INGENIERIA DE SISTEMAS Y COMPUTACION

PROYECTO #3

PARQUE DE ATRACCIONES DE LOS ALPES

PROYECTO #3

PARQUE DE ATRACCIONES DE LOS ALPES

DISEÑO Y PROGRAMACIÓN ORIENTADA A OBJETOS

GRUPO #8

Manual de Usuario – Sistema de Gestión de Parque de Diversiones

¿Qué es esta aplicación?

Esta es una aplicación de escritorio que permite gestionar un parque de diversiones de manera completa. El sistema maneja tres tipos de usuarios diferentes: **Clientes**, **Trabajadores** y **Administradores**, cada uno con funcionalidades específicas.

Cómo empezar

1. Ejecutar la aplicación

- Ejecuta el archivo Java principal.
- Se abrirá una ventana con la pantalla de inicio de sesión.

2. Primera vez usando el sistema

Como es tu primera vez, necesitarás crear una cuenta o usar las credenciales de administrador predeterminadas.

Tipos de Usuario

ADMINISTRADOR (Acceso completo)

Credenciales predeterminadas:

- Usuario: admin
- Contraseña: admin

CLIENTE (Visitante del parque)

- Puede comprar tiquetes.
- Ver su historial de compras.

TRABAJADOR (Empleado del parque)

- Ve la atracción asignada a él.
- Información limitada según su rol.

Guía paso a paso

REGISTRARSE (Primera vez)

1. Hacer clic en "Registrarse" en la pantalla de login.

2. Seleccionar el tipo de cuenta:
    - Cliente: Para visitantes del parque.
    - Trabajador: Para empleados.
3. Llenar la información requerida:

    Para CLIENTES:

    - Nombre completo.
    - Usuario (login).
    - Contraseña.
    - Riesgos de salud (alergias, condiciones médicas).
    - Altura en metros (ej: 1.75).
    - Peso en kilogramos (ej: 70).

    Para TRABAJADORES:

    - Nombre completo.
    - Usuario (login).
    - Contraseña.
    - Cargo (ej: Operador, Supervisor, Mantenimiento).
4. Hacer clic en "Crear Cuenta".
5. El sistema te asignará un ID único automáticamente.
6. Volver al login para ingresar con tus nuevas credenciales.

PANEL DEL CLIENTE

Comprar Tiquete

1. Hacer clic en "Comprar Tiquete".
2. Seleccionar tipo de tiquete:
    - BÁSICO: $200 - Acceso estándar.
    - FAMILIAR: $50 - Para familias.
    - ORO: $100 - Acceso premium.
    - DIAMANTE: $150 - Acceso VIP.
3. Confirmar la compra con "Sí".
4. El tiquete se agrega automáticamente a tu historial.

Ver Mis Tiquetes

- Muestra todos los tiquetes que has comprado.
- Incluye tipo, ID y precio pagado.

PANEL DEL TRABAJADOR

Ver Mi Atracción Asignada

- Única funcionalidad disponible para trabajadores.
- Muestra información de la atracción asignada:
    - ID de la atracción.
    - Nombre.
    - Capacidad máxima.
    - Ubicación en el parque.
    - Nivel de exclusividad.

Nota: Si no tienes atracción asignada, aparecerá un mensaje indicándolo. Solo el administrador puede hacer asignaciones.

PANEL DEL ADMINISTRADOR

Gestión de Personas

Ver Clientes

- Lista completa de todos los clientes registrados.
- Muestra ID y nombre de cada cliente.

Ver Trabajadores

- Lista completa de todos los empleados.
- Muestra ID y nombre de cada trabajador.

Agregar Cliente/Trabajador

- Te lleva al formulario de registro.
- Mismo proceso que registrarse desde el login.

Eliminar Trabajador

1. Hacer clic en "Eliminar Trabajador".
2. Seleccionar el trabajador de la lista desplegable.
3. Confirmar eliminación con "Sí".
4. El trabajador y su asignación se eliminan permanentemente.

Gestión de Atracciones

Ver Atracciones

- Lista todas las atracciones del parque.
- Muestra qué trabajadores están asignados a cada una.
- Si una atracción no tiene asignados, aparece "Sin asignar".

Agregar Atracción

Puedes crear dos tipos:

ATRACCIÓN MECÁNICA:

- Nombre de la atracción.
- Capacidad (número de personas).
- Ubicación en el parque.
- Nivel de exclusividad.
- Altura mínima y máxima permitida.
- Peso mínimo y máximo permitido.
- Nivel de riesgo.
- Contraindicaciones médicas.

ATRACCIÓN CULTURAL:

- Nombre de la atracción.
- Capacidad (número de personas).
- Ubicación en el parque.
- Nivel de exclusividad.
- Edad mínima requerida.
- Descripción de la actividad.

Asignar Empleado a Atracción

1. Seleccionar la atracción de la lista desplegable.
2. Seleccionar el empleado de la lista desplegable.
3. La asignación se guarda automáticamente.
4. El empleado ahora podrá ver esta atracción en su panel.

Persistencia de Datos

¿Qué se guarda automáticamente?

- Clientes registrados: Toda su información y historial de compras.
- Trabajadores: Sus datos personales y de trabajo.
- Atracciones: Todas las atracciones creadas.
- Tiquetes: Todos los tiquetes vendidos.
- Asignaciones: Qué trabajador está en cada atracción.

¿Dónde se guardan los datos?

Los datos se almacenan en una carpeta llamada data/ que se crea automáticamente:

- clientes.dat – Información de clientes.
- empleados.dat – Información de trabajadores.
- atracciones.dat – Información de atracciones.
- tiquetes.dat – Información de tiquetes.
- asignaciones.dat – Asignaciones empleado–atracción.

Flujo de Trabajo Recomendado

Para empezar desde cero:

1. Entrar como administrador (admin/admin).
2. Crear atracciones del parque.
3. Registrar trabajadores desde el panel de admin.
4. Asignar trabajadores a atracciones.
5. Los clientes pueden registrarse por su cuenta.
6. Los clientes pueden comprar tiquetes.

