Documentación de Pruebas JUnit para el Sistema de Gestión de Parque de Diversiones

Equipo 7

30 de marzo de 2025

Índice

1. Introducción

Este documento describe los programas de prueba (casos de prueba) diseñados para verificar la correcta implementación de las funcionalidades y reglas de negocio del sistema de gestión del parque de diversiones. Las pruebas se basan en el modelo de dominio conceptual y los requisitos funcionales definidos (`FR1`, `FR2`, `FR3`). Cada caso de prueba detalla su propósito, el escenario bajo el cual se ejecuta, los pasos a seguir, los datos de entrada, el resultado esperado y su trazabilidad con los requisitos y el modelo.

Pruebas de Gestión de Elementos del Parque (FR1)

2. Pruebas de Gestión de Elementos del Parque (FR1)

Estas pruebas verifican la funcionalidad relacionada con la creación, consulta, modificación y eliminación de atracciones (mecánicas, culturales) y espectáculos en el parque.

ID: TC_FR1_CREAR_ATR_MEC_EXITO_01

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionMecanica`, `AtraccionRepository` (Mock)
Método Probado: `crearAtraccionMecanica(...)`
Suite de Pruebas: Gestión de Elementos del Parque - Creación
Descripción/Propósito: Verificar la creación exitosa de una atracción mecánica de riesgo medio con datos válidos.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia mock de `AtraccionRepository`.
- Crear una instancia de `ServicioGestionElementosParque` inyectando el mock del repositorio.
- Definir datos válidos para una atracción mecánica de riesgo medio (ID=."M001", Nombre=Çarrusel", ..., NivelRiesgo=`MEDIO`, capacitacionEspecifica=null).

Pasos de Ejecución:
1. Llamar al método `servicio.crearAtraccionMecanica(...)` con los datos válidos.

Datos de Entrada:
- Parámetros correspondientes a una atracción mecánica válida de riesgo medio.

Resultado Esperado:
- El método debe retornar un objeto `AtraccionMecanica` con los datos proporcionados.
- El método `atraccionRepository.save(Atraccion)` debe ser llamado una vez con la atracción creada.
- No se deben lanzar excepciones.

Trazabilidad: FR1 (Gestión de Atracciones), Clase `ServicioGestionElementosParque`, Clase `AtraccionMecanica`.

ID: TC_FR1_CREAR_ATR_MEC_RIESGO_ALTO_EXITO_02

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionMecanica`, `AtraccionRepository` (Mock)
Método Probado: `crearAtraccionMecanica(...)`
Suite de Pruebas: Gestión de Elementos del Parque - Creación
Descripción/Propósito: Verificar la creación exitosa de una atracción mecánica de riesgo alto con capacitación específica válida.

Precondiciones/Estado Inicial (Setup):
- Crear una instancia mock de `AtraccionRepository`.
- Crear una instancia de `ServicioGestionElementosParque` con el mock.
- Definir datos válidos para una atracción mecánica de riesgo alto (ID=."M002", Nombre="Montaña Rusa X", ..., NivelRiesgo=`ALTO`, capacitacionEspecifica=`MONTAÑA_RUSA_A1`).

Pasos de Ejecución:
1. Llamar a `servicio.crearAtraccionMecanica(...)` con los datos válidos.

Datos de Entrada:
- Parámetros correspondientes a una atracción mecánica válida de riesgo alto, incluyendo `Capacitacion` específica.

Resultado Esperado:
- Retorna un objeto `AtraccionMecanica` válido.
- `atraccionRepository.save(Atraccion)` es llamado una vez.
- No se lanzan excepciones.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`, Clase `AtraccionMecanica`.

ID: TC_FR1_CREAR_ATR_MEC_ERROR_RIESGO_ALTO_SIN_CAP_03

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionMecanica`
Método Probado: `crearAtraccionMecanica(...)`
Suite de Pruebas: Gestión de Elementos del Parque - Creación - Errores
Descripción/Propósito: Verificar que se lanza `DatosInvalidosException` al intentar crear una atracción de riesgo alto sin especificar la capacitación requerida.

Precondiciones/Estado Inicial (Setup):
- Crear instancia mock de `AtraccionRepository`.
- Crear instancia de `ServicioGestionElementosParque` con el mock.
- Definir datos para atracción mecánica de riesgo alto pero con `capacitacionEspecifica = null`.

Pasos de Ejecución:
1. Intentar llamar a `servicio.crearAtraccionMecanica(...)` con los datos inválidos.

Datos de Entrada:
- Parámetros para atracción de riesgo ALTO, pero `capacitacionEspecifica` es `null`.

Resultado Esperado:
- Se lanza una excepción de tipo `DatosInvalidosException`.
- El método `atraccionRepository.save(Atraccion)` no debe ser llamado.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`, Clase `AtraccionMecanica`.

ID: TC_FR1_CREAR_ATR_CULT_EXITO_04

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionCultural`, `AtraccionRepository` (Mock)
Método Probado: `crearAtraccionCultural(...)`
Suite de Pruebas: Gestión de Elementos del Parque - Creación
Descripción/Propósito: Verificar la creación exitosa de una atracción cultural con datos válidos.

Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Instancia de `ServicioGestionElementosParque` con el mock.

- Datos válidos para una atracción cultural (ID=."AC001", Nombre="Show de Mascotas", ..., EdadMinima=0, NivelExclusividad=`FAMILIAR`).

Pasos de Ejecución:
1. Llamar a `servicio.crearAtraccionCultural(...)` con los datos válidos.

Datos de Entrada:
- Parámetros correspondientes a una atracción cultural válida.

Resultado Esperado:
- Retorna un objeto `AtraccionCultural` válido.
- `atraccionRepository.save(Atraccion)` es llamado una vez.
- No se lanzan excepciones.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`, Clase `AtraccionCultural`.

ID: TC_FR1_CREAR_ESP_EXITO_05

Componente/Clase Probada: `ServicioGestionElementosParque`, `Espectaculo`, `AtraccionRepository` (Mock)
Método Probado: `crearEspectaculo(...)`
Suite de Pruebas: Gestión de Elementos del Parque - Creación
Descripción/Propósito: Verificar la creación exitosa de un espectáculo con datos válidos.

Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Instancia de `ServicioGestionElementosParque` con el mock.
- Datos válidos para un espectáculo (ID=."ES001", Nombre="Desfile Nocturno", ..., horarios, climaNoPermitido).

Pasos de Ejecución:
1. Llamar a `servicio.crearEspectaculo(...)` con los datos válidos.

Datos de Entrada:
- Parámetros correspondientes a un espectáculo válido.

