Pablo Medina Forero – 202323276

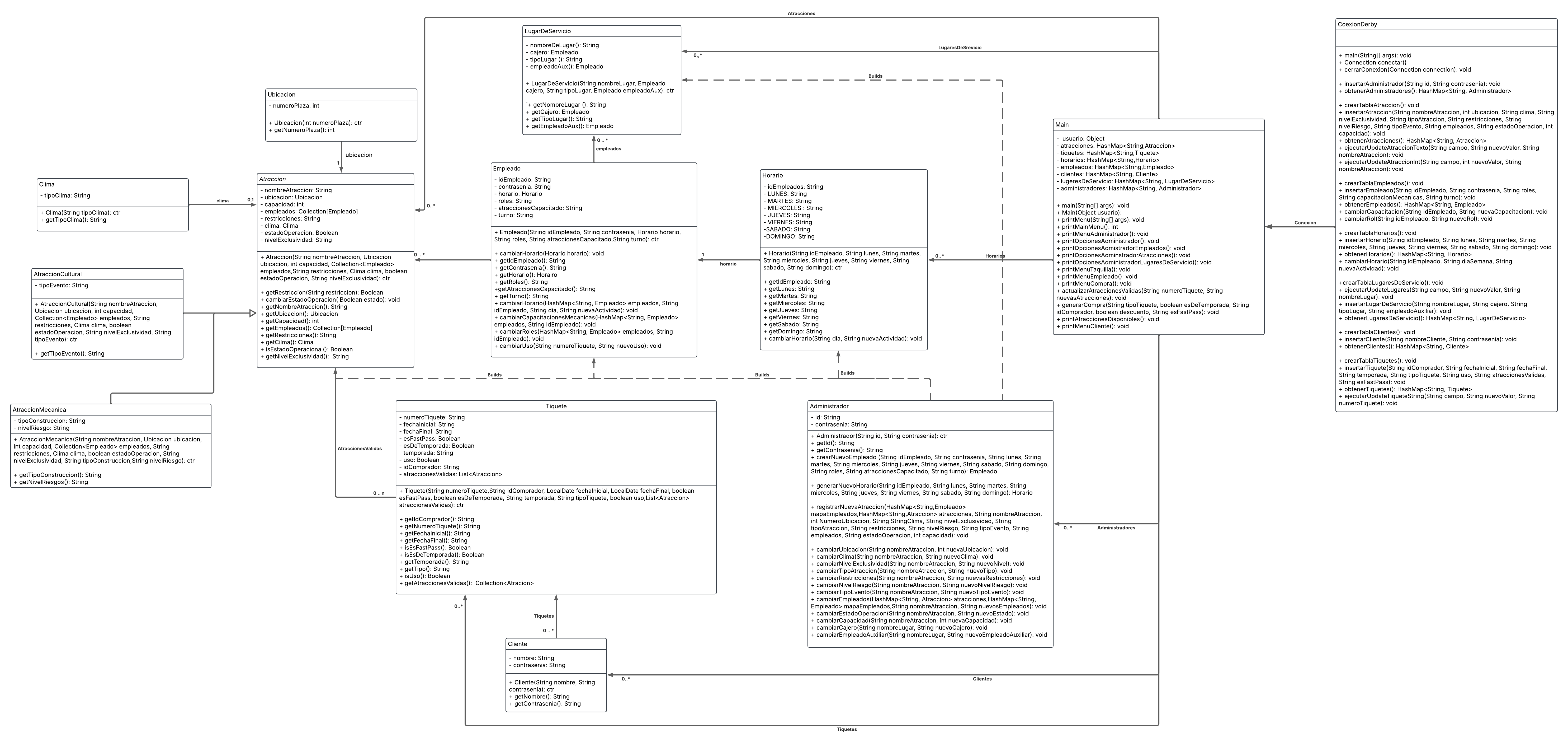
Juan David Suárez - 202223801

Sebastián Barbosa – 202316512

Proyecto 1 entrega final

1. Modelo UML:

Para este proyecto se tiene en cuenta el siguiente diagrama UML:



Donde encontramos las siguientes clases con sus respectivas descripciones y atributos, junto con las relaciones que estas mantienen entre ellas.

Empezando por la clase Main, la cual se encarga de iniciar la aplicación de manera virtual y permitirle al usuario interactuar con el sistema que se está ofreciendo.

**1. Main**

**Responsabilidad:** Punto de entrada de la aplicación; gestiona el flujo de interacción con el usuario (menús, entrada de datos, llamadas a la lógica de negocio).  
 **Justificación:** Centralizar la inicialización y orquestación de todas las operaciones (usuarios, atra­cciones, tiquetes, horarios, etc.) para mantener separada la lógica de presentación de la lógica de dominio.

