	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: POO

Grupo: 01

No de Práctica(s): Practica 9 y 10

Integrante(s): 322125337

322080869

322059179

323629814

322113536

No. de lista o brigada: 6

Semestre: 2025

Fecha de entrega: 16/11/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del problema**: La práctica se centra en analizar una aplicación existente desarrollada en Dart el nuevo lenguaje que estamos implementando que simula un sistema de gestión para un taller mecánico. El sistema maneja diferentes tipos de vehículos como 'Auto', 'Moto' y 'Camion'. Nuestro propósito es elaborar un reporte que documente esta aplicación, creando diagramas UML (uno estático y uno dinámico) y proporcionando una interpretación de los conceptos de programación utilizados, con un enfoque específico en el manejo de excepciones.

- **Motivación**: Analizar una aplicación funcional es fundamental para comprender cómo los conceptos teóricos antiguos vistos en clase se implementan en un caso real y el como todo lo que hemos aprendido podemos aplicarlo sin importar el lenguaje con el que estemos trabajando. Además, estudiar el manejo de excepciones nos permite entender cómo construir aplicaciones robustas y fiables que gestionan errores de forma controlada. [main.dart]

- **Objetivos**:

  - Elaborar un diagrama UML estático y dinamico, como un diagrama de clases y de secuencia, que represente la estructura, relaciones entre las clases y el flujo del código proporcionado.
  - Analizar e interpretar cómo se aplica el concepto de excepciones en el código Dart para validar los datos de entrada en los constructores y setters.

2. Marco Teórico

Trabajado por Juan Prado

Flutter: Flutter es un kit de desarrollo de software (SDK) de código abierto creado por Google. Es un framework que utiliza el lenguaje de programación Dart. Su principal objetivo es permitir a los desarrolladores construir aplicaciones compiladas nativamente para múltiples plataformas a partir de una única base de código.[1]

Dart:Dart es el lenguaje de programación creado por Google que sirve como base para el framework Flutter, aunque también se puede usar para desarrollo web y de servidor. Es un lenguaje orientado a objetos, de tipado fuerte. Dart está optimizado para el desarrollo de interfaces de usuario, ofreciendo características como la compilación "Just-In-Time"(JIT) durante el desarrollo que habilita el "Hot Reload"de Flutter y la compilación .^head-Of-Time"(AOT) a código nativo rápido (ARM o x64) para la versión de producción, lo que garantiza un alto rendimiento [2]

Excepciones: Las Excepciones son eventos anómalos o errores que ocurren durante la ejecución de un programa y que interrumpen el flujo normal de las instrucciones. Representan situaciones inesperadas, como intentar dividir por cero, acceder a un archivo que no existe, o intentar leer datos de una conexión de red fallida. En lugar de permitir que el programa se detenga abruptamente, los lenguajes de programación proporcionan un mecanismo para gestionar estos errores de forma controlada. Esto sería como tal el manejo de excepciones, y típicamente utiliza bloques de código como try (para envolver el código que podría fallar), catch (para .ªtraparz ejecutar un plan de contingencia si ocurre un error específico) y finally (para ejecutar código de limpieza, como cerrar un archivo, sin importar si hubo o no una excepción).[3]

3. Desarrollo [Analisis]

Trabajado por Edgar Saúl Bautista y Said Correa Sam

Interpretacion de conceptos aplicados en el codigo.:

Los conceptos y temas vistos a lo largo del curso se veran reflejados en esta nueva herramienta llamada dart. Cómo referencia usaremos el ejemplo del código main.dart visto anteriormente en clase donde se interpretan los conceptos del curso aplicados al código.

En primer lugar se observa el uso de clases, objetos, encapsulación y abstracción (temas ya antes vistos). La abstracción se refleja en la interfaz "ServicioTaller"que define el comportamiento general esperado para cualquier tipo de vehículo en el taller y también en la clase abstracta "Vehiculo"que reúne atributos y métodos comunes a todos los tipos de vehículos. Luego tenemos la encapsulación la cual se logra mediante atributos privados acompañado de getters y setters que incorporan validaciones para mantener bien los datos.

También está presente el control de flujo y los tipos de datos donde el programa utiliza estructuras como condicionales (if, switch) y ciclos (while, for) además de trabajar con tipos básicos como String, int, double y bool, y colecciones como listas genéricas (List<Vehiculo>), esto permite gestionar la interacción con el usuario, validar entradas y operar sobre la flotilla de vehículos de manera dinámica y segura.

