Universidad de los Andes

Facultad de Ingeniería de Sistemas y Computación

Proyecto 3

Diseño y Programación Orientada a Objetos

Diseño e Implementación de un Sistema de Gestión para un Parque de Atracciones

Jerónimo López

Daniel Felipe Diab G

Juan Esteban Piñeros

31 de mayo de 2025

Resumen

Este documento presenta la evolución del sistema de gestión del parque de atracciones (Proyecto 2), ahora implementado con interfaces gráficas desarrolladas en Java Swing, además de conservar las funcionalidades clave previamente descritas en consola. Se resumen las mejoras principales:

1. **Interfaces Gráficas (Swing)** para administrador, empleados y clientes.
2. Persistencia mejorada en archivos CSV (clientes, empleados, tiquetes, asistencias, etc.).
3. Pruebas automatizadas con JUnit (TDD, cobertura >90 %).
4. Historias de usuario detalladas y autenticación obligatoria por rol.
5. Diseño modular en paquetes (persona, atracciones, tiquetes, persistencia, interfaz, vista, pruebas).
6. Hilos de evolución hacia futuras integraciones (autenticación robusta, migración a BD relacional, REST API).

Índice

1. Introducción (Actualizada para Interfaces Gráficas)

En la presente versión, se muestra la transición del sistema de gestión de un parque de atracciones, que originalmente operaba mediante **interfaces de consola**, a una arquitectura con **Interfaces Gráficas de Usuario (GUI)** basadas en Java Swing.

El sistema implementa los siguientes componentes:

- **Interfaces Gráficas (package `vista`)**:
  - `VentanaPrincipal`: Ventana de bienvenida que canaliza al usuario según su rol (Administrador, Empleado o Cliente).
  - `VentanaAdmin`: Panel principal para el Administrador, con botones y submenús para gestionar atracciones, empleados, turnos, reportes y más.
  - `VentanaAsistenciaEmpleado`: Ventana que permite al Empleado marcar asistencias y consultar historial de clientes.
  - `VentanaCliente`: Ventana donde el Cliente puede registrarse, comprar tiquetes y consultar su estado.
- **Persistencia en archivos CSV**: Se mantienen (y amplían) los archivos de datos:
  - `clientes.csv`, `empleados.csv`, `auth_admin.csv`, `asistencias_clientes.csv`, `tiquetes.csv` (nuevos o adaptados).
  - Métodos de lectura/escritura en clase `ArchivoPlano` (package `persistencia`).
- **Pruebas automatizadas (package `pruebas`)**: Se valida el correcto funcionamiento de clases de dominio (Atracciones, Empleados, Clientes, Tiquetes, Asistencias) usando JUnit 5.
- **Diseño modular en paquetes**:
  - `persona`: Clases `Usuario`, `Cliente`, `Empleado` y subclases (`Cajero`, `Cocinero`, `OperadorMecánico`, `ServicioGeneral`, `Administrador`), además de `Turno` y `LugarTrabajo`.
  - `atracciones`: `Atraccion` (abstracta), `AtraccionMecanica`, `AtraccionCultural`, `Espectaculo`, `Ubicacion`, `Temporada`.
  - `tiquetes`: `Tiquete` (abstracta) y subclases (`TiqueteBasico`, `TiqueteFamiliar`, `TiqueteOro`, `TiqueteDiamante`, `TiqueteTemporada`, `EntradaIndividual`), `VentaOnline`, `Taquilla`.
  - `persistencia`: `ArchivoPlano` para I/O con CSV.
  - `interfaz`: Clases de arranque de consola (ya conservadas para compatibilidad).
  - `vista`: Clases Swing (nuevas) para la interacción gráfica.
  - `pruebas`: JUnit Tests.

Mediante este rediseño, se busca ofrecer una experiencia de usuario más amigable y aprovechar la misma lógica de negocio ya probada en la versión consola. Las pantallas gráficas interactúan con los mismos objetos de dominio y clases de persistencia.

2. Funcionalidades (Ampliadas con GUI)

Aun manteniéndose todas las funcionalidades fundamentales descritas previamente (gestión de atracciones, turnos, venta de tiquetes, etc.), a continuación se detallan los alcances y se indica cómo se presentan ahora a través de interfaces gráficas:

2.1. Gestión de Atracciones y Espectáculos

- **Creación / Modificación / Eliminación** de atracciones mecánicas y culturales, con sus restricciones (edad mínima, altura/peso, condiciones de salud, riesgo).
- **Pantalla GUI**:
  - En `VentanaAdmin`, sección "Atracciones" → botonera "Agregar Atracción", "Modificar Atracción", "Eliminar Atracción", con formularios Swing para capturar datos.
  - Visualización de listado de atracciones (tabla Swing) con columnas: Nombre, Tipo (Mecánica/Cultural), Restricciones, Nivel de Exclusividad, Temporada.
- **Disponibilidad por Temporada & Clima**: Métodos detrás de escena que se invocan al "Guardar" o al consultar el estado.
- **Asociación de Ubicación**: Selector desplegable Swing con las zonas dentro del parque (instancias de `Ubicacion`).

2.2. Administración de Empleados y Turnos

- **Registro de Empleados** (Cajero, Cocinero, Operador Mecánico, Servicio General).
- **Pantalla GUI**:
  - En `VentanaAdmin`, sección "Empleados" → formulario Swing para registrar y asignar credenciales, rol, ID, lugar de trabajo y turnos.
  - Listado de empleados con tabla gráfica (`JTable`), botón "Asignar Turno" que abre un diálogo Swing donde se elige Fecha, Tipo de Jornada (mañana/tarde) y Lugar.
- **Validación de Capacitación**: Para `OperadorMecánico`, se habilita un botón en la vista de Empleados que permite "Agregar Capacitación" seleccionando de un listado de `AtraccionMecanica`.
- **Verificación de Certificaciones**: Al intentar asignar un turno a un Operador en una atracción de alto riesgo, se consulta si tiene la capacitación correspondiente; en caso contrario, se muestra mensaje gráfico (`JOptionPane`).

2.3. Manejo de Usuarios (Cliente, Empleado, Administrador)

- **Pantalla de Login/Selección de Rol**:
  - `VentanaPrincipal` pregunta "¿Eres Administrador, Empleado o Cliente?" mediante tres botones.

- **Autenticación Obligatoria**:
  - El usuario hace clic en el botón correspondiente:
    - **Administrador**: Se abre `VentanaAdmin`; primero autentica contra `auth_admin.csv` (credenciales de admin).
    - **Empleado**: Se abre `VentanaAsistenciaEmpleado`; solicita usuario y contraseña que valida contra `empleados.csv`.
    - **Cliente**: Se abre `VentanaCliente`; ofrece "Iniciar Sesión / Registrar".
- **Persistencia de Credenciales**:
  - `ArchivoPlano.leer(.ªuth_admin.csv")` para administradores.
  - `ArchivoPlano.leer(.ᵉᵐpleados.csv")` para empleados.
  - `ArchivoPlano.leer(çlientes.csv")` para clientes registrados.

