Iván Estévez Sabucedo

Informe PROG05

Para crear el programa solicitado he dividido la tarea en tres partes.

- Estructura inicial
- Implementación de todos los requisitos
- Comprobación y corrección

Estructura inicial

Inicialmente se generan las tres clases (`Principal`, `Vehiculo` y `Validar`) y se introducen los datos básicos en cada una de ellas.

En `Vehiculo` hay dos consideraciones:
- Se crea un único constructor con todos los parámetros necesarios.
- No se crea ningún setter ya que únicamente se introducirán los datos del objeto al crearlo.
    - Únicamente se añade el método `addKilometros()` que nos permite incrementar en una cantidad dada los kilómetros del vehículo.

En `Validar` se incluye la validación de NIF, obtenida de los materiales del aula virtual. También se implementa el método `esMayorQueCero()` que valida que el número introducido sea mayor que cero.

También se crea `obtenerEnteroPorTeclado()` que comprueba que introducimos un número entero.

En `Principal` se escriben las instrucciones para instanciar el vehículo. Para mostrar el menú se crea un método propio, así como para la solicitud de fecha de matriculación y del incremento de kilómetros (opción 4 del menú).

Implementación de todos los requisitos

En esta parte se verifican todos los requisitos del enunciado. Opto por la mejora sugerida de que cuando un dato introducido no es correcto, se muestre un mensaje y se vuelva a solicitar.

También se incorpora el método `getAnios()` en la clase `Vehiculo` que permite conocer los años que han transcurrido entre la matriculación y la fecha actual. Se ha optado por emplear la clase `ChronoUnit`, pues permite calcular la diferencia de años entre dos fechas dadas.

Para almacenar la fecha se ha optado, tal y como se sugería en el enunciado, por emplear `LocalDate`, pues permite almacenar día, mes y año y su constructor comprueba que la fecha sea correcta (no permite introducir un 30 de febrero, por ejemplo).

Por último, se crea el método `actualizarKilometros()` que valida que el incremento es un entero mayor que cero. Posteriormente, este dato lo almacena en el objeto `Vehiculo` mediante el método `addKilometros()` arriba mencionado.

Comprobación y corrección

Una vez creado el programa, se han ejecutado diversas comprobaciones introduciendo datos erróneos para verificar que el comportamiento es el correcto.

Se realizan diversos cambios conceptuales. Por ejemplo, en el campo fecha únicamente se revisaba que la fecha no fuese igual a la actual (equals), pero eso dejaba la puerta abierta a que se pudiese introducir una fecha futura.

Otro error detectado fue no se habían implementado correctamente las excepciones, ya que al querer saltase un mensaje de error al querer consultar o modificar valores de un objeto cuando este todavía no está creado, salía una excepción `NullPointerException` que no se había previsto. Para corregirlo, se envolvió todo el menú en un `try/catch` que, a su vez, está en un bucle del que solo se sale cuando el usuario así lo solicita.

Una vez comprobado todo, se añadió la documentación requerida, se comentó el código y se escribió este informe.

Apuntes finales

Durante la realización de esta tarea he aprovechado para aprender sobre GitHub y he optado por sincronizar la tarea en este sistema, de forma que pueda realizar un seguimiento de los cambios y de la evolución del proyecto.

También he podido descubrir que `==` no sirve para comparar objetos.

Por último en el propio código incluí la validación de matrícula con el método `esMatriculaCorrecta()`, pero no la implementé ya que no permitiría introducir otro tipo de vehículos (guarda civil, motocicletas, etc.). De todos modos, el código aparece comentado para posibles mejoras sobre el mismo.