La herencia y el polimorfismo de igual forma están presentes más que nada. Las clases Auto, Moto y Camion extienden a la clase abstracta Vehiculo heredando sus atributos y comportamiento básico pero redefiniendo métodos como calcularServicio, generarReporteServicio y descripcion haciendo uso del @override. Ya con esto se ve claramente reflejado el polimorfismo ya que diferentes tipos de vehículos pueden ser tratados de manera uniforme a través de referencias del tipo Vehiculo pero responden de forma distinta según su implementación particular.

El manejo de excepciones también se encuentra implementado de manera correcta ya que se utilizan tanto excepciones lanzadas explícitamente mediante throw ArgumentError(...) en los setters, como bloques try/catch al momento de registrar nuevos vehículos, lo cual permite controlar errores y así poder evitar fallos inesperados durante la ejecución del programa.

El flujo de entrada y salida se realiza mediante la librería dart:io capturando datos introducidos por el usuario a través de stdin.readLineSync() y mostrando información por consola mediante stdout.write y print, así se complementa la parte interactiva del sistema permitiendo la recolección de datos y la presentación de reportes.

También es importante mencionar que el código ya presenta elementos cercanos a patrones de diseño especialmente el patrón Factory, pues las funciones que crean vehículos específicos funcionan como métodos de construcción especializados.
De igual forma se observa un uso implícito de un patrón de comportamiento similar a Strategy al permitir que cada tipo de vehículo defina su propia lógica de cálculo de servicio.

Por último la estructura de clases e interfaces presentada podría representarse sin dificultades mediante diagramas UML.

Enfasis en interpretacion de concepto de excepciones:
El programa maneja exepciones utilizando try-catch (aunque tiene errores)

Al momento de registrar vehículos se utiliza un try, por lo que iniciamos la creación del objeto. Dependiendo del tipo de vehiculo se crea un objeto y se llama a su función correspondiente

Esa función tiene inputs para registrar los datos del vehículo, al registrar los datos se regresa el tipo de dato listo para que el objeto los reciba como argumentos

En esta parte es donde deberian entrar los throw ArgumentError, ya que estamos ingresando los valores para la creación del objeto (los trow se encuentran en los setters)

Si esto pasa de manera correcta, en caso de detectar un valor no deseado en el setter(nombre o modelo vacío o año menor a 1900), se lanza el ArgumentError en lugar de asignarlo al objeto, por lo que el programa detiene su funciónamiento y se va al catch más cercano

El codigo se detiene hasta la creción del objeto [Ej: Vehiculo nuevoAuto = crearAutoInteractivo()], por lo que no se ejecuta la siguiente linea donde se agrega el objeto vehiculo a nuestra flotilla [flotilla.add(nuevoAuto)], y en su lugar se dirige al catch

Dentro del catch simplemente se imprime el mensaje de [ERROR] junto con el tipo de error que cometimos

ERROR EN EL CÓDIGO: La aplicación al momento de llamar al constructor se salta por completo los setters (ya que usa this.), por lo que no hace las validaciones y los objetos aceptan cualquier valor. Para solucionar esto solo haria falta cambiar la asignación de this. a usar los setters que establecimos con las excepciones