Resultado Esperado:
- Retorna un objeto `Espectaculo` válido.
- `atraccionRepository.save(Atraccion)` es llamado una vez.
- No se lanzan excepciones.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`, Clase `Espectaculo`.

ID: TC_FR1_CONSULTAR_ID_EXITO_06

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `consultarAtraccionPorId(String id)`
Suite de Pruebas: Gestión de Elementos del Parque - Consulta
Descripción/Propósito: Verificar la consulta exitosa de una atracción existente por su ID.

Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Configurar el mock para que `findById(."AM001")` retorne un `Optional` conteniendo una `AtraccionMecanica` válida.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.consultarAtraccionPorId(."AM001")`.

Datos de Entrada:
- `id`: ."AM001"(ID existente).

Resultado Esperado:
- El método retorna un `Optional` que contiene la `AtraccionMecanica` configurada en el mock.
- El método `atraccionRepository.findById(.ªM001")` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_CONSULTAR_ID_NO_EXISTE_07

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `consultarAtraccionPorId(String id)`
Suite de Pruebas: Gestión de Elementos del Parque - Consulta - Errores
Descripción/Propósito: Verificar que la consulta por un ID inexistente retorna un Optional vacío.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Configurar el mock para que `findById(ÏD_INEXISTENTE")` retorne `Optional.empty()`.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.consultarAtraccionPorId(ÏD_INEXISTENTE")`.

Datos de Entrada:
- `id`: ÏD_INEXISTENTE".

Resultado Esperado:
- El método retorna `Optional.empty()`.
- El método `atraccionRepository.findById(ÏD_INEXISTENTE")` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_CONSULTAR_NOMBRE_EXITO_08

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `consultarAtraccionPorNombre(String nombre)`
Suite de Pruebas: Gestión de Elementos del Parque - Consulta
Descripción/Propósito: Verificar la consulta exitosa de una atracción existente por su nombre.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Configurar el mock para que `findByNombre(Çarrusel")` retorne un `Optional` con una `Atraccion` válida.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.consultarAtraccionPorNombre(Çarrusel")`.

Datos de Entrada:
- `nombre`: Çarrusel"(Nombre existente).

Resultado Esperado:
- Retorna un `Optional` que contiene la `Atraccion` configurada.
- `atraccionRepository.findByNombre(Çarrusel")` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_CONSULTAR_TODAS_EXITO_09

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `consultarTodasLasAtracciones()`
Suite de Pruebas: Gestión de Elementos del Parque - Consulta
Descripción/Propósito: Verificar que se retornan todas las atracciones existentes.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Crear una lista de atracciones (ej. una mecánica, una cultural).
- Configurar el mock para que `findAll()` retorne la lista creada.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.consultarTodasLasAtracciones()`.

Datos de Entrada: N/A
Resultado Esperado:
- Retorna la lista de atracciones configurada en el mock.
- `atraccionRepository.findAll()` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_ACTUALIZAR_ATR_EXITO_10

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `actualizarAtraccion(Atraccion atraccion)`
Suite de Pruebas: Gestión de Elementos del Parque - Modificación
Descripción/Propósito: Verificar la actualización exitosa de una atracción existente.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Crear una `AtraccionMecanica` existente (ID=."AM001").
- Configurar el mock para que `findById(."AM001")` retorne `Optional.of(atraccionExistente)`.
- Crear una versión modificada de la atracción (ej. cambio de cupo máximo).
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.actualizarAtraccion(atraccionModificada)`.

Datos de Entrada:
- `atraccionModificada`: Objeto `Atraccion` con los datos actualizados.

Resultado Esperado:
- El método no lanza excepciones.
- `atraccionRepository.findById(."AM001")` es llamado una vez.
- `atraccionRepository.save(atraccionModificada)` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_ACTUALIZAR_ATR_ERROR_NO_EXISTE_11

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `actualizarAtraccion(Atraccion atraccion)`
Suite de Pruebas: Gestión de Elementos del Parque - Modificación - Errores
Descripción/Propósito: Verificar que se lanza `AtraccionNoEncontradaException` al intentar actualizar una atracción inexistente.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Crear un objeto `Atraccion` con un ID que no existe (ID=ÏD_INEXISTENTE").
- Configurar el mock para que `findById(ÏD_INEXISTENTE")` retorne `Optional.empty()`.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Intentar llamar a `servicio.actualizarAtraccion(atraccionInexistente)`.

Datos de Entrada:
- `atraccionInexistente`: Objeto `Atraccion` con ID=ÏD_INEXISTENTE".

Resultado Esperado:
- Se lanza una excepción de tipo `AtraccionNoEncontradaException`.
- `atraccionRepository.findById(ÏD_INEXISTENTE")` es llamado una vez.
- `atraccionRepository.save(...)` no es llamado.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_DEFINIR_TEMP_EXITO_12

Componente/Clase Probada: `ServicioGestionElementosParque`, `Atraccion`, `AtraccionRepository` (Mock)
Método Probado: `definirTemporadaAtraccion(String idAtraccion, LocalDateTime fechaInicio, LocalDateTime fechaFin)`
Suite de Pruebas: Gestión de Elementos del Parque - Modificación
Descripción/Propósito: Verificar que se puede definir correctamente la temporada de una atracción existente.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Crear una `Atraccion` existente (ID=.ᴬM001") que permita temporada.
- Configurar el mock para que `findById(.ᴬM001")` retorne `Optional.of(atraccionExistente)`.
- Definir fechas de inicio y fin válidas para la temporada.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.definirTemporadaAtraccion(.ᴬM001", fechaInicio, fechaFin)`.

Datos de Entrada:
- `idAtraccion`: .ᴬM001".
- `fechaInicio`: Fecha/hora de inicio válida.
- `fechaFin`: Fecha/hora de fin válida (posterior a inicio).

Resultado Esperado:
- El método no lanza excepciones.
- `atraccionRepository.findById(.ᴬM001")` es llamado una vez.
- El objeto `Atraccion` recuperado tiene sus fechas de temporada actualizadas.

- `atraccionRepository.save(atraccionActualizada)` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`, Clase `ElementoParque`.

ID: TC_FR1_ELIMINAR_ATR_EXITO_13

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Método Probado: `eliminarAtraccion(String idAtraccion)`
Suite de Pruebas: Gestión de Elementos del Parque - Eliminación
Descripción/Propósito: Verificar la eliminación exitosa de una atracción existente por su ID.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Instancia de `ServicioGestionElementosParque` con el mock.
- (Opcional: Configurar `findById` para verificar existencia si se desea lanzar excepción si no existe).

Pasos de Ejecución:
1. Llamar a `servicio.eliminarAtraccion(.ªM001")`.