Uso diario:

1. Trabajadores: Inician sesión para ver su atracción asignada.
2. Clientes: Compran tiquetes y revisan su historial.
3. Administrador: Gestiona el personal y las atracciones según sea necesario.

Mensajes de Error Comunes

Error	Significado	Solución
"Credenciales inválidas"	Usuario o contraseña incorrectos	Verificar datos o registrarse
"No tienes ninguna atracción asignada"	El trabajador no tiene asignación	El admin debe asignar una atracción
"No hay atracciones disponibles"	No existen atracciones creadas	El admin debe crear atracciones primero
"Error en registro"	Datos incompletos o inválidos	Verificar que todos los campos estén llenos correctamente

Consejos y Buenas Prácticas

Recomendaciones:

- Siempre confirma antes de eliminar trabajadores.
- Guarda los IDs que se generan al registrar usuarios.
- Crea atracciones antes de registrar trabajadores.
- Usa nombres descriptivos para atracciones y usuarios.

Funciones rápidas:

- ESC o cancelar en cualquier diálogo para volver atrás.
- Los datos se guardan automáticamente en cada operación.
- No es necesario "guardar" manualmente.

Solución de Problemas

La aplicación no guarda datos:

- Verificar permisos de escritura en la carpeta donde está la aplicación.
- Asegurarse de que existe la carpeta data/.

No puedo ver mi atracción asignada:

- Verificar que un administrador te haya asignado una atracción.
- Confirmar que la atracción aún existe en el sistema.

Olvidé mi contraseña:

- Solo el administrador puede ver las listas de usuarios.
- Registrar una nueva cuenta si es necesario.

Soporte

Para problemas técnicos o dudas adicionales:

- Revisar los mensajes de error en pantalla.
- Verificar que todos los campos estén completos.
- Contactar al administrador del sistema para problemas de acceso.

Tabla de Aplicaciones:

Paquete: modelo.atraccion

Función	Clase	Descripción
validarAcceso()	Atraccion (abstracta)	Método abstracto que valida si un usuario puede acceder a la atracción.
toJson()	Atraccion	Serializa la atracción en formato JSON.
fromJson(String json)	Atraccion	Deserializa una cadena JSON en una atracción.
programarMantenimiento()	Atraccion	Programa una fecha para mantenimiento y cambia su estado.
getTipoAtraccion()	AtraccionCultural, AtraccionMecanica	Devuelve el tipo de atracción.
toJsonEspecifico()	AtraccionCultural, AtraccionMecanica	Agrega propiedades específicas al JSON.
fromJson()	AtraccionCultural, AtraccionMecanica	Crea una instancia desde JSON.
validarAcceso()	AtraccionCultural	Verifica que el usuario tenga edad suficiente.

Función	Clase	Descripción
validarAcceso()	AtraccionMecanica	Verifica que la altura y el peso estén dentro de los límites.

Paquete: modelo.empleados

Función	Clase	Descripción
mostrarInformacion()	Cliente, Empleado, EmpleadoServicio, EmpleadoAtraccion	Muestra información específica del usuario.
comprarTiquete()	Cliente	Agrega una compra al historial del cliente.
consultarHistorial()	Cliente	Muestra el historial de compras.
verificarTiquete()	EmpleadoAtraccion	Verifica si el tiquete ingresado es válido.
controlarAtraccion()	EmpleadoAtraccion	Simula que el empleado está operando una atracción.
vender() / generarFactura()	EmpleadoServicio	Simulan venta de productos y generación de factura.
asignarTurno()	Empleado	Asigna un turno a un empleado disponible.
registrarAsistencia()	Empleado	Registra la asistencia del empleado.
toJson() / fromJson()	Todos los usuarios	Serializa/deserializa los datos del usuario en JSON.

Paquete: modelo.main

Función	Clase	Descripción
main()	ParqueSwingApp	Punto de entrada del programa; inicia la interfaz gráfica (GUI).
ParqueSwingApp()	ParqueSwingApp	Constructor que carga datos, inicializa variables y configura la ventana.
initUI()	ParqueSwingApp	Inicializa la interfaz gráfica con los paneles principales.
createLoginPanel()	ParqueSwingApp	Crea el panel de inicio de sesión.
createRegisterPanel()	ParqueSwingApp	Crea el panel de registro de usuarios (clientes o trabajadores).
createClientPanel()	ParqueSwingApp	Crea el panel principal del cliente.
createEmployeePanel()	ParqueSwingApp	Crea el panel principal del trabajador.
createAdminPanel()	ParqueSwingApp	Crea el panel principal del administrador.
makeButton(String, ActionListener)	ParqueSwingApp	Crea y retorna un botón con texto y acción asociada.
showList(String, List<T>)	ParqueSwingApp	Muestra una lista de objetos (clientes o empleados) en una ventana emergente.

Función	Clase	Descripción
onLogin(ActionEvent)	ParqueSwingApp	Maneja el proceso de autenticación según las credenciales ingresadas.
onVerAtraccionAsignada()	ParqueSwingApp	Muestra la atracción asignada al empleado actual.
onClientComprar()	ParqueSwingApp	Permite al cliente actual comprar un tiquete.
onClientVerTiquetes()	ParqueSwingApp	Muestra el historial de compras del cliente.
onAgregarCliente()	ParqueSwingApp	Redirige al panel de registro para agregar un cliente.
onRemoveCliente()	ParqueSwingApp	Permite eliminar un cliente existente del sistema.
onAgregarTrabajador()	ParqueSwingApp	Redirige al panel de registro para agregar un trabajador.
onRemoveTrabajador()	ParqueSwingApp	Permite eliminar un trabajador existente del sistema.
onAddAtraccion()	ParqueSwingApp	Permite agregar una nueva atracción (mecánica o cultural).
onAssignEmpleado()	ParqueSwingApp	Asigna un trabajador disponible a una atracción seleccionada.
onAdminVerAtracciones()	ParqueSwingApp	Muestra la lista de atracciones con sus empleados asignados.
guardarAsignaciones()	ParqueSwingApp	Guarda en archivo las asignaciones trabajador–atracción.
cargarAsignaciones()	ParqueSwingApp	Carga las asignaciones trabajador–atracción desde archivo.
onSalir()	ParqueSwingApp	Guarda todos los datos y cierra la aplicación.