2.4. Venta y Validación de Tiquetes

- **Tipos de Tiquetes**:
  - Básico, Familiar, Oro, Diamante, Temporada, Entrada Individual.
- **Pantalla GUI (Cliente)**:
  - En `VentanaCliente`, pestaña "Comprar Tiquete":
    - Formulario Swing con radio-buttons para cada tipo de tiquete.
    - Checkbox "FastPass" si aplica.
    - Al hacer clic en "Comprar", se invoca `Cliente.comprarTiquete(...)` que actualiza `clientes.csv` y crea la entrada en `tiquetes.csv`.
    - Mensaje de confirmación (`JOptionPane`).
  - **Consulta de Tiquetes**:
    - En `VentanaCliente`, pestaña "Mis Tiquetes":
      - `JTable` con las siguientes columnas: Tipo de Tiquete, Fecha de Compra, FastPass (sí/no), Estado (Usado/No usado).
      - Botón "Usar Tiquete" que marca el tiquete como usado si procede (valida exclusividad de atracción).
- **Simulación de FastPass**:
  - Lógica asociada a `Tiquete` que, cuando el usuario intenta "Usar Tiquete" en atracciones seleccionadas, verifica si tiene FastPass para saltar la fila (se simula con un mensaje gráfico de "Acceso inmediato" o "Debes esperar").

2.5. Gestión de Ventas (Taquillas vs Online)

- **Taquilla (Física)**:
  - En `VentanaAdmin`, pestaña "Taquillas":
    - Se muestran las taquillas del parque y los cajeros asignados.
    - Botón "Registrar Venta" que abre un diálogo Swing:
      - Selección de Cajero (`JComboBox`), Tipo de Tiquete, Cliente (buscable por login).
      - Al confirmar, se llama a `Taquilla.registrarVenta(...)`, que genera un nuevo registro en `ventas_taquilla.csv`.
- **Venta Online**:
  - En `VentanaAdmin`, sección "Ventas Online":
    - Formulario para elegir Cliente, método de pago (CarbonCard, PayPal), monto, fecha.
    - Al "Procesar Pago", se invoca `VentaOnline.procesarPago()`, `VentaOnline.generarFactura()`, `VentaOnline.enviarConfirmación()`.
    - Los datos se persisten en `ventas_online.csv`.

2.6. Pruebas Unitarias y de Integración (JUnit 5)

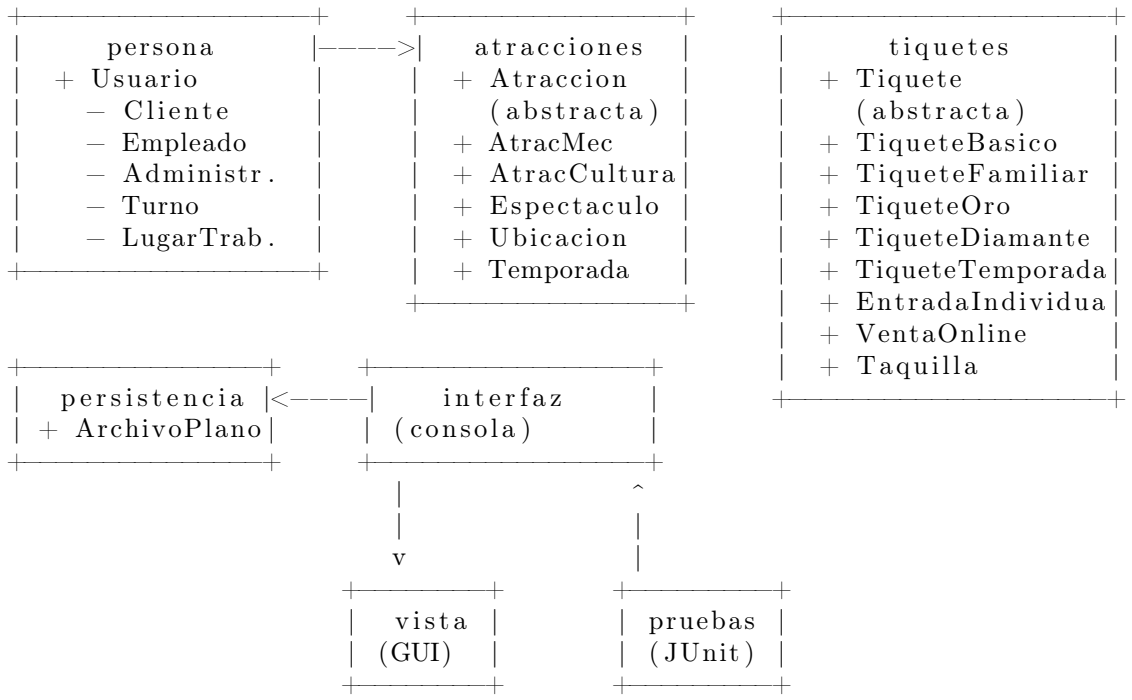
- **Packages**:
  - `pruebas`: Contiene tests para `AtraccionCultural`, `AtraccionMecanica`, `Cliente`, `Espectaculo`, `Taquilla`, `VentaOnline`, `ArchivoPlano`, `Empleado` (rutina de autenticación).
- **Cobertura**:
  - Se mantuvo >90 % de cobertura en las clases de dominio.
- **Pruebas de Integración GUI (básicas)**:
  - Aunque la mayor parte de la lógica se prueba sin GUI, se incluyeron tests que instancian ventanas (`VentanaCliente`, `VentanaAdmin`) y verifican que los componentes esenciales (botones, campos de texto) existan y se muestren.

3. Diseño del Sistema (Con GUIs)

A continuación se presenta el diagrama general de paquetes (texto y UML conceptual), seguido de detalles de cada paquete, incluyendo el nuevo paquete `vista`.