Diagrama UML estático:
Trabajado por Alfaro Carreto Irving Alberto

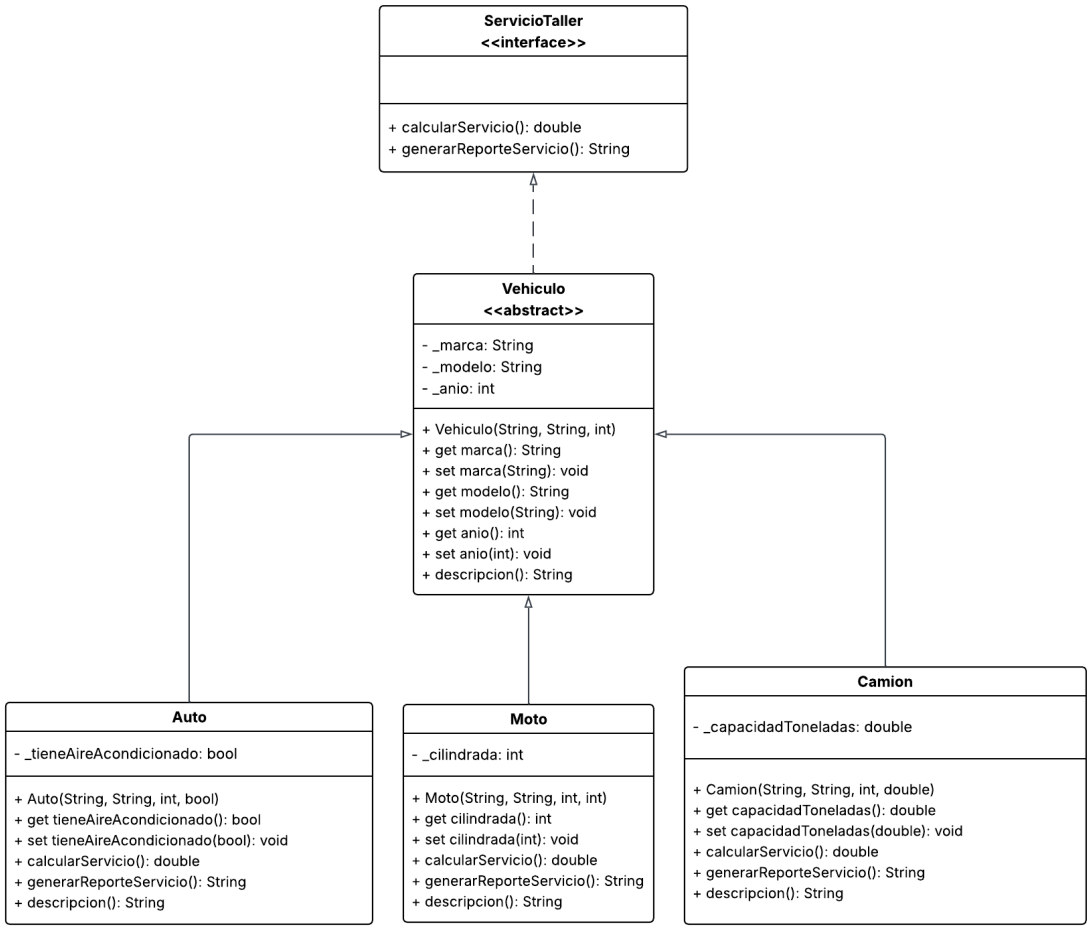
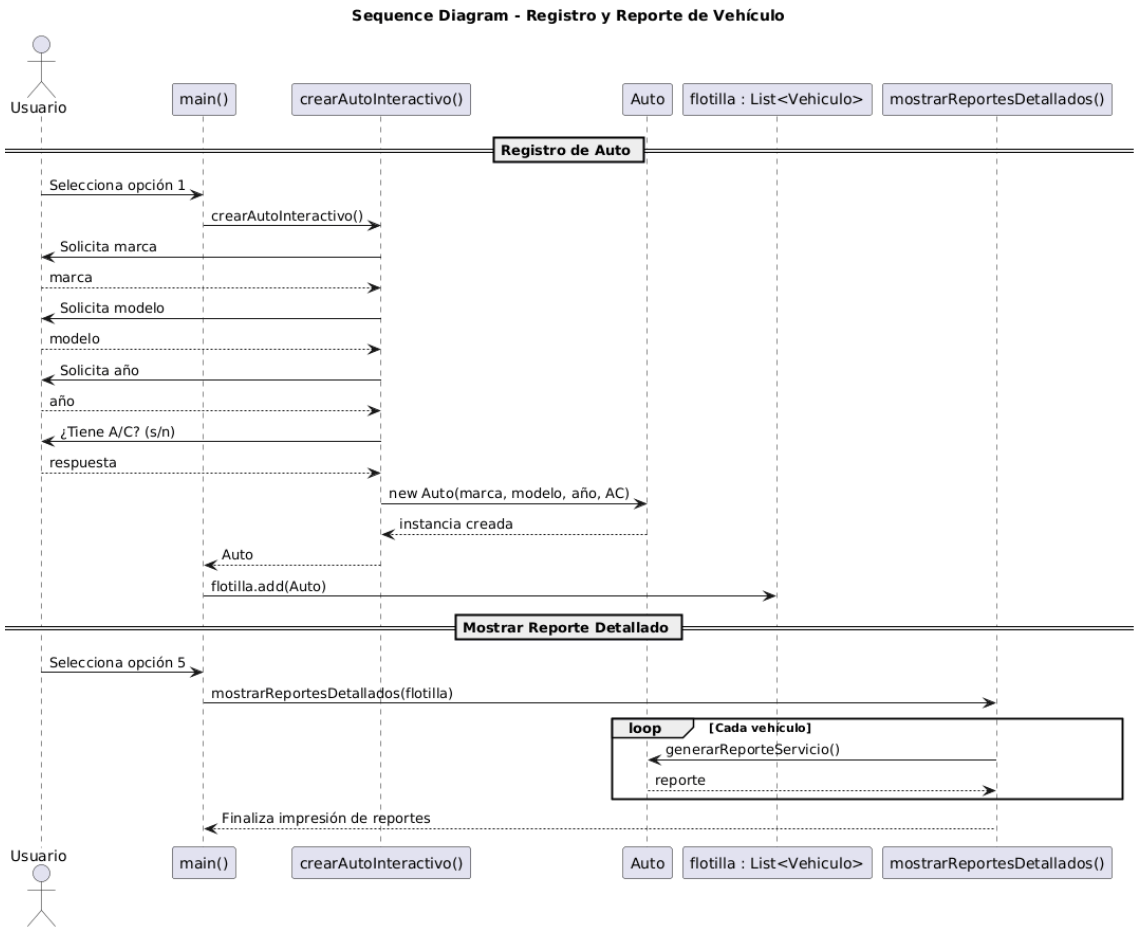