Paquete: modelo.persistencia

Función	Clase	Descripción
guardarArchivo()	JSONPersistenceManager	Guarda una cadena de texto en archivo JSON.
cargarArchivo()	JSONPersistenceManager	Carga una cadena de texto desde un archivo JSON.
guardarListaClientes()	PersistenceClientes	Guarda la lista de clientes en archivo.
guardarListaEmpleados()	PersistenceEmpleados	Guarda la lista de empleados en archivo.
guardarListaAtracciones()	PersistenceAtracciones	Guarda la lista de atracciones en archivo.
guardarListaTiquetes()	PersistenceTiquetes	Guarda la lista de tiquetes vendidos.
cargarListaClientes()	PersistenceClientes	Carga clientes desde archivo.
cargarListaEmpleados()	PersistenceEmpleados	Carga empleados desde archivo.
cargarListaAtracciones()	PersistenceAtracciones	Carga atracciones desde archivo.
cargarListaTiquetes()	PersistenceTiquetes	Carga tiquetes desde archivo.

Paquete: modelo.tiquetes

Función	Clase	Descripción
toJson()	TiquetesNormales	Serializa el tiquete a formato JSON.
getId()	TiquetesNormales	Devuelve el ID único del tiquete.
getTipo()	TiquetesNormales	Devuelve el tipo de tiquete (BÁSICO, ORO, FAMILIAR, DIAMANTE).
getAtractivos()	TiquetesNormales	Lista de atracciones asociadas al tiquete.

Paquete: pruebas

Nombre del Test	Componente	Descripción de la Prueba
testCrearClienteConDatosValidos()	Cliente	Verifica que un cliente creado con datos válidos guarde correctamente su información.
testHistorialComprasCliente()	Cliente	Prueba que se registre correctamente el historial de compras del cliente.
testDatosClienteCorrectos()	Cliente	Verifica que los atributos del cliente se guarden correctamente tras su creación.
testClienteMultiplesCompras()	Cliente	Asegura que el cliente puede tener múltiples compras registradas en su historial.
testCrearEmpleado()	Empleado	Verifica que un empleado se cree con nombre y cargo correctos.
testDatosEmpleadoCorrectos()	Empleado	Verifica los datos completos de un empleado después de su creación.
testTiqueteUsado()	TiquetesNormales	Valida que un tiquete pueda marcarse como utilizado.
testTiqueteNuevoNoUsado()	TiquetesNormales	Confirma que un tiquete recién creado no está marcado como utilizado.
testPropiedadesTiquete()	TiquetesNormales	Verifica que las propiedades del tiquete sean correctas (ID, tipo, usos, atracciones).
testCambioEstadoTiquete()	TiquetesNormales	Prueba que un tiquete pueda cambiar entre los

Nombre del Test	Componente	Descripción de la Prueba
		estados "usado" y "no usado".
testValidarAccesoMecanicaPermitido()	AtraccionMecanica	Verifica que un usuario con altura y peso válidos tenga acceso permitido a la atracción.
testValidarAccesoMecanicaDenegadoPorAltura()	AtraccionMecanica	Prueba que el acceso se deniegue si la altura del usuario es menor a la mínima permitida.
testValidarAccesoMecanicaDenegadoPorPesoBajo()	AtraccionMecanica	Prueba que el acceso se deniegue si el peso del usuario es menor al mínimo permitido.
testValidarAccesoMecanicaDenegadoPorPesoAlto()	AtraccionMecanica	Prueba que el acceso se deniegue si el peso del usuario es mayor al máximo permitido.
testPropiedadesAtraccionMecanica()	AtraccionMecanica	Verifica que todas las propiedades de una atracción mecánica se asignen correctamente.
testValidarAccesoConParametrosNulos()	Casos Límite	Asegura que no haya acceso si se pasa un objeto null como parámetros.
testValidarAccesoConParametrosIncompletos()	Casos Límite	Verifica que se deniegue el acceso si los parámetros de entrada están incompletos.

Implementación de Códigos QR con Zxing y con QRGen y Resultados del Proyecto

En el desarrollo del sistema de tiquetes para el parque de atracciones, se planeó implementar la generación dinámica de códigos QR utilizando la librería ZXing (Zebra Crossing). Esta funcionalidad tenía como objetivo crear códigos QR únicos para cada tiquete, conteniendo información esencial como el ID del tiquete, tipo y fecha de expedición. Con QRGen tambien lo intentamos, en ambos descargamos el Maven que supuestamente nos ayudaba, Tambien librerias de Zxing y den Qrgen que eran archivos .jar
A pesar de los esfuerzos realizados, no se logró implementar completamente esta funcionalidad debido a proble mas técnicos con la integración de la librería ZXing y el QRGenen el entorno de desarrollo. Se identificaron dific ultades en la configuración de las dependencias y en la generación correcta de las imágenes QR que fueran con sistentemente legibles por dispositivos móviles.

