	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 7

No de Práctica(s): Práctica 9 y 10

Integrante(s): 322044339

322094152

322114461

425093384

322150984

No. de brigada: 6

Semestre: 2026-1

Fecha de entrega: 16 de noviembre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

En muchos talleres mecánicos, llevar un registro ordenado de los vehículos que ingresan y de los trabajos que se les realizan puede volverse complicado cuando no existe un sistema que concentre toda la información. Sin una organización mínima, cada vehículo termina manejándose de manera distinta y es común que haya errores en los costos, confusiones al capturar datos o resultados poco claros al momento de atender al cliente.

Con el fin de evitar estos problemas, en esta práctica se desarrolla un pequeño sistema en Dart que permite registrar distintos tipos de vehículos usando los principios básicos de la Programación Orientada a Objetos. Para ello se parte de una clase abstracta que reúne los elementos generales y de la cual se derivan las clases que representan autos, motos y camiones. Cada una define su forma de calcular el costo del servicio y de generar un reporte, lo que permite tratarlas de manera uniforme sin perder sus diferencias.

Además, el sistema revisa la información que el usuario va capturando —por ejemplo, la marca, el modelo, el año o la cilindrada— antes de aceptarla. Gracias a estas revisiones evitamos que entren datos que no tengan sentido o no coincidan con el tipo de dato que debería ser ingresado y el programa puede avisar de inmediato cuando alguna entrada no coincide. Esto ayuda a que todo funcione con mayor estabilidad y disminuye el riesgo de guardar información errónea.

El propósito general es contar con una herramienta sencilla que permita agregar vehículos, mostrar su información y obtener reportes de servicio de manera clara. La estructura modular también facilita ampliar el sistema en prácticas posteriores.

322044339

2. Marco Teórico

2.1. Flutter

Flutter es un framework de código libre desarrollado por Google para construir aplicaciones multiplataforma compiladas de forma nativa, a partir de una sola base de código, esto quiere decir que no necesitas programar una nueva aplicación para cada plataforma, convirtiendo el programa escrito en lenguaje Dart en lenguaje máquina.[3]

2.2. Dart

Dart es un lenguaje optimizado para el desarrollo de aplicaciones rápidas en cualquier plataforma; su objetivo es ofrecer al cliente un lenguaje productivo para el de-

sarrollo multiplataforma junto con una plataforma flexible de ejecución, pues permite compilar tu código de dos formas distintas.

1. Plataforma nativa

   Para aplicaciones dirigidas a dispositivos móviles o de escritorio (Dart incluye una máquina virtual con compilación "just-in-time" y compilación "ahead-in-time" para poder producir código máquina).

2. Plataforma web

   Para aplicaciones dirigidas a la web (El compilador web, traduce el código Dart en JavaScript o WebAssembly).

Además, Dart es un lenguaje seguro, que utiliza comprobación de tipo estático para garantizar que el valor de una variable siempre corresponda con su tipo estático, sin embargo su sistema de typing es flexible, permitiendo el uso de un "dynamic" un tipo de dato que realiza comprobaciones de tipo en tiempo de ejecución, también, el lenguaje Dart impone una seguridad estricta de valores nulos que evita errores que resultan del acceso no intencional de las variables establecidas en null, en otras palabras, los valores no pueden ser nulos a menos que se indique que pueden serlo, a diferencia de otros lenguajes de seguridad estricta de valores nulos, cuando Dart determina que una variable no es anulada, esa variable nunca puede ser nula, protegiéndonos de excepciones nulas en tiempo de ejecución.[2]

2.3. Manejo de excepciones

Una excepción es la indicación de que se produjo un error en el programa. Las excepciones, se producen cuando se ejecuta un método que no concluye su trabajo correctamente, y en su lugar termina de forma excepcional como consecuencia de una situación que no se esperaba.

Cuando se produce dicha situación debemos saber manejarla para evitar que nuestro programa se detenga, causando confusión en el usuario que pruebe nuestra aplicación.