Datos de Entrada:
- `idAtraccion`: .ªM001"(ID existente).

Resultado Esperado:
- El método no lanza excepciones.
- El método `atraccionRepository.deleteById(.ªM001")` es llamado una vez.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

ID: TC_FR1_CONSULTAR_TIPOS_EXITO_14

Componente/Clase Probada: `ServicioGestionElementosParque`, `AtraccionRepository` (Mock)
Métodos Probados: `consultarAtraccionesMecanicas()`, `consultarAtraccionesCulturales()`, `consultarEspectaculos()`
Suite de Pruebas: Gestión de Elementos del Parque - Consulta por Tipo
Descripción/Propósito: Verificar que las consultas por tipo específico de atracción/espectáculo funcionan correctamente.
Precondiciones/Estado Inicial (Setup):
- Instancia mock de `AtraccionRepository`.
- Crear listas separadas para atracciones mecánicas, culturales y espectáculos.
- Configurar el mock para que `findAllMecanicas()`, `findAllCulturales()`, `findAllEspectaculos()` retornen las listas correspondientes.
- Instancia de `ServicioGestionElementosParque` con el mock.

Pasos de Ejecución:
1. Llamar a `servicio.consultarAtraccionesMecanicas()`.
2. Llamar a `servicio.consultarAtraccionesCulturales()`.
3. Llamar a `servicio.consultarEspectaculos()`.

Datos de Entrada: N/A
Resultado Esperado:
- Cada llamada retorna la lista correspondiente configurada en el mock.
- Los métodos `findAllMecanicas()`, `findAllCulturales()`, `findAllEspectaculos()` del repositorio son llamados una vez cada uno.

Trazabilidad: FR1, Clase `ServicioGestionElementosParque`.

3. Pruebas de Gestión de Empleados y Asignaciones (FR2)

Estas pruebas verifican la funcionalidad relacionada con el registro de empleados, la consulta de su información y la gestión de sus asignaciones diarias por turnos a diferentes lugares de trabajo dentro del parque.

ID: TC_FR2_REG_EMP_EXITO_01

Componente/Clase Probada: `ServicioGestionEmpleados`, `Empleado`, `Cajero`
Método Probado: `registrarEmpleado(Empleado empleado)`
Suite de Pruebas: Gestión de Empleados - Registro
Descripción/Propósito: Verificar el registro exitoso de un nuevo empleado de tipo Cajero con datos válidos.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioGestionEmpleados`.
- Crear una instancia de `Cajero` con datos válidos: identificacion=."E0"01", nombre=."A"na Torres", capacitaciones=[`MANEJO_CAJA`].

Pasos de Ejecución:
1. Llamar al método `servicio.registrarEmpleado(empleadoCajero)`.

Datos de Entrada:
- `empleadoCajero`: Objeto `Cajero` configurado.

Resultado Esperado:
- El método retorna el objeto `Cajero` registrado.
- El empleado ."E0"01"debe existir en la colección interna de empleados del servicio.
- No se deben lanzar excepciones.

Trazabilidad: FR2 (Gestión de Empleados), Clase `ServicioGestionEmpleados`, Clase `Empleado`, Clase `Cajero`.

ID: TC_FR2_REG_EMP_ERROR_NULL_02

Componente/Clase Probada: `ServicioGestionEmpleados`
Método Probado: `registrarEmpleado(Empleado empleado)`
Suite de Pruebas: Gestión de Empleados - Registro - Errores
Descripción/Propósito: Verificar que se lanza `DatosInvalidosException` al intentar registrar un empleado nulo.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioGestionEmpleados`.

Pasos de Ejecución:
1. Llamar al método `servicio.registrarEmpleado(null)`.

Datos de Entrada:
- `empleado`: `null`.

Resultado Esperado:
- Se lanza una excepción de tipo `DatosInvalidosException`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

ID: TC_FR2_REG_EMP_ERROR_DUPLICADO_03

Componente/Clase Probada: `ServicioGestionEmpleados`, `Empleado`, `Cajero`

Método Probado: `registrarEmpleado(Empleado empleado)`
Suite de Pruebas: Gestión de Empleados - Registro - Errores
Descripción/Propósito: Verificar que se lanza `DatosInvalidosException` al intentar registrar un empleado con una identificación que ya existe.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioGestionEmpleados`.
- Registrar un empleado inicial: `Cajero` con identificacion=."E0"01".
- Crear otra instancia de `Empleado` (puede ser `Cocinero`) con la misma identificacion=."E0"01".

Pasos de Ejecución:
1. Llamar al método `servicio.registrarEmpleado(empleadoDuplicado)`.

Datos de Entrada:
- `empleadoDuplicado`: Objeto `Empleado` con identificacion=."E0"01".

Resultado Esperado:
- Se lanza una excepción de tipo `DatosInvalidosException`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

ID: TC_FR2_CONS_EMP_ID_EXITO_04

Componente/Clase Probada: `ServicioGestionEmpleados`
Método Probado: `consultarEmpleadoPorIdentificacion(String identificacion)`
Suite de Pruebas: Gestión de Empleados - Consulta
Descripción/Propósito: Verificar la consulta exitosa de un empleado existente por su identificación.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioGestionEmpleados`.
- Registrar un empleado: `Cajero` con identificacion=."E0"01", nombre=."A"na Torres".

Pasos de Ejecución:
1. Llamar al método `servicio.consultarEmpleadoPorIdentificacion(."E0"01")`.

Datos de Entrada:
- `identificacion`: ."E0"01".

Resultado Esperado:
- El método retorna un `Optional` que contiene al empleado ."E0"01".
- El empleado recuperado tiene el nombre ."A"na Torres".

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

ID: TC_FR2_CONS_EMP_ID_NO_EXISTE_05

Componente/Clase Probada: `ServicioGestionEmpleados`
Método Probado: `consultarEmpleadoPorIdentificacion(String identificacion)`
Suite de Pruebas: Gestión de Empleados - Consulta - Errores
Descripción/Propósito: Verificar que la consulta por una identificación inexistente retorna un Optional vacío.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioGestionEmpleados`.
- (Opcional) Registrar otros empleados con IDs diferentes.

Pasos de Ejecución:
1. Llamar al método `servicio.consultarEmpleadoPorIdentificacion(ÏD_INEXISTENTE")`.

Datos de Entrada:
- `identificacion`: ÏD_INEXISTENTE".