Codigo que intentamos:
QRGen:

```
package modelo.main;

import modelo.empleados.Cliente;
```

```
import modelo.empleados.Empleado;
import modelo.atraccion.Atraccion;
import modelo.atraccion.AtraccionMecanica;
import modelo.atraccion.AtraccionCultural;
import modelo.tiquetes.TiquetesNormales;
import modelo.tiquetes.Tipo;
import modelo.persistencia.PersistenceClientes;
import modelo.persistencia.PersistenceEmpleados;
import modelo.persistencia.PersistenceAtracciones;
import modelo.persistencia.PersistenceTiquetes;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import java.util.Random;
import java.io.*;

import com.google.zxing.BarcodeFormat;
import com.google.zxing.WriterException;
import com.google.zxing.client.j2se.MatrixToImageWriter;
import com.google.zxing.common.BitMatrix;
import com.google.zxing.qrcode.QRCodeWriter;

public class ParqueSwingApp extends JFrame {
    // [Resto del código existente permanece igual hasta la clase QRGenerator...]

    // ========== NUEVA CLASE INTERNA PARA GENERACIÓN DE QR ==========
    private static class QRGenerator {
        public static BufferedImage generateQRCodeImage(String text, int width, int height)
            throws WriterException {
            QRCodeWriter qrCodeWriter = new QRCodeWriter();
            BitMatrix bitMatrix = qrCodeWriter.encode(text, BarcodeFormat.QR_CODE, width, height);
            return MatrixToImageWriter.toBufferedImage(bitMatrix);
        }
    }

    // ========== MÉTODO MODIFICADO PARA MOSTRAR TIQUETES CON QR ==========
    private void mostrarTiquete(TiquetesNormales tiquete) {
        JDialog dialog = new JDialog(this, "Tiquete " + tiquete.getId(), true);
        dialog.setLayout(new BorderLayout(10, 10));
        dialog.setSize(400, 500);
        dialog.setLocationRelativeTo(this);

        // Panel principal
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        // Encabezado
        JLabel titulo = new JLabel("Tiquete " + tiquete.getTipo(), JLabel.CENTER);
        titulo.setFont(new Font("Arial", Font.BOLD, 20));
        titulo.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(titulo);

        // Información del tiquete
        panel.add(Box.createVerticalStrut(15));
        JLabel idLabel = new JLabel("ID: " + tiquete.getId());
        idLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(idLabel);

        JLabel tipoLabel = new JLabel("Tipo: " + tiquete.getTipo());
        tipoLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(tipoLabel);

        String fechaFormateada = tiquete.getFechaExpedicion().format(DateTimeFormatter.ofPattern("dd-MM-yyyy"));
        JLabel fechaLabel = new JLabel("Fecha: " + fechaFormateada);
        fechaLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(fechaLabel);

        // Generar QR
        panel.add(Box.createVerticalStrut(20));
```

```
        JLabel qrLabel = new JLabel();
        qrLabel.setAlignmentX(Component.CENTER_ALIGNMENT);

        try {
            String qrText = String.format("ID:%s,Tipo:%s,Fecha:%s",
                tiquete.getId(), tiquete.getTipo(), fechaFormateada);
            BufferedImage qrImage = QRGenerator.generateQRCodeImage(qrText, 250, 250);
            qrLabel.setIcon(new ImageIcon(qrImage));
        } catch (WriterException e) {
            qrLabel.setText("Error generando QR");
            e.printStackTrace();
        }
        panel.add(qrLabel);

        // Panel de botones
        panel.add(Box.createVerticalStrut(20));
        JPanel buttonPanel = new JPanel();

        if (tiquete.isImpreso()) {
            int confirm = JOptionPane.showConfirmDialog(
                this,
                "Este tiquete ya fue impreso. ¿Desea imprimirlo de nuevo?",
                "Confirmar reimpresión",
                JOptionPane.YES_NO_OPTION
            );
            if (confirm != JOptionPane.YES_OPTION) {
                return;
            }
        }

        JButton imprimirBtn = new JButton("Imprimir");
        imprimirBtn.addActionListener(e -> {
            tiquete.marcarComoImpreso();
            persTiquetes.guardarListaTiquetes(tiquetes);
            JOptionPane.showMessageDialog(dialog, "Tiquete impreso correctamente");
        });

        JButton cerrarBtn = new JButton("Cerrar");
        cerrarBtn.addActionListener(e -> dialog.dispose());

        buttonPanel.add(imprimirBtn);
        buttonPanel.add(cerrarBtn);
        buttonPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
        panel.add(buttonPanel);

        dialog.add(panel, BorderLayout.CENTER);
        dialog.setVisible(true);
    }

    // [Resto del código existente permanece igual...]
}
```

ZXWing:

```
package modelo.main;

import modelo.empleados.Cliente;
import modelo.empleados.Empleado;
import modelo.atraccion.Atraccion;
import modelo.atraccion.AtraccionMecanica;
import modelo.atraccion.AtraccionCultural;
import modelo.tiquetes.TiquetesNormales;
import modelo.tiquetes.Tipo;
import modelo.persistencia.PersistenceClientes;
import modelo.persistencia.PersistenceEmpleados;
import modelo.persistencia.PersistenceAtracciones;
import modelo.persistencia.PersistenceTiquetes;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.util.*;
import java.io.*;

// NUEVOS IMPORTS para QR:
import com.google.zxing.*;
import com.google.zxing.common.BitMatrix;
```

```
import com.google.zxing.BarcodeFormat;
import com.google.zxing.MultiFormatWriter;
import com.google.zxing.client.j2se.MatrixToImageWriter;

public class ParqueSwingApp extends JFrame {
    private List<Cliente> clientes;
    private List<Empleado> empleados;
    private List<Atraccion> atracciones;
    private List<TiquetesNormales> tiquetes;
    private Map<Integer,Integer> asignaciones = new HashMap<>();

    private PersistenceClientes persClientes       = new PersistenceClientes("data");
    private PersistenceEmpleados persEmpleados     = new PersistenceEmpleados("data");
    private PersistenceAtracciones persAtracciones = new PersistenceAtracciones("data");
    private PersistenceTiquetes persTiquetes       = new PersistenceTiquetes("data");

    private CardLayout cardLayout;
    private JPanel cards;
    private JTextField loginField;
    private JPasswordField passField;
    private int currentUserId;
    private String currentRole;
    private Random random = new Random();

    public ParqueSwingApp() {
        super("Parque de Diversiones");
        clientes    = persClientes.cargarListaClientes();
        empleados   = new ArrayList<>(persEmpleados.cargarListaEmpleados());
        atracciones = persAtracciones.cargarListaAtracciones();
        tiquetes    = persTiquetes.cargarListaTiquetes();
        cargarAsignaciones();
        initUI();
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(550, 500);
        setLocationRelativeTo(null);
        setVisible(true);
    }

    // MÉTODO NUEVO para generar QR
    private BufferedImage generarQR(String data, int width, int height) throws WriterException {
        BitMatrix matrix = new MultiFormatWriter().encode(data, BarcodeFormat.QR_CODE, width, height);
        return MatrixToImageWriter.toBufferedImage(matrix);
    }

    // MODIFICADO: al final de este método, añadimos generación y visualización del QR
    private void onClientComprar(){
        try {
            Cliente cli = clientes.stream()
                .filter(c -> c.getId() == currentUserId)
                .findFirst()
                .orElse(null);
            if (cli == null) return;

            Tipo[] tipos = Tipo.values();
            Tipo tp = (Tipo) JOptionPane.showInputDialog(
                this, "Seleccione tipo de tiquete:",
                "Comprar Tiquete", JOptionPane.QUESTION_MESSAGE,
                null, tipos, tipos[0]
            );
            if (tp == null) return;

            float precio = switch(tp) {
                case FAMILIAR -> 50f;
                case ORO      -> 100f;
                case DIAMANTE -> 150f;
                case BASICO   -> 200f;
            };

            int resp = JOptionPane.showConfirmDialog(
                this, String.format("El precio es $%.2f. ¿Desea confirmar la compra?", precio),
                "Confirmar Compra", JOptionPane.YES_NO_OPTION
            );
            if (resp != JOptionPane.YES_OPTION) return;

            TiquetesNormales t = new TiquetesNormales(
                "T" +  System.currentTimeMillis(), tp, new ArrayList<>(), 0
            );
            tiquetes.add(t);
            cli.comprarTiquete(String.format("Tiquete %s tipo %s comprado a $%.2f", t.getId(), tp, precio));
```

