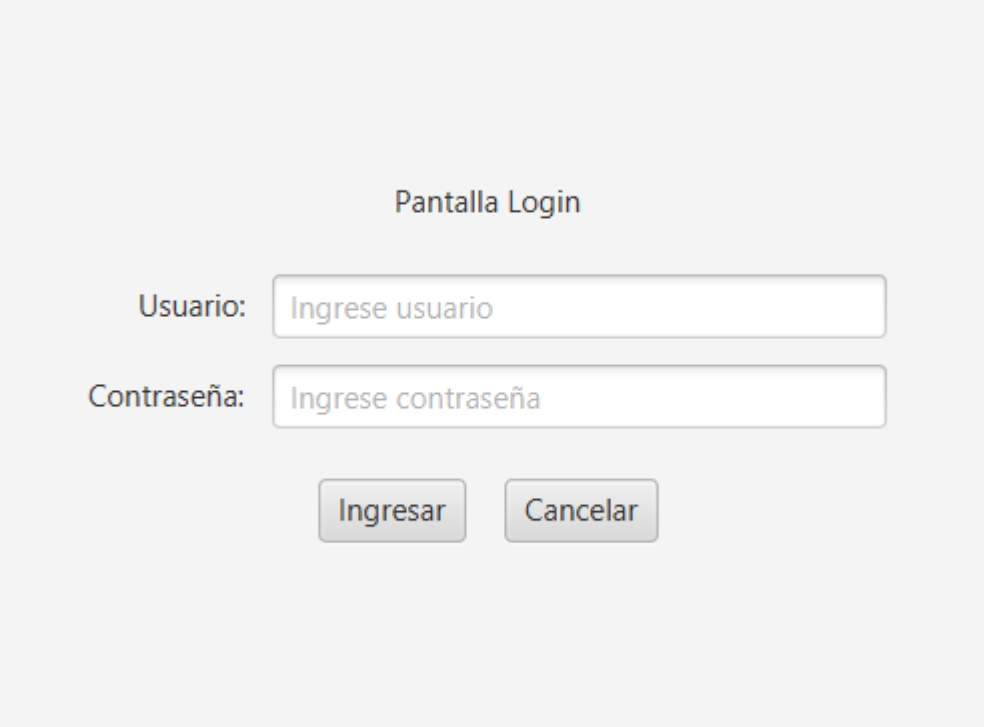
  Aplicado para separar el Subsistema de Ventas del resto de la Aplicación de una forma leve. Permitiendo que los Objetos no tengan que interactuar directamente con los Elementos dentro del Subsistema, simplificando el uso de este y del Sistema entero.

  Esta es una Representación del Subsistema Entero del lado de los Modelos, con los Patrones aplicados.

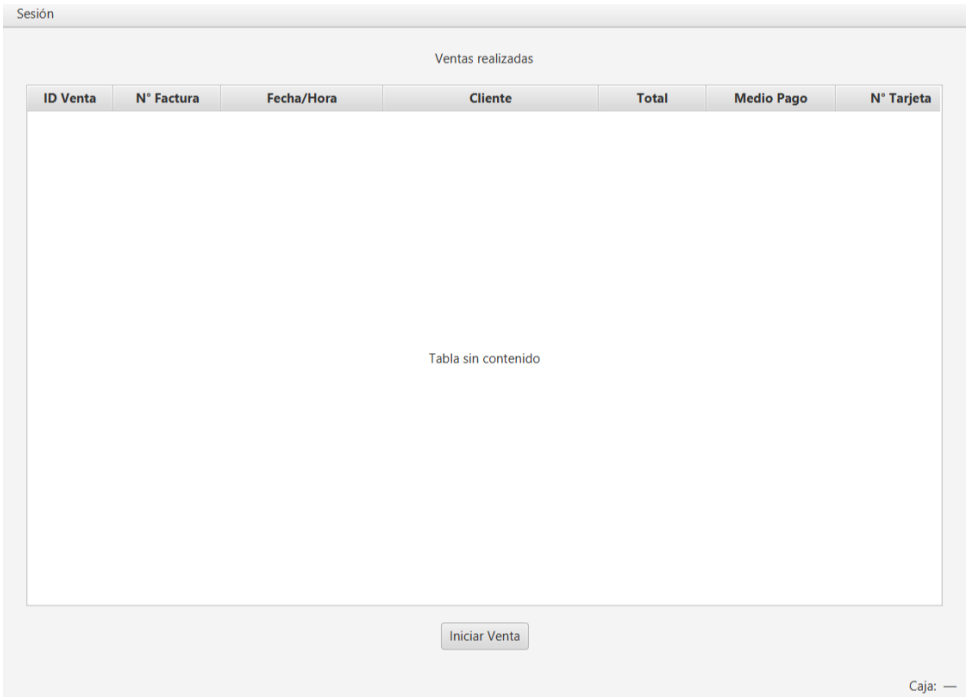


Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- # Prototipos de Interfaz del Sistema Entero:
  - **Login:** Inicio de la Aplicación