Resultado Esperado:
- El método retorna `Optional.empty()`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

ID: TC_FR2_ASIG_TURNO_CAJERO_TAQUILLA_EXITO_06

Componente/Clase Probada: `ServicioGestionEmpleados`, `Cajero`, `Taquilla`, `AsignacionTurno`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Éxito
Descripción/Propósito: Verificar la asignación exitosa de un Cajero capacitado a una Taquilla.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `Cajero` (ID=."E0"01") con `Capacitacion.MANEJO_CAJA`.
- Crear una instancia de `Taquilla` (ID="TQ01", Nombre="Taquilla Principal").
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(."E0"01", taquilla, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ."E0"01".
- `lugar`: Objeto `Taquilla` "TQ01".
- `fecha`: Fecha válida.
- `turno`: Turno válido (e.g., `APERTURA`).

Resultado Esperado:
- Retorna un objeto `AsignacionTurno` válido.
- La asignación contiene al empleado ."E0"01", la taquilla "TQ01", la fecha y el turno especificados.
- La colección interna de asignaciones contiene la nueva asignación.
- No se lanzan excepciones.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `AsignacionTurno`, Clase `Cajero`, Clase `Taquilla`.

ID: TC_FR2_ASIG_TURNO_OPERARIO_ATR_MEDIO_EXITO_07

Componente/Clase Probada: `ServicioGestionEmpleados`, `OperarioAtraccion`, `AtraccionMecanica`, `AsignacionTurno`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Éxito
Descripción/Propósito: Verificar la asignación exitosa de un Operario con capacitación general a una Atracción Mecánica de riesgo medio.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `OperarioAtraccion` (ID=."E0"02") con `Capacitacion.OPERACION_ATRACCIONES`.
- Crear una `AtraccionMecanica` (ID=."A"M001", Nombre=Çarrusel", NivelRiesgo=`MEDIO`).
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(."E0"02", atraccionMecanica, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ."E0"02".
- `lugar`: Objeto `AtraccionMecanica` ."A"M001".
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado:
- Retorna un objeto `AsignacionTurno` válido.
- La asignación contiene al empleado ."E0"02τ la atracción ."A"M001".
- No se lanzan excepciones.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `AsignacionTurno`, Clase `OperarioAtraccion`, Clase `AtraccionMecanica`.

ID: TC_FR2_ASIG_TURNO_OPERARIO_ATR_ALTO_EXITO_08

Componente/Clase Probada: `ServicioGestionEmpleados`, `OperarioAtraccion`, `AtraccionMecanica`, `AsignacionTurno`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Éxito
Descripción/Propósito: Verificar la asignación exitosa de un Operario con capacitación específica a una Atracción Mecánica de riesgo alto que requiere esa capacitación.

Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `OperarioAtraccion` (ID=."E0"03") con `Capacitacion.OPERACION_ATRACCIONES` y `Capacitacion.MONTAÑA_RUSA_A1`.
- Crear una `AtraccionMecanica` (ID=."A"M002", Nombre="Montaña Rusa X", NivelRiesgo=`ALTO`, capacitacionEspecifica=`MONTAÑA_RUSA_A1`).
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(."E0"03", atraccionAltoRiesgo, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ."E0"03".
- `lugar`: Objeto `AtraccionMecanica` ."A"M002".
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado:
- Retorna un objeto `AsignacionTurno` válido.
- La asignación contiene al empleado ."E0"03τ la atracción ."A"M002".
- No se lanzan excepciones.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `AsignacionTurno`, Clase `OperarioAtraccion`, Clase `AtraccionMecanica`.

ID: TC_FR2_ASIG_TURNO_SERV_GENERAL_EXITO_09

Componente/Clase Probada: `ServicioGestionEmpleados`, `ServicioGeneral`, `AsignacionTurno`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`

Suite de Pruebas: Gestión de Asignaciones - Éxito
Descripción/Propósito: Verificar la asignación exitosa de un empleado de Servicio General sin un lugar de trabajo específico.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `ServicioGeneral` (ID=."E0"04").
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(."E0"04", null, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ."E0"04".
- `lugar`: `null`.
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado:
- Retorna un objeto `AsignacionTurno` válido.
- La asignación contiene al empleado ."E0"04", `lugarTrabajo` es `null`.
- No se lanzan excepciones.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `AsignacionTurno`, Clase `ServicioGeneral`.

ID: TC_FR2_ASIG_TURNO_ERROR_NO_EMP_10

Componente/Clase Probada: `ServicioGestionEmpleados`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Errores
Descripción/Propósito: Verificar que se lanza `EmpleadoNoEncontradoException` al intentar asignar un empleado inexistente.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Crear una instancia de `Taquilla` (ID="TQ01").
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(ÏD_INEXISTENTE", taquilla, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ÏD_INEXISTENTE".
- `lugar`: Objeto `Taquilla` "TQ01".
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado:
- Se lanza una excepción de tipo `EmpleadoNoEncontradoException`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

ID: TC_FR2_ASIG_TURNO_ERROR_CAPACITACION_11

Componente/Clase Probada: `ServicioGestionEmpleados`, `Cajero`, `Taquilla`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Errores
Descripción/Propósito: Verificar que se lanza `CapacitacionInsuficienteException` al asignar un Cajero sin capacitación de caja a una Taquilla.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `Cajero` (ID=".E005") sin `Capacitacion.MANEJO_CAJA`.
- Crear una instancia de `Taquilla` (ID="TQ02").
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(".E005", taquilla, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ".E005".
- `lugar`: Objeto `Taquilla` "TQ02".
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado:
- Se lanza una excepción de tipo `CapacitacionInsuficienteException`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `Cajero`, Clase `Taquilla`.

ID: TC_FR2_ASIG_TURNO_ERROR_ATR_ALTO_SIN_CAP_ESP_12

Componente/Clase Probada: `ServicioGestionEmpleados`, `OperarioAtraccion`, `AtraccionMecanica`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Errores
Descripción/Propósito: Verificar que se lanza `CapacitacionInsuficienteException` al asignar un Operario sin capacitación específica a una Atracción de riesgo alto que la requiere.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `OperarioAtraccion` (ID=".E006") solo con `Capacitacion.OPERACION_ATRACCIONES`.
- Crear una `AtraccionMecanica` (ID=".AM002", Nombre="Montaña Rusa X", NivelRiesgo=`ALTO`, capacitacionEspecifica=`MONTAÑA_RUSA_A1`).
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(".E006", atraccionAltoRiesgo, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ".E006".
- `lugar`: Objeto `AtraccionMecanica` ".AM002".
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado: • Se lanza una excepción de tipo `CapacitacionInsuficienteException`.
Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `OperarioAtraccion`, Clase `AtraccionMecanica`.

