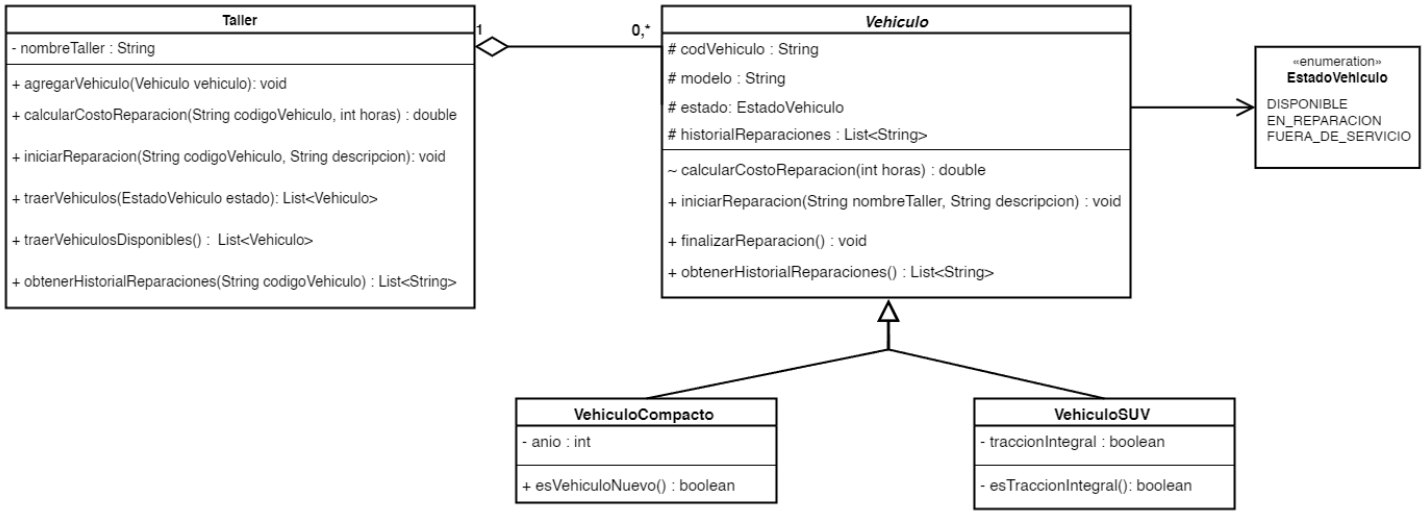
Materia:	Programación II		
Nivel:	2º Cuatrimestre		
Tipo de Examen:	Modelo de Primer Parcial		
Apellido(1):		Fecha:	15 oct 2024
Nombre/s(1):		Docente a cargo(2):	
División(1):		Nota(2):	
DNI(1):		Firma(2):	

(1) Campos a completar solo por el estudiante en caso de imprimir este enunciado en papel.
(2) Campos a completar solo por el docente en caso de imprimir este enunciado en papel.

Enunciado: Sistema para un Taller de Reparación

Un taller de reparación de vehículos necesita un sistema para gestionar diferentes tipos de vehículos, cada uno con características particulares, gestionar las reparaciones, calcular costos de servicio, y gestionar el historial de reparaciones. El sistema también debe manejar la disponibilidad de los vehículos y permitir consultas detalladas sobre el estado de las reparaciones y el historial de cada vehículo. La clase `Taller` será responsable de gestionar los vehículos y las reparaciones.



Capa Modelo

Clase Abstracta `Vehiculo`:

- **Atributos:**
  - `codigoVehiculo` (String): Código único del vehículo, con una longitud exacta de 7 caracteres. Se debe validar esta restricción al crear un nuevo vehículo, lanzando una excepción en caso contrario (IllegalArgumentException).
  - `modelo` (String): Modelo del vehículo.
  - `precioBase` (double): Precio base del servicio de reparación del vehículo.
  - `estado` (enum EstadoVehiculo):
    - `DISPONIBLE, EN_REPARACION, FUERA_DE_SERVICIO.`
  - `historialReparaciones` (List<String>): Lista que guarda una descripción de las reparaciones realizadas en el vehículo.
    - **Formato** "YYYY-MM-DD: [NombreTaller] Descripción"
- **Métodos abstractos:**
  - `calcularCostoReparacion(int horas)`: Método abstracto que será implementado por las subclases para calcular el costo total de la reparación según las características del vehículo.
- **Métodos:**
  - `iniciarReparacion(String nombreTaller, String descripcion)`: Marca el vehículo como en reparación y añade la fecha(Fecha Actual), el nombre del taller junto con una descripción al historial.
  - `finalizarReparacion()`: Marca el vehículo como disponible nuevamente y añade una entrada en el historial indicando el fin de la reparación.
  - `obtenerHistorialReparaciones()`: Retorna el historial de reparaciones del vehículo.