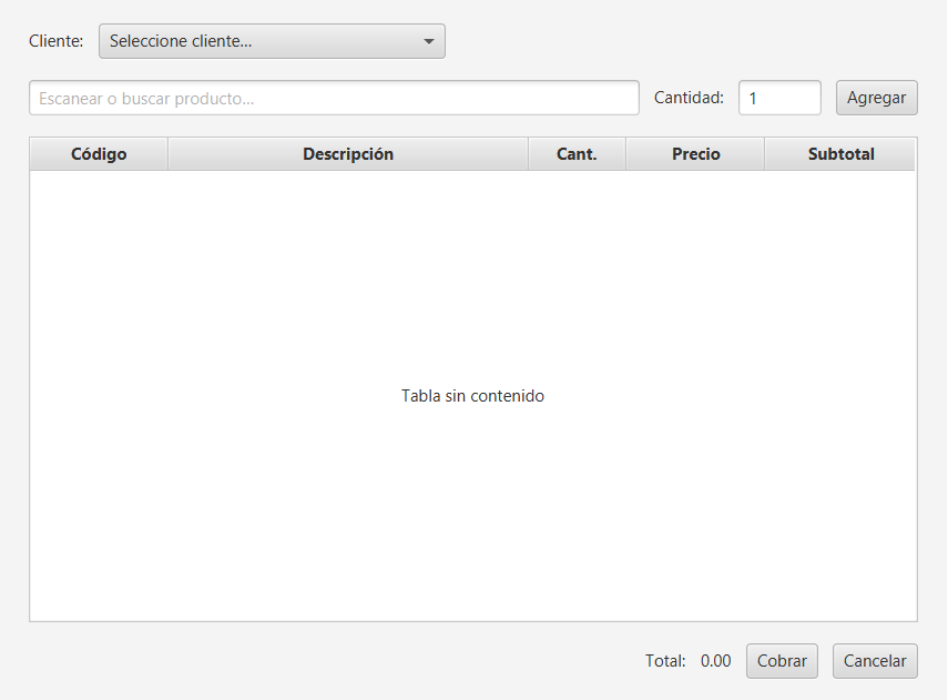


  - **CashierRoot.FXML:** 1 – Flujo Cajero

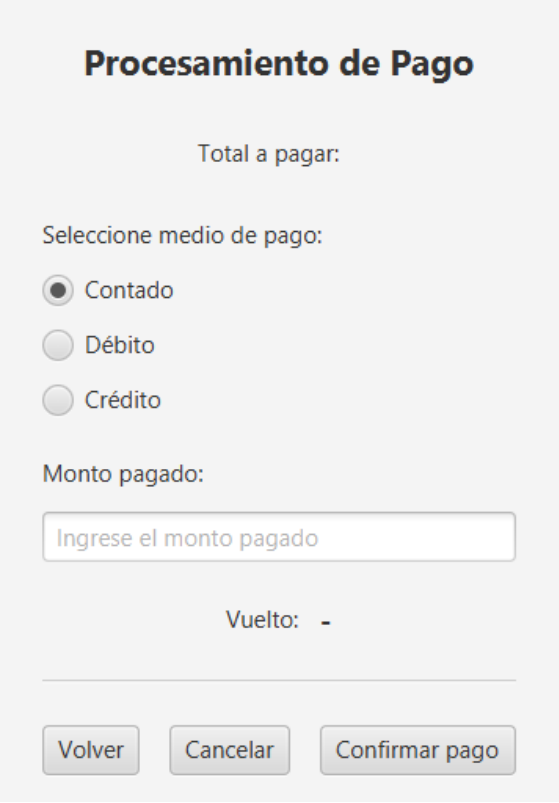


Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- **CashierSale.FXML:** 2 – Flujo Cajero



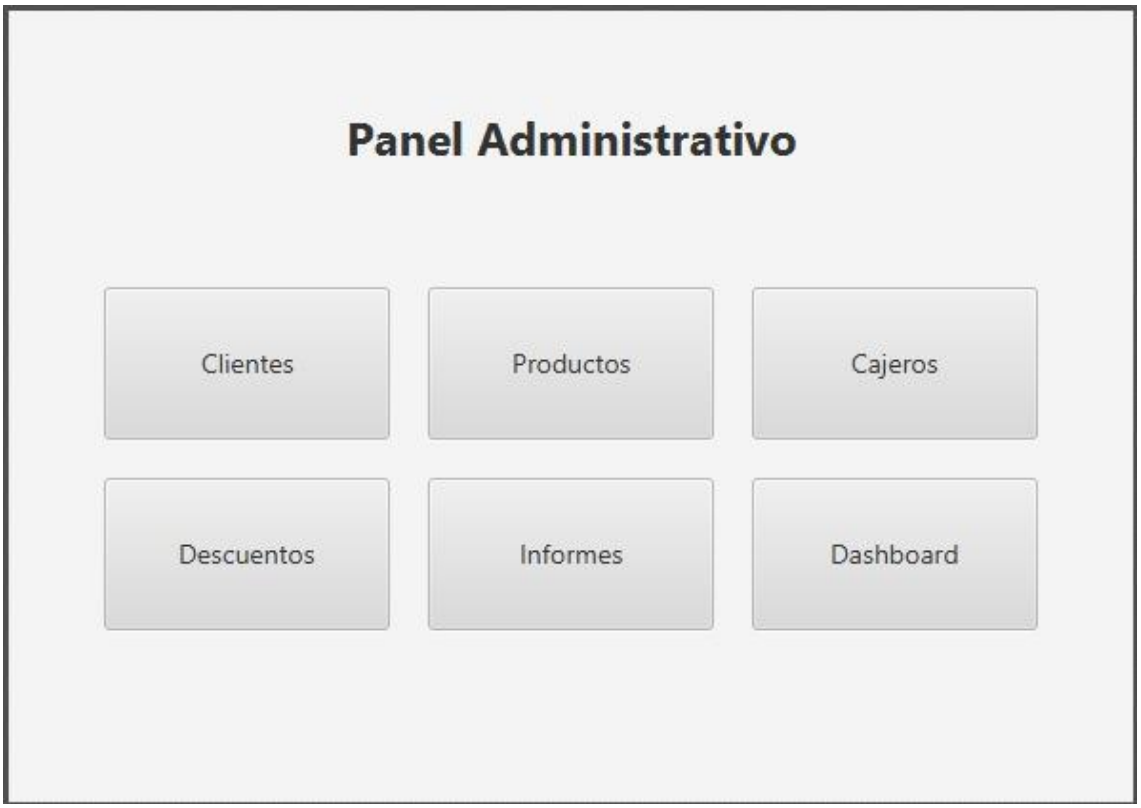
- **CashierPayment.FXML:** 3 – Flujo Cajero



Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

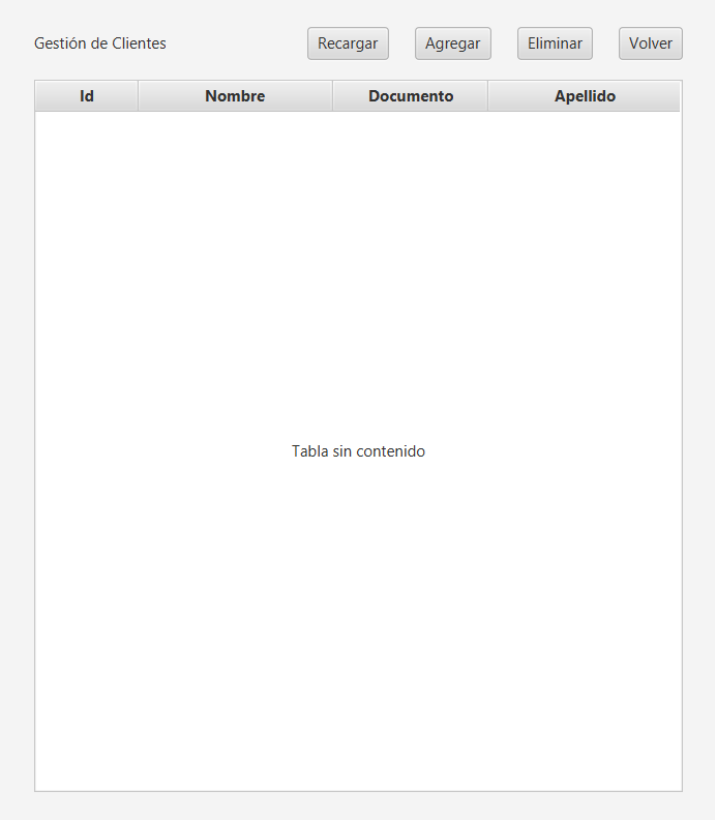
- **CashierConfirm.FXML:** 4 – Flujo Cajero

- **AdminRoot.FXML:** 1 – Flujo Administración

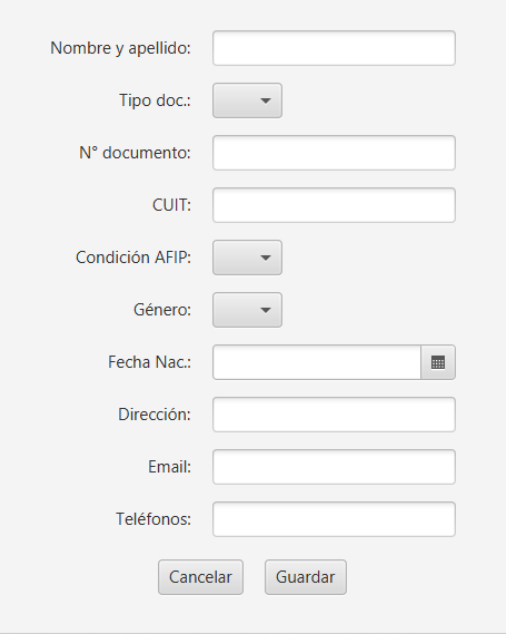


Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- **AdminCliente.FXML:** 2.1 – Flujo Administración