ID: TC_FR2_ASIG_TURNO_ERROR_TIPO_INCORRECTO_13

Componente/Clase Probada: `ServicioGestionEmpleados`, `Cocinero`, `Taquilla`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Errores
Descripción/Propósito: Verificar que se lanza `AsignacionInvalidaException` al intentar asignar un empleado a un lugar incompatible con su rol (e.g., Cocinero a Taquilla).
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `Cocinero` (ID=."E0"07").
- Crear una instancia de `Taquilla` (ID="TQ03").
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(."E0"07", taquilla, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ."E0"07".
- `lugar`: Objeto `Taquilla` "TQ03".
- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado: • Se lanza una excepción de tipo `AsignacionInvalidaException`.
Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `Cocinero`, Clase `Taquilla`.

ID: TC_FR2_ASIG_TURNO_ERROR_SERV_GENERAL_LUGAR_14

Componente/Clase Probada: `ServicioGestionEmpleados`, `Cajero`, `AsignacionTurno`
Método Probado: `asignarTurno(String idEmpleado, LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Gestión de Asignaciones - Errores
Descripción/Propósito: Verificar que se lanza `AsignacionInvalidaException` al intentar asignar un empleado que no es de Servicio General sin especificar un lugar de trabajo.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `Cajero` (ID=."E0"01").
- Definir `LocalDate fecha` y `Turno turno`.

Pasos de Ejecución:
1. Llamar a `servicio.asignarTurno(."E0"01", null, fecha, turno)`.

Datos de Entrada:
- `idEmpleado`: ."E0"01".
- `lugar`: `null`.

- `fecha`: Fecha válida.
- `turno`: Turno válido.

Resultado Esperado:
- Se lanza una excepción de tipo `AsignacionInvalidaException`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `Cajero`.

ID: TC_FR2_CONS_ASIG_EMP_DIA_EXITO_15

Componente/Clase Probada: `ServicioGestionEmpleados`
Método Probado: `consultarAsignacionesEmpleadoDia(String idEmpleado, LocalDate fecha)`
Suite de Pruebas: Gestión de Asignaciones - Consulta
Descripción/Propósito: Verificar la consulta de las asignaciones de un empleado para un día específico.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar empleados E001 (Cajero) y E002 (Operario).
- Crear Taquilla TQ01 y Atraccion AM001.
- Definir `LocalDate fecha`.
- Asignar E001 a TQ01 para `fecha`, turno `APERTURA`.
- Asignar E001 a TQ01 para `fecha`, turno `CIERRE` (horas extras).
- Asignar E002 a AM001 para `fecha`, turno `APERTURA`.

Pasos de Ejecución:
1. Llamar a `servicio.consultarAsignacionesEmpleadoDia(."E0"01", fecha)`.

Datos de Entrada:
- `idEmpleado`: ."E0"01".
- `fecha`: Fecha utilizada en las asignaciones.

Resultado Esperado:
- Retorna una lista con 2 objetos `AsignacionTurno`.
- Una asignación es para el turno `APERTURA` en TQ01.
- La otra asignación es para el turno `CIERRE` en TQ01.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

ID: TC_FR2_VERIF_REQ_CAFETERIA_EXITO_16

Componente/Clase Probada: `ServicioGestionEmpleados`, `Cafeteria`, `Cajero`, `Cocinero`
Método Probado: `verificarRequisitosPersonalLugar(LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Verificación de Requisitos
Descripción/Propósito: Verificar que una Cafetería cumple los requisitos de personal (Cajero + Cocinero) para un turno.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `Cajero` (E001) con `MANEJO_CAJA`.

- Registrar un `Cocinero` (E007) con `MANEJO_ALIMENTOS`.
- Crear una `Cafeteria` (CF01).
- Definir `LocalDate fecha` y `Turno turno`.
- Asignar E001 a CF01 para `fecha`, `turno`.
- Asignar E007 a CF01 para `fecha`, `turno`.

Pasos de Ejecución:
1. Llamar a `servicio.verificarRequisitosPersonalLugar(cafeteria, fecha, turno)`.

Datos de Entrada:
- `lugar`: Objeto `Cafeteria` CF01.
- `fecha`: Fecha de las asignaciones.
- `turno`: Turno de las asignaciones.

Resultado Esperado:
- El método retorna `true`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `Cafeteria`.

ID: TC_FR2_VERIF_REQ_ATRACCION_ERROR_MINIMO_17

Componente/Clase Probada: `ServicioGestionEmpleados`, `AtraccionMecanica`, `OperarioAtraccion`
Método Probado: `verificarRequisitosPersonalLugar(LugarTrabajo lugar, LocalDate fecha, Turno turno)`
Suite de Pruebas: Verificación de Requisitos - Errores
Descripción/Propósito: Verificar que una Atracción no cumple los requisitos si no tiene el número mínimo de empleados asignados.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `OperarioAtraccion` (E002) con `OPERACION_ATRACCIONES`.
- Crear una `AtraccionMecanica` (AM001) con `empleadosMinimos = 2`.
- Definir `LocalDate fecha` y `Turno turno`.
- Asignar solo E002 a AM001 para `fecha`, `turno`.

Pasos de Ejecución:
1. Llamar a `servicio.verificarRequisitosPersonalLugar(atraccion, fecha, turno)`.

Datos de Entrada:
- `lugar`: Objeto `AtraccionMecanica` AM001.
- `fecha`: Fecha de la asignación.
- `turno`: Turno de la asignación.

Resultado Esperado:
- El método retorna `false`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `AtraccionMecanica`.

ID: TC_FR2_AUTORIZAR_CAP_EXITO_18

Componente/Clase Probada: `ServicioGestionEmpleados`, `Empleado`
Método Probado: `autorizarCapacitacion(String idEmpleado, Capacitacion capacitacion)`

Suite de Pruebas: Gestión de Empleados - Capacitaciones
Descripción/Propósito: Verificar que se puede autorizar y agregar una nueva capacitación a un empleado existente.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.
- Registrar un `Cajero` (E001) inicialmente solo con `MANEJO_CAJA`.

Pasos de Ejecución:
1. Llamar a `servicio.autorizarCapacitacion(."E01", Capacitacion.PRIMEROS_AUXILIOS)`.
2. (Verificación) Consultar el empleado E001.

Datos de Entrada:
- `idEmpleado`: ."E01".
- `capacitacion`: `Capacitacion.PRIMEROS_AUXILIOS`.

Resultado Esperado:
- El método no lanza excepciones.
- El empleado E001 consultado posteriormente debe tener la capacitación `PRIMEROS_AUXILIOS` además de `MANEJO_CAJA`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`, Clase `Empleado`.