**Métodos:**

main(String[] args): void

Punto de entrada de la aplicación; inicializa datos y arranca el menú.

Main(Object usuario)

Constructor que recibe el tipo de usuario (admin, empleado o cliente) y lanza su menú.

printMenu(String[] args): void

Despliega el menú principal según el role detectado.

printMainMenu(): int

Muestra opciones iniciales (login, registro) y devuelve la selección.

printMenuAdministrador(): void

Despliega menú de administrador.

printOpcionesAdministrador(): void

Lista acciones CRUD (empleados, atracciones, servicios).

printOpcionesAdmistradorEmpleados(): void

Submenú para gestión de empleados.

printOpcionesAdminstradorAtracciones(): void

Submenú para gestión de atracciones.

printOpcionesAdministradorLugaresDeServicio(): void

Submenú para gestión de lugares de servicio.

printMenuTaquilla(): void

Opciones de taquilla (venta, devolución, consulta).

printMenuEmpleado(): void

Opciones de empleado (consultar turno, reportar estado).

printMenuCompra(): void

Flujo de compra de tiquetes.

actualizarAtraccionesValidas(String numeroTiquete, String nuevasAtracciones): void

Actualiza la lista de atracciones de un tiquete.

generarCompra(String tipoTiquete, boolean esDeTemporada, String idComprador, boolean descuento, String esFastPass): void

Crea y almacena un nuevo tiquete con sus parámetros.

printAtraccionesDisponibles(): void

Muestra atracciones operativas y aptas para el usuario.

printMenuCliente(): void

Menú para cliente (comprar, ver tiquetes, usar pase).

**2. Ubicación**

**Responsabilidad:** Representar la posición física (número de plaza) de atracciones o lugares de servicio dentro del parque.  
 **Justificación:** Necesaria para mapear y mostrar en qué zona del parque se ubica cada recurso, facilitar rutas y planificación logística.

**Constructor:**

Ubicacion(int numeroPlaza)

**Getter:**

getNumeroPlaza(): int

### **3. Clima**

**Responsabilidad:** Modelar las condiciones meteorológicas actuales para evaluar la operatividad de atracciones.  
 **Justificación:** Algunas atracciones (p.ej. mecánicas o al aire libre) dependen del clima; el cambio de clima puede activar cierres automáticos.

**Constructor:**

Ubicacion(int numeroPlaza)

**Getter:**

getNumeroPlaza(): int

→ Devuelve el número de plaza; permite conocer la ubicación sin exponer el campo.

### **4. Tiquete**

**Responsabilidad:** Representar un pase de acceso al parque, con sus características (vigencia, tipo, descuentos, uso y atracciones autorizadas).  
 **Justificación:** Necesario para validar entradas, aplicar descuentos (empleados, temporada), y controlar qué atracciones puede visitar cada cliente.

**Constructor:**

Tiquete(String numeroTiquete, String idComprador, LocalDate fechaInicial, LocalDate fechaFinal, boolean esFastPass, boolean esDeTemporada, String temporada, String tipoTiquete, boolean uso, List<Atraccion> atraccionesValidas)

**Getters:**

getIdComprador(): String

→ ID del cliente dueño del tiquete.

getNumeroTiquete(): String

→ Clave única de identificación del pase.

getFechaInicial(): String

→ Fecha de inicio de validez.

getFechaFinal(): String

→ Fecha de fin de validez.

isEsFastPass(): Boolean

→ Indica si el pase es de acceso rápido.

isEsDeTemporada(): Boolean

→ Dice si es pase de temporada.

getTemporada(): String

→ Nombre de la temporada asignada.

getTipo(): String

→ Categoría del tiquete (adulto, niño, VIP).

isUso(): Boolean

→ Estado de uso (ya utilizado o no).

getAtraccionesValidas(): Collection<Atraccion>

→ Lista de atracciones permitidas con ese tiquete.

**5. Atracción (Abstracta)**

**Responsabilidad:** Modelar cualquier tipo de atracción del parque, con atributos comunes: nombre, ubicación, capacidad, personal asignado, restricciones, clima, estado operativo y nivel de exclusividad.  
 **Justificación:** Facilita la reutilización de código y la extensión para subtipos (cultural, mecánica), asegurando que todas compartan comportamiento esencial (apertura/cierre, asignación de empleados, control de acceso).