Clase `VehiculoCompacto` (subclase de `Vehiculo`):

- **Atributos:**
  - `anio` (int): Año de fabricación del vehículo.
- **Métodos:**
  - `calcularCostoReparacion(int horas)`: Implementa el método para calcular el costo de reparación, considerando un costo base multiplicado por el número de horas y un descuento del 5% si el vehículo es del año actual.
  - `esVehiculoNuevo()`: Retorna `true` si el vehículo fue fabricado en el año actual, indicando que es un modelo reciente. (`LocalDate.now().getYear()`)

Clase `VehiculoSUV` (subclase de `Vehiculo`):

- **Atributos:**
  - `traccionIntegral` (boolean): Indica si el vehículo tiene tracción integral.
- **Métodos:**
  - `calcularCostoReparacion(int horas)`: Implementa el método para calcular el costo de reparación, considerando un costo base multiplicado por el número de horas, con un aumento del 10% si tiene tracción integral.
  - `esTraccionIntegral()`: Retorna `true` si el vehículo tiene tracción integral.

Clase `Taller`:

- **Atributos:**
  - `nombreTaller` (String): Nombre del taller.
  - `List<Vehiculo> inventarioVehiculos`: Lista que contiene todos los vehículos disponibles en el taller.
  - `List<Vehiculo> vehiculosEnReparacion`: Lista de vehículos que están actualmente en reparación.

- **Métodos:**
  - `agregarVehiculo(Vehiculo vehiculo)`: Permite agregar un nuevo vehículo al inventario del taller, lanzando una excepción si ya existe un vehículo con el mismo código.
  - `calcularCostoReparacion(String codigoVehiculo, int horas)`: Busca un vehículo en el inventario usando `codigoVehiculo`, verifica si existe y calcula el costo de reparación utilizando el método correspondiente de la subclase de `Vehiculo`. Lanza una excepción si el vehículo no está en reparación.
  - `iniciarReparacion(String codigoVehiculo, String descripcion)`: Marca el vehículo como en reparación y lo añade a la lista de vehículos en reparación. Lanza una excepción si ya está en reparación.
  - `finalizarReparacion(String codigoVehiculo)`: Marca un vehículo como disponible nuevamente y lo remueve de la lista de vehículos en reparación.
  - `traerVehiculosDisponibles()`: Retorna una lista de vehículos que no están en reparación.
  - `traerVehiculos(EstadoVehiculo estado)`: Retorna una lista de vehículos que tengan el estado que se indica.
  - `obtenerHistorialReparaciones(String codigoVehiculo)`: Retorna el historial completo de reparaciones de un vehículo, lanzando una excepción si el vehículo no existe.

Capa Test

Nota: Al comenzar cada test, indicar el número correspondiente al test. Por ejemplo: `System.out.println("1-1");` y luego realizar la implementación.

Test.java

1 - Creación de vehículos y validación de datos

1-1: Intentar crear el objeto `VehiculoCompacto` con los siguientes datos:

- **Datos**: `codigoVehiculo=ABC123`, `modelo=Toyota Corolla`, `precioBase=2000.0`, `anio=2024`.
- Validar que el código del vehículo tenga exactamente 7 caracteres. Se espera una excepción por longitud incorrecta.

1-2: Crear e imprimir el objeto `VehiculoCompacto` con los siguientes datos:

- **Datos**: `codigoVehiculo=XYZ5678`, `modelo=Honda Civic`, `precioBase=2200.0`, `anio=2023`.
- Validar que se cree el objeto correctamente y mostrar sus datos.

1-3: Crear e imprimir el objeto `VehiculoSUV` con los siguientes datos:

- **Datos**: `codigoVehiculo=QRS7890`, `modelo=Chevrolet Tahoe`, `precioBase=4000.0`, `traccionIntegral=false`.
- Validar que se cree el objeto correctamente y mostrar sus datos.