Diagrama UML Dinámico:
Trabajado por Urzúa Sánchez Enrique



4. Resultados

Despues del analisis realizado del codigo para poder poner a prueba los temas vistos en clase pusimos a prueba el codigo realizado por el profesor en el que se puede observar el manejo de las excepciones ademas de como funciona el lenguaje dart.

Prueba de registro

```
PS C:\Users\gato2\OneDrive\Escritorio\GIT> dart main.dart
==================================
   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: 1

== Registro de Auto ==
Marca: Ford
Modelo: Mustang
Año: 1965
¿Tiene aire acondicionado? (s/n): n

[OK] Auto agregado.

Presiona ENTER para continuar...
```

En este caso los casos de registro fueron realizados con datos privados y clases heredadas lo que nos permite tener un manejo de los datos más seguro para futuros proyectos con el lenguaje, además de que podemos comprobar como se manejas algunos problemas que podría llegar a generar el usuario gracias al manejo de excepciones.

```
Elige una opción: 1

== Registro de Auto ==
Marca: Volkswagen
Modelo: Beetle
Año: 1968
¿Tiene aire acondicionado? (s/n): n

[OK] Auto agregado.

Presiona ENTER para continuar...


================================
   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: Harley-Davidson
Modelo: Fat Boy
Año: 1990
Cilindrara (cc): 1340

[OK] Moto agregada.
```

Ya con los casos finales casos verificamos que los datos de la flotilla se hayan guardado correctamente observando los datos guardados.