**Constructor:**

Atraccion(String nombreAtraccion, Ubicacion ubicacion, int capacidad, Collection<Empleado> empleados, String restricciones, Clima clima, boolean estadoOperacion, String nivelExclusividad)

**Métodos:**

getRestriccion(String restriccion): Boolean

→ Valida que la atracción cumpla cierta restricción (edad, altura).

cambiarEstadoOperacion(Boolean estado): void

→ Activa o desactiva la atracción.

Getters:

getNombreAtraccion(): String

→ Nombre identificador.

getUbicacion(): Ubicacion

→ Objeto de ubicación asociado.

getCapacidad(): int

→ Capacidad máxima de usuarios.

getEmpleados(): Collection<Empleado>

→ Personal asignado a la atracción.

getRestricciones(): String

→ Descripción de restricciones.

getClima(): Clima

→ Objeto clima requerido.

isEstadoOperacional(): Boolean

→ Indica si está activa.

getNivelExclusividad(): String

→ Nivel de acceso/exclusividad.

### **6. AtracciónCultural**

Añade el atributo tipoEvento para representaciones artísticas o shows.

**Constructor:**

AtraccionCultural(..., String tipoEvento)

**Getter:**

getTipoEvento(): String

→ Tipo de evento (teatro, concierto…).

### **7. AtracciónMecánica**

Incluye tipoConstruccion y nivelRiesgo para evaluar la seguridad y el tipo de maquinaria.

**Constructor:**

AtraccionMecanica(..., String tipoConstruccion, String nivelRiesgo)

**Getters:**

getTipoConstruccion(): String

getNivelRiesgos(): String

→ Nivel de riesgo asociado.

**8. Administrador**

**Responsabilidad:** Gestión integral del parque: creación/edición de empleados, atracciones, lugares de servicio, horarios y clima.  
 **Justificación:** Rol con privilegios elevados para mantener configuraciones y reglas de negocio actualizadas, asegurar disponibilidad operativa y seguridad.

**Constructor:**

Administrador(String id, String contrasenia)

**Getters:**

getId(): String

→ Identificador único de admin.

getContrasenia(): String

→ Clave de acceso.

Métodos de creación y configuración:

crearNuevoEmpleado(...): Empleado

generarNuevoHorario(...): Horario

registrarNuevaAtraccion(...): void

Múltiples cambiarX(...) para modificar atributos de atracciones y lugares.

### **9. Cliente**

**Responsabilidad:** Representar al visitante del parque, con credenciales y su colección de tiquetes.  
 **Justificación:** Modelar al usuario final que interactúa con sistemas de venta y control de acceso; fundamental para procesos de autenticación y registro de visitas.

**Constructor:**

Cliente(String nombre, String contrasenia)

**Getters:**

getNombre(): String

→ Nombre de usuario.

getContrasenia(): String

→ Contraseña.

### **9. Empleado**

**Responsabilidad:** Personal asignado a atracciones o lugares de servicio; valida entradas y ejecuta tareas según rol y capacitación.  
 **Justificación:** Necesario para asignar responsabilidades operativas, controlar acceso a atracciones (verificarEntradaCliente(...)) y garantizar que solo personal calificado cubra funciones especializadas.

**Constructor:**

Empleado(String idEmpleado, String contrasenia, Horario horario, String roles, String atraccionesCapacitado, String turno)

**Getters:**

getIdEmpleado(): String

getContrasenia(): String

getHorario(): Horario

getRoles(): String

getAtraccionesCapacitado(): String

getTurno(): String

**Métodos de actualización:**

cambiarHorario(Horario horario)

Sobrecarga de cambiarHorario(...), cambiarCapacitacionesMecanicas(...), cambiarRoles(...), cambiarUso(...) (modifica uso de tiquetes).

### **10. Horario**

**Responsabilidad:** Representar el calendario semanal de actividades (Lunes–Domingo) de un empleado.  
 **Justificación:** Permite planificar y consultar turnos, actividades diarias y modificar en tiempo de ejecución mediante el Administrador o el propio empleado.

**Constructor:**

Horario(String idEmpleado, String lunes, String martes, String miercoles, String jueves, String viernes, String sabado, String domingo)

**Getters:**

getIdEmpleado(): String

getLunes(): String

getMartes(): String

getMiercoles(): String

getJueves(): String

getViernes(): String

getSabado(): String

getDomingo(): String

Método:

cambiarHorario(String dia, String nuevaActividad): void

→ Cambia el horario del empleado

**11. LugarDeServicio**

**Responsabilidad:** Modelar puntos de venta o atención (cafetería, tienda, taquilla), asignando cajero y empleado auxiliar.  
 **Justificación:** Refleja la infraestructura de servicios al cliente y permite gestionar personal y roles en cada ubicación.

