# Evaluación Contínua UF2404: Sistema de Gestión de Taller de Vehículos - Ejercicio en Java

Desarrolla una aplicación en Java para gestionar un taller de vehículos. El sistema permitirá registrar vehículos, clientes y reparaciones, aplicando conceptos de herencia, polimorfismo, agregación, composición, `static`, `final`, y uso de colecciones (`ArrayList`).

Se debe entregar el proyecto completo de Java Maven que se cree. Deberéis subirlo a un repositorio o carpeta en la nube y compartir el enlace en el espacio dedicado a ello en el campus de Ironhack.

---

## Enunciado Completo

1. **Clase `Persona` (abstracta)**: Representa a cualquier persona en el sistema (clientes o mecánicos).
    - **Atributos**:
        - `nombre` (String, final): nombre de la persona.
        - `telefono` (String, final): número de teléfono.
    - **Métodos**:
        - Constructor que reciba `nombre` y `telefono`.
        - Métodos `getters` para los atributos.
        - Método abstracto `mostrarInfo()`, que será implementado por las clases hijas para mostrar detalles específicos.

2. **Clase `Cliente`** (hereda de `Persona`): Representa a un cliente que posee uno o más vehículos.
    - **Atributos**:
        - `vehiculos` (ArrayList de `Vehiculo`): lista de vehículos pertenecientes al cliente.
    - **Métodos**:
        - Constructor que reciba `nombre` y `telefono`.
        - Método `agregarVehiculo(Vehiculo vehiculo)` que agregue un vehículo a la lista de vehículos.
        - Implementación de `mostrarInfo()`, que imprima el nombre, teléfono y lista de vehículos del cliente.

- **Agregación**: Un cliente posee uno o más vehículos, pero los vehículos existen independientemente del cliente. Los vehículos pueden añadirse o eliminarse sin afectar la existencia de la clase `Cliente`.

3. **Clase `Vehiculo`**: Representa un vehículo en el sistema.
   - **Atributos**:
     - `matricula` (String, final): matrícula del vehículo.
     - `marca` (String): marca del vehículo.
     - `modelo` (String): modelo del vehículo.
     - `propietario` (asociación con `Cliente`): referencia al propietario del vehículo.
   - **Métodos**:
     - Constructor que reciba `matricula`, `marca`, `modelo` y `propietario`.
     - Métodos `getters` y `setters` para `marca` y `modelo`.
     - Método `mostrarInfoVehiculo()` que imprima la matrícula, marca, modelo y propietario del vehículo.
   - **Asociación**: Un vehículo está asociado a un cliente, pero el cliente no necesita ser creado dentro de `Vehiculo`; es una relación externa que puede cambiar sin que el vehículo dependa directamente de `Cliente` para existir.

4. **Clase `Reparacion`** (composición): Representa una reparación realizada a un vehículo.
   - **Atributos**:
     - `vehiculo` (composición con `Vehiculo`): el vehículo al que se le realizará la reparación. La reparación crea el vehículo internamente.
     - `descripcion` (String): descripción del trabajo de reparación.
     - `costo` (double): costo de la reparación.
   - **Métodos**:
     - Constructor que reciba `descripcion`, `costo`, `matricula`, `marca` y `modelo`, creando internamente un objeto `Vehiculo`.
     - Método `mostrarInfoReparacion()` que muestre los detalles de la reparación, incluyendo los datos del vehículo.
   - **Composición**: `Reparacion` crea un nuevo objeto `Vehiculo` internamente, indicando que la reparación está vinculada al vehículo específico que se describe, y depende de esta instancia creada dentro de `Reparacion`.

5. **Clase `Taller`**: Administra toda la información del taller.

