Sistema de Gestión de Vehículos en un Taller

Contexto

El Taller “MotorPro” necesita un mini-sistema para gestionar vehículos que ingresan a mantención. Se deben registrar autos y motos, calcular costos, aplicar descuentos, calcular impuestos y prima de seguro, y administrar los registros en memoria. La jefatura quiere ver un demo funcional por consola.

Objetivo de aprendizaje

Evaluar la correcta aplicación de:

- **Herencia y abstracción** (clase base abstracta y 2 hijas).
- **Interfaces** (implementadas por las clases hijas).
- **Sobrecarga** (constructores y/o métodos).
- **Sobreescritura** (métodos polimórficos).
- **DAO** con `ArrayList<Vehiculo>` y **CRUD** completo.
- **Polimorfismo demostrado en `main`**.

Clases e interfaces (nombres y requisitos)

1) Clase abstracta (padre): `Vehiculo` debe implementar la interface

- **Atributos privados**:
  - `String patente` (única, obligatoria)
  - `String marca`
  - `int anio`
  - `double kilometraje`
  - `double valorBaseMantenimiento`
- **Constructores** (sobrecarga mínima):
  - `Vehiculo(String patente)`
  - `Vehiculo(String patente, String marca, int anio, double kilometraje, double valorBaseMantenimiento)`
- **Métodos**:
  - `public String getPatente()` / `setPatente(String)`
  - getters/setters restantes
  - **abstract** `public double calcularCostoMantenimiento();`
  - **abstract** `public String descripcionBreve();`

- **overridden** `public String toString()` → incluir patente, marca, año, tipo, costo mantenimiento

2) Interface

- `Tributable`
  - `double calcularImpuesto();` *(impuesto asociado al mantenimiento, p.ej., 19% del costo)*
  - `double calcularPrimaSeguro();` *(prima en función de año/km/tipo)*

3) Clases hijas

`Auto ->Vehiculo`

- **Atributos extra**:
  - `int numeroPuertas`
  - `boolean hibrido`
- **Constructores** (sobrecarga):
  - `Auto(String patente)`
  - `Auto(String patente, String marca, int anio, double kilometraje, double valorBaseMantenimiento, int numeroPuertas, boolean hibrido)`
- **Implementaciones requeridas**:
  - `@Override double calcularCostoMantenimiento()`
    - Regla sugerida: `valorBaseMantenimiento` + recargo por kilometraje (tramos) + recargo si `hibrido`
  - `@Override String descripcionBreve()`
    - Ej.: `"Auto 4P Híbrido - Marca X (2019)"`
  - `@Override double calcularImpuesto()`
    - Ej.: `calcularCostoMantenimiento() * 0.19`
  - `@Override double calcularPrimaSeguro()`
    - Ej.: base por tipo + ajuste por año (vehículos más nuevos pagan más) + km
- **Sobrecarga de método (requerida)**:
  - `public double aplicarDescuento(double porcentaje)` → porcentaje [0..100]
  - `public double aplicarDescuento(double porcentaje, boolean clienteFiel)` → si `clienteFiel`, aplicar porcentaje + bonus adicional (p.ej., +2%)

`Moto -> Vehiculo`

- **Atributos extra**:
  - `int cilindradaCC`
  - `boolean usoDeportivo`
- **Constructores**:
  - `Moto(String patente)`

    - `Moto(String patente, String marca, int anio, double kilometraje, double valorBaseMantenimiento, int cilindradaCC, boolean usoDeportivo)`
- **Implementaciones requeridas**:
    - `@Override double calcularCostoMantenimiento()`
        - Regla sugerida: base + ajuste por cilindrada + ajuste por uso deportivo
    - `@Override String descripcionBreve()`
    - `@Override double calcularImpuesto()`
    - `@Override double calcularPrimaSeguro()`
- **Sobrecarga (opcional pero recomendable)**:
    - `public double aplicarDescuento(double porcentaje)`
    - `public double aplicarDescuento(double porcentaje, boolean clubMoteros)`

DAO: `VehiculoDAO`

- **Estructura interna**:
    - `private  ArrayList<Vehiculo> vehiculos = new ArrayList<>();`
- **CRUD obligatorio**:
    - **Create**: `public boolean agregar(Vehiculo v)`
        - Debe **validar unicidad** por `patente`; si existe, **no** agrega y retorna `false`.
    - **Read**:
        - `public Vehiculo buscar(String patente)`
        - `public ArrayList<Vehiculo> obtenerDatos()`
    - **Update**:
        - `public boolean actualizar(Vehiculo v)`
            - Reemplaza el existente con misma `patente`.
    - **Delete**:
        - `public boolean eliminar(String patente)`
- **Búsquedas/consultas adicionales (mínimo 2)**:
    - `public ArrayList<Vehiculo> buscarPorMarca(String marca)`
    - `public ArrayList<Vehiculo> listarPorAnioDesc()` (ordenamiento)

Reglas de negocio sugeridas (para tus fórmulas)

- **Costo mantenimiento**:
    - Base + tramo por km (`<20k`, `20–80k`, `>80k`) + recargo por rasgos (híbrido / deportivo / alta cilindrada).
- **Impuesto**: 19% del costo.

- **Prima seguro**: base por tipo + ajuste por antigüedad (año actual − `anio`) + ajuste por km.

*(Puedes elegir valores concretos, pero **documenta** las fórmulas en comentarios.)*

Demostración en `Main`

Debe mostrar por consola, **sin leer archivos**, al menos:

1. Crear `VehiculoDAO` y **agregar** 4 objetos (2 `Auto`, 2 `Moto`) usando **constructores sobrecargados**.
2. **Listar todos** y mostrar `toString()` (polimorfismo dinámico).
3. Para cada `Vehiculo`:
   - Imprimir `descripcionBreve()`, `calcularCostoMantenimiento()`, `calcularImpuesto()`, `calcularPrimaSeguro()`.
   - Aplicar **sobrecarga** de `aplicarDescuento(...)` en al menos un `Auto` y una `Moto`.
4. **Buscar** por `patente` y **actualizar** un registro (p.ej., cambiar km).
5. **Eliminar** un registro y mostrar el listado final.
6. **Demostración de polimorfismo por interfaz**:
   - Crear una lista `ArrayList<Vehiculo>` con instancias `Auto` y `Moto`, recorrerla e invocar `calcularPrimaSeguro()`.