ID: TC_FR2_AUTORIZAR_CAP_ERROR_NO_EMP_19

Componente/Clase Probada: `ServicioGestionEmpleados`
Método Probado: `autorizarCapacitacion(String idEmpleado, Capacitacion capacitacion)`
Suite de Pruebas: Gestión de Empleados - Capacitaciones - Errores
Descripción/Propósito: Verificar que se lanza `EmpleadoNoEncontradoException` al intentar autorizar capacitación para un empleado inexistente.
Precondiciones/Estado Inicial (Setup):
- Crear instancia de `ServicioGestionEmpleados`.

Pasos de Ejecución:
1. Llamar a `servicio.autorizarCapacitacion(ÏD_INEXISTENTE", Capacitacion.PRIMEROS_AUXILIOS)`.

Datos de Entrada:
- `idEmpleado`: ÏD_INEXISTENTE".
- `capacitacion`: `Capacitacion.PRIMEROS_AUXILIOS`.

Resultado Esperado:
- Se lanza una excepción de tipo `EmpleadoNoEncontradoException`.

Trazabilidad: FR2, Clase `ServicioGestionEmpleados`.

4. Pruebas de Venta y Validación de Tiquetes (FR3)

Estas pruebas verifican la funcionalidad relacionada con la venta de diferentes tipos de tiquetes, su validación para el acceso al parque y atracciones, y el registro de su uso.

ID: TC_FR3_VENTA_GEN_ORO_EXITO_01

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`, `Usuario`
Método Probado: `venderTiqueteGeneral(Usuario comprador, CategoriaTiquete categoria, double precioBase)`
Suite de Pruebas: Venta de Tiquetes - General - Éxito

Descripción/Propósito: Verificar la venta exitosa de un Tiquete General categoría Oro a un usuario regular.

Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una instancia de `Usuario` (no Empleado) con ID=Ü001", Nombre="Juan Perez".
- Definir `precioBase = 50.0`.

Pasos de Ejecución:
1. Llamar a `servicio.venderTiqueteGeneral(usuario, CategoriaTiquete.ORO, precioBase)`.

Datos de Entrada:
- `comprador`: Objeto `Usuario` Ü001".
- `categoria`: `CategoriaTiquete.ORO`.
- `precioBase`: 50.0.

Resultado Esperado:
- Retorna un objeto `TiqueteGeneral` válido.
- El tiquete tiene la categoría `ORO`.
- El precio final del tiquete es 50.0 (sin descuento).
- El tiquete está asociado al comprador Ü001".
- El tiquete indica que el comprador no es empleado.
- No se lanzan excepciones.

Trazabilidad: FR3 (Venta de Tiquetes), Clase `ServicioVentaTiquetes`, Clase `TiqueteGeneral`.

ID: TC_FR3_VENTA_GEN_EMP_DESC_EXITO_02

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`, `Empleado`

Método Probado: `venderTiqueteGeneral(Usuario comprador, CategoriaTiquete categoria, double precioBase)`

Suite de Pruebas: Venta de Tiquetes - General - Éxito

Descripción/Propósito: Verificar la venta exitosa de un Tiquete General con descuento aplicado a un Empleado.

Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una instancia de `Empleado` (e.g., `Cajero`) con ID=.E0 01", Nombre=.Ana Torres".
- Definir `precioBase = 50.0`.
- Constante `DESCUENTO_EMPLEADO` = 0,50.

Pasos de Ejecución:
1. Llamar a `servicio.venderTiqueteGeneral(empleado, CategoriaTiquete.DIAMANTE, precioBase)`.

Datos de Entrada:
- `comprador`: Objeto `Empleado` .E0 01".
- `categoria`: `CategoriaTiquete.DIAMANTE`.
- `precioBase`: 50.0.

Resultado Esperado:
- Retorna un objeto `TiqueteGeneral` válido.
- El tiquete tiene la categoría `DIAMANTE`.
- El precio final del tiquete es 25.0 (50.0 * (1 - 0.50)).

- El tiquete está asociado al comprador .E001".
- El tiquete indica que el comprador es empleado.
- No se lanzan excepciones.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteGeneral`, Clase `Empleado`.

ID: TC_FR3_VENTA_GEN_ERROR_NULL_03

Componente/Clase Probada: `ServicioVentaTiquetes`
Método Probado: `venderTiqueteGeneral(Usuario comprador, CategoriaTiquete categoria, double precioBase)`
Suite de Pruebas: Venta de Tiquetes - General - Errores
Descripción/Propósito: Verificar que se lanza `DatosInvalidosException` al intentar vender un Tiquete General con comprador nulo.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.

Pasos de Ejecución:
1. Llamar a `servicio.venderTiqueteGeneral(null, CategoriaTiquete.FAMILIAR, 30.0)`.

Datos de Entrada:
- `comprador`: `null`.
- `categoria`: `CategoriaTiquete.FAMILIAR`.
- `precioBase`: 30.0.

Resultado Esperado:
- Se lanza una excepción de tipo `DatosInvalidosException`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`.

ID: TC_FR3_VENTA_TEMP_EXITO_04

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteTemporada`, `Usuario`
Método Probado: `venderTiqueteTemporada(Usuario comprador, CategoriaTiquete categoria, LocalDateTime fechaInicio, LocalDateTime fechaFin, double precioBase)`
Suite de Pruebas: Venta de Tiquetes - Temporada - Éxito
Descripción/Propósito: Verificar la venta exitosa de un Tiquete de Temporada a un usuario regular.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una instancia de `Usuario` (ID=Ü002").
- Definir `LocalDateTime fechaInicio` y `LocalDateTime fechaFin` válidas (fin >inicio).
- Definir `precioBase = 200.0`.

Pasos de Ejecución:
1. Llamar a `servicio.venderTiqueteTemporada(usuario, CategoriaTiquete.ORO, fechaInicio, fechaFin, precioBase)`.

Datos de Entrada:
- `comprador`: Objeto `Usuario` Ü002".
- `categoria`: `CategoriaTiquete.ORO`.
- `fechaInicio`: Fecha/hora de inicio válida.

- `fechaFin`: Fecha/hora de fin válida.
- `precioBase`: 200.0.

Resultado Esperado:
- Retorna un objeto `TiqueteTemporada` válido.
- El tiquete tiene la categoría `ORO`.
- El precio final es 200.0.
- Las fechas de inicio y fin coinciden con las proporcionadas.
- No se lanzan excepciones.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteTemporada`.

ID: TC_FR3_VENTA_TEMP_ERROR_BASICO_05

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteTemporada`
Método Probado: `venderTiqueteTemporada(Usuario comprador, CategoriaTiquete categoria, LocalDateTime fechaInicio, LocalDateTime fechaFin, double precioBase)`
Suite de Pruebas: Venta de Tiquetes - Temporada - Errores
Descripción/Propósito: Verificar que se lanza `DatosInvalidosException` (o similar desde el constructor del tiquete) al intentar vender un Tiquete de Temporada con categoría BASICO.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una instancia de `Usuario` (ID=Ü003").
- Definir fechas válidas.

