# CASO PRÁCTICO 1

**DESARROLLAR E IMPLEMENTAR UNA APLICACIÓN JAVA PARA CONCESIONARIO DE COCHES**

## Contexto

Trabajas en una empresa como programador Java, y vuestros principales clientes son grandes empresas multinacionales con grandes volúmenes de venta. El principal cliente de vuestra empresa, un famoso concesionario de coches os ha encargado que desarrolléis una aplicación Java para ellos, adaptada tanto a los productos y servicios que ofrecen, como a sus principales clientes.

El encargo es el siguiente: debéis elaborar un programa Java permita la gestión del concesionario que trabaja con tres tipos de coches: km0, nuevos y segunda mano. Todos ellos tienen una información común: matrícula, color,modelo, precio y kilómetros recorridos.

A su vez, los coches de segunda mano y nuevos tienen una información específica: los coches de segunda mano, el nombre del antiguo propietario. y los coches nuevos, la duración de la garantía.

Los coches km0 pueden ser alquilados y los coches de segunda mano y nuevos solo vendidos.

## Cuestiones a resolver

1. Crea una clase abstracta Coche, donde se definen los atributos mínimos para representar las características elementales de cualquier coche, que serán matricula, color, modelo, precio y kilómetros recorridos, además de un atributo de clase llamado numCochesStock, utilizado para contar los objetos de tipo Coche(que son, Km0, segunda mano o nuevos) que se instancien, este será un entero y se incrementará en 1 cada vez que se crea un objeto de una de las clases de coches que posee el concesionario.
2. Crea el código de las clases Java implementando una relación de herencia desde la superclase abstracta Coche hasta las subclases CocheKm0, CocheNuevo, CocheSegundaMano.
3. Cada clase debe disponer de constructor, así como permitir establecer y recuperar el valor de sus atributos (en caso de que los tenga) mediante métodos getter y setter, además de un método para mostrar la información del objeto, para ello deberás sobreescribir el método toString en las clases que sea necesario para mostrar la información correctamente y utilizando el principio de herencia según convenga para no volver a repetir código ya implementado.
4. Crea dos interfaces, una llamada Vendible y otra llamada Alquilable. Una con un método void vender () que muestre en consola “El coche X se ha vendido” y otra con un método void alquilar() que muestre “El coche X se ha alquilado”, que implementarán las clases de coches que puedan alquilar o vender, donde X será la clase de coche, que corresponderá a km0, de segunda mano o nuevo.
5. Crea una clase principal con el método main, donde se cree un objeto de cada tipo y se realicen las acciones necesarias para que se venda o alquile un coche km0, de segunda mano o nuevo según el tipo de coche, siempre teniendo en cuenta las especificaciones y restricciones que se han planteado en el encargo, el usuario sabrá las acciones realizadas en cada momento por los mensajes que aparecerán en consola de cada uno de los objetos creados.
6. Crea las clases, las relaciones de herencia e implementa las interfaces en las clases correspondientes según las indicaciones y teniendo en cuenta las restricciones y reglas a la hora de implementar clases convencionales, clases abstractas, interfaces y atributos de clase.

Para el desarrollo del código debe usarse el IDE Eclipse y se debe entregar preferiblemente solo un archivo con **extensión .java** que contenga un método main() para que se pueda ejecutar.

En este enlace se encuentra la explicación de cómo sobrescribir y utilizar el método toString: <https://www.youtube.com/watch?v=EqiUftNakIs>

## Recursos

Se deberá consultar el contenido de la unidad, internet, libros, revistas y utilizar medios informáticos para la presentación del caso práctico (Word, Power-Point…).

## Objetivos

Comprender y aplicar la sintaxis básica de las clases en Java, incluyendo la declaración de propiedades y métodos.

Crear constructores para inicializar objetos de clase y comprender su importancia en la creación de instancias.

Comprender los modificadores de acceso y cómo se utilizan para controlar la visibilidad de las clases y sus miembros.

Definir y utilizar variables y métodos estáticos que pertenecen a la clase en lugar de una instancia específica.

Identificar y utilizar interfaces para definir contratos comunes que las clases deben implementar.

Utilizar clases abstractas y conocer cómo se declaran.

## Resultados de aprendizaje y criterios de evaluación

Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.

* Se ha reconocido la sintaxis, estructura y componentes típicos de una clase.
* Se han definido clases.
* Se han definido propiedades y métodos.
* Se han creado constructores.
* Se han desarrollado programas que instancien y utilicen objetos de las clases creadas anteriormente.
* Se han utilizado mecanismos para controlar la visibilidad de las clases y de sus miembros.
* Se han definido y utilizado clases heredadas.
* Se han creado y utilizado métodos estáticos.
* Se han creado y utilizado conjuntos y librerías de clases.