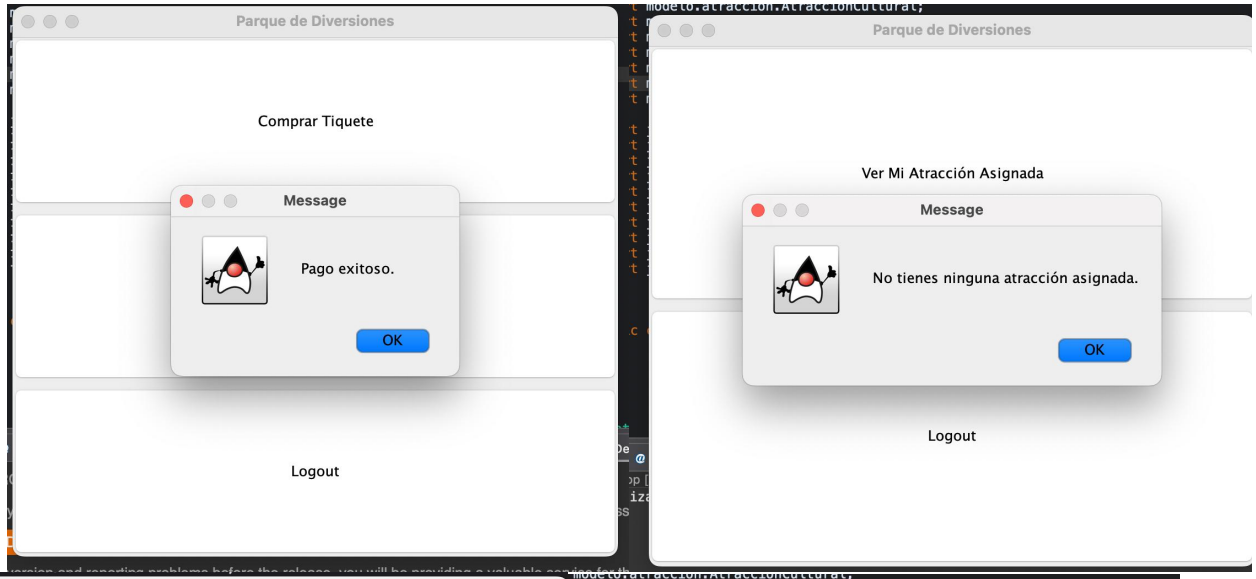
```
            persTiquetes.guardarListaTiquetes(tiquetes);
            persClientes.guardarListaClientes(clientes);

            JOptionPane.showMessageDialog(this, "Compra realizada correctamente.");

            // NUEVO BLOQUE: generar y mostrar QR
            String msgQR = "ID: " + t.getId() + "\nTipo: " + t.getTipo() +
                           "\nFecha: " + java.time.LocalDateTime.now();

            BufferedImage qrImage = generarQR(msgQR, 150, 150);

            JPanel panel = new JPanel(new BorderLayout());
            JTextArea area = new JTextArea(msgQR);
            area.setEditable(false);
            JLabel qrLabel = new JLabel(new ImageIcon(qrImage));
            panel.add(area, BorderLayout.CENTER);
            panel.add(qrLabel, BorderLayout.EAST);

            JOptionPane.showMessageDialog(this, panel, "Tiquete QR", JOptionPane.INFORMATION_MESSAGE);

        } catch(Exception ex) {
            JOptionPane.showMessageDialog(this, "Error al procesar la compra o generar QR");
            ex.printStackTrace();
        }
    }

    // ... (todo el resto del código permanece igual, como métodos de login, paneles, administración, etc.)

    public static void main(String[] args){ SwingUtilities.invokeLater(ParqueSwingApp::new); }
}
```