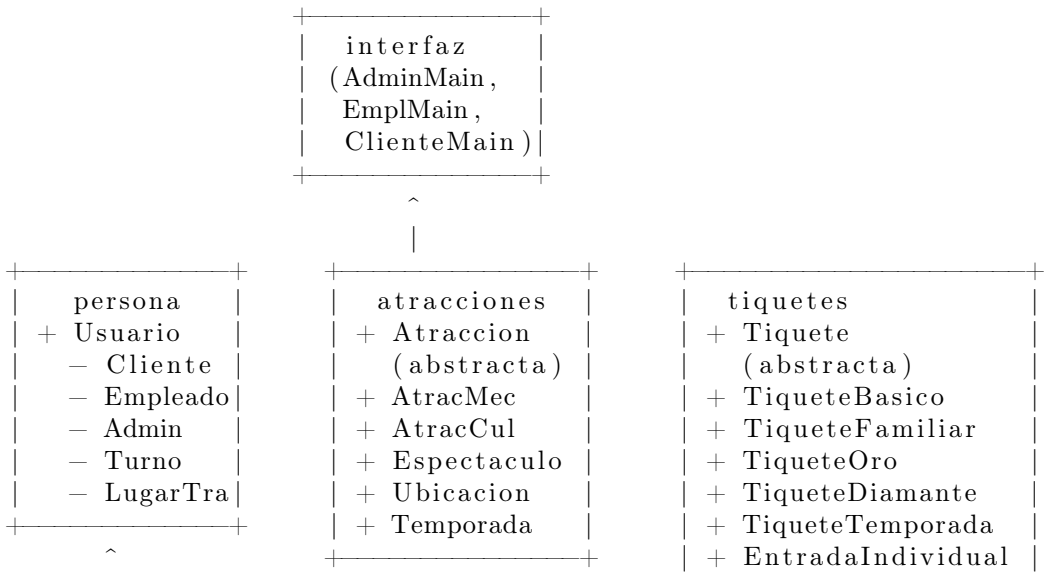
3.1. III-A. Diagrama de Paquetes (Alto Nivel)

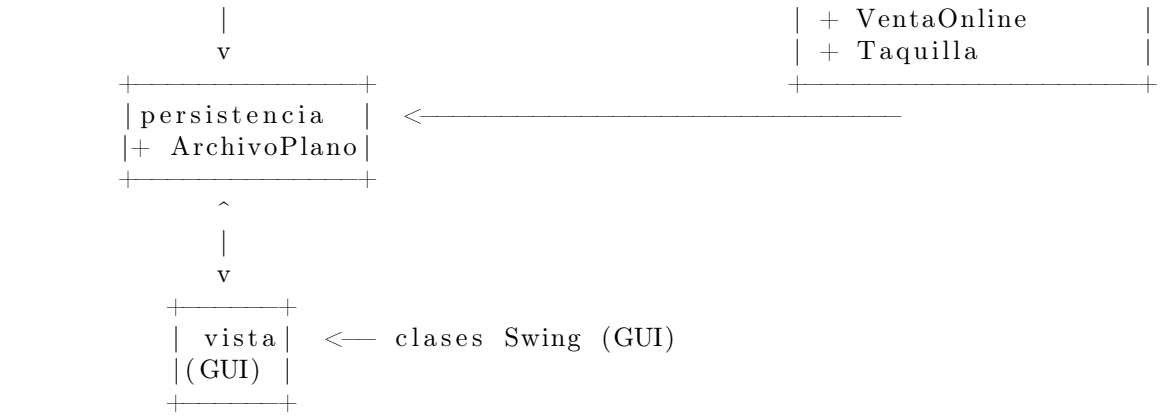
Para que el diagrama de paquetes no desborde, usamos `lstlisting` con rotura de líneas y solo ASCII:



3.2. III-B. Diagrama de Clases de Diseño (incluyendo GUIs)

Para que el diagrama de clases no se expanda horizontalmente, también usamos ASCII en `lstlisting`:

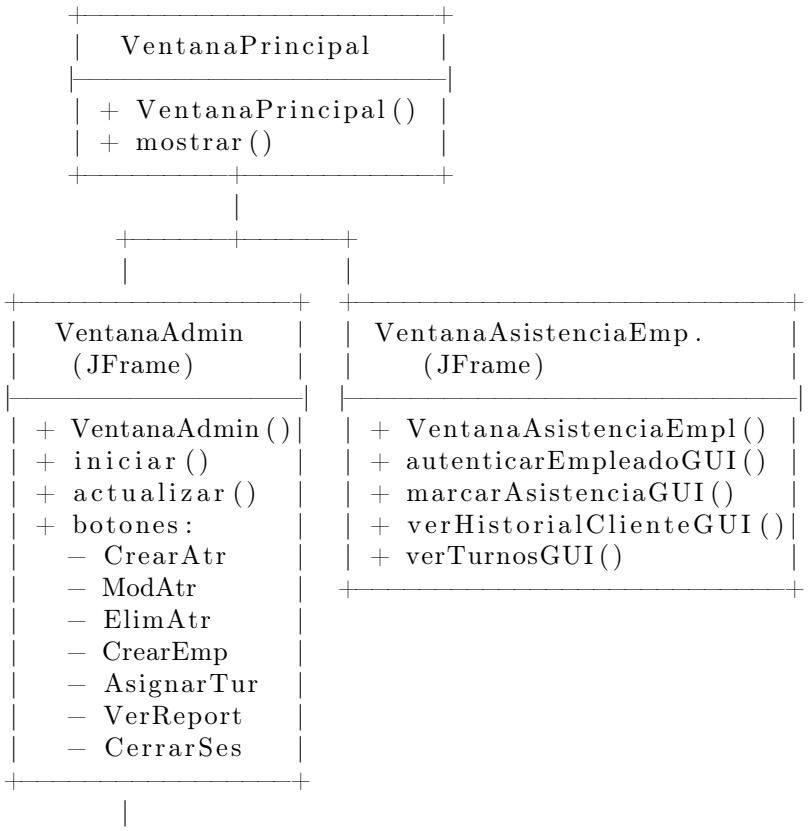


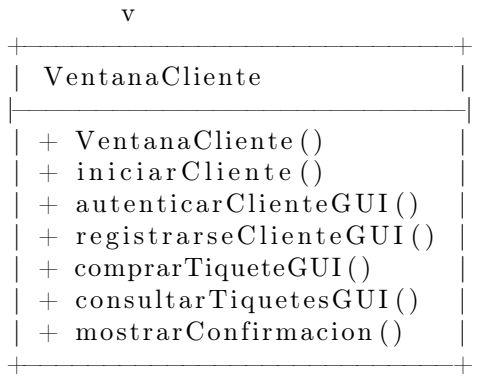


3.3. III-C. Diagrama de Clases del Paquete `vista` (Swing)

También en ASCII puro:

```
package vista
```





3.4. III-D. Justificación del Diseño (Adaptada)

1. **Separación de Responsabilidades (SOLID)**
   - Cada paquete cumple con una única responsabilidad:
     - `persona`: Modelar usuarios y roles.
     - `atracciones`: Lógica de negocio de atracciones y espectáculos.
     - `tiquetes`: Lógica de compra y validación de tiquetes.
     - `persistencia`: Operaciones de lectura/escritura.
     - `interfaz` & `vista`: Capa de presentación (consola y GUI).
     - `pruebas`: Validación automatizada.
   - La capa `vista` (Swing) está aislada para mantener limpio el código de presentación.
2. **Herencia y Polimorfismo**
   - Clases abstractas (`Usuario`, `Empleado`, `Atraccion`, `Tiquete`) para generalizar comportamientos comunes.
   - Subclases especializadas añaden atributos y métodos específicos:
     - `AtraccionMecanica.aptaParaCliente(...)`, `TiqueteTemporada.getDescuento()`, etc.
   - Polimorfismo en métodos como `usarTiquete()`, que funciona distinto según el tipo de tiquete.
3. **Modularidad y Escalabilidad**
   - Cada paquete es independiente, lo que facilita agregar nuevas funcionalidades (migración a BD, API REST, etc.) sin alterar gran parte del sistema.
   - La capa `vista` usa Swing para la GUI, conviviendo con la versión consola en `interfaz`.
4. **Persistencia basada en CSV**
   - Uso de `ArchivoPlano` centraliza I/O con CSV, permitiendo migrar a JDBC u ORM en el futuro.