El proceso por el que pasa un lenguaje de programación al lidiar con una excepción es el siguiente: Al detectarse un error, el computador genera un objeto de una clase específica de excepciones, dicho objeto, contiene toda la información sobre el problema a partir del punto donde el programa genero el objeto, después, se "lanza" el objeto con la esperanza de que alguien lo atrape, si no se atrapa, el programa finaliza su ejecución y en la consola aparecerá toda la información contenida en el objeto.

Para evitar que nuestro programa se detenga cuando ocurre una excepción, Dart nos ofrece la estructura try-catch, el cual funciona de la siguiente forma:

Dentro del bloque de código contenido en "try" se escribe una parte del código que se cree que podría ocasionar una excepción. Por otra parte dentro de catch, primeramente puedes especificar el tipo de excepción que esperas recibir, aunque esto no es necesario, además puedes pasarle una variable como argumento para guardar

el error que se genere aunque, nuevamente, esto no es necesario, y por ultimo dentro del bloque de código de "catch" se determina la acción que se realizara al ocurrir la excepción, en pseudocódigo se vería de la siguiente manera:[1]

```
try {
    // Código que podría causar una excepción
}
catch (e) {
    // Acción a realizar cuando ocurre la excepción
}
```

322150984

3. Desarrollo

En esta ocasión se analizó un código en lenguaje Dart y haciendo uso de Flutter dedicado a un taller, siendo un proyecto útil para ver los conceptos de programación orientada a objetos en un nuevo lenguaje a la vez que explotamos este paradigma con un ejemplo realista.

3.1. taller.dart

El código cuenta con 5 clases, de las cuales la clase **ServicioTaller** es abstracta, lo que significa que más tarde la sobrescribiremos, esta clase tiene dos métodos, calcularServicio() y generarReportesServicio(). La clase **Vehiculo** implementa a **ServicioTaller** pero también es abstracta, esta servirá como clase padre o molde para crea tres subclases a través de extends, esta clase tiene tres atributos, siendo estos *marca*, *modelo* y *anio* las cuales están protegidas por un getter y un setter en el cuál hay un manejo de excepción en el que salta un aviso cuando alguno de estos datos falta en un Vehículo, también tiene un método String description() para mostrar los datos antes mencionados. Los hijos de **Vehiculo** son **Auto**, **Moto** y **Camion**, dado que si bien todos son vehículos tienen diferencias aunque usar un molde para crearlas es más eficiente y una característica del paradigma.

La clase **Auto** tiene un atributo booleano para denotar si tiene aire acondicionado o no obteniéndolo a través de un get y usando un método set. Después para sus métodos heredados utiliza la sobre escritura con @Override para calcularServicio(), generarReporteServicio y descripción() siendo que esta última pertenece a Vehiculo y no a servicioTaller, dándonos como resultado nuestra descripción y un reporte. Por su parte la clase **Moto** hija de **Vehiculo** sigue una estructura bastante similar a **Auto**, solo que en lugar de ver si es que tiene aire acondicionado, esta maneja un booleano *cilindrada* que verifica si es cilindrada y de cuanto, es este caso al momento del set compara si esta es menor o igual a cero, puesto que el número debe ser positivo y si no lo es hay que manejar la situación con un aviso, de ahí en fuera hace la misma operación de sobrescribir métodos ajustado a lo que una moto necesita como el ajuste

de cadena o el recargo de alto desempeño, cosas que si son verdaderas entonces se cargaran a la cuenta. Por último en lo que respecta a clases tenemos a la subclase camión el cuál ahora tiene un atributo double para su capacidad en toneladas, y como las toneladas no pueden ser negativas tenemos que manejar con un mensaje si es que se llega a dar el caso de un valor de ese tipo en el set, dentro del cálculo del servicio encontramos variables diferentes siendo la revisión de frenos, de suspensión y si el atributo capacidadToneladas es mayor que 10, lo que significará otro cargo.