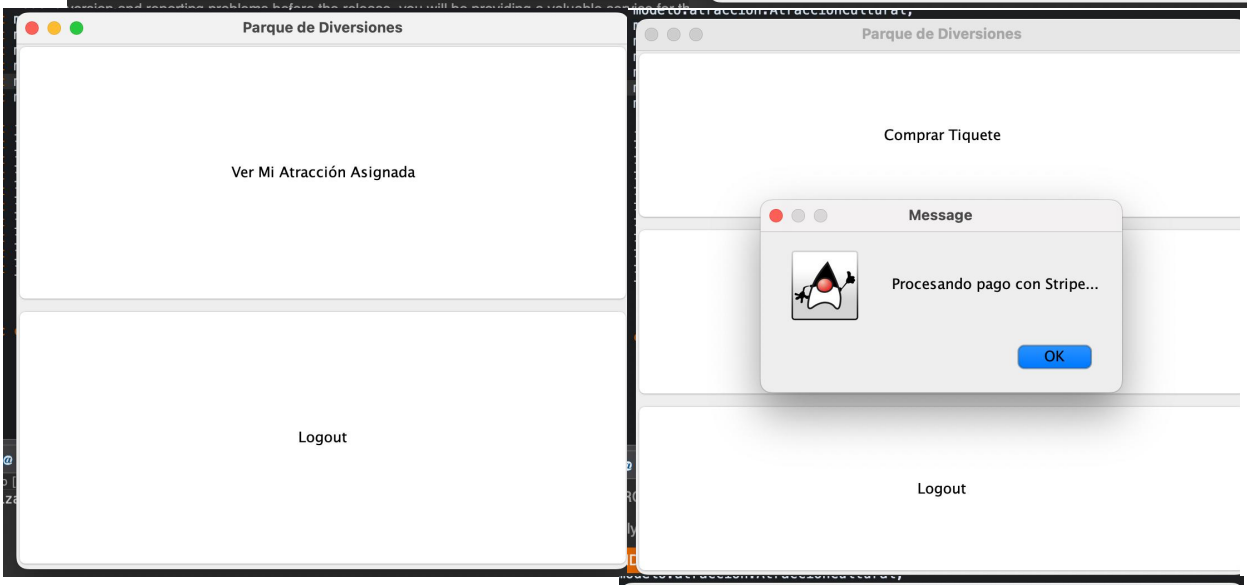
Tambien se Hizo un pom.xml:

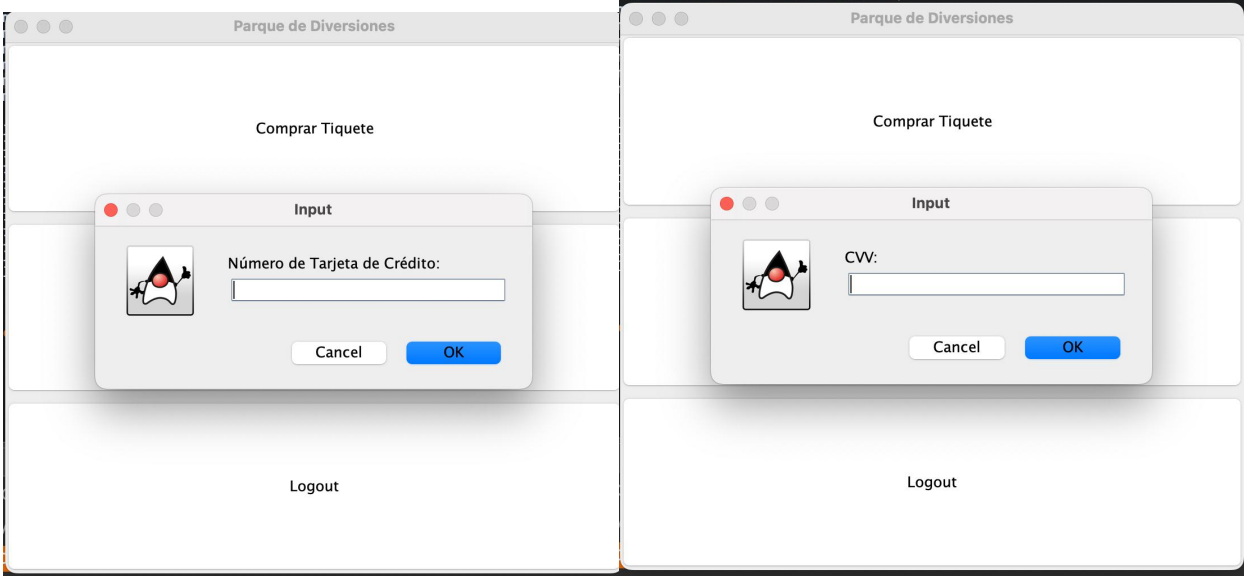
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ejemplo</groupId>
    <artifactId>parque-diversiones</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source> <!-- o 1.8 si usas Java 8 -->
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- QRGen (incluye ZXing internamente) -->
        <dependency>
            <groupId>net.glxn</groupId>
            <artifactId>qrgen</artifactId>
            <version>2.6.0</version>
        </dependency>

        <!-- ZXing JavaSE (por si usas BufferedImage directamente) -->
        <dependency>
```

```
            <groupId>com.google.zxing</groupId>
            <artifactId>javase</artifactId>
            <version>3.3.0</version>
        </dependency>

        <!-- Swing ya viene con Java, no es necesario incluirlo -->
    </dependencies>

    <build>
        <plugins>
            <!-- Plugin de compilación estándar -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>

            <!-- Plugin para crear ejecutables -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.3.0</version>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>modelo.main.ParqueSwingApp</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Imágenes de funcionamiento del codigo:


Parque de Diversiones
Comprar Tiquete
Message
Pago exitoso.
OK
Logout
Parque de Diversiones
Ver Mi Atracción Asignada
Message
No tienes ninguna atracción asignada.
OK
Logout