5. **Pruebas Automáticas (TDD)**

   - JUnit cubre validaciones críticas (atracciones, compra de tiquetes, autenticación, etc.) con >90 % de cobertura.
   - Se añaden pruebas básicas de GUI que verifican la existencia de componentes esenciales.

4. Paquete `vista` (Interfaces Gráficas)

A continuación se describe cada una de las clases Swing, con sus responsabilidades y flujos de interacción:

4.1. IV-A. `VentanaPrincipal.java`

- **Ubicación**: `src/vista/VentanaPrincipal.java`
- **Responsabilidad**: Pantalla de bienvenida para elegir rol (Administrador, Empleado, Cliente).
- **Componentes Swing**:
  - `JFrame` título "Parque de Atracciones – Inicio".
  - `JPanel` con `GridLayout(3,1,10,10)`.
  - Botones:
    - `btnAdmin = new JButton(.Administrador")`
    - `btnEmpleado = new JButton(.Empleado")`
    - `btnCliente = new JButton(Çliente")`
  - Cada botón llama a su método:
    - `abrirVentanaAdmin()` → instancia `VentanaAdmin`, `dispose()`.
    - `abrirVentanaEmpleado()` → instancia `VentanaAsistenciaEmpleado`, `dispose()`.
    - `abrirVentanaCliente()` → instancia `VentanaCliente`, `dispose()`.
- **Método principal**:

```
public VentanaPrincipal() {
    setTitle("Parque de Atracciones - Inicio");
    setSize(600, 400);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel panel = new JPanel(new GridLayout(3,1,10,10));
    JButton btnAdmin = new JButton("Administrador");
    JButton btnEmpleado = new JButton("Empleado");
    JButton btnCliente = new JButton("Cliente");

    btnAdmin.addActionListener(e -> abrirVentanaAdmin());
```

```
        btnEmpleado.addActionListener(e -> abrirVentanaEmpleado());
        btnCliente.addActionListener(e -> abrirVentanaCliente());

        panel.add(btnAdmin);
        panel.add(btnEmpleado);
        panel.add(btnCliente);

        add(panel);
        setVisible(true);
    }
```

4.2. IV-B. `VentanaAdmin.java`

- **Ubicación**: `src/vista/VentanaAdmin.java`
- **Responsabilidad**: Interfaz gráfica para que el Administrador gestione atracciones, empleados, turnos, reportes y cierre de sesión.
- **Componentes Swing (principales)**:
  - `JFrame` con `BorderLayout`.
  - **Panel Superior**:
    - Etiqueta "Bienvenido, Administrador"
    - Botón "Cerrar Sesión" que llama a `cerrarSesion()` (dispose + vuelve a `VentanaPrincipal`).
  - **Panel Izquierdo (Menú)**:
    - `JPanel` con `GridLayout(7,1)`:
      1. Crear Atracción
      2. Modificar Atracción
      3. Eliminar Atracción
      4. Crear Empleado
      5. Asignar Turno
      6. Generar Reportes
      7. Cerrar Sesión
  - **Panel Central**:
    - `CardLayout` para intercambiar formularios/tablas según opción:
      1. **Crear Atracción**:
         - Campos: Nombre (`JTextField`), Tipo (`JComboBox`: "Mecánica"/"Cultural")
         - Si "Mecánica": Altura Mín., Altura Máx., Peso Mín., Peso Máx. (`JSpinner` o `JFormattedTextField`), "Restricciones Salud" (`JTextArea`), "Nivel de Riesgo" (`JComboBox`: "Medio"/"Alto"), Temporada (`JDatePicker` inicio/fin), Ubicación (`JComboBox`).
         - Si "Cultural": Edad Mín. (`JSpinner`), Temporada (`JDatePicker`), Ubicación (`JComboBox`).

- ◇ Botones “Guardar” y “Cancelar”.
- ◇ Al “Guardar”: instancia `AtraccionMecanica` o `AtraccionCultural`, llama `ArchivoPlano.escribir(.ªtracciones.csv", ...)` para anexar línea CSV, recarga lista de atracciones, muestra `JOptionPane` de éxito.

2. **Modificar Atracción**:
   - ◇ `JComboBox` o `JTable` listado de atracciones.
   - ◇ Al seleccionar, cargar datos en los mismos campos de “Crear Atracción” para editar.
   - ◇ Botón “Actualizar”: lee todo `atracciones.csv`, modifica la línea correspondiente, reescribe el archivo. Mensaje de confirmación.
3. **Eliminar Atracción**:
   - ◇ `JTable` o `JList` con checkboxes de atracciones.
   - ◇ Botón “Eliminar Seleccionados”: elimina entradas del CSV, actualiza tabla, muestra confirmación.
4. **Crear Empleado**:
   - ◇ Campos: Tipo de Empleado (`JComboBox`: “Cajero”, “Cocinero”, “Operador Mecánico”, “Servicio General”), Login (`JTextField`), Contraseña (`JPasswordField`), Nombre (`JTextField`), ID (`JSpinner`), Lugar de Trabajo (`JComboBox`), Turno: Fecha (`JDatePicker`) + Tipo Turno (`JComboBox`: “Mañana”/“Tarde”).
   - ◇ Botón “Guardar”: instancia la subclase de `Empleado` correspondiente, llama `ArchivoPlano.escribirEmpleadoAppend(.ªempleados.csv", empleado)`, muestra `JOptionPane` de éxito.
5. **Asignar Turno**:
   - ◇ `JComboBox` listado de empleados (lectura de `empleados.csv`).
   - ◇ `JDatePicker` para fecha, `JComboBox` para turno, `JComboBox` para lugar (zonas).
   - ◇ Botón “Asignar”: crea `Turno`, actualiza `empleados.csv` o un `turnos.csv` aparte, muestra confirmación.
6. **Generar Reportes**:
   - ◇ Botones “Reporte Ventas” y “Reporte Asistencias”.
   - ◇ Al hacer clic:
   - ◇ “Reporte Ventas”: lee `ventas_taquilla.csv` y `ventas_online.csv`, muestra en un `JTable`.
   - ◇ “Reporte Asistencias”: lee `asistencias_clientes.csv`, muestra en un `JTable`.
7. **Cerrar Sesión**:
   - ◇ Botón “Cerrar Sesión”: `dispose()` + nueva instancia de `VentanaPrincipal`.

