Índice

1. Introducción

1.0.1. Planteamiento del problema

La práctica se enfoca en aplicar el Lenguaje Unificado Modelado (UML) para visualizar y documentar de una forma clara un sistema de gestión de vehículos. Ademas de que implementa el manejo de excepciones en Dart para garantizar que la aplicación gestione los errores genrados por datos de entrada erroneos o fallos en la ejecución.

1.0.2. Motivación

La motivación de esta práctica es la necesidad de formalizar el diseño de una aplicación mediante los diagramas UML, facilitando la comunicación y documentacion de un sistema para un mejor desarrollo. Por otra parte, la correcta implementacion de las excepciones evita que en la lógica de la aplicación haya complicaciones imprevistas.

1.0.3. Objetivos

- Documentar la estructura de un sistema mediante un diagrama UML (de clases o estático) que muestre las clases, atributos, operaciones y relaciones estáticas.
- Documentar la estructura de un sistema mediante un diagrama UML (de secuencia o dinámica) que represente el flujo de interacción y mensajes entre objetos.
- Aplicar el concepto de excepciones dentro de un programa y comprender como las palabras clave se utilizan para validar los datos de entrada y prevenir el cierre de la aplicacion ante errores.

2. Marco Teórico

2.1. Diagrama UML (Lenguaje Unificado de Modelado)

[1] El **Lenguaje Unificado de Modelado** (UML, por sus siglas en inglés) es un lenguaje gráfico estándar y formal para la industria del software. No es un lenguaje de programación, sino un conjunto de *diagramas* que actúan como planos. Se utiliza para visualizar, especificar, construir y documentar los diferentes artefactos y vistas de un sistema de software complejo, facilitando la comunicación entre desarrolladores, analistas y clientes.

2.1.1. Diagrama de Clases

Es el diagrama más común de UML y pertenece a los diagramas de **estructura** (estáticos). Describe la estructura del sistema mostrando sus clases, atributos (datos), operaciones (métodos) y las relaciones estáticas entre ellas (como herencia, asociación, agregación o composición). [1]

2.1.2. Diagrama de Secuencia

Este es un diagrama de **comportamiento** (dinámico), enfocado en la interacción. Muestra cómo los objetos colaboran entre sí a lo largo del *tiempo*. Se centra en el orden cronológico (la secuencia) de los mensajes que se envían y reciben entre diferentes objetos (representados como "líneas de vida") para realizar una tarea o un caso de uso específico. [1]

2.2. Errores

[3] Una aplicación puede tener diversos tipos de errores, los cuales se pueden clasificar en tres grandes grupos:

- Errores sintácticos: fallas generadas por infringir las normas de escritura de un lenguaje o sintaxis
- Errores semánticos o lógicos: defectos más sutiles, se producen cuando la sintaxis del código es correcta, pero la semántica o significado no es el que se pretendía.
- ***Errores de ejecución:*** se presentan cuando la aplicación se está ejecutando e impiden que continúe con este proceso. Su origen es diverso, se pueden producir por un uso incorrecto del programa por parte del usuario , o se pueden presentar debido a errores de programación como acceder a localidades no reservadas o hacer divisiones entre cero, o debido a algún recurso externo al programa como acceder a un archivo o cuando se acaba el espacio en la pila o stack de la memoria.

2.2.1. Excepciones

[3] Una excepción es una condición anormal que cuando ocurre altera el flujo normal del programa en ejecución. Estos errores pueden ser generados por la lógica del programa, como un índice de un arreglo fuera de su rango, una división entre cero o errores generados por los propios objetos que denuncian algún tipo de estado no previsto o condición que no pueden manejar.
Beneficios:

- *Separación de lógica*: Permite separar el código de lo que el programa debe hacer del código de manejo de errores
- Las excepciones pueden "subir" a través de la pila de llamadas (call stack) hasta encontrar un método que sepa cómo manejarlas, centralizando la gestión de errores.
- Evita que el programa colapse ante datos de entrada incorrectos o fallos de recursos externos (como un archivo no encontrado)

2.2.2. Manejo de excepciones en Dart

[2] En Dart, las excepciones son objetos. El lenguaje proporciona las palabras clave *try, catch, on, finally y throw*:

- *throw:* Se utiliza para detener la ejecución y generar una excepción explícitamente cuando se detecta un problema.
- *try:* Define un bloque de código donde podría ocurrir una excepción.
- *catch:* Bloque que captura la excepción lanzada en el try y permite manejarla (por ejemplo, imprimiendo un error).
- *on* Se usa cuando se quiere capturar un tipo específico de excepción.

