Enunciado.

A lo largo de esta unidad has ido aprendiendo a crear tus propias clases así como sus distintos miembros (atributos y métodos). Has experimentando con la encapsulación y accesibilidad (modificadores de acceso a miembros), has creado miembros estáticos (de clase) y de instancia (de objeto), has escrito constructores para tus clases, has sobrecargado métodos y los has utilizado en pequeñas aplicaciones. También has tenido tu primer encuentro el concepto de herencia, que ya desarrollarás en unidades más avanzadas junto con otros conceptos avanzados de la Programación Orientada a Objetos.

Una vez finalizada la unidad se puede decir que tienes un dominio adecuado del lenguaje Java como para desarrollar tus propias clases y utilizarlas en una aplicación final que sea capaz de manipular un conjunto de datos simple. Dada esa premisa, esta tarea tendrá como objetivo escribir una pequeña aplicación en Java empleando algunos de los elementos que has aprendido a utilizar.

Se trata de desarrollar una aplicación Java denominado **PROG05_Tarea** que permita gestionar un vehículo. Mediante un menú que aparecerá en pantalla se podrán realizar determinas operaciones:

1. **Nuevo Vehículo.**
2. **Ver datos identificativos**
3. **Ver estado del vehículo**
4. **Viajar**
5. **Repostar**
6. **Llenar depósito.**
7. **Actualizar precio combustible.**
8. **Mostrar antigüedad**.
9. **Mostrar propietario**.
10. **Salir.**

La funcionalidad será la siguiente:

- Al iniciar la aplicación se mostrará el menú propuesto.
- Dependiendo de la opción seleccionada por el usuario:
    - **Nuevo Vehículo:** Se creará un nuevo Vehículo, si los datos introducidos por el usuario son correctos, que contendrá la siguiente información (marca, modelo, matrícula, número de kilómetros, fecha de matriculación, capacidad de depósito, descripción, precio, nombre del propietario, dni del propietario). Al crear un nuevo vehículo se hará con el depósito vacío. Todos los datos serán solicitados por teclado y tan solo habrá que comprobar:
        - Que la fecha de matriculación es anterior a la actual: puedes solicitar por separado día, mes y año y construir un objeto LocalDate (tienes

una referencia en el apartado Consejos y recomendaciones).

- Que el número de kilómetros es mayor o igual que 0.
- Que el DNI del propietario es correcto.
- Que la capacidad de depósito es un número positivo y menor que 100.
- Que la marca, modelo y matrícula no se han dejado en blanco.
- Que el consumo medio del vehículo (expresado en litros de combustible por cada 100 km) es positivo e inferior a 20.
- Si no se cumple algunas de las condiciones se deberá mostrar el correspondiente mensaje de error. En ese caso habrá se mostrará de nuevo el menú principal.

- **Datos identificativos: D**evolverá una cadena con la marca, modelo, matrícula y precio del vehículo.
- **Estado del vehículo**: Devolverá una cadena con el número de kilómetros, así como el contenido actual del depósito.
- **Viajar**: Se deberá comprobar si hay combustible suficiente para viajar los kilómetros solicitados, en caso contrario solo permitirá viajar hasta dejar el depósito a cero. Deberá actualizar el número de kilómetros del vehículo con los kilómetros efectivamente realizados (pueden ser menos de los solicitados) y restar del estado del depósito los litros consumidos en el viaje. Habrá que tener en cuenta que solo se podrán sumar kilómetros. Devolverá una cadena informando de los Km realmente realizados y de cómo queda el cuenta kilómetros tras el viaje.
- **Repostar**. Se deben incrementar al contenido del depósito los litros que se pide repostar, pero teniendo en cuenta que no se podrá sobrepasar la capacidad máxima del depósito. Es decir, si el coche tiene 50 l de capacidad y actualmente 20 l en el depósito, si se pide repostar 40 l solo se podrán incrementar 30 l más (50 que caben menos 20 que hay antes). El método devolverá como cadena el importe de la operación (litros realmente repostados por el precio del combustible).
- **Llenar**. Es similar a repostar, pero en este caso no se indican los litros ya que se trata de incrementar los litros que quepan hasta la capacidad máxima. Se puede hacer uso del método anterior o implementarlo completamente de nuevo. Como Repostar (…) también debe devolver una cadena con el importe de los litros repostados.
- **Actualizar precio combustible**. Debe permitir cambiar el dato precio_combustible. Hay que tener en cuenta que el precio de combustible puede tener decimales y que es el mismo para todos, por lo que la clase deberá contemplar este dato compartido para todos los objetos de la clase coche, sin almacenar una copia de este valor para cada coche.
- **Mostrar antigüedad**: devolverá en una cadena un mensaje informando del

número de años del vehículo desde que se matriculó, no la fecha de matriculación.

- **Mostrar propietario:** Devolverá el nombre del propietario del vehículo junto a su dni.
- **Salir**: El programa finalizará.