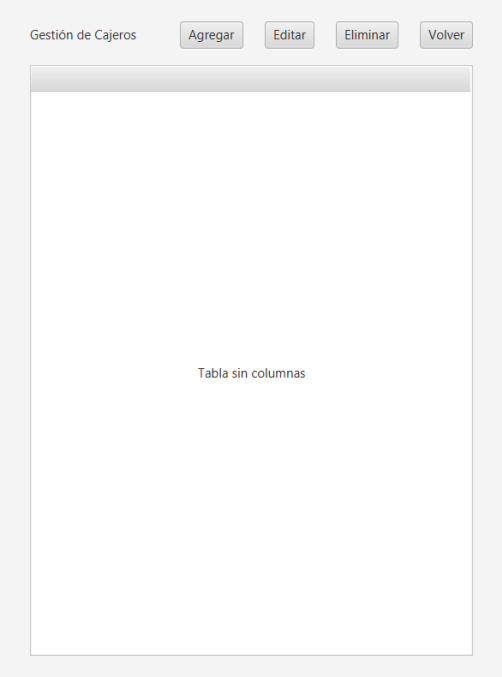


- **AdminClientesForm.FXML:** 3.1 – Flujo Administración

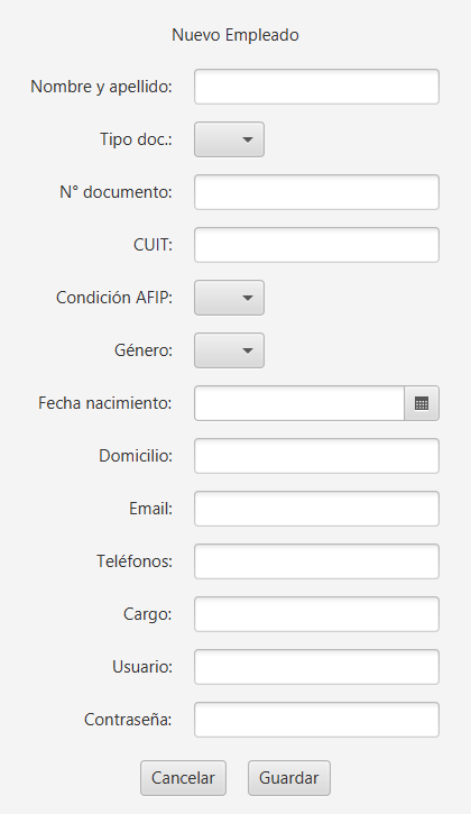


Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- **AdminCajero.FXML:** 2.2 – Flujo Administración

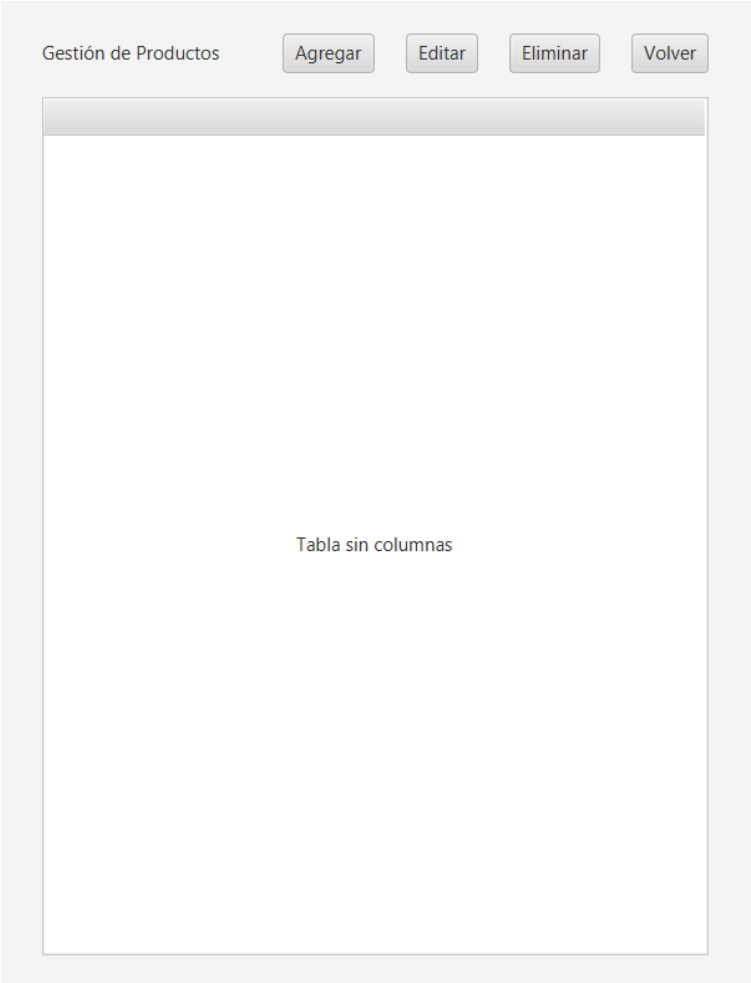


- **AdminCajeroForm.FXML:** 3.2 – Flujo Administración



Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

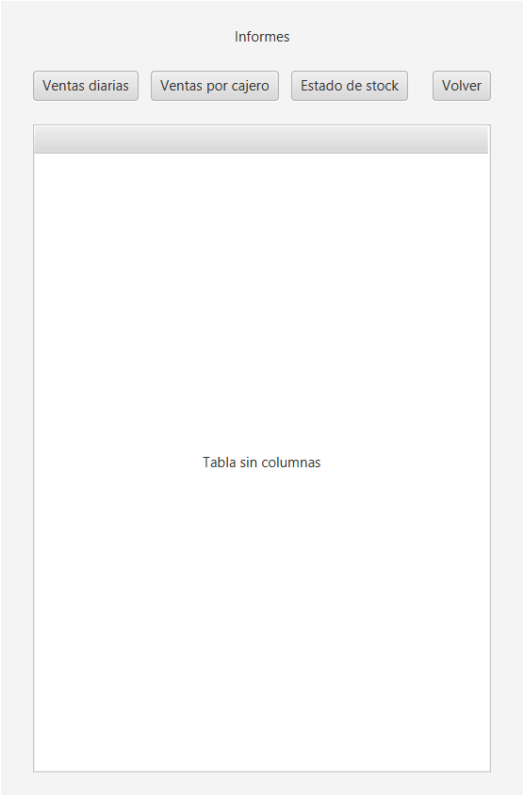
- **AdminProductos.FXML:** 2.3 – Flujo Administración