- **Atributos**:
  - `nombre` (String, final): nombre del taller.
  - `clientes` (ArrayList de `Cliente`): lista de clientes registrados.
  - `reparaciones` (ArrayList de `Reparacion`): lista de reparaciones realizadas.
- **Métodos**:
  - Constructor que reciba el `nombre` del taller e inicialice las listas.
  - Métodos `registrarCliente(Cliente cliente)` y `registrarReparacion(Reparacion reparacion)` para añadir nuevos clientes y reparaciones.
  - Método `listarReparaciones()` que imprima todas las reparaciones realizadas.
  - Método estático `contarReparaciones()` que devuelva el número total de reparaciones registradas.
- **Agregación**: `Taller` tiene una lista de `Cliente` y `Reparacion`, pero el taller y los clientes existen de manera independiente entre sí. Se pueden añadir o quitar clientes sin que afecte a la existencia del taller.

6. **Clase `AppTaller`**: Implementa el método `main` para simular la operación del sistema.
   - Crea una instancia de `Taller`.
   - Registra algunos clientes y vehículos.
   - Realiza algunas reparaciones usando `Scanner` para recibir detalles de entrada.
   - Muestra la lista de reparaciones activas.

---

## Notas

- Usa `final` para atributos que no deben cambiar, como `matricula` en `Vehiculo`.
- La **agregación** se encuentra en la relación entre `Cliente` y `Vehiculo`, así como entre `Taller` y `Cliente` y `Reparacion`.
- La **composición** se usa dentro de `Reparacion`, creando un `Vehiculo` específico internamente como una parte dependiente de la reparación.
- Utiliza `Scanner` en `AppTaller` para recibir datos de vehículos y reparaciones de parte del usuario.

# Rúbrica de Evaluación

**Estructura y Organización del Código (2 puntos)**

- 2 puntos: El código está bien estructurado, con nombres claros para las clases, métodos y variables, y con un uso adecuado de la indentación y comentarios.
- 1 punto: El código es funcional, pero la organización y los nombres pueden ser confusos o difíciles de seguir.
- 0 puntos: El código es difícil de leer y entender debido a la falta de estructura o de claridad en nombres y organización.

**Implementación de Herencia y Polimorfismo (2 puntos)**

- 2 puntos: La herencia entre `Persona`, `Cliente`, y `Mecanico` (si se ha implementado) es correcta, y el polimorfismo se aplica adecuadamente mediante el método `mostrarInfo()`.
- 1 punto: La herencia está implementada, pero faltan algunos aspectos de polimorfismo o el uso de `mostrarInfo()`.
- 0 puntos: No se ha implementado herencia ni polimorfismo de manera correcta.

**Implementación de Composición y Agregación (2 puntos)**

- 2 puntos: La composición y agregación se han implementado correctamente en las clases `Reparacion` y `Taller`, con el uso adecuado de la creación de objetos en sus constructores.
- 1 punto: La composición y agregación están presentes, pero tienen errores o implementaciones confusas.
- 0 puntos: No se ha implementado la composición o agregación de manera adecuada.

**Uso de ArrayList y Gestión de Colecciones (2 puntos)**

- 2 puntos: Se han utilizado correctamente `ArrayList` para almacenar clientes, vehículos y reparaciones, y se han gestionado adecuadamente las operaciones sobre estas colecciones.
- 1 punto: Se ha utilizado `ArrayList`, pero con errores en su implementación o sin gestionar adecuadamente las operaciones.
- 0 puntos: No se han utilizado `ArrayList` o no funcionan de manera correcta en el contexto del ejercicio.

**Captura de Entrada con Scanner y Ejecución en `AppTaller` (2 puntos)**

- 2 puntos: La clase `AppTaller` utiliza `Scanner` para recibir datos de entrada, permite registrar clientes y reparaciones, y muestra la lista de reparaciones.

- 1 punto: La clase `AppTaller` permite registrar clientes y reparaciones, pero la captura de entrada con `Scanner` es parcial o incorrecta.
- 0 puntos: No se ha implementado la captura de entrada con `Scanner` o no funciona de manera correcta.

Puntuación máxima: **10 puntos**