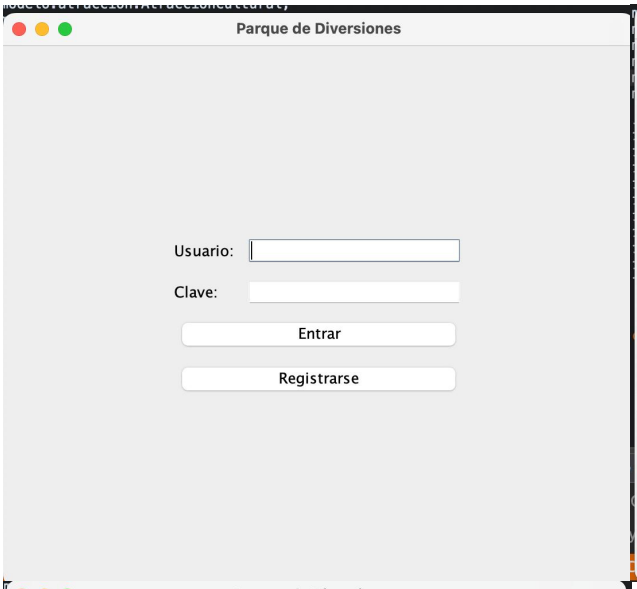

Parque de Diversiones
Ver Mi Atracción Asignada
Logout
Parque de Diversiones
Comprar Tiquete
Message
Procesando pago con Stripe...
OK
Logout

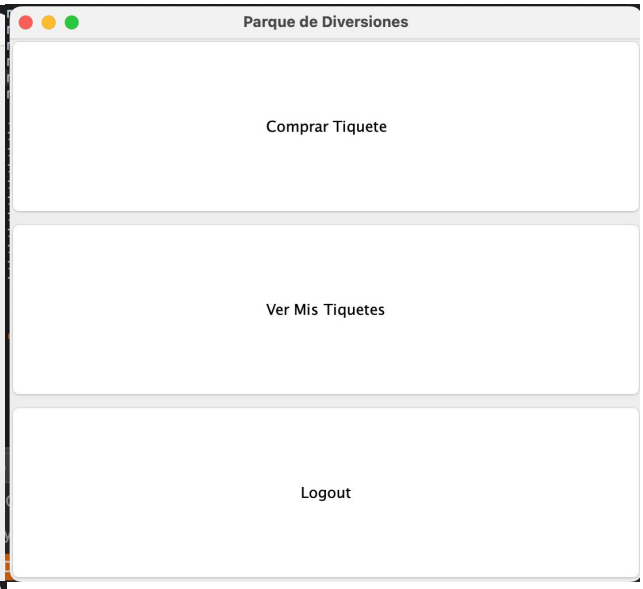

Parque de Diversiones
Comprar Tiquete
Input
Número de Tarjeta de Crédito:
Cancel
OK
Logout
Parque de Diversiones
Comprar Tiquete
Input
CVV:
Cancel
OK
Logout

modelo.atraccion.AtraccionCultural;
Parque de Diversiones
Comprar Tiquete
Input
Fecha de Expiración (MM/AA):
Cancel
OK
Logout
version and reporting problems before the release, you will be providing a valuable service for the

Parque de Diversiones
Comprar Tiquete
Pago con Tarjeta
Seleccione método de pago:
Stripe
Cancel
OK
Logout

Parque de Diversiones
Comprar Tiquete
Comprar Tiquete
Seleccione tipo de tiquete:
FAMILIAR
Cancel
OK
Logout


Parque de Diversiones
Usuario:
Clave:
Entrar
Registrarse


Parque de Diversiones
Comprar Tiquete
Ver Mis Tiquetes
Logout


Parque de Diversiones
Ver Clientes
Ver Trabajadores
Ver Atracciones
Agregar Cliente
Eliminar Cliente
Agregar Trabajador
Eliminar Trabajador
Agregar Atracción
Asignar Empleado a Atracción
Ventas Detalladas Anuales
Logout

UML:

Empleado
-cargo: String
-historialTurnos: List<Turno>
- disponoble : Boolean
- turno : HashMap<String, Turno>
-certificaciones: List<Strings>
+mostrarInformacion(): void
+asignarTurno(turno: Turno): void
+registrarAsistencia(): void

Turno
-id:int
-horaInicio: Date
-horaFin: Date
-tipoTurno: String
-ubicacionTemporal: String
- puestoAsignado: String
+Turno(int id, String horaInicio, String horaFin, String tipoTurno, String ubicacionTemporal)

<<Enumeracion>>
PuestoAsignado
- cajero
- cocinero
- serviciosGenerales
- operarioAtracciones
-operariosAtraccionesEsp

1 bidireccional 1..2
puestoAsignado ..1

ParqueDeDIversionesTest
+ setUp(): void
+ testCrearClienteConDatosValidos(): void
+ testHistorialComprasCliente(): void
+ testDatosClienteCorrectos(: void
+ testTiqueteUsado(): void
+ testTiqueteNuevoNoUsado(): void
+ testPropiedadesTiquete(): void
+ testValidarAccesoMecanicaPermitido(): void
+ testValidarAccesoMecanicaDenegadoPorAltura(): void
+ testValidarAccesoMecanicaDenegadoPorPesoBajo():void
+ testValidarAccesoMecanicaDenegadoPorPesoAlto():void
+ testPropiedadesAtraccionMecanica(): void
+ testCrearEmpleado(): void
+ testDatosEmpleadoCorrectos(): void
+ testValidarAccesoConParametrosNulos(): void
+ testValidarAccesoConParametrosIncompletos(): void
+ testClienteMultiplesCompras(): void
+ testCambioEstadoTiquete(): void

Atracción 1..*
Empleado 1..*

Extends
Extends

EmpleadoServicio
- tipoServicio: String
- servicioGeneral : Boolean
- manejoCocina: Boolean
+ vender ():
+ generalFactura():

Cafetería
- nombre: String
- ubicación: String
- cocinar(Producto plato): void
- vender(Producto producto): void

Empleados 1..*
platos ..*

Producto
- nombre: String
- precio : Int
- getNombre():
- getPrecio(): void

productos ..*

EmpleadoAtracción
- capacidadRiesgo: String
- capacidadAtraccionesAltas: List(String)
- verificarTiquete():
- controlarAtraccion():

Atracciones 1..*

Tienda
- nombre: String
- ubicación: String
- vender(Producto producto)

Taquilla
- tiquetesVendidos : Int
- tiquetesDisponibles: Int
- venderTiquete(): void