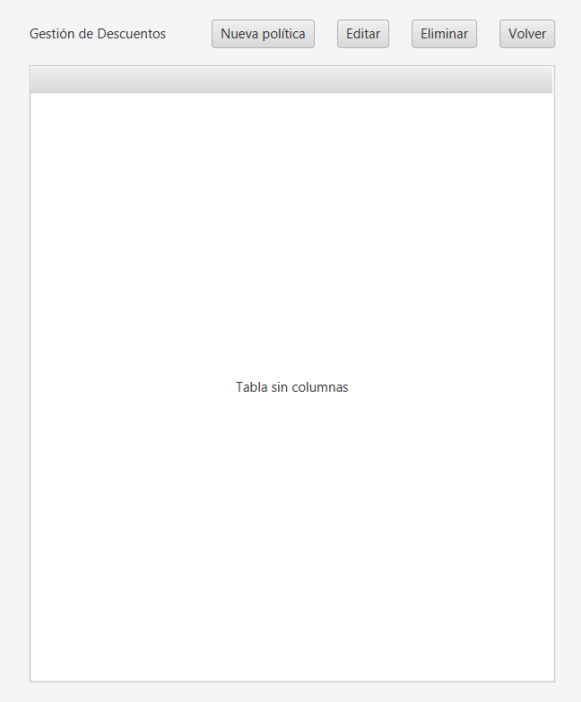
- **AdminProductosForm.FXML:** 3.3 – Flujo Administración

Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- **AdminReports.FXML:** 2.4 – Flujo Administración

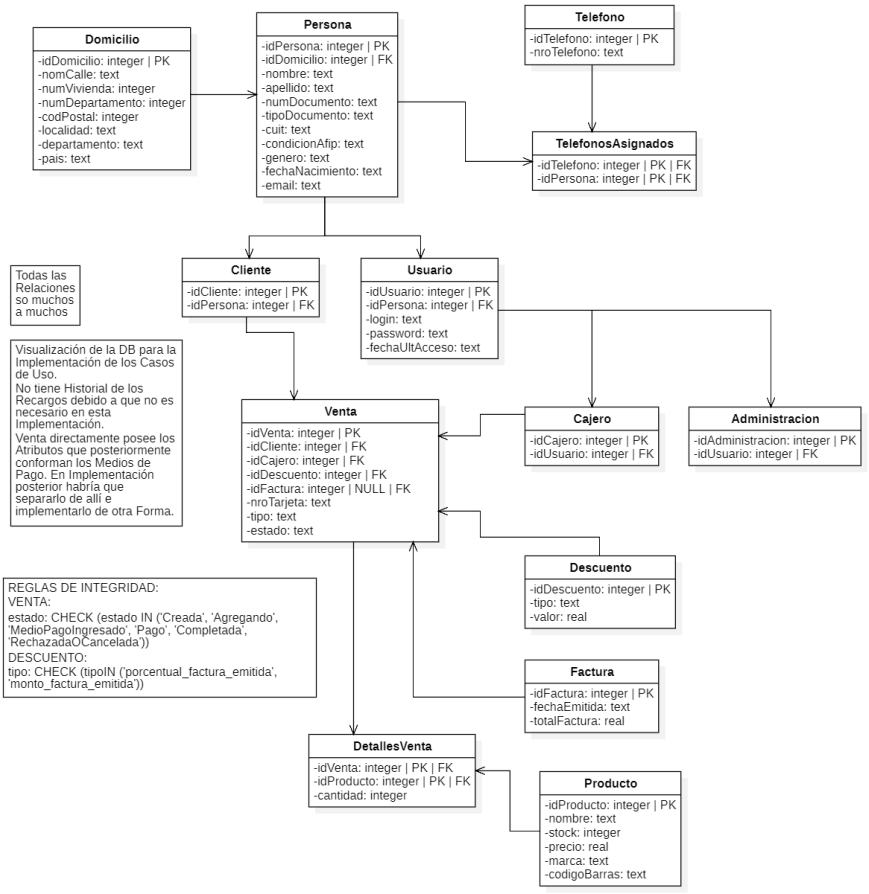


- **AdminDescuentos.FXML:** 2.5 – Flujo Administración



Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- # Diagrama de la Base de Datos del Sistema:



- # Implementación de Casos de Uso Principales:

Los Principales Casos de Uso implementados fueron:

- **Ingreso de una Venta,** por parte del Cajero, con todos sus Datos Relevantes, y su posterior manejo hasta la generación de la Factura. Generándose adecuadamente las respectivas entradas en la DB.
- **Ingreso de un Cliente,** por parte del Administrador, en el Sistema. Incluyendo todos los Datos del Individuo, desde su Nombre, Documento, Domicilio, entre otros. Generándose adecuadamente las respectivas entradas en la DB.
- **Para revisar la Implementación, ver el Código Adjuntado programado en NetBeans.**

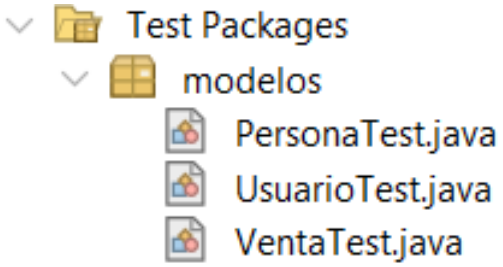
- **Tener en cuenta que en ConexionSQL.java hay dos implementaciones de la Conexión a la Base de Datos. Una que utiliza un Archivo .db almacenado en Disco que es persistente que requiere configuración de Equipo a Equipo. Y otra que utiliza la .db integrada en resources del Proyecto, la cual no es persistente, pero sirve para testear funcionalidades rápidamente. Esta ultima es la de DEFAULT.**

- ## Tests de Unidad:

Se implementaron Tests de Unidad de algunas de las Clases Principales del Sistema, incluyendo:

- **Venta**
- **Usuario**
- **Persona**

En estos Unit Tests o Tests de Unidad, se testearon no solo las Funciones Básicas, tales como Getters y Setters, pero también algunas funciones más especializadas de las Clases ya mencionadas.