Para leer desde CLI o Command Line Interface, establecemos ciertos métodos para darle el formato o capacidades necesarias para que se recojan datos adecuadamente, todas tienen como parámetro (String Promt). La función leerLinea muestra un mensaje, lee una línea escrita por el usuario y lo devuelve con un ternario. LeerEntero que recoge un entero, si no lo obtiene lo vuelve a pedir y si no es un entero manda un mensaje solicitándolo así, por lo que maneja la excepción de otros tipos de datos, LeerDouble hace lo mismo pero con datos con punto flotante. Por su parte leerBollSN transforma un carácter 's' a True y 'n' a False y si se ingresa otra cosa manda un mensaje de error.

Ahora siguen los métodos para registrar objetos **Vehículo**, o en estos casos unos Autos crearAutoInteractivo(), Motos crearMotoInteractiva() o Camiónes crearCamionInteractivo(), estos están hechos como formularios para que el usuario ingrese datos por lo que los atributos usan métodos para leer los datos ingresados y previniendo cualquier mal uso o error como introducir tipos de datos erróneos o cadenas vacías.

Para mostrar el listado básico se usa un método vacío con el parámetro de la lista de vehículos, por lo que primero ve si es que la flotilla o nuestro taller está vacío, si lo está manda un mensaje, si no, con una iteración nos muestra la descripción y el costo total. Si lo queremos detallado se usa otro método el cual itera mostrando el método reporte de servicio.

En el main se crea la lista vacía de vehículos el cuál recibe el nombre de flotilla. Se crea un clásico menú con un while true y las opciones de registrar un auto, moto, camión, ver la flotilla, ver el reporte detallado y salir, si la opción es cero simplemente sale, pero si no actúa de acuerdo al caso aunque en los primeros tres se auxilia de un try-catch para manejar errores de ejecución o excepciones, después se crea un objeto vehículo dependiendo del caso y se añade a la flotilla, en los otros dos casos muestra listados específicos. Finalmente el sistema hace una pausa intermedia antes de volver a mostrar el menú.