-Caso de los datos de la flotilla resumidos

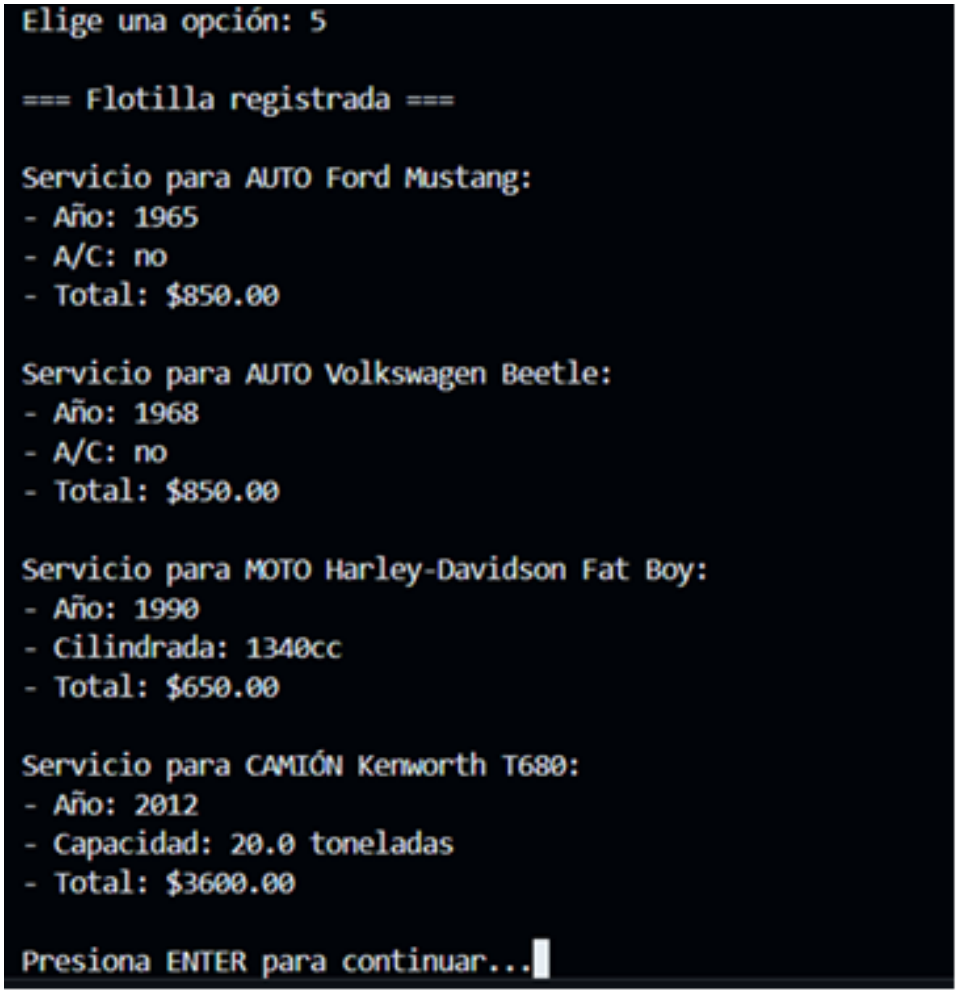
```
================================
   SISTEMA PARA TALLER MECÁNICO
================================
1) Registrar Auto
2) Registrar Moto
3) Registrar Camión
4) Ver flotilla (resumen)
5) Ver reportes detallados
0) Salir
Elige una opción: 4

=== Flotilla registrada ===

[0] Auto: Ford Mustang (1965) - A/C: no | Servicio: $850.00
[1] Auto: Volkswagen Beetle (1968) - A/C: no | Servicio: $850.00
[2] Moto: Harley-Davidson Fat Boy (1990) - 1340cc | Servicio: $650.00
[3] Camión: Kenworth T680 (2012) - Capacidad: 20.0 toneladas | Servicio: $3600.00

Presiona ENTER para continuar...
```

-Vista de los reportes completos


Elige una opción: 5
=== Flotilla registrada ===
Servicio para AUTO Ford Mustang:
- Año: 1965
- A/C: no
- Total: $850.00
Servicio para AUTO Volkswagen Beetle:
- Año: 1968
- A/C: no
- Total: $850.00
Servicio para MOTO Harley-Davidson Fat Boy:
- Año: 1990
- Cilindrada: 1340cc
- Total: $650.00
Servicio para CAMIÓN Kenworth T680:
- Año: 2012
- Capacidad: 20.0 toneladas
- Total: $3600.00
Presiona ENTER para continuar...

5. Conclusiones

El análisis de la aplicación demuestra cómo la implementación del manejo de excepciones es fundamental para poder construir y manejar software más complejo. La arquitectura del programa. Este enfoque y el uso de las excepciones previene la creación de objetos con estados inválidos, este mecanismo de validación coexiste con los principios aprendidos en clase, como la abstracción y el polimorfismo. El polimorfismo permite que la lógica principal, mediante los bloques try-catch en la función main, gestione la creación de cualquier tipo de vehículo de forma genérica, capturando errores de manera uniforme sin necesidad de conocer los detalles de validación de cada subclase.
Los resultados de ejecución, que muestran el registro exitoso de la flotilla y la generación de reportes detallados nos dejan ver un diseño que equilibra la flexibilidad (polimorfismo) con la seguridad (excepciones). El análisis confirma que la aplicación de estos conceptos teóricos es lo que permite al sistema gestionar los errores de forma controlada.

6. Referencias bibliográficas

[1] Google. (s.f.). Flutter - Build apps for any screen. flutter.dev. https://flutter.dev/

[2] Google. (s.f.). Dart programming language. dart.dev. https://dart.dev/

[3] Oracle. (s.f.). Lesson: Exceptions (The Java™ Tutorials >Essential Java Classes). Oracle Help Center.
https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html