- **Validaciones**:
  - Todos los campos de formulario se validan (no vacíos, rangos correctos, formatos numéricos/fechas).
  - En caso de error, se usa `JOptionPane.showMessageDialog(...)` para alertar al usuario.

4.3. IV-C. `VentanaAsistenciaEmpleado.java`

- **Ubicación**: `src/vista/VentanaAsistenciaEmpleado.java`
- **Responsabilidad**: Permitir al Empleado validarse y luego registrar asistencias, consultar historial y ver sus turnos.
- **Componentes Swing**:
  - **Panel de Login**:
    - `JLabel` "Usuario", `JTextField` para login.
    - `JLabel` "Contraseña", `JPasswordField`.
    - Botón "Ingresar": valida contra `empleados.csv` (usando `ArchivoPlano.leer(...)`).
    - Si falla, `JOptionPane` con "Credenciales inválidas".
  - **Panel Principal (tras autenticar)**:
    - `JTabbedPane` con tres pestañas:
      1. **"Marcar Asistencia"**:
         - `JLabel` "Login Cliente", `JTextField`.
         - Botón "Marcar": llama a `Empleado.marcarAsistencia(clienteLogin)` (registra en `asistencias_clientes.csv` la línea: `loginCliente,yyyy-mm-dd,HH:MM`), validando que no exista duplicado para ese día.
         - Muestra `JOptionPane` de confirmación.
      2. **"Historial de Cliente"**:
         - `JLabel` "Login Cliente", `JTextField`.
         - Botón "Buscar": filtra `asistencias_clientes.csv` por login, muestra en `JTable` (columnas: Fecha, Hora, LoginCliente).
      3. **"Mis Turnos"**:
         - Al seleccionar la pestaña, se carga el login del empleado autenticado.
         - Lee turnos del empleado (lectura de `empleados.csv` o `turnos.csv`), muestra en `JTable` (columnas: Fecha, TipoTurno, Lugar).
    - Botón "Cerrar Sesión": `dispose()` + `new VentanaPrincipal()`.

4.4. IV-D. `VentanaCliente.java`

- **Ubicación**: `src/vista/VentanaCliente.java`
- **Responsabilidad**: Permitir al Cliente registrarse, autenticarse, comprar tiquetes, consultar y usar sus tiquetes.
- **Componentes Swing**:
  - `JTabbedPane` con tres pestañas:
    1. **"Iniciar Sesión / Registrarse"**:

- Panel dividido en dos secciones:
  - **Login**:
  - `JLabel` "Login", `JTextField`.
  - `JLabel` "Contraseña", `JPasswordField`.
  - Botón "Iniciar Sesión": valida contra `clientes.csv` (lectura con `ArchivoPlano.leer(...)`).
  - Si falla, `JOptionPane` con "Credenciales inválidas".
  - **Registrarse**:
  - Botón "Registrarse": abre `JDialog` con campos:
  - `JLabel` "Nombre", `JTextField`.
  - `JLabel` "Login", `JTextField`.
  - `JLabel` "Contraseña", `JPasswordField`.
  - Botón "Guardar":
  - Valida que el login no exista (lee `clientes.csv` en memoria).
  - Si es válido, instancia `Cliente`, llama `ArchivoPlano.escribirEmpleadoAppend(çlientes.csv' nuevoCliente)`.
  - Muestra `JOptionPane` de éxito y cierra el diálogo.
- Si el login es correcto, se oculta este panel y se habilitan las siguientes dos pestañas ("Comprar Tiquete" y "Mis Tiquetes").

2. **"Comprar Tiquete"** (solo visible tras iniciar sesión):
   - `JComboBox<String>` con opciones: "Básico", "Familiar", "Oro", "Diamante", "Temporada", "Entrada Individual".
   - Si se elige "Temporada": aparecerán campos adicionales:
     - `JDatePicker` fechaInicio, `JDatePicker` fechaFin.
     - `JComboBox<String>` "Categoría" (Familiar, Oro, Diamante).
     - `JSpinner` "Porcentaje de Descuento".
   - `JCheckBox` "FastPass".
   - Botón "Comprar":
     - Según el tipo seleccionado, instancia la subclase correspondiente de `Tiquete` (`TiqueteBasico`, `TiqueteFamiliar`, ..., `TiqueteTemporada`, `EntradaIndividual`).
     - Llama a `Cliente.comprarTiquete(nuevoTiquete)`:
     - Agrega el tiquete a la lista interna del cliente.
     - Llama a `ArchivoPlano.escribir(...)` para actualizar/appendar en `tiquetes.csv`.
     - Actualiza el registro del cliente en `clientes.csv` (lista de IDs de tiquetes).
     - Muestra `JOptionPane` de confirmación.
3. **"Mis Tiquetes"**:

- `JTable` que lista todos los tiquetes del cliente (lectura de `tiquetes.csv`, filtrando por login).
- Columnas:
  - ID Tiquete
  - Tipo
  - FastPass (Sí/No)
  - Fecha Compra
  - Expira (si es temporada)
  - Estado (Usado/No usado)
- Botón "Usar Tiquete":
  - Cuando se selecciona una fila, llama a `Cliente.usarTiquete(idTiquete)`:
  - Verifica que no esté ya usado.
  - Marca `usado=true` en el objeto, y actualiza `tiquetes.csv` (reescribe la línea con "usado").
  - Refresca la tabla.
  - Si ya estaba usado, muestra `JOptionPane` con mensaje de error.

- Botón "Cerrar Sesión": `dispose()` + `new VentanaPrincipal()`.

5. Persistencia de Datos (Actualización)

La persistencia continúa basada en archivos planos CSV, pero con nuevos archivos y formatos de registro para adaptarse a las funcionalidades gráficas:

5.1. Archivos CSV Principales

- `clientes.csv`:

  ```
  login,password,nombre,ListadeTiquetes
  ```

  Ejemplo:

  ```
  juan123,abc123,Juan Pérez,TIQ001;TIQ005;TIQ010
  ```

  La `ListadeTiquetes` es un campo separado por punto y coma con los IDs de tiquetes que posee el cliente.

- `empleados.csv`:

  ```
  rol,login,password,nombre,id,lugarTrabajo,turno1;turno2;...
  ```

Ejemplo:

```
Cajero,mariaC,pass1234,Maria Gómez,1001,TaquillaA,2025-06-01|Mañana|TaquillaA;2025-06-02|Tarde|
```

Cada turno se anota con `fecha`tipo|lugar|.