3. Desarrollo

Conceptos de la Programacion Orientada a Objetos en el ejemplo "**main.dart**":

1. **Abstracción**: Se define la *clase Vehiculo* como abstracta. No se pueden crear objetos de tipo "Vehículo", obligando a definir si es Auto, Moto o Camión. También se usa una interfaz **ServicioTaller**.

2. **Herencia**: Las clases Auto, Moto y Camion extienden (heredan) de Vehiculo, reutilizando los atributos marca, modelo y año

3. **Encapsulamiento:** Los atributos como **marca** o **anio** son privados (empiezan con guion bajo). El acceso a ellos se controla a través de Setters y Getters.

4. **Polimorfismo:** Cada subclase implementa de forma distinta los métodos @override calcularServicio() y @override generarReporteServicio()

5. **Manejo de excepciones:**

   a) **Validación en los Setters (Clases Vehiculo, Moto, Camion):**
   Con las validaciones en if, se revisan los atributos que se tratan de ingresar al constructor del Objeto, si al validarse no se cumple con las especificaciones de cada atributo, con un **throw** detiene la ejecución del bloque *try* donde se intentó crear el objeto, y con ArgumentError crea un objeto de error que leerá el bloque catch, que tiene como constructor el *mensaje* que indica la razón por la que no pudo construirse el objeto, permitiendo que el menú maneje el error elegantemente en lugar de cerrarse.

   b) **Manejo en el menú (funcion main):**
   En los case 1, 2 y 3:

   - El código intenta crear el objeto dentro de un bloque try
   - Si el usuario ingresa un dato no válido, el setter lanza la excepción
   - el bloque catch (e) atrapa ese error

   Entonces, en lugar de que el programa se cierre con un error rojo en la consola, el programa imprime el error correspondiente, y regresa al menú principal

3.1. Diagramas UML

A continuacion se muestran las representaciones de la estructura de los programas en diagramas UML estatico (clases) y dinamico (secuencia).