El proyecto de Netbeans constará de dos paquetes, donde se crearán las clases oportunas:

- **PROG05_Ejerc1**: que contendrá la clase **Vehículo** y la clase **Principal**.
- **PROG05_Ejerc1_util**: contendrá una **clase** con un **métodos estáticos** para realizar validaciones.

La clase Vehículo dispondrá de los siguientes métodos:

- **Constructor o constructores.**
- **Métodos get y set para acceder a sus propiedades.**
- **Método get_Anios():** Retorna un entero con el número de años del vehículo.
- **Método mostrarDatosIdentificativos()**
- **Método mostrarEstadoVehiculo()**
- **Método viajar(km)**
- **Método respotar(litros)**
- **Método llenar()**
- **Método actualizarPrecio(nuevoPrecio)**

TEN EN CUENTA

- En la clase vehículo no debe solicitar datos por teclado ni escribir datos en pantalla. Esas operaciones se realizarán en la clase Principal. Piensa por tanto en los tipos de datos que deben retornar los métodos para que el programa de la clase Principal pueda mostrar la información que se pide.
- En la clase Vehículo no se deben hacer validaciones de datos. Los datos se validan en la clase Principal y si son correcto, se instancia el objeto Vehículo.
- Debes incluir **una excepción** para la validación del DNI. Es decir, cuando no sea válido, se lanzará una excepción que se gestionará en la clase Principal, desde donde se mostrará el correspondiente mensaje de error.
- La aplicación solo trabajará con un vehículo, por lo tanto, solo utilizará una referencia a un objeto de tipo **Vehículo** en la clase **Principal**. Si existe un vehículo y el usuario selecciona **Nuevo Vehículo** en el menú, se perderá la información del vehículo existente y se guardará la del nuevo.
- No será necesario realizar comprobaciones de tipo en los datos solicitados por teclado.

- No se podrán mostrar datos de un vehículo si aún no se ha creado, y obviamente ninguna de las operaciones que cambian su estado (viajar, repostar, etc.): en ese caso habrá que mostrar un mensaje por pantalla.

Piensa en los modificadores de acceso que debes utilizar tanto en atributos y métodos de la clase.

Para poder explicar las decisiones de diseño tomadas, así como el funcionamiento del código, deberás realizar un vídeo de captura de pantalla en el que vayas explicando los detalles del diseño de la clase (atributos argumentando el tipo elegido y modificador si fuera el caso) así como los métodos (parámetros de entrada y valor retornado). Se trata de explicar el código como si lo estuvieras mostrando directamente al profesor. No hace falta editar el vídeo y no pasa nada si hay algún error que se rectifica a continuación. La tarea de hacer el vídeo no os debe llevar más tiempo que el de grabarlo una vez (tened antes preparado el código en Netbeans) y el de subirlo y compartirlo en Youtube. Es importante grabarlo directamente de pantalla, no utilizando un móvil o algo así ya que es frecuente que se pierda el enfoque y no se pueda ver con detalle el código.

IMPORTANTE

- En la cabecera de las clases añade documentación indicando autor y descripción de la clase.
- En la cabecera de cada método añade documentación indicando la funcionalidad que implementa y el valor que devuelve.
- El código fuente Java de esta clase debería incluir **comentarios** en cada atributo (o en cada conjunto de atributos) y método (o en cada conjunto de métodos del mismo tipo) indicando su utilidad.

MEJORA

- Una mejora consistiría en, cuando un dato solicitado no es correcto, mostrar un mensaje y volver a solicitarlo hasta que se introduzca correctamente.

Consejos y recomendaciones.

Para realizar la aplicación te realizamos la siguiente serie de recomendaciones:

- Básate en los diferentes ejemplos que has tenido que probar durante el estudio de la unidad. Algunos de ellos te podrán servir de mucha ayuda, así que aprovéchalos.
- Puedes crear las clases que creas conveniente para llegar a una solución estructurada (por ejemplo, crear la clase DNI).
- El ejercicio resuelto de la clase DNI, en el cual se hacen comprobaciones de entrada, puede servirte de base para lanzar excepciones cuando no se cumplen ciertas condiciones. Además, puedes utilizar ese código para realizar la validación el DNI del propietario del vehículo.
- En la Unidad 2, recuerda que hicimos una pequeña introducción al trabajo con cadenas de caracteres en Java.

- Si utilizas la clase Scanner, después de leer un entero lee una nueva línea para evitar errores de salto de línea. Es decir, para leer un entero siempre se deben ejecutar estas dos instrucciones:

valor=sca.nextInt();

sca.nextLine();

- Para trabajar con fechas, utiliza las clases disponibles en el API Java a partir de la versión 8 (LocalDate y LocalTime). En el siguiente enlace tienes ejemplos de uso que te servirán para desarrollar la tarea.

    - https://experto.dev/java-8-fechas-horas/
    - https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/time/LocalDate.html