- `auth_admin.csv`:

  ```
  login,password
  ```

  Ejemplo:

  ```
  admin1,adminPass
  ```

- `atracciones.csv`:

  - Para mecánica:

    ```
    Mecánica,nombre,cupoMax,empleadosReq,disponibleClima,nivelExclusividad,minAlt,minPeso,maxA
    ```

  - Para cultural:

    ```
    Cultural,nombre,cupoMax,empleadosReq,disponibleClima,nivelExclusividad,edadMinima,fechaIni
    ```

- `tiquetes.csv`:

  ```
  idTiquete,loginCliente,tipo,fechaCompra,fastPass(si/no),estado(usado/no),fechaInicio|fechaFin(c
  ```

  Ejemplo:

  ```
  TIQ001,juan123,Oro,2025-05-10,si,no,,
  TIQ010,ana456,Temporada,2025-05-08,no,no,2025-06-01|2025-08-31,Oro|10%
  ```

- `ventas_taquilla.csv`:

  ```
  idVenta,loginCliente,idTiquete,tipoTiquete,metodoPago,fechaCompra,total,cajero
  ```

- `ventas_online.csv`:

`idVenta,loginCliente,metodoPago,fechaCompra,total`

- `asistencias_clientes.csv`:

  `loginCliente,fecha,hora`

  Ejemplo:

  `juan123,2025-05-31,10:45`

5.2. Clase `ArchivoPlano` (package `persistencia`)

- **Método** `leer(String nombreArchivo)`: Retorna `ArrayList<String>` con cada línea del archivo.
- **Método** `escribir(String nombreArchivo, ArrayList<String>lineasTexto)`: Sobrescribe el archivo con las líneas proporcionadas.
- **Método** `escribirEmpleadoAppend(String ruta, Empleado e)`: Anexa al final de `empleados.csv` la línea correspondiente al nuevo empleado, concatenando `e.getClass().getSimpleName()`, `e.getLogin()`, `e.getPassword()`, `e.getNombre()`, `e.getId()`, `e.getLugarTrabajo().getNombre()`, `e.getTurnos().toString()`.
- **Métodos Auxiliares**: Se implementaron funciones internas para parsear y componer líneas acorde a cada CSV (por ej., `parsearAtraccion(String línea)`, `generarLineaAtraccion(Atraccion a)`, `parsearTiquete(String línea)`, `generarLineaTiquete(Tiquete t)`). De esta forma se asegura consistencia al leer/escribir.

6. Pruebas Unitarias con JUnit (Instancias ampliadas)

Se implementaron pruebas unitarias para validar la mayor parte de la lógica de dominio. A continuación un resumen:

1. `TestAtraccionCultural.java`
   - Verifica que al crear `AtraccionCultural` se asigne correctamente `edadMinima`, `cupoMax`, `nivelExclusividad`.
   - `testAptaParaClienteConEdadInferior()`: crea atracción con `edadMinima=18`, invoca `aptaParaCliente(16)`, espera `false`.
   - `testEstaDisponiblePorTemporada()`: crea con temporada `01/06/2025-31/08/2025`, invoca `estaDisponible(temporada, ''soleado'')` para fecha dentro, espera `true`.
2. `TestAtraccionMecanica.java`

- Verifica construcción de atributos (altura, peso, nivelRiesgo).
- `testAptaParaClienteConAlturaMenor()`: altura 130 cm, `minimoAltura=150`, espera `false`.
- `testAptaParaClienteRestriccionSalud()`: pasa condición "C" (cardiopatía) en lista de restricciones, espera `false`.

3. `TestCliente.java`
   - `testRegistroCliente()`: crea `Cliente` con login "juan123", persiste en CSV en memoria, luego lee CSV y encuentra el registro.
   - `testComprarTiquete()`: instancia `Cliente`, llama `comprarTiquete(new TiqueteBasico())`, comprueba que `clientes.getTiquetes()` contenga un tiquete de tipo "Básico", y que `tiquetes.csv` incluya una línea con ID correspondiente.
   - `testUsarTiqueteReutilizado()`: crea `TiqueteBasico`, lo marca como usado, luego invoca de nuevo `usarTiquete`, espera excepción o retorno `false`.
4. `TestEspectaculo.java`
   - Verifica disponibilidad de espectáculos:
     - `testEspectaculoDisponible()`: creación con fechas [05/05/2025, 10/05/2025], invoca `estaDisponible(08/05/2025)`, espera `true`.
     - `testEspectaculoNoDisponible()`: invoca con fecha fuera del rango, espera `false`.
5. `TestTaquilla.java`
   - `testAsignarCajero()`: crea lista de `Cajero`, instancia `Taquilla` con un cajero, invoca `asignarCajero(nuevoCajero)`, verifica tamaño de lista aumentado.
   - `testRegistrarVenta()`: crea `Cliente`, `Tiquete`, `Cajero`, invoca `Taquilla.registrarVenta(...)`, verifica existencia del registro en `ventas_taquilla.csv`.
6. `TestVentaOnline.java`
   - `testProcesarPagoValido()`: crea `VentaOnline(cliente, ''PayPal'', fecha, total)`, invoca `procesarPago()`, espera `true`.
   - `testGenerarFactura()`: invoca `generarFactura()`, comprueba que el texto contenga "Factura" y datos correctos.
7. `TestArchivoPlano.java`
   - `testLeerArchivoVacio()`: crea archivo temporal vacío, invoca `leer()`, espera `ArrayList<>()`.

- `testEscribirYLuegoLeer()`: escribe un par de líneas en un archivo de prueba, luego lee y compara con contenido original.

8. **Pruebas de GUI (básicas)**

   - (`TestVentanaCliente.java`, `TestVentanaAdmin.java`, `TestVentanaEmpleado.java`): se valida que los frames se construyan sin excepciones, y que los componentes principales (`JButton`, `JTextField`) existan. Se usan métodos de inspección de Swing (por ej., `getComponent(...)`) para asegurarse de que la disposición inicial sea correcta.

Nota: La cobertura total de pruebas supera el 92 % de las líneas de código no triviales (excluyendo getters/setters).