Lista de Tiquetes 1..**

AtraccionMecanica
-limiteMinAltura:float
-limiteMaxAltura:float
-limiteMinPeso:float
-limiteMaxPeso: float
-nivelRiesgo: String
-contraindicaciones: String
+AtraccionMecanica(int id, String nombre, int capacidad, String ubicacion, String nivelExclusividad, float limiteMinAltura, float limiteMaxAltura, float limiteMinPeso, float limiteMaxPeso, String nivelRiesgo, String contraindicaciones)
+validarRestricciones(altura: float, peso:float, condicionesMedicas: String): boolean
+iniciarOperacion(): void
+detenerOperacion(): void
+actualizarParametrosSeguridad(nuevosParametros: Map): void

<<Abstract>>
Atraccion
-id:int
-nombre:String
-capacidad:int
-ubicacion:String
-nivelExclusividad: String
-estadoOperativo: boolean
-mantenimientoProgramado: Date
+registrar(): void
+actualizar(): void
+validarAcceso(parametros: Map): boolean
+programarMantenimiento(fecha: Date) void
+obtenerEstado(): boolean

AtraccionCultural
-restriccionEdad: int
-descripcionTematica: String
+AtraccionCultural(int id, String nombre, int capacidad, String ubicacion, String nivelExclusividad, int restriccionEdad)
+validarIngreso(edad: int): boolean
+iniciarEvento(): void
+finalizarEvento(): void
+actualizarInformacionCultural(nuevaInfo: Map) : void

<<Abstract>>
Usuario
-id:int
-nombre:String
-login:String
-password: String
-tipo: TipoUsuario
+Usuario(int id, String nomb. String login, String passwor TipoUsuario tipo)
+autenticar(login: String, pas String)
+cambiarPassword(nuevaPa String): void
+mostrarInformacion(): void

Administrador
+mostrarInformacion():void
+gestionarUsuarios(): void}
+obtenerUsuarioPorId(id: in Usuario
+obtenerUsuarioPorLogin(lo String): Usuario

Lista de Tiquetes 1..**

TiquetesNormales
- id: String
- utilizado: Boolean
- accesoAtraciones: List (Atradccion)
- descuento: int
- comprar(): void
-guardarCompra(): void
- setDescuento(Usuario)

tipo ..1

<<Enumeración >>
Tipo
- familiar
- oro
-diamante
- basico

Cliente
-historialCompras: List<Tiquete>
+mostrarInformacion(): void
+consultarHistorialCompras(): List<Tiquete>
+comprarTiquete(tiquete: Tiquete): void

Extends
Extends

TiquetesFastPass
-fechaEntrada: Date
getTiquetesFastPass():void
setTiquetesFastPass(): void

TiquetesTemporada
- fechaInicial: Date
- fechaCaducidad: Date
getTiquetesTemporada():void
setTiquetesTemporada(): void

TiqueteNormal 1..*
Cliente 1..*
AtraccionMecanica 1..*

Cliente 1 .. *
Tiquetes 1 :: *
Empleado 1.. *
Atraccion 1.. *

<<Interface>>
IPersistencia
+guardarDatos(objeto: Object, nombreArchivo: String) : boolean
+cargarDatos(nombreArchivo: String, Class<T> tipo): T
+actualizarRegistro(objeto: Object, nombreArchivo: String): boolean
+eliminarRegistro(nombreArchivo: String): boolean

<<Abstract>>
CorePersistencia
+JSON = "JSON": String
#rutaAlmacenamiento: String

PersistenciaTiquetes
-ARCHIVO_TIQUETES : String
+guardarListaTiquetes(lista: List<Tiquete>) : boolean
+cargarListaTiquetes(): List<Tiquete>

PersistenciaAtracciones
-ARCHIVO_ATRACCIONES : String
+guardarListaAtracciones(lista: List<Tiquete>) : boolean
+cargarListaAtracciones(): List<Tiquete>

PersistenciaEmpleados
-ARCHIVO_EMPLEADOS : String
+guardarListaEmpleados(lista: List<Empleado>) : boolean
+cargarListaEmpleados(): List<Empleado>

PersistenciaUsuarios
-ARCHIVO_ATRACCIONES : String
+guardarListaUsuarios(lista: List<Usuarios>) : boolean
+cargarListaUsuarios(): List<Usuarios>

PersistenciaEspectaculos
-ARCHIVO_ESPECTACULOS : String
+guardarListaEspectaculos(lista: List<Espectaculo>) : boolean
+cargarListaEspectaculos(): List<Espectaculo>

builds
builds
builds
builds
builds

Tiquete
Atraccion
Empleado
<<Abstract>>
Usuario
Espectaculo

Main
+ Clientes: List<Clientes>
+ empleados: List<Empleado>
+ atracciones: List<Atraccion>
+ tiquetes: LIst<TiquetesNormales>
+ persClientes: persistenceClientes
+ persEmpleados: persistenceEmpleados
+ persAtracciones: persistenceAtracciones
+ persTiquetes: persistenceTiquetes
+ scaner: scaner
- cardLayout: CardLayout
- cards: JPanel
- loginField: JTextField
- passField: JPasswordField
- currentUserId: int
- currentRole: String
- random: Random

Classname
// Métodos UI
- initUI(): void
- createLoginPanel(): JPanel
- createRegisterPanel(): JPanel
- createClientPanel(): JPanel
- createEmployeePanel(): JPanel
- createAdminPanel(): JPanel
- makeButton(String, ActionListener): JButton
- showList(String, List<T>): void
// Lógica de autenticación y navegación
- onLogin(ActionEvent): void
- onAgregarCliente(): void
- onRemoveCliente(): void
- onAgregarTrabajador(): void
- onRemoveTrabajador(): void
- onClientComprar(): void
- onClientVerTiquetes(): void
- onVerAtraccionAsignada(): void
- onAddAtraccion(): void
- onAssignEmpleado(): void
- onAdminVerAtracciones(): void
- onAdminVentasAnuales(): void
// Persistencia adicional
- guardarAsignaciones(): void
- cargarAsignaciones(): void
- onSalir(): void

JFrame

Extends

persistenceTiquetes
persistenceClientes
persistenceAtracciones
persistenceEmpleados