Desarrolla programas aplicando características avanzadas de los lenguajes orientados a objetos y del entorno de programación.

* Se han identificado los conceptos de herencia, superclase y subclase.
* Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.
* Se ha reconocido la incidencia de los constructores en la herencia.
* Se han creado clases heredadas que sobrescriben la implementación de métodos de la superclase.
* Se han diseñado y aplicado jerarquías de clases.
* Se han probado y depurado las jerarquías de clases.
* Se han realizado programas que implementen y utilicen jerarquías de clases.
* Se ha comentado y documentado el código.
* Se han identificado y evaluado los escenarios de uso de interfaces.
* Se han identificado y evaluado los escenarios de utilización de la herencia y la composición.

## Rúbrica para su evaluación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Excelente** | **Satisfactorio** | **Mejorable** | **Insuficiente** |
| **Definición de la clase abstracta y atributos estáticos.** | Crea la clase abstracta clara y bien estructurada. El atributo estático está bien definido y correctamente implementado y gestionado en el constructor, con una lógica clara para su incremento. | Crea la clase abstracta y atributo estático de manera adecuada. El atributo estático se incrementa, pero la implementación podría optimizarse. | Crea la clase abstracta pero su estructura es confusa o incompleta. Puede fallar el atributo estático, o su uso es incorrecto. La herencia no se realiza de forma correcta. | No define correctamente la clase abstracta o los atributos estáticos. No se respeta el principio de herencia y la implementación es ineficiente o incorrecta. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Implementación de las subclases y relaciones de herencia.** | Diseña subclases que respetan correctamente la jerarquía de herencia, reutilizando código de la clase base y aplicando sobreescritura con precisión. | Implementa subclases y herencia adecuadamente, aunque con menor aprovechamiento del código base y aplica sobreescritura correctamente. | Crea subclases básicas, pero con errores o sin aprovechar la herencia correctamente. | No implementa correctamente las subclases ni respeta las relaciones de herencia. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Uso de interfaces para definir acciones.** | Integra correctamente interfaces, definiendo y aplicando métodos como vender o alquilar con mensajes claros en consola. | Implementa interfaces y métodos de manera adecuada. | Utiliza interfaces y métodos con errores. | No define ni utiliza correctamente interfaces o métodos necesarios y/o estos son irrelevantes para el caso práctico. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Implementación del método main y ejecución del programa.** | Se crean correctamente los objetos de cada tipo (Km0, Nuevo y SegundaMano). Se ejecutan correctamente las acciones de venta y alquiler según corresponda. Se muestra información clara y organizada en la consola. El código es estructurado y fácil de entender. | Se crean los objetos y se realizan las acciones correctamente, pero necesita algún tipo de mejora. Se implementan correctamente los métodos de venta y alquiler, aunque hay ligeras posibles mejoras en la organización del código. | Se crean los objetos, pero falta ejecutar alguna acción importante. La salida en consola es confusa o poco organizada. Hay errores menores en la lógica del main. | No se crean correctamente los objetos o faltan acciones esenciales. La ejecución del programa no refleja adecuadamente las funciones de venta y alquiler. El main no sigue una estructura lógica o tiene errores graves. |
| 20% | 2 | 1.5 | 1 | 0.5 |
| **Código limpio con estructuras necesarias.** | Presenta código limpio con estructuras necesarias y sin redundancia. | Presenta código limpio, pero hay algunas estructuras que se pueden simplificar, eliminar o tabular para evitar redundancia. | Presenta código más o menos limpio, pero hay estructuras que se pueden simplificar, eliminar o tabular para evitar redundancia. | Presenta código desordenado, hay muchas estructuras que se deben simplificar o eliminar para evitar redundancia. |
| 10% | 1 | 0.75 | 0.5 | 0.25 |
| **Errores de compilación y/o ejecución.** | El programa se ejecuta sin errores de compilación o ejecución, realizando la función esperada. | El programa se ejecuta sin errores de compilación, pero sí con algunos errores en la ejecución, la función realizada varía ligeramente de la función esperada. | El programa se ejecuta sin errores de compilación, pero sí con varios errores en la ejecución, la función realizada varía de la función esperada. | El programa se ejecuta con errores de compilación por tanto no es ejecutable y/o el programa ejecuta sin errores de compilación, pero con numerosos errores en la ejecución, la función realizada no coincide con la función esperada. |
| 10% | 1 | 0.75 | 0.5 | 0.25 |