Pasos de Ejecución:
1. Intentar llamar a `servicio.venderTiqueteTemporada(usuario, CategoriaTiquete.BASICO, fechaInicio, fechaFin, 100.0)`.

Datos de Entrada:
- `comprador`: Objeto `Usuario` Ü003".
- `categoria`: `CategoriaTiquete.BASICO`.
- `fechaInicio`: Fecha/hora de inicio válida.
- `fechaFin`: Fecha/hora de fin válida.
- `precioBase`: 100.0.

Resultado Esperado:
- Se lanza una excepción (probablemente `DatosInvalidosException` o una específica del dominio `Tiquete`).

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteTemporada`.

ID: TC_FR3_VENTA_INDIV_EXITO_06

Componente/Clase Probada: `ServicioVentaTiquetes`, `EntradaIndividual`, `Usuario`, `AtraccionMecanica`
Método Probado: `venderEntradaIndividual(Usuario comprador, Atraccion atraccion, double precioBase)`
Suite de Pruebas: Venta de Tiquetes - Individual - Éxito

Descripción/Propósito: Verificar la venta exitosa de una Entrada Individual para una atracción específica.

Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una instancia de `Usuario` (ID=Ü004").
- Crear una instancia de `AtraccionMecanica` (ID=.ªM001", Nombre=Çarrusel").
- Definir `precioBase = 10.0`.

Pasos de Ejecución:
1. Llamar a `servicio.venderEntradaIndividual(usuario, atraccion, precioBase)`.

Datos de Entrada:
- `comprador`: Objeto `Usuario` Ü004".
- `atraccion`: Objeto `AtraccionMecanica` .ªM001".
- `precioBase`: 10.0.

Resultado Esperado:
- Retorna un objeto `EntradaIndividual` válido.
- El tiquete está asociado a la atracción .ªM001".
- El precio final es 10.0.
- No se lanzan excepciones.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `EntradaIndividual`.

ID: TC_FR3_VENTA_FASTPASS_EXITO_07

Componente/Clase Probada: `ServicioVentaTiquetes`, `FastPass`, `Usuario`

Método Probado: `venderFastPass(Usuario comprador, LocalDateTime fechaValida, double precioBase)`

Suite de Pruebas: Venta de Tiquetes - FastPass - Éxito

Descripción/Propósito: Verificar la venta exitosa de un FastPass para una fecha específica.

Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una instancia de `Usuario` (ID=Ü005").
- Definir `LocalDateTime fechaValida`.
- Definir `precioBase = 25.0`.

Pasos de Ejecución:
1. Llamar a `servicio.venderFastPass(usuario, fechaValida, precioBase)`.

Datos de Entrada:
- `comprador`: Objeto `Usuario` Ü005".
- `fechaValida`: Fecha/hora válida.
- `precioBase`: 25.0.

Resultado Esperado:
- Retorna un objeto `FastPass` válido.
- La fecha de validez coincide con la proporcionada.
- El precio final es 25.0.
- No se lanzan excepciones.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `FastPass`.

ID: TC_FR3_VALIDAR_TIQ_GEN_EXITO_08

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`
Método Probado: `validarTiquete(Tiquete tiquete, LocalDateTime fechaUso)`
Suite de Pruebas: Validación de Tiquetes - General
Descripción/Propósito: Verificar que un Tiquete General recién comprado y no utilizado es válido para la fecha actual.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Vender un `TiqueteGeneral` (cualquier categoría) a un usuario.
- Definir `LocalDateTime fechaUso` como la fecha/hora actual.

Pasos de Ejecución:
1. Llamar a `servicio.validarTiquete(tiqueteGeneral, fechaUso)`.

Datos de Entrada:
- `tiquete`: El `TiqueteGeneral` recién creado.
- `fechaUso`: Fecha/hora actual.

Resultado Esperado:
- El método retorna `true`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteGeneral`.

ID: TC_FR3_VALIDAR_TIQ_TEMP_EXITO_09

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteTemporada`
Método Probado: `validarTiquete(Tiquete tiquete, LocalDateTime fechaUso)`
Suite de Pruebas: Validación de Tiquetes - Temporada
Descripción/Propósito: Verificar que un Tiquete de Temporada es válido dentro de su rango de fechas.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Vender un `TiqueteTemporada` con un rango de fechas que incluya la fecha actual.
- Definir `LocalDateTime fechaUso` como una fecha/hora dentro del rango de validez del tiquete.

Pasos de Ejecución:
1. Llamar a `servicio.validarTiquete(tiqueteTemporada, fechaUso)`.

Datos de Entrada:
- `tiquete`: El `TiqueteTemporada` creado.
- `fechaUso`: Fecha/hora dentro del rango válido.

Resultado Esperado:
- El método retorna `true`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteTemporada`.

ID: TC_FR3_VALIDAR_TIQ_TEMP_ERROR_FECHA_10

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteTemporada`
Método Probado: `validarTiquete(Tiquete tiquete, LocalDateTime fechaUso)`
Suite de Pruebas: Validación de Tiquetes - Temporada - Errores
Descripción/Propósito: Verificar que un Tiquete de Temporada no es válido fuera de su rango de fechas.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.

- Vender un `TiqueteTemporada` con un rango de fechas específico (e.g., mes pasado).
- Definir `LocalDateTime fechaUso` como una fecha/hora fuera del rango de validez (e.g., fecha actual).

Pasos de Ejecución:
1. Llamar a `servicio.validarTiquete(tiqueteTemporada, fechaUso)`.

Datos de Entrada:
- `tiquete`: El `TiqueteTemporada` creado.
- `fechaUso`: Fecha/hora fuera del rango válido.

Resultado Esperado:
- El método retorna `false`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteTemporada`.

ID: TC_FR3_VALIDAR_ACCESO_ORO_ATR_FAMILIAR_EXITO_11

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`, `AtraccionCultural`
Método Probado: `validarAccesoAtraccion(Tiquete tiquete, Atraccion atraccion)`
Suite de Pruebas: Validación de Acceso - Categoría/Nivel
Descripción/Propósito: Verificar que un Tiquete General ORO permite acceso a una Atracción de nivel FAMILIAR.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Vender un `TiqueteGeneral` de categoría `ORO`.
- Crear una `AtraccionCultural` con `NivelExclusividad.FAMILIAR`.