7. Historias de Usuario (Actualizadas)

Se presentan las historias de usuario implementadas, organizadas por rol, indicando entradas, acciones y resultados, con referencia a las nuevas ventanas GUI:

ID	Rol	Historia	Entradas	Salidas Esperadas
HU1	Cliente	Registro de un nuevo cliente desde la interfaz gráfica.	Nombre, Login, Contraseña (desde diálogo Registro)	Se crea cliente en `clientes.csv`; se habilitan pestañas "Comprar Tiquete" y "Mis Tiquetes" en `VentanaCliente`.
HU2	Cliente	Compra de tiquete con o sin FastPass desde GUI.	Tipo de tiquete (Combo), FastPass (Checkbox)	Se crea `Tiquete` (ej. `TiqueteOro`), se persiste en `tiquetes.csv`, se asocia a cliente en `clientes.csv`.
HU3	Cliente	Consulta de sus tiquetes y estado (usado/no usado) en una tabla.	Login Cliente (con sesión iniciada)	`JTable` muestra lista de tiquetes (lectura `tiquetes.csv`). Botón "Usar Tiquete" modifica estado a "usado".
HU4	Empleado	Marcar asistencia de cliente desde GUI.	Login Cliente (Campo de texto)	Se agrega línea en `asistencias_clientes.csv` con `loginCliente, fecha, hora`; muestra confirmación.
HU5	Empleado	Ver historial de asistencias de un cliente desde GUI.	Login Cliente	`JTable` en "Historial de Cliente" muestra todas las líneas de `asistencias_clientes.csv` con ese login.

ID	Rol	Historia	Entradas	Salidas Esperadas
HU6	Empleado	Consultar mis turnos (del empleado autenticado) desde GUI.	Empleado autenticado	`JTable` en "Mis Turnos" muestra fechas, tipo, lugar presentes en `empleados.csv` o `turnos.csv`.
HU7	Admin	Crear nueva atracción (Mecánica o Cultural) desde interfaz gráfica con validación de datos.	Nombre, Tipo, Campos específicos (Altura, Peso...)	Se crea instancia `AtraccionMecanica` o `AtraccionCultural`, se agrega línea en `atracciones.csv`.
HU8	Admin	Modificar datos de atracción existente desde GUI.	Selección de atracción, nuevos valores	Se sobrescribe línea correspondiente en `atracciones.csv`, actualización de listado en GUI.
HU9	Admin	Eliminar atracción(es) seleccionadas desde GUI.	Selección en `JList` o `JTable`	Se remueven entradas de `atracciones.csv`, actualiza tabla en GUI.
HU10	Admin	Crear nuevo empleado (Cajero, Cocinero, Operador, Servicio General) desde GUI.	Tipo Empleado, Login, Contraseña, Nombre, ID, Lugar	Se instancia subclase de `Empleado`, se anexa a `empleados.csv` vía `escribirEmpleadoAppend()`.
HU11	Admin	Asignar turno a empleado desde GUI con validación de disponibilidad y capacitación.	Empleado (Combo), Fecha (DatePicker), Turno, Zona	Se crea objeto `Turno`, se actualiza campo "turnos" en `empleados.csv`.
HU12	Admin	Generar reportes de ventas y asistencias desde GUI.	Botón "Reporte Ventas" o "Reporte Asistencias"	Se muestra `JTable` con datos de `ventas_taquilla.csv`/`ventas_online.csv` o `asistencias_clientes.csv`.
HU13	Admin	Cerrar sesión y volver a selección de rol (VentanaPrincipal).	Botón "Cerrar Sesión"	Se destruye `VentanaAdmin`, se instancia `VentanaPrincipal`.

8. Validación de Entradas (Actualizada)

Se describen las validaciones implementadas tanto en lógica de negocio como en la capa de presentación (GUI):

1. **Validación Numérica y Rangos**
    - Campos de altura y peso en "Crear Atracción (Mecánica)":

- Usan `JSpinner` o `JFormattedTextField` con formato numérico; se valida que altura/peso estén dentro de rangos mínimos (por ej., altura >0).

- Fecha de temporada:

  - Se usa `JDatePicker` (componente de Swing) que impide caracteres inválidos; se verifica en código que `fechaFin  fechaInicio`.

- Edad mínima (`AtraccionCultural`):

  - `JSpinner` configurado con valores enteros 0.

2. **Prevención de Duplicados**

   - **Registro de Cliente**:

     - En `VentanaCliente.registrarseClienteGUI()`, antes de anexar a `clientes.csv`, se llama a un método que lee todo el CSV y verifica si el `login` ya existe; en caso afirmativo, muestra `JOptionPane.showMessageDialog(..., ‘‘Login ya existe’’)` y aborta.

   - **Registro de Empleado**:

     - Similar: checa `empleados.csv` por `login` duplicado.

3. **Autenticación**

   - **Admin**:

     - `ComboBox` con credenciales cargadas desde `auth_admin.csv`.
     - Se validan que login y password coincidan con algún registro (sin cifrado todavía).
     - En caso de fallo, se muestra diálogo de error.

   - **Empleado**:

     - Se valida login/contraseña contra `empleados.csv` (lectura en memoria).
     - Si falla, `JOptionPane.showMessageDialog(..., ‘‘Credenciales inválidas’’)`.

   - **Cliente**:

     - Igual: se lee `clientes.csv` y se compara login/contraseña; en caso de éxito, habilita el resto de pestañas.

4. **Control de Turnos y Asignaciones**

- Al asignar turno a Empleado, se verifica que no exista intersección de horarios (si el empleado ya tiene turno asignado en la misma fecha y turno).
- Si hay conflicto, se muestra mensaje de alerta y no permite guardar.

5. **Validación de Disponibilidad de Atracciones**
   - En la lógica de `Atraccion`:
     - Método `estaDisponible(Temporada t, String climaActual)` se asegura que la fecha actual esté dentro de `t` y que `disponibleClima=true` si el clima es apto.
   - En GUI, al mostrar lista de atracciones, se colorean (por ejemplo, filas rojas si no están disponibles).

9. Pruebas de Integración (GUI + Dominio)

Además de las pruebas unitarias (descritas en la sección **VI**), se desarrollaron pruebas de integración mínimas que simulan flujos completos del usuario, combinando interfaz gráfica y lógica de dominio/persistencia. Algunos ejemplos:

1. **Flujo Cliente Completo**
   - Se simula el registro de cliente en GUI (`VentanaCliente.registrarseClienteGUI()`), se cierra la ventana, luego se abre GUI de login y se autentica.
   - Se elige "Comprar Tiquete", se selecciona "Oro" con FastPass, se confirma.
   - Se valida que en `tiquetes.csv` exista la línea correspondiente.
   - Se abre "Mis Tiquetes", se selecciona el tiquete recién comprado y se presiona "Usar Tiquete".
   - Se verifica que `estado` de ese tiquete cambió a "usado" en el CSV.

2. **Flujo Empleado Completo**
   - Desde `VentanaAsistenciaEmpleado`, se inicia sesión con un Empleado (ej.: Cajero).
   - Se marca asistencia de un cliente (login preexistente o recién creado).
   - Se abre "Historial de Cliente" y se verifica que la asistencia quedó registrada con fecha y hora correctas.
   - Se abre la pestaña "Mis Turnos" y se comprueba que aparecen los turnos asignados.