3.2. Diagrama estático

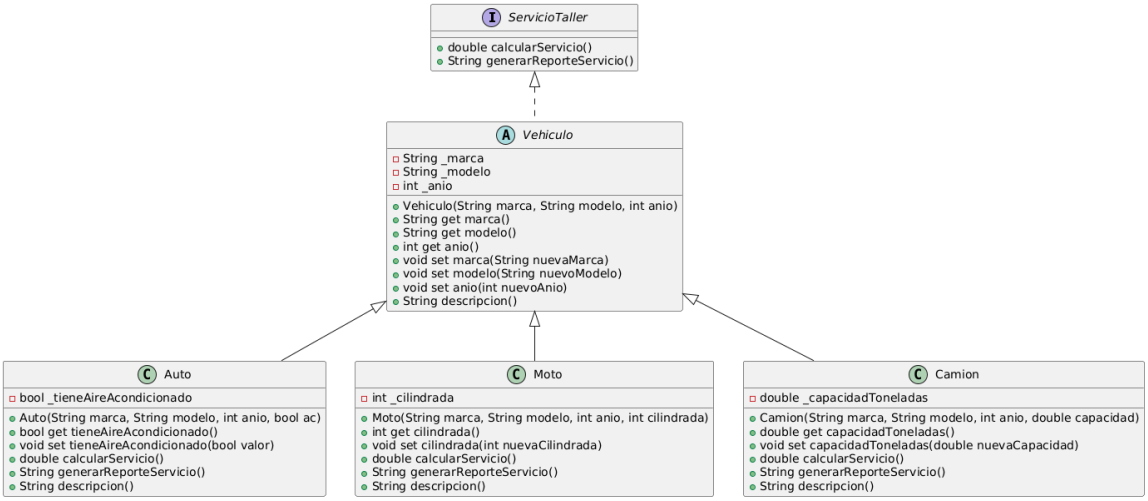


Figura 1: UML Diagrama de clases

3.3. Diagrama dinámico

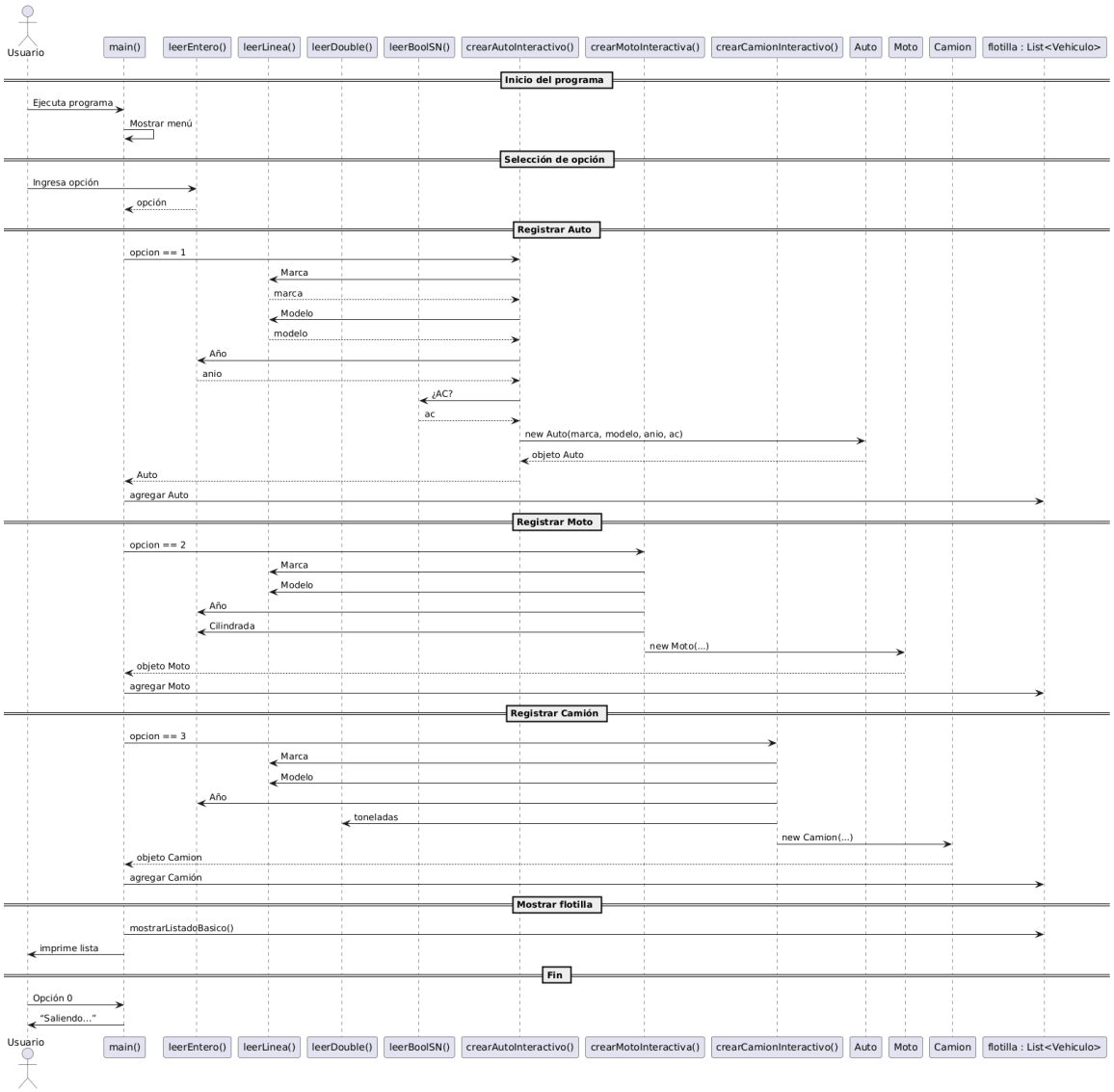


Figura 2: UML Diagrama de secuencia

* Recomendamos hacer zoom para una mejor visualización

322094152

4. Resultados

4.1. Menu inicial

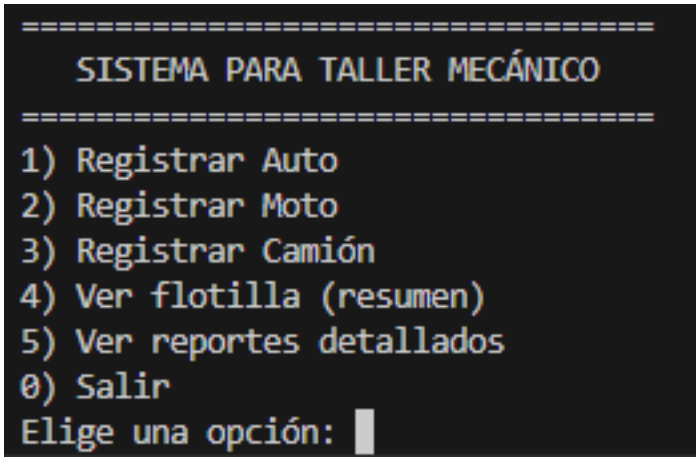


Figura 3: Interfaz de inicio

Una vez se compila y se ejecuta el programa, se muestra el siguiente menú inicial solicitando que se elija la opción deseada.

4.2. Registrar Auto