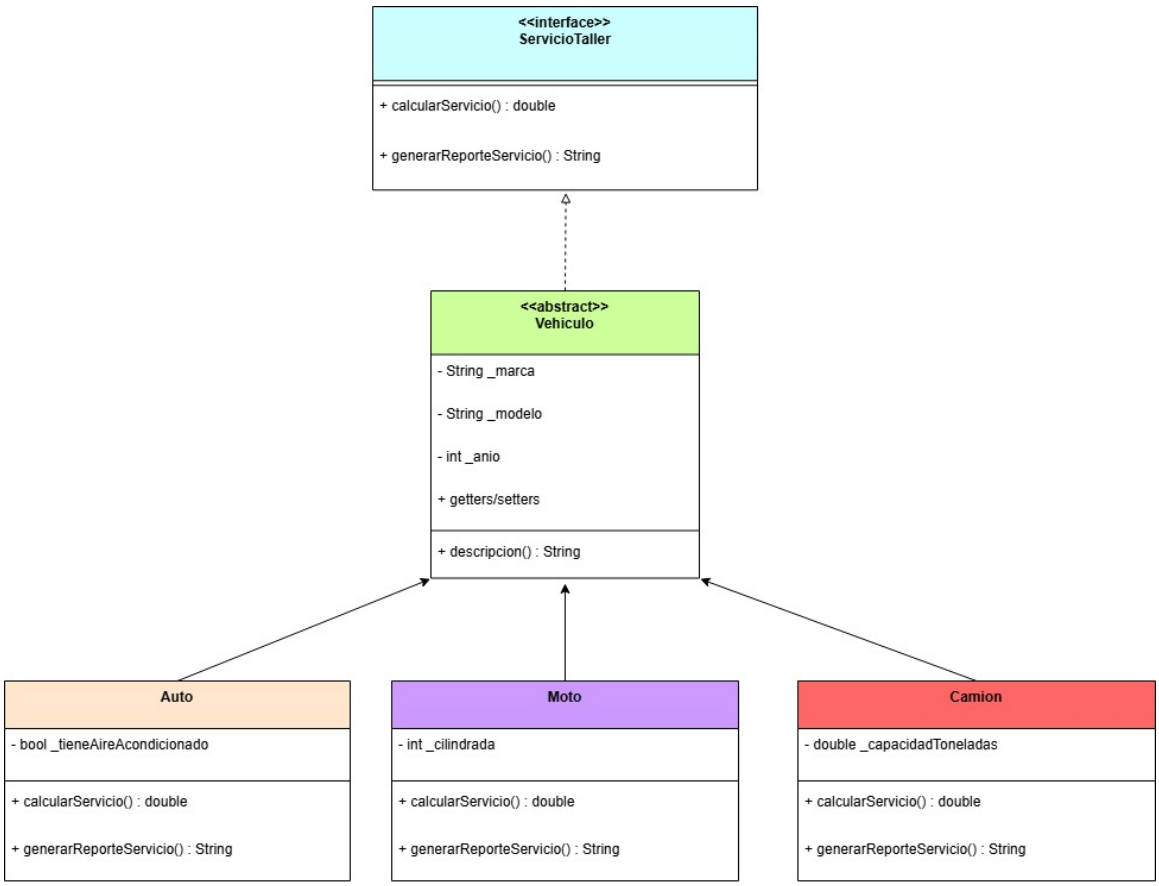

Figura 1: Diagrama de clases

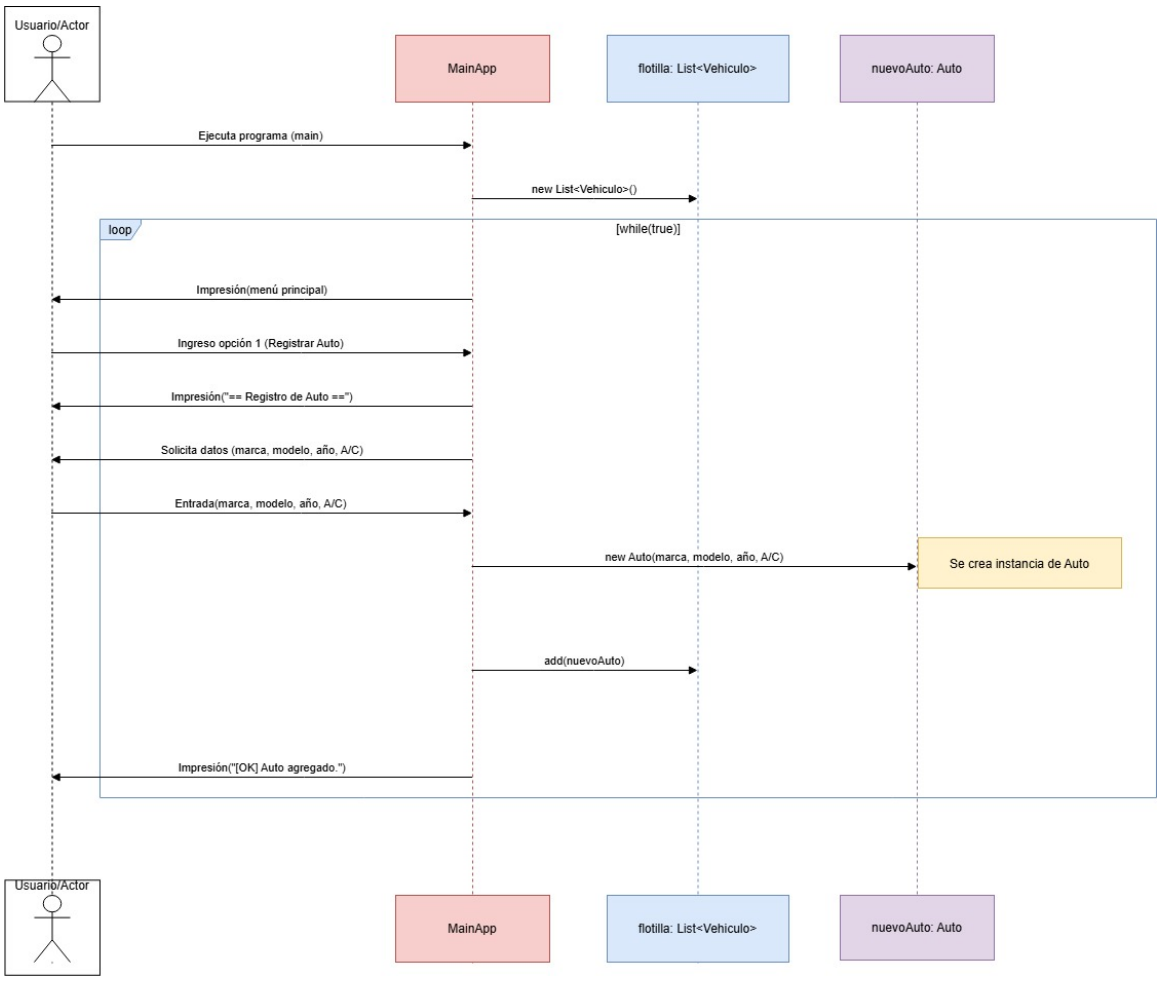

Figura 2: Diagrama de secuencia

4. Resultados

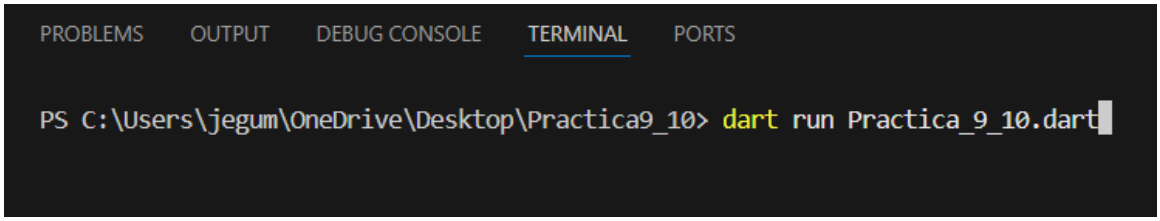

Figura 3: Comando de copilacion y ejecucion

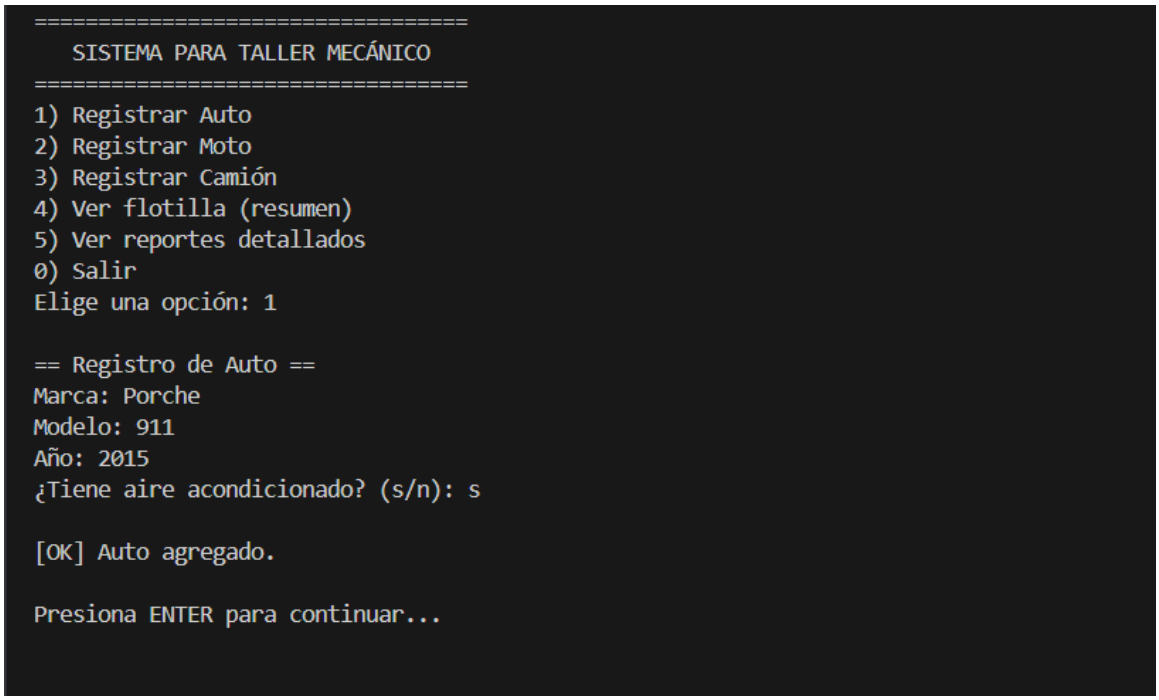

Figura 4: Ejemplo de ejecución para el caso de Registrar Auto.

```
==================================
   SISTEMA PARA TALLER MECÁNICO
==================================
1) Registrar Auto
2) Registrar Moto
3) Registrar Camión
4) Ver flotilla (resumen)
5) Ver reportes detallados
0) Salir
Elige una opción: 2

== Registro de Moto ==
Marca: Kawasaki
Modelo: H2
Año: 2024
Cilindrara (cc): 998

[OK] Moto agregada.

Presiona ENTER para continuar...
```

Figura 5: Ejemplo de ejecución para el caso de Registrar Moto.

```
==================================
   SISTEMA PARA TALLER MECÁNICO
==================================
1) Registrar Auto
2) Registrar Moto
3) Registrar Camión
4) Ver flotilla (resumen)
5) Ver reportes detallados
0) Salir
Elige una opción: 3

== Registro de Camion ==
Marca: Mercedes-Benz
Modelo: Atego
Año: 2020
Capacidad de carga (toneladas): 500

[OK] Camión agregado.

Presiona ENTER para continuar...
```

Figura 6: Ejemplo de ejecución para el caso de Registrar Camión.

```
==============================
   SISTEMA PARA TALLER MECÁNICO
==============================
1) Registrar Auto
2) Registrar Moto
3) Registrar Camión
4) Ver flotilla (resumen)
5) Ver reportes detallados
0) Salir
Elige una opción: 4

=== Flotilla registrada ===

[0] Auto: Porche 911 (2015) - A/C: sí | Servicio: $1250.00
[1] Moto: Kawasaki H2 (2024) - 998cc | Servicio: $650.00
[2] Camión: Mercedez-Benz Atego (2020) - Capacidad: 500.0 toneladas | Servicio: $3600.00
```