- **UsuarioTest.java:**

```
package modelos;

import enums.Genero;
import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;

import static org.junit.Assert.*;

public class UsuarioTest {

    private Usuario usuario;
    private Domicilio domicilioInicial;

    @Before
```

Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

```
public void setUp() {
    // Ahora instanciamos también un Domicilio real:
   domicilioInicial = new Domicilio(
      "Calle 123",   // nomCalle
      10,            // numVivienda
      2,             // numDepartamento
      5000,          // codPostal
      "Ciudad",      // localidad
      "Provincia",   // departamento
      "Argentina"    // pais
   );

   // Construimos el Usuario pasando por el super constructor de Persona
   usuario = new Usuario(
      "Laura",                 // nombre
      "Ramírez",               // apellido
      "87654321",              // numDocumento
      "DNI",                   // tipoDocumento
      "27-87654321-0",         // cuit
      "Monotributo",           // condicionAfip
      Genero.Femenino,         // género
      "1988-07-15",            // fechaNacimiento
      domicilioInicial,        // domicilio
      "laura@ejemplo.com",     // email
      Arrays.asList("4242-4242") // teléfonos
   );
}

@Test
public void testGetCargo_devuelveNull() {
   // Según la implementación actual, getCargo() siempre retorna null.
   assertNull("getCargo() debe devolver null en la clase Usuario tal como está igualado",
           usuario.getCargo());
}

@Test
public void testLogin_getYSet() {
   // Al inicio login es null
   assertNull("Inicialmente login es null", usuario.getLogin());

   // Seteamos y comprobamos
   usuario.setLogin("laura123");
   assertEquals("El getter debe devolver el mismo login seteado",
            "laura123", usuario.getLogin());
}

@Test
public void testPassword_getYSet() {
   // Al inicio password es null
   assertNull("Inicialmente password es null", usuario.getPassword());

   // Seteamos y comprobamos
   usuario.setPassword("P@ssw0rd");
   assertEquals("El getter debe devolver la contraseña seteada",
            "P@ssw0rd", usuario.getPassword());
}

@Test
public void testFechaUltAcceso_getYSet() {
   // Al inicio fechaUltAcceso es null
   assertNull("Inicialmente fechaUltAcceso es null", usuario.getFechaUltAcceso());

   // Seteamos y comprobamos
```

```
        usuario.setFechaUltAcceso("2025-06-05 10:30");
        assertEquals("El getter debe devolver exactamente la fecha seteada",
                "2025-06-05 10:30", usuario.getFechaUltAcceso());
    }

    @Test
    public void testConstructorHeredaCamposDePersona() {
        // Verificamos que los campos heredados de Persona están bien
        assertEquals("Laura", usuario.getNombre());
        assertEquals("Ramírez", usuario.getApellido());
        assertEquals("87654321", usuario.getNumDocumento());
        assertEquals("DNI", usuario.getTipoDocumento());
        assertEquals("27-87654321-0", usuario.getCuit());
        assertEquals("Monotributo", usuario.getCondicionAfip());
        assertEquals(Genero.Femenino, usuario.getGenero());
        assertEquals("Femenino", usuario.getGeneroStr());
        assertEquals("1988-07-15", usuario.getFechaNacimiento());

        // Comprobamos cada campo de domicilio heredado:
        Domicilio d = usuario.getDomicilio();
        assertNotNull(d);
        assertEquals("Calle 123", d.getNomCalle());
        assertEquals(10, d.getNumVivienda());
        assertEquals(2, d.getNumDepartamento());
        assertEquals(5000, d.getCodPostal());
        assertEquals("Ciudad", d.getLocalidad());
        assertEquals("Provincia", d.getDepartamento());
        assertEquals("Argentina", d.getPais());

        assertEquals("laura@ejemplo.com", usuario.getEmail());
        assertEquals(1, usuario.getTelefonos().size());
        assertEquals("4242-4242", usuario.getTelefonos().get(0));
    }

    @Test
    public void testSettersDePersona_enUsuario() throws Exception {
        // Cambiamos y verificamos algunos campos heredados desde Persona
        usuario.setNombre("Lucía");
        usuario.setApellido("González");
        usuario.setNumDocumento("11223344");
        usuario.setTipoDocumento("LC");
        usuario.setCuit("23-11223344-1");
        usuario.setCondicionAfip("Responsable Inscripto");
        assertEquals("Lucía", usuario.getNombre());
        assertEquals("González", usuario.getApellido());
        assertEquals("11223344", usuario.getNumDocumento());
        assertEquals("LC", usuario.getTipoDocumento());
        assertEquals("23-11223344-1", usuario.getCuit());
        assertEquals("Responsable Inscripto", usuario.getCondicionAfip());

        // Probamos setGenero(String) heredado
        usuario.setGenero("Masculino");
        assertEquals(Genero.Masculino, usuario.getGenero());
        assertEquals("Masculino", usuario.getGeneroStr());

        // Testeamos algún caso inválido para setGenero(String)
        try {
            usuario.setGenero("INEXISTENTE");
            fail("Debe lanzar Exception si el String no coincide con algún valor enum Genero");
        } catch (Exception ex) {
            assertTrue(ex.getMessage().contains("No se pudo convertir"));
        }
```

```
        // Cambiamos fecha de nacimiento y email
        usuario.setFechaNacimiento("2000-01-01");
        usuario.setEmail("lucia@ejemplo.org");
        assertEquals("2000-01-01", usuario.getFechaNacimiento());
        assertEquals("lucia@ejemplo.org", usuario.getEmail());

        // Cambiamos domicilio a uno distinto
        Domicilio nuevoDom = new Domicilio(
            "Calle Nueva",  // nomCalle
            55,             // numVivienda
            3,              // numDepartamento
            1000,           // codPostal
            "OtraCiudad",   // localidad
            "OtraProv",     // departamento
            "Argentina"     // pais
        );
        usuario.setDomicilio(nuevoDom);
        Domicilio d2 = usuario.getDomicilio();
        assertNotNull(d2);
        assertEquals("Calle Nueva", d2.getNomCalle());
        assertEquals(55, d2.getNumVivienda());
        assertEquals(3, d2.getNumDepartamento());
        assertEquals(1000, d2.getCodPostal());
        assertEquals("OtraCiudad", d2.getLocalidad());
        assertEquals("OtraProv", d2.getDepartamento());
        assertEquals("Argentina", d2.getPais());

        // Cambiamos lista de teléfonos
        usuario.setTelefonos(Arrays.asList("9999-0000", "3333-2222"));
        assertEquals(2, usuario.getTelefonos().size());
        assertTrue(usuario.getTelefonos().contains("9999-0000"));
    }
}
```