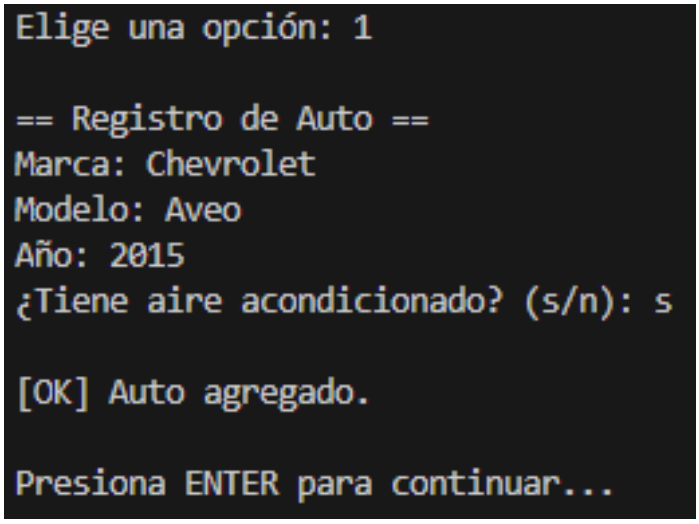


Figura 4: Ingreso de los datos del auto a registrar

Al seleccionar la opción 1, se solicita al usuario ingresar los datos del auto que se desea registrar, se registraron 3 autos en total en esta práctica, cada uno con sus datos correspondientes.

4.3. Vuelta al menú

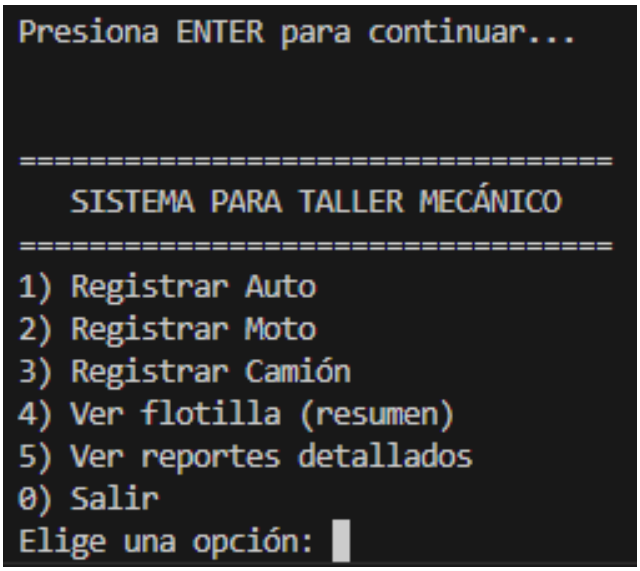


Figura 5: Menú inicial de vuelta

Al terminar de registrar cualquier vehículo se solicita presionar ”ENTER” para avanzar, al realizar esto el programa nos devolverá al menú de inicio.