2 - Cálculo de costo de reparación

2-1: Calcular e imprimir el costo de reparación del `VehiculoCompacto`:

- **Vehículo**: `codigoVehiculo=XYZ5678`, `modelo=Honda Civic`, `precioBase=2200.0`, `anio=2023`.
- **Horas**: 5 horas.
- Verificar el cálculo del costo de reparación para un vehículo que no es del año actual.

2-2: Calcular e imprimir el costo de reparación del `VehiculoSUV`:

- **Vehículo**: `codigoVehiculo=QRS7890`, `modelo=Chevrolet Tahoe`, `precioBase=4000.0`, `traccionIntegral=false`.

- **Horas**: 3 horas.
- Verificar el cálculo del costo de reparación sin tracción integral.

3 - Gestión de vehículos en el taller

3-1: Agregar los siguientes vehículos al taller:

- **Vehículos**:
  - `VehiculoCompacto` [codigoVehiculo=LMN1111, modelo=Volkswagen Polo, precioBase=1800.0, anio=2024].
  - `VehiculoSUV` [codigoVehiculo=OPQ2222, modelo=Toyota RAV4, precioBase=3000.0, traccionIntegral=true].
- Validar que se agreguen correctamente al inventario del taller.

3-2: Reintentar agregar el siguiente vehículo:

- **Vehículo**: `VehiculoSUV` [codigoVehiculo=LMN1111, modelo=Ford Explorer, precioBase=3500.0, traccionIntegral=true].
- Verificar que se lance una excepción por intentar agregar un vehículo con un código que ya existe en el taller.

4 - Consulta de vehículos disponibles en el taller

4-1: Traer todos los vehículos disponibles en el taller:

- Verificar que el sistema retorne correctamente los vehículos que están `DISPONIBLE`.
- Verificar que el sistema retorne correctamente los vehículos que estén con el estado se les pase por parámetros.

Objetivos de Aprobación No Directa (Calificación de 4 a 5 puntos)

1) **Comprensión básica de la herencia:**

   El estudiante debe demostrar una comprensión básica de la herencia, identificando correctamente las relaciones entre la clase abstracta `Vehiculo` y sus subclases `VehiculoCompacto` y `VehiculoSUV`. Debe ser capaz de entender cómo las subclases heredan atributos y métodos de la superclase `Vehiculo`.

2) **Implementación de métodos abstractos:**

   El estudiante debe ser capaz de implementar correctamente el método abstracto `calcularCostoReparacion()` en las subclases `VehiculoCompacto` y `VehiculoSUV`, asegurando que cada subclase tenga su propia lógica de cálculo de costo.

3) **Manejo de excepciones simples:**

   El estudiante debe demostrar la habilidad de manejar excepciones en situaciones básicas, como al crear un vehículo con un código de longitud incorrecta, usando la validación de longitud de `codigoVehiculo`.

4) **Uso correcto de estados de enumeración:**

   El estudiante debe ser capaz de usar correctamente la enumeración `EstadoVehiculo (DISPONIBLE, EN_REPARACION, FUERA_DE_SERVICIO)` para cambiar el estado de los vehículos de manera coherente al iniciar o finalizar una reparación.

Objetivos de Aprobación Directa (Calificación de 6 a 10 puntos)

1) **Aplicación eficiente del polimorfismo:**

   El estudiante debe demostrar una implementación eficiente de polimorfismo al usar el método `calcularCostoReparacion()` en las subclases `VehiculoCompacto` y `VehiculoSUV`, asegurando que cada subclase aplique sus reglas específicas de costo (como el descuento en vehículos nuevos o el aumento por tracción integral).

2) **Gestión avanzada de excepciones:**

   El estudiante debe ser capaz de manejar casos de excepción más complejos en el taller, como evitar la duplicación de vehículos en el inventario o lanzar una excepción si se intenta finalizar una reparación de un vehículo que no está en reparación.

3) **Implementación completa de la lógica del taller:**

   El estudiante debe ser capaz de implementar correctamente los métodos de la clase `Taller`, como agregar vehículos, calcular costos de reparación, gestionar reparaciones e historial de vehículos, garantizando que el sistema funcione en su totalidad sin errores lógicos o de ejecución.

4) **Consultas eficientes del estado de los vehículos:**

   El estudiante debe implementar correctamente los métodos de consulta en el taller, como `traerVehiculosDisponibles()` y `traerVehiculos()`, asegurando que los datos retornados sean precisos y reflejen el estado real de los vehículos.