- ## PersonaTest.java:

```
package modelos;

import org.junit.Before;
import org.junit.Test;

import java.util.Arrays;
import java.util.List;

import enums.Genero;

import static org.junit.Assert.*;

public class PersonaTest {

    private Persona persona;
    private Domicilio domicilio;

    @Before
    public void setUp() {
        // Ahora creamos un Domicilio real, usando el constructor de Domicilio.java:
        domicilio = new Domicilio(
            "Av. Siempre Viva",  // nomCalle
            123,                 // numVivienda
            4,                   // numDepartamento
            5000,                // codPostal
            "Springfield",       // localidad
            "Capital",           // departamento
            "Argentina"          // pais
```

```
        );

        // Valores iniciales para el constructor completo
        persona = new Persona(
            "Juan",                 // nombre
            "Pérez",                // apellido
            "12345678",             // numDocumento
            "DNI",                  // tipoDocumento
            "20-12345678-9",        // cuit
            "Responsable Inscripto",  // condicionAfip
            Genero.Masculino,       // género
            "1990-01-01",           // fechaNacimiento
            domicilio,          // domicilio
            "juan@ejemplo.com",     // email
            Arrays.asList("1234-5678", "8765-4321") // teléfonos
        );
    }

    @Test
    public void testGettersIniciales() {
        assertEquals("Juan", persona.getNombre());
        assertEquals("Pérez", persona.getApellido());
        assertEquals("12345678", persona.getNumDocumento());
        assertEquals("DNI", persona.getTipoDocumento());
        assertEquals("20-12345678-9", persona.getCuit());
        assertEquals("Responsable Inscripto", persona.getCondicionAfip());
        assertEquals(Genero.Masculino, persona.getGenero());
        assertEquals("Masculino", persona.getGeneroStr());
        assertEquals("1990-01-01", persona.getFechaNacimiento());

        // Verificamos cada campo de Domicilio que guardamos:
        Domicilio d = persona.getDomicilio();
        assertNotNull(d);
        assertEquals("Av. Siempre Viva", d.getNomCalle());
        assertEquals(123, d.getNumVivienda());
        assertEquals(4, d.getNumDepartamento());
        assertEquals(5000, d.getCodPostal());
        assertEquals("Springfield", d.getLocalidad());
        assertEquals("Capital", d.getDepartamento());
        assertEquals("Argentina", d.getPais());

        assertEquals("juan@ejemplo.com", persona.getEmail());
        List<String> telefonos = persona.getTelefonos();
        assertEquals(2, telefonos.size());
        assertTrue(telefonos.contains("1234-5678"));
        assertTrue(telefonos.contains("8765-4321"));
    }

    @Test
    public void testSettersSimples() {
        // Nombre y apellido
        persona.setNombre("María");
        persona.setApellido("Gómez");
        assertEquals("María", persona.getNombre());
        assertEquals("Gómez", persona.getApellido());

        // Documento y tipo
        persona.setNumDocumento("87654321");
        persona.setTipoDocumento("LC");
        assertEquals("87654321", persona.getNumDocumento());
        assertEquals("LC", persona.getTipoDocumento());

        // CUIT y condición AFIP
```

```
        persona.setCuit("23-87654321-0");
        persona.setCondicionAfip("Monotributo");
        assertEquals("23-87654321-0", persona.getCuit());
        assertEquals("Monotributo", persona.getCondicionAfip());

        // Fecha de nacimiento
        persona.setFechaNacimiento("1985-12-31");
        assertEquals("1985-12-31", persona.getFechaNacimiento());

        // Email
        persona.setEmail("maria@ejemplo.org");
        assertEquals("maria@ejemplo.org", persona.getEmail());

        // Teléfonos
        persona.setTelefonos(Arrays.asList("1111-2222"));
        List<String> tel = persona.getTelefonos();
        assertEquals(1, tel.size());
        assertEquals("1111-2222", tel.get(0));
    }

    @Test
    public void testSetYGetDomicilio() {
        // Cambiamos el domicilio a uno distinto
        Domicilio otroDom = new Domicilio(
            "Calle Falsa",  // nomCalle
            742,            // numVivienda
            1,              // numDepartamento
            1000,           // codPostal
            "Shelbyville",  // localidad
            "Capital",      // departamento
            "Argentina"     // pais
        );
        persona.setDomicilio(otroDom);
        Domicilio d2 = persona.getDomicilio();
        assertNotNull(d2);
        assertEquals("Calle Falsa", d2.getNomCalle());
        assertEquals(742, d2.getNumVivienda());
        assertEquals(1, d2.getNumDepartamento());
        assertEquals(1000, d2.getCodPostal());
        assertEquals("Shelbyville", d2.getLocalidad());
        assertEquals("Capital", d2.getDepartamento());
        assertEquals("Argentina", d2.getPais());
    }

    @Test
    public void testSetGenero_porString_valido() throws Exception {
        // Pongo el género como String (coincidente con algún valor del enum)
        persona.setGenero("Femenino");
        assertEquals(Genero.Femenino, persona.getGenero());
        assertEquals("Femenino", persona.getGeneroStr());
    }

    @Test
    public void testSetGenero_porString_invalido_lanzaException() {
        try {
            persona.setGenero("OTRO_NO_EXISTE");
            fail("Debe lanzar excepción si el String no coincide con ningún valor de enum Genero");
        } catch (Exception ex) {
            assertTrue(ex.getMessage().contains("No se pudo convertir"));
        }
    }
}
```