4.4. Registrar moto

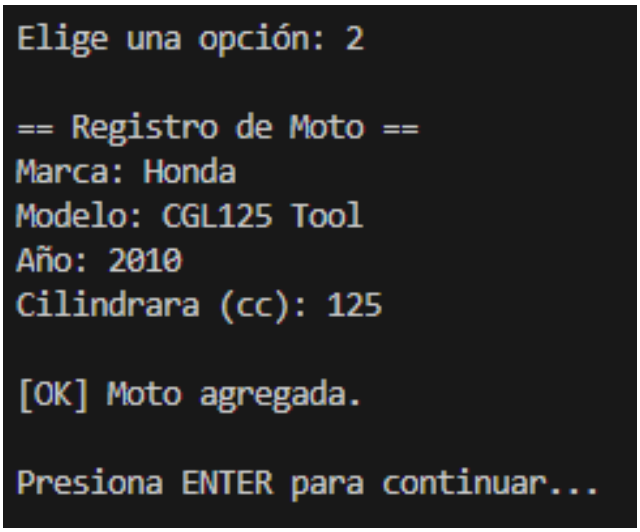


Figura 6: Ingreso de los datos de la moto a registrar

Al seleccionar la opción 2, se solicita al usuario ingresar los datos de la moto que se desea registrar, se registraron 3 motos en total en esta práctica, cada una con sus datos correspondientes.

4.5. Registrar camión

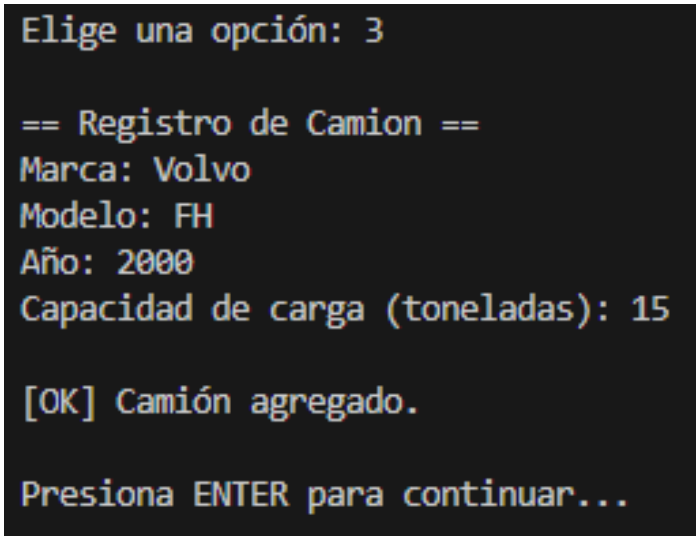


Figura 7: Ingreso de los datos del camión a registrar

Al seleccionar la opción 3, se solicita al usuario ingresar los datos del camión que se desea registrar, se registraron 3 camiones en total en esta práctica, cada uno con sus datos correspondientes.