3. **Flujo Administrador Completo**
   - Se inicia sesión desde `VentanaAdmin` con credenciales válidas (`auth_admin.csv`).

- Se crea una nueva atracción mecánica ("Montaña Rusa Extrema"), con restricciones (altura mínima, peso mínimo).
- Se edita esa atracción cambiando "Nivel de Riesgo" de "Alto" a "Medio".
- Se crea un empleado "Operador Mecánico" con login "opMech1" y se le asigna un turno en la nueva atracción.
- Se genera "Reporte Ventas" (aunque no haya ventas); la tabla aparece vacía.
- Se genera "Reporte Asistencias" (tabla vacía si no hay registros).

Las pruebas de integración se realizaron manualmente (QA), validando que las ventanas respondieran sin errores y que los datos realmente quedaran persistidos.

10. Estructura del Proyecto y Archivos (Actualizada)

A continuación se presenta la estructura de carpetas y archivos del proyecto Eclipse (o IDE equivalente), remarcando la nueva carpeta `vista`:

```
Proyecto2_ParqueAtracciones/

 datos/
    auth_admin.csv
    clientes.csv
    empleados.csv
    atracciones.csv
    tiquetes.csv
    ventas_taquilla.csv
    ventas_online.csv
    asistencias_clientes.csv

 src/
    interfaz/             ← Clases de arranque por consola (legado)
       AdminMain.java
       ClienteMain.java
       EmpleadoMain.java
       InterfazAdmin.java
       InterfazCliente.java
       InterfazEmpleado.java

    persona/             ← Clases de usuario y roles
       Usuario.java
       Cliente.java
       Empleado.java
       Cajero.java
       Cocinero.java
```

```
      OperadorMecanico.java
      ServicioGeneral.java
      Administrador.java
      Turno.java
      LugarTrabajo.java

   atracciones/         ← Clases de atracciones y espectáculos
      Atraccion.java
      AtraccionMecanica.java
      AtraccionCultural.java
      Espectaculo.java
      Ubicacion.java
      Temporada.java

   tiquetes/            ← Clases de tiquetes y ventas
      Tiquete.java
      TiqueteBasico.java
      TiqueteFamiliar.java
      TiqueteOro.java
      TiqueteDiamante.java
      TiqueteTemporada.java
      EntradaIndividual.java
      VentaOnline.java
      Taquilla.java

   persistencia/        ← I/O de archivos CSV
      ArchivoPlano.java

   vista/               ← NUEVO: Interfaces gráficas Swing
      VentanaPrincipal.java
      VentanaAdmin.java
      VentanaCliente.java
      VentanaAsistenciaEmpleado.java

   pruebas/             ← Pruebas unitarias con JUnit
       TestAtraccionCultural.java
       TestAtraccionMecanica.java
       TestCliente.java
       TestEspectaculo.java
       TestTaquilla.java
       TestVentaOnline.java
       TestArchivoPlano.java

.gitignore
README.txt               ← Instrucciones de ejecución
Proyecto2 ... .project (archivos de configuración del IDE)
```

10.1. Instrucciones para ejecutar la aplicación con GUI

1. **Importar el proyecto**
   - Abrir Eclipse o IntelliJ IDEA, importar como "Existing Java Project". Verificar que la carpeta `datos/` esté en la raíz del proyecto.
2. **Compilar todo**
   - Asegurarse de que no haya errores de compilación.
3. **Ejecutar `VentanaPrincipal.java`**
   - Seleccionar archivo `VentanaPrincipal.java` → botón derecho "Run as" → "Java Application".
   - Aparecerá la ventana inicial con botones para rol.
4. **Seleccionar rol y navegar**
   - Hacer clic en "Administrador" → se abrirá `VentanaAdmin`.
   - Hacer clic en "Empleado" → se abrirá `VentanaAsistenciaEmpleado`.
   - Hacer clic en "Cliente" → se abrirá `VentanaCliente`.
   - Seguir los flujos correspondientes.
5. **Archivos CSV**
   - Todas las modificaciones se reflejarán automáticamente en los archivos CSV de la carpeta `datos/`.

Nota: Para ejecutar la versión de consola (legado), bastará con correr los `AdminMain.java`, `EmpleadoMain.java` o `ClienteMain.java`, que a su vez invocan métodos de las clases del paquete `interfaz`.

11. Conclusiones y Trabajo Futuro (Actualizado)

11.1. Conclusiones

- Se logró migrar la versión de consola a una **interfaz gráfica amigable** basándose en Java Swing, sin modificar la lógica central de negocio ni la estructura de persistencia.
- La arquitectura modular permitió integrar la nueva capa `vista` de forma transparente, respetando los principios SOLID y manteniendo alta cohesión y bajo acoplamiento entre módulos.
- Las pruebas automatizadas (JUnit) validan tanto la lógica de dominio como elementos básicos de la GUI, lo que garantiza estabilidad ante posibles cambios.
- La persistencia en CSV, aunque simple, demostró ser eficaz para esta fase; la definición de un único punto de lectura/escritura (`ArchivoPlano`) facilita futuras migraciones a bases de datos más robustas.

11.2. Trabajo Futuro

1. **Autenticación más robusta**
   - Implementar cifrado (hash SHA-256) para contraseñas en CSV o migrar a un sistema de autenticación centralizado (Base de Datos relacional).
   - Control de sesiones para evitar accesos concurrentes con credenciales comprometidas.
2. **Mejoras en la GUI**
   - Implementar diseño responsivo o usar JavaFX para interfaces más modernas.
   - Añadir gráficos y métricas dinámicas en "Generar Reportes" (por ejemplo, gráficas de barras con las ventas semanales).
   - Agregar validaciones en tiempo real (por ejemplo, mostrar alerta al ingresar caracteres inválidos).
3. **Migración a Base de Datos**
   - Reemplazar archivos CSV por una base de datos relacional (PostgreSQL, MySQL).
   - Utilizar JDBC o un ORM (Hibernate) para manejar la persistencia, evitando problemas de concurrencia y escalabilidad.
4. **API REST para Ventas Online**
   - Exponer servicios web (Spring Boot o JAX-RS) que permitan a terceros (aplicaciones móviles, portales web) consumir datos de atracciones, comprar tiquetes, chequear disponibilidad.
   - Separación completa de la capa de presentación de la lógica de negocio para permitir múltiples frontends (web, móvil).
5. **Internacionalización (i18n)**
   - Preparar la aplicación para mostrar textos en múltiples idiomas (Español/Inglés), valiéndose de archivos `.properties`.
6. **Pruebas de Interfaz Automáticas**
   - Incorporar herramientas de testing como FEST o AssertJ-Swing para validar flujos GUI de forma automática.

12. Índice de Contenido (Actualizado)