Figura 7: Salida del sistema mostrando los vehículos registrados.

```
================================
   SISTEMA PARA TALLER MECÁNICO
================================
1) Registrar Auto
2) Registrar Moto
3) Registrar Camión
4) Ver flotilla (resumen)
5) Ver reportes detallados
0) Salir
Elige una opción: 5

=== Flotilla registrada ===

Servicio para AUTO Porche 911:
- Año: 2015
- A/C: sí
- Total: $1250.00

Servicio para MOTO Kawasaki H2:
- Año: 2024
- Cilindrada: 998cc
- Total: $650.00

Servicio para CAMIÓN Mercedez-Benz Atego:
- Año: 2020
- Capacidad: 500.0 toneladas
- Total: $3600.00

Presiona ENTER para continuar...
```

Figura 8: Ejecución de la opción Ver reportes detallados.

```
==================================
   SISTEMA PARA TALLER MECÁNICO
==================================
1) Registrar Auto
2) Registrar Moto
3) Registrar Camión
4) Ver flotilla (resumen)
5) Ver reportes detallados
0) Salir
Elige una opción: 0

Saliendo del sistema. Buen día.
```

Figura 9: Finalización del programa desde el menú principal.

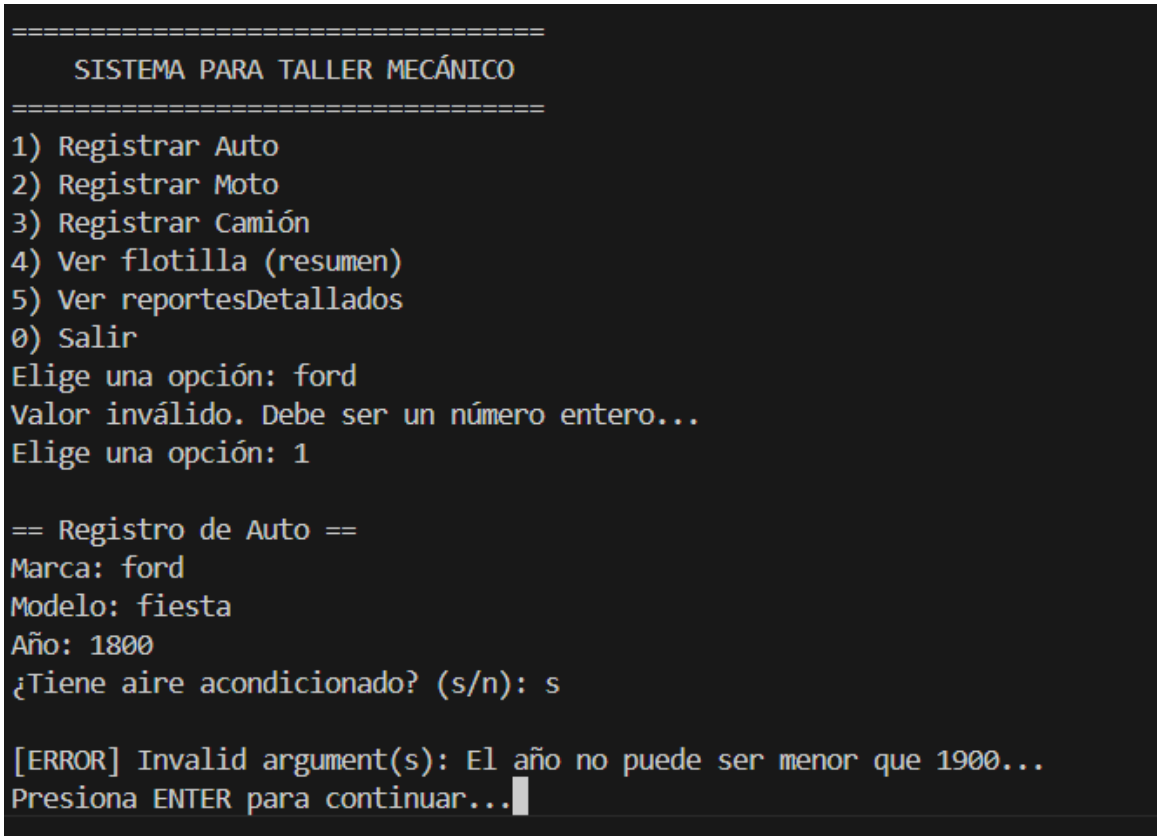

Figura 10: Error en la ejecucion para el caso uno

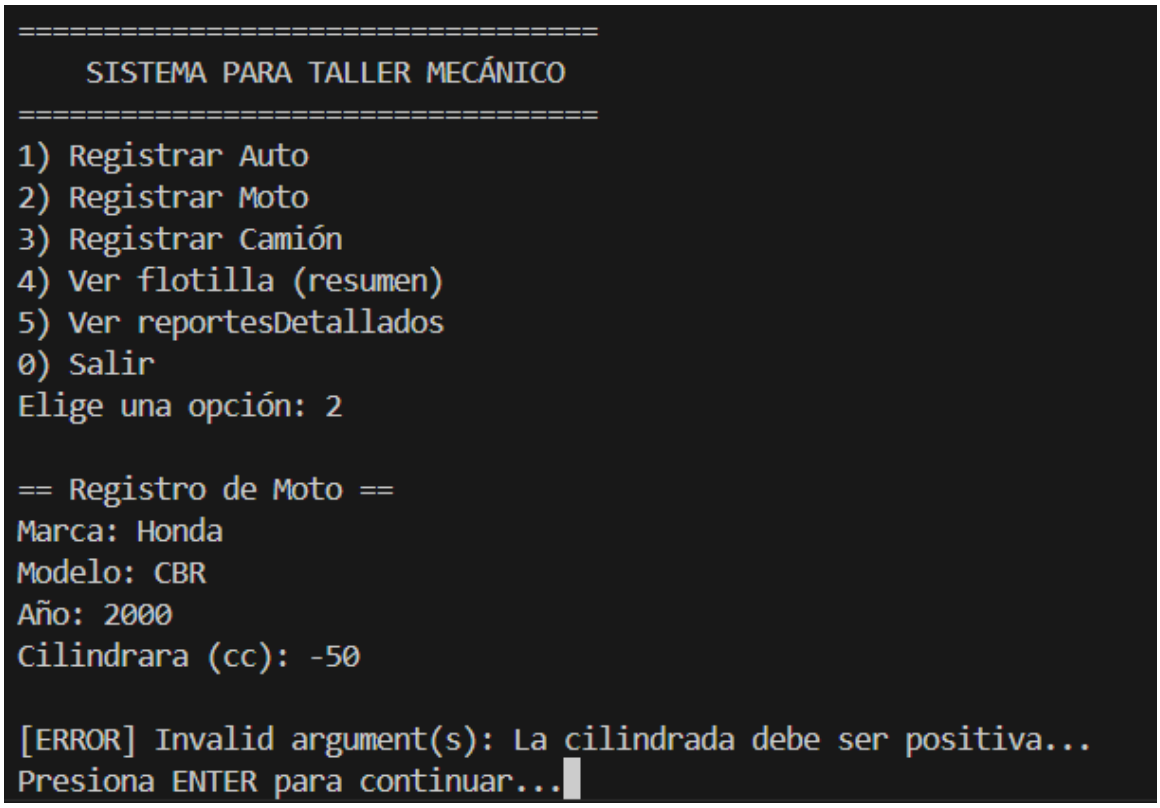

Figura 11: Error en la ejecucion para el caso dos

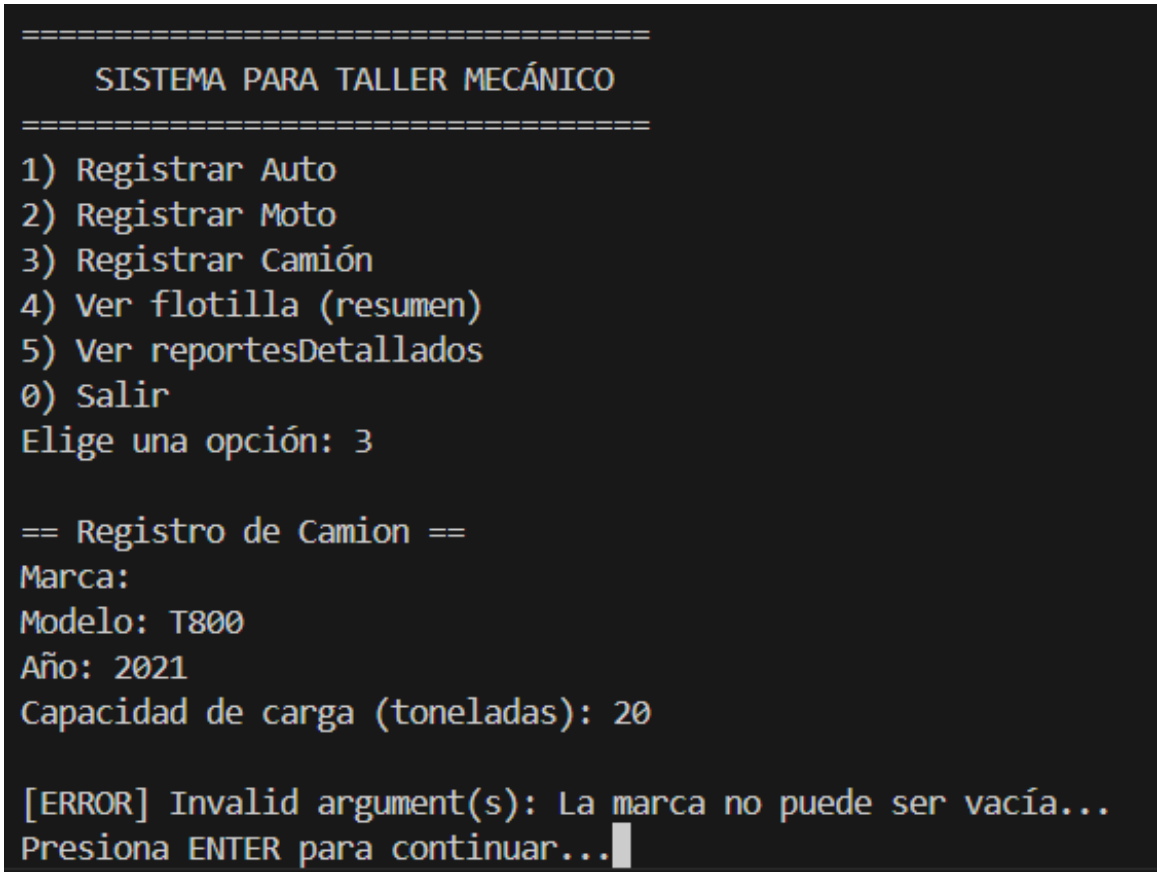

Figura 12: Error en la ejecucion para el caso tres

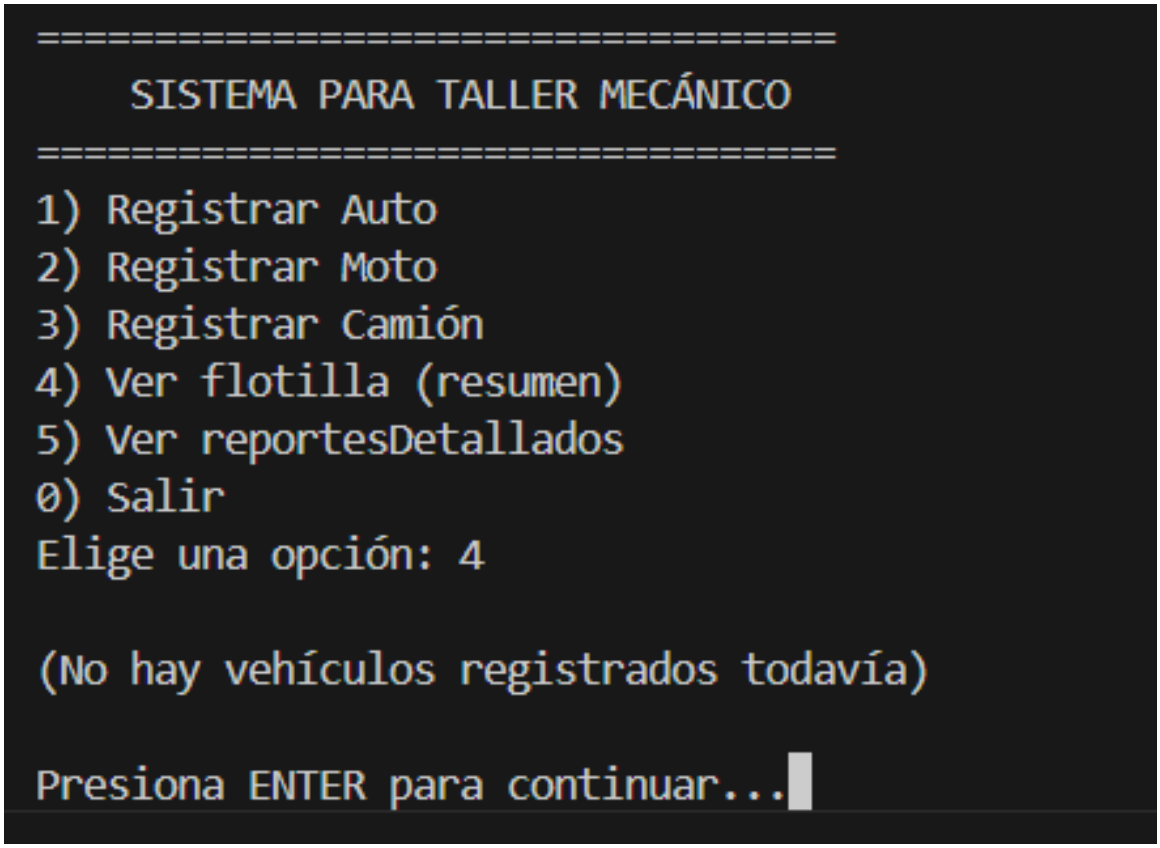

Figura 13: Error en la ejecucion para el caso cuatro

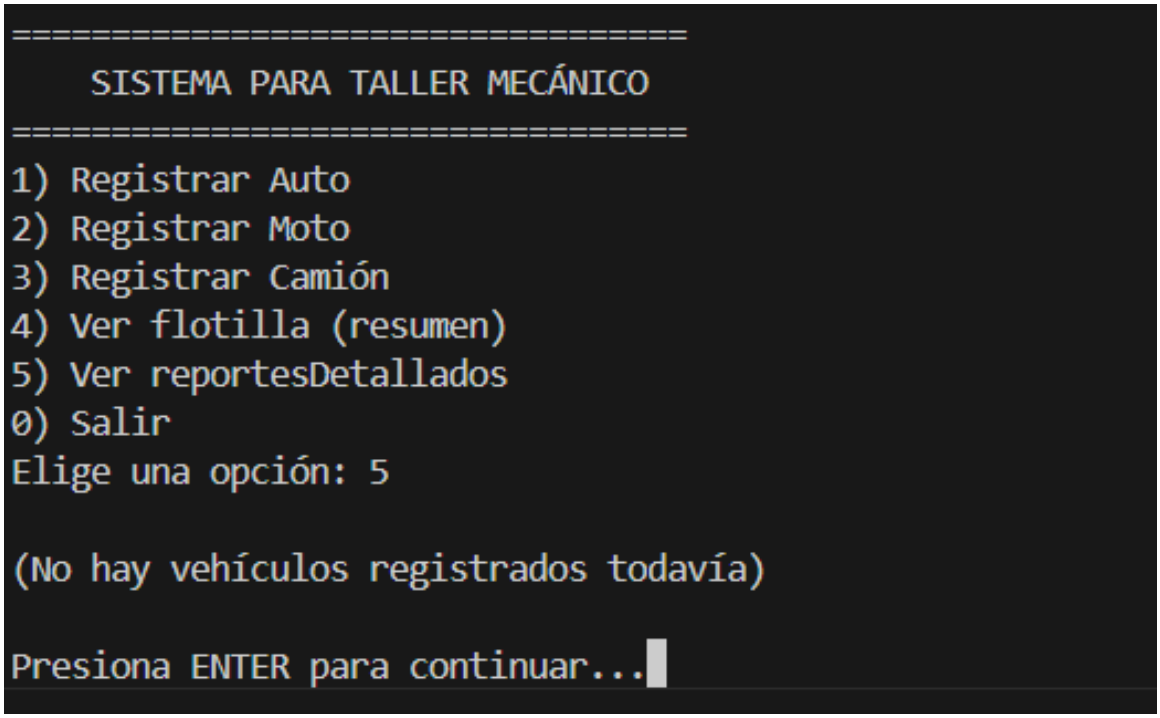

Figura 14: Error en la ejecucion para el caso cinco

5. Conclusiones