4.6. Ver flotilla (resumen)

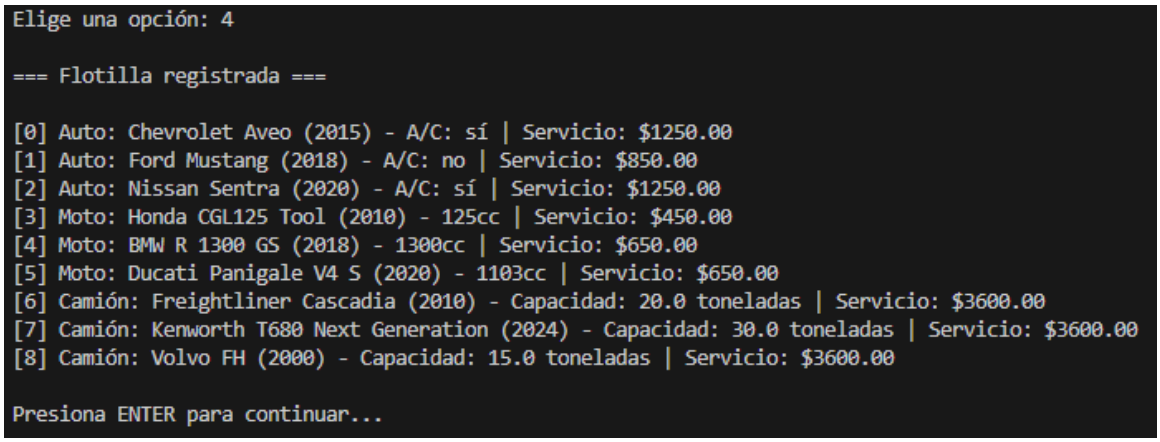


Figura 8: Resumen de los vehículos ingresados

Al seleccionar la opción 4, se muestran los datos de todos los vehículos registrados, así como el precio de su servicio.

4.7. Ver reportes detallados

```
Elige una opción: 5

=== Flotilla registrada ===

Servicio para AUTO Chevrolet Aveo:
- Año: 2015
- A/C: sí
- Total: $1250.00

Servicio para AUTO Ford Mustang:
- Año: 2018
- A/C: no
- Total: $850.00

Servicio para AUTO Nissan Sentra:
- Año: 2020
- A/C: sí
- Total: $1250.00
```

(a) Detalles de los autos

```
Servicio para MOTO Honda CGL125 Tool:
- Año: 2010
- Cilindrada: 125cc
- Total: $450.00

Servicio para MOTO BMW R 1300 GS:
- Año: 2018
- Cilindrada: 1300cc
- Total: $650.00

Servicio para MOTO Ducati Panigale V4 S:
- Año: 2020
- Cilindrada: 1103cc
- Total: $650.00
```

(b) Detalles de las motos

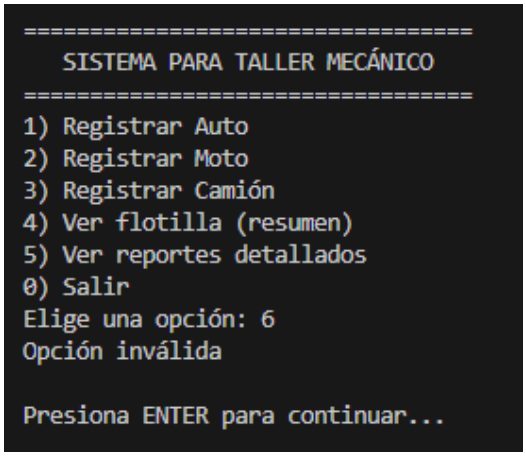
```
Servicio para CAMIÓN Freightliner Cascadia:
- Año: 2010
- Capacidad: 20.0 toneladas
- Total: $3600.00

Servicio para CAMIÓN Kenworth T680 Next Generation:
- Año: 2024
- Capacidad: 30.0 toneladas
- Total: $3600.00

Servicio para CAMIÓN Volvo FH:
- Año: 2000
- Capacidad: 15.0 toneladas
- Total: $3600.00

Presiona ENTER para continuar...
```