Materia: Diseño Orientado a Objetos.
Institución: Universidad Blas Pascal.
Profesor: Esp. Ing. Agustín Fernández.
Alumnos: Gastón Molina, Tomas Molina, Facundo Gómez, y Edgar Karpowicz.

- ## VentaTest.java:

```
package modelos;

import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

import java.util.List;

public class VentaTest {

   private Venta venta;
   private Producto prodA;
   private Producto prodB;
   private DetallesVenta detA;
   private DetallesVenta detB;

   @Before
   public void setUp() {
      // Creamos dos productos con diferentes precios
      prodA = new Producto("Manzana", 100, 10.0f, "FrutasSA", "COD123");
      prodB = new Producto("Banana", 50, 5.0f, "FrutasSA", "COD456");

      // Creamos los DetallesVenta correspondientes
      detA = new DetallesVenta(prodA, 2); // 2 * 10 = 20
      detB = new DetallesVenta(prodB, 3); // 3 * 5  = 15

      // Instanciamos una Venta “vacía”. Pasamos null a cliente/cajero
      venta = new Venta(null, null);

      // Agregamos los detalles manualmente
      venta.agregarDetalleVenta(detA);
      venta.agregarDetalleVenta(detB);
      // Ahora la venta tiene dos detalles: [detA, detB] y su total base = 20 + 15 = 35
   }

   @Test
   public void testCalcularTotal_sumaExacta() {
      float total = venta.calcularTotal();
      assertEquals("El total debe ser 35.0 (20 + 15).", 35.0f, total, 0.0001f);
   }

   @Test
   public void testCalcularTotalConRecargo_sinMedioPago() {
      float totalRec = venta.calcularTotalConRecargo();
      assertEquals("Sin medio de pago, debe ser igual al total base (35).", 35.0f, totalRec, 0.0001f);
   }

   @Test
   public void testCalcularTotalConDescuentoYRecargo_sinDescuentoYMedioPago() {
      float totalFinal = venta.calcularTotalConDescuentoYRecargo();
      assertEquals("Sin descuento y sin medio de pago, debe ser 35.", 35.0f, totalFinal, 0.0001f);
   }

   @Test
   public void testEmitirFactura_y_getFactura() {
      assertNull("Inicialmente no debe haber factura.", venta.getFactura());
      venta.emitirFactura("2025-06-05", 35.0f);
      Factura f = venta.getFactura();
      assertNotNull("Luego de emitirFactura, no debe ser null.", f);
      assertEquals("La fecha debe guardarse correctamente.", "2025-06-05", f.getFechaEmitida());
      assertEquals("El totalFactura debe ser 35.0.", 35.0f, f.getTotalFactura(), 0.0001f);
```

```
    }

    @Test
    public void testBorrarDetalleVenta_porPosiciónValida() {
        venta.borrarDetalleVenta(0);
        assertEquals("Después de borrar índice 0, queda solo detB (15).", 15.0f, venta.calcularTotal(), 0.0001f);
        List<DetallesVenta> lista = venta.getDetalles();
        assertEquals("Debe quedar 1 detalle.", 1, lista.size());
        assertEquals("El detalle restante debe corresponder a prodB.", prodB, lista.get(0).getProducto());
    }

    @Test
    public void testBorrarDetalleVenta_posiciónNegativa_oFueraDeRango() {
        venta.borrarDetalleVenta(-1);
        venta.borrarDetalleVenta(999);
        assertEquals("Índices inválidos no deben alterar total.", 35.0f, venta.calcularTotal(), 0.0001f);
        assertEquals("La lista de detalles sigue con 2 elementos.", 2, venta.getDetalles().size());
    }

    @Test
    public void testBorrarDetalleProducto_existente() {
        venta.borrarDetalleProducto(prodA);
        assertEquals("Al borrar prodA, queda solo detB con total 15.", 15.0f, venta.calcularTotal(), 0.0001f);
        assertEquals("Lista debe tener 1 detalle.", 1, venta.getDetalles().size());
        assertEquals("El detalle restante debe ser prodB.", prodB, venta.getDetalles().get(0).getProducto());
    }

    @Test
    public void testBorrarDetalleProducto_noExistente() {
        Producto fantasma = new Producto("Pera", 10, 7.0f, "FrutasSA", "COD999");
        venta.borrarDetalleProducto(fantasma);
        assertEquals("Si el producto no existe, total sigue 35.", 35.0f, venta.calcularTotal(), 0.0001f);
        assertEquals("Lista sigue con 2 detalles.", 2, venta.getDetalles().size());
    }
}
```