Podemos concluir que los objetivos fueron cumplidos, ya que logramos documentar la estructura y el comportamiento del sistema mediante la elaboración de los diagramas UML, el **diagrama de clases** representa la estructura del programa, mostrando las clases, la interfaz y las relaciones estáticas como la herencia y la implementación de la interfaz. El **diagrama de secuencia** representa el comportamiento del programa, ya que modela la interacción y el flujo de mensajes al registrar un nuevo vehículo.
Por otra parte, comprendimos de una mejor manera como se usa el mecanismo de excepciones de Dart como: try, throw, catch. Las validaciones en los setters lanzan una excepción explicíta *(throw ArgumentError)* cuando se intenta crear un objeto con datos inválidos. Mientras que la función *catch* "captura" esas excepciones dentro de un bloque *try*, esto permite que el programa imprima el mensaje de error y evita el cierre inesperado del menú principal por tener datos de entrada incorrectos.

6. Referencias

[1] J. Avila, J. y Bailón. Diagramas uml. en análisis y diseño en poo. (https://portalacademico.cch.unam.mx/cibernetica1/analisis-y-diseno-en-poo/diagramas-UML), 2022. Consultado el 15-11-2025.

[2] The Dart Project Authors (Google). Dart language documentation: Error handling. (https://dart.dev/language/error-handling), 2024. Consultado el 15-11-2025.

[3] J. A. Lozano. Excepciones y errores. (https://repositorio-uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/3085/mod$_r esource/content/1/UAPA-Excepciones-Errores/index.html), 2020. Consultadoel 15-11-2025.