(c) Detalles de los camiones

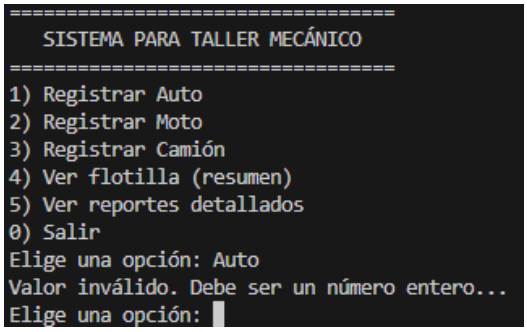
Figura 9: Reporte detallado de los vehículos

Al seleccionar la opción 5 se despliega un reporte de los vehículos especificando marca, modelo, año, aire acondicionado, cilindrada o capacidad (según sea el caso) y el precio total del servicio

4.8. Entrada inválida



(a) Número de opción no válido



(b) Tipo de entrada inválido

Figura 10: Ingreso de una entrada no válida y mensaje de error

Al ingresar una opción no válida o una entrada que no coincide con el tipo de dato que se debería ingresar, se muestra el mensaje de error correspondiente

4.9. Salida del programa

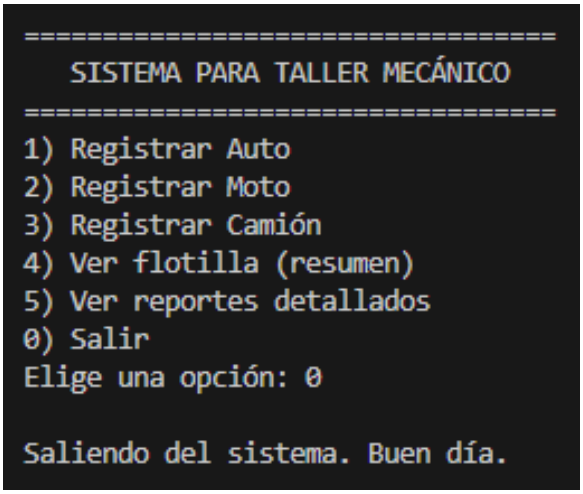


Figura 11: Termino de la ejecución del programa

Al seleccionar la opción 0, se muestra un mensaje de salida y el programa termina su ejecución

322114461

5. Reto

https://vt.tiktok.com/ZSf1MNF7s/

6. Conclusiones

La implementación del sistema de registro de vehículos permitió aplicar de manera íntegra los principios del lenguaje Dart, identificando sus similitudes y diferencias con respecto al lenguaje utilizado antes de la migración, Java. Mediante el uso de clases abstractas, herencia, encapsulamiento y *overriding* de métodos, fue posible comprender cómo una estructura modular puede representar distintos tipos de vehículos mientras conserva una interfaz común. Esto demuestra de forma práctica cómo el paradigma orientado a objetos facilita la construcción de software extensible y organizado.

Asimismo, el proyecto funcionó como una introducción al manejo de excepciones, mostrando la importancia de anticipar escenarios no controlados y garantizar que el programa responda adecuadamente ante entradas inválidas. Con ello se refuerza la relevancia de diseñar aplicaciones que mantengan estabilidad operativa y brinden retroalimentación clara al usuario.

En conjunto, la práctica permitió consolidar el entendimiento del paradigma orientado a objetos dentro de un nuevo lenguaje, evidenciando que, aún con cambios en la plataforma o sintaxis, los principios fundamentales estudiados a lo largo del curso continúan siendo aplicables y esenciales para el desarrollo de soluciones.

425093384

Referencias

[1] Universidad de los Andes. *Manejo de las excepciones*. [Online; accessado: 16-Nov-2025]. n.d. URL: `https://universidad-de-los-andes.gitbooks.io/fundamentos-de-programacion/content/Nivel4/5_ManejoDeLasExcepciones.html`.

[2] Dart. *Dart Language Documentation*. [Accessed: 2025-11-16]. 2025. URL: `https://dart.dev/docs`.

[3] Flutter. *Flutter — Build apps for any screen*. [Online; accessed: 16-Nov-2025]. n.d. URL: `https://flutter.dev/`.