Pasos de Ejecución:
1. Llamar a `servicio.validarAccesoAtraccion(tiqueteOro, atraccionFamiliar)`.

Datos de Entrada:
- `tiquete`: Tiquete ORO.
- `atraccion`: Atracción FAMILIAR.

Resultado Esperado:
- El método retorna `true`.
- No se lanzan excepciones.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteGeneral`, Clase `NivelExclusividad`.

ID: TC_FR3_VALIDAR_ACCESO_FAMILIAR_ATR_ORO_ERROR_12

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`, `AtraccionMecanica`
Método Probado: `validarAccesoAtraccion(Tiquete tiquete, Atraccion atraccion)`
Suite de Pruebas: Validación de Acceso - Categoría/Nivel - Errores
Descripción/Propósito: Verificar que se lanza `TiqueteInvalidoException` al intentar acceder a una Atracción ORO con un Tiquete FAMILIAR.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Vender un `TiqueteGeneral` de categoría `FAMILIAR`.
- Crear una `AtraccionMecanica` con `NivelExclusividad.ORO`.

Pasos de Ejecución:
1. Intentar llamar a `servicio.validarAccesoAtraccion(tiqueteFamiliar, atraccionOro)`.

Datos de Entrada:
- `tiquete`: Tiquete FAMILIAR.

- `atraccion`: Atracción ORO.

Resultado Esperado:
- Se lanza una excepción de tipo `TiqueteInvalidoException`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `TiqueteGeneral`, Clase `NivelExclusividad`.

ID: TC_FR3_VALIDAR_ACCESO_INDIV_ATR_CORRECTA_EXITO_13

Componente/Clase Probada: `ServicioVentaTiquetes`, `EntradaIndividual`, `AtraccionMecanica`
Método Probado: `validarAccesoAtraccion(Tiquete tiquete, Ataccion atraccion)`
Suite de Pruebas: Validación de Acceso - Individual
Descripción/Propósito: Verificar que una Entrada Individual permite el acceso a la atracción específica para la que fue comprada.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear una `AtraccionMecanica` (ID=."AM001").
- Vender una `EntradaIndividual` para la atracción ."AM001".

Pasos de Ejecución:
1. Llamar a `servicio.validarAccesoAtraccion(entradaIndividual, atraccionAM001)`.

Datos de Entrada:
- `tiquete`: La `EntradaIndividual` para ."AM001".
- `atraccion`: La instancia de `AtraccionMecanica` ."AM001".

Resultado Esperado:
- El método retorna `true`.
- No se lanzan excepciones.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `EntradaIndividual`.

ID: TC_FR3_VALIDAR_ACCESO_INDIV_ATR_INCORRECTA_ERROR_14

Componente/Clase Probada: `ServicioVentaTiquetes`, `EntradaIndividual`, `AtraccionMecanica`
Método Probado: `validarAccesoAtraccion(Tiquete tiquete, Ataccion atraccion)`
Suite de Pruebas: Validación de Acceso - Individual - Errores
Descripción/Propósito: Verificar que se lanza `TiqueteInvalidoException` al intentar usar una Entrada Individual en una atracción diferente a la comprada.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Crear dos atracciones: `AtraccionMecanica` ."AM001τ `AtraccionCultural` ."AC001".
- Vender una `EntradaIndividual` para la atracción ."AM001".

Pasos de Ejecución:
1. Intentar llamar a `servicio.validarAccesoAtraccion(entradaIndividualAM001, atraccionAC001)`.

Datos de Entrada:
- `tiquete`: La `EntradaIndividual` para ."AM001".
- `atraccion`: La instancia de `AtraccionCultural` ."AC001".

Resultado Esperado:
- Se lanza una excepción de tipo `TiqueteInvalidoException`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `EntradaIndividual`.

ID: TC_FR3_VALIDAR_ACCESO_FASTPASS_ERROR_15

Componente/Clase Probada: `ServicioVentaTiquetes`, `FastPass`, `AtraccionMecanica`
Método Probado: `validarAccesoAtraccion(Tiquete tiquete, Atraccion atraccion)`
Suite de Pruebas: Validación de Acceso - FastPass - Errores
Descripción/Propósito: Verificar que se lanza `TiqueteInvalidoException` al intentar usar un FastPass solo para acceder a una atracción.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Vender un `FastPass`.
- Crear una `AtraccionMecanica`.

Pasos de Ejecución:
1. Intentar llamar a `servicio.validarAccesoAtraccion(fastPass, atraccion)`.

Datos de Entrada:
- `tiquete`: El `FastPass` vendido.
- `atraccion`: La instancia de `AtraccionMecanica`.

Resultado Esperado:
- Se lanza una excepción de tipo `TiqueteInvalidoException`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `FastPass`.

ID: TC_FR3_REGISTRAR_USO_EXITO_16

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`
Método Probado: `registrarUsoTiquete(Tiquete tiquete)`
Suite de Pruebas: Registro de Uso - Éxito
Descripción/Propósito: Verificar que se puede registrar el uso de un tiquete válido por primera vez.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.
- Vender un `TiqueteGeneral` (no utilizado).

Pasos de Ejecución:
1. Llamar a `servicio.registrarUsoTiquete(tiqueteGeneral)`.
2. (Verificación) Consultar el estado del tiquete (debería estar marcado como usado).

Datos de Entrada:
- `tiquete`: El `TiqueteGeneral` no utilizado.

Resultado Esperado:
- El método no lanza excepciones.
- El método `marcarComoUtilizado()` del tiquete es invocado.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `Tiquete`.

ID: TC_FR3_REGISTRAR_USO_ERROR_YA_USADO_17

Componente/Clase Probada: `ServicioVentaTiquetes`, `TiqueteGeneral`
Método Probado: `registrarUsoTiquete(Tiquete tiquete)`
Suite de Pruebas: Registro de Uso - Errores
Descripción/Propósito: Verificar que se lanza `TiqueteYaUtilizadoException` al intentar registrar el uso de un tiquete que ya fue utilizado.
Precondiciones/Estado Inicial (Setup):
- Crear una instancia de `ServicioVentaTiquetes`.

- Vender un `TiqueteGeneral`.
- Registrar el uso del tiquete una vez: `servicio.registrarUsoTiquete(tiqueteGeneral)`.

Pasos de Ejecución:
1. Intentar llamar a `servicio.registrarUsoTiquete(tiqueteGeneral)` por segunda vez.

Datos de Entrada:
- `tiquete`: El `TiqueteGeneral` ya utilizado.

Resultado Esperado:
- Se lanza una excepción de tipo `TiqueteYaUtilizadoException`.

Trazabilidad: FR3, Clase `ServicioVentaTiquetes`, Clase `Tiquete`.