**Constructor:**

LugarDeServicio(String nombreLugar, Empleado cajero, String tipoLugar, Empleado empleadoAux)

**Getters:**

getNombreLugar(): String

getCajero(): Empleado

getTipoLugar(): String

getEmpleadoAux(): EmpleadonuevaActividad): void

**11. ConexionDerby**

**Responsabilidad:** Abstraer la persistencia en base de datos Derby.

**Justificación:** Centraliza acceso JDBC y operaciones CRUD.

Métodos:

main(String[] args): void

conectar(): Connection

cerrarConexion(Connection): void

crearTablaX(), insertarX(...), obtenerX() para cada entidad

ejecutarUpdateAtraccionTexto(...), ejecutarUpdateAtraccionInt(...), etc

**Restricciones del sistema:**

1. **Estado de Operación de Atracción**
   1. Una atracción **no puede admitir clientes** si estadoOperacion == false.
   2. El clima adverso puede forzar el cambio de estadoOperacion a false.
2. **Capacidad de la Atracción**
   1. No se debe **superar** el número máximo de clientes definido por capacidad.
   2. Si se alcanza el límite, **ningún cliente extra** puede agregarse.
3. **Validación de Tiquetes**
   1. Un cliente **debe poseer** un tiquete válido (fecha, tipo, uso) para ingresar a la atracción deseada.
   2. Esta validación se realiza en Empleado.verificarEntradaCliente(Cliente, Atraccion) o en la clase Taquilla.
4. **Gestión de Empleados y Capacitación**
   1. Solo empleados con la **capacitación adecuada** (capacitadoParaCocinar, capacitadoNivelAlto, o en atraccionesCapacitado) pueden asignarse a ciertas atracciones o lugares de servicio.
   2. Si pierden los requisitos, deben **removerse** de esa asignación.
5. **Restricciones de Clima**
   1. El Administrador puede cambiar el clima con cambiarClima(...).
   2. Ciertas atracciones (especialmente mecánicas o al aire libre) se **cierran automáticamente** si el clima no es apto.
6. **Ingreso y Salida de Clientes**
   1. No se permite la entrada de un cliente si **no ha registrado** correctamente su salida anterior.
   2. Se debe registrar fecha de ingreso y de salida en SistemaVentas/Taquilla para controlar permanencia y accesos.
7. **Casos de prueba**  
   **Creación y Configuración de Atracciones**
   1. *Objetivo:* Instanciar Atraccion, AtraccionMecanica y AtraccionCultural con distintos niveles de exclusividad, ubicaciones, clima y capacidad.
   2. *Verifica:* Que los constructores asignen correctamente atributos (nivelExclusividad, ubicacion, restricciones, estadoOperacion).
8. **Asignación de Empleados a Atracciones/Lugares**
   1. *Objetivo:* Que Administrador.agregarEmpleadoAtraccion(...) y agregarEmpleadoLDS(...) asignen personal solo si cumple capacitación.
   2. *Verifica:* Que un empleado sin la capacitación requerida no quede asignado y que el listado de empleados de la atracción/lugar se actualice correctamente.
9. **Verificación de Entrada de Clientes**
   1. *Objetivo:* Probar Empleado.verificarEntradaCliente(Cliente, Atraccion) y Taquilla.validarEntrada(Cliente).
   2. *Casos:* Cliente con tiquete caducado / sin FastPass / sin temporada; atracción cerrada; cliente válido.
   3. *Verifica:* Retorno false en condiciones inválidas y true cuando todo es correcto.
10. **Clima y Estado de Operación**
    1. *Objetivo:* Usar Administrador.cambiarClima(...) y luego Atraccion.cambiarEstadoOperacion(...) de forma automática.
    2. *Verifica:* Que tras un clima “adverso” el método cierre la atracción (estadoOperacion = false) y que al restablecer clima “favorable” se reabra si procede.
11. **Manejo de Horarios**
    1. *Objetivo:* Llamar a Horario.cambiarHorario(dia, actividad) y asociar el horario modificado a un Empleado.
    2. *Verifica:* Que Empleado.getHorario() refleje la actividad nueva en el día indicado.
12. **Ingreso y Salida de Clientes (Taquilla / SistemaVentas)**
    1. *Objetivo:* Registrar la venta de tiquetes (SistemaVentas.realizarVenta(...)), la entrada y la salida del cliente.
    2. *Verifica:* Que se agregue correctamente el cliente a clientesAprovados, que marcarSalidaCliente(...) actualice la fecha de salida y que no permita reingreso sin comprar nuevo tiquete.

Para finalizar, dejamos el diagrama de dominio:

