ANALISTA PROGRAMADOR

PRACTICO 2

LOIS MORALES

2023

Resumen:

En este trabajo se desarrolló un sistema para gestionar una empresa de alquileres de vehículos mediante la programación orientada a objetos. Se implementaron conceptos como: objetos, clases, métodos, herencia, y composición, agregación y asociación. Se creó una clase padre llamada Vehículo y tres clases hijas para representar a los tipos del vehículo: Familiar, Utilitario y Deportivo. Cada clase incluye atributos que tienen información importante de un vehículo como un número identificador, matricula, marca, color, capacidad del tanque, estado (es decir, si se puede alquilar o ya está alquilado), el precio del alquiler diario y km por litro, además dependiendo del tipo de vehículo, se debe registrar un dato adicional, la capacidad del maletero para los Familiares, la capacidad de carga para los Utilitarios, y la velocidad máxima para los Deportivos. También se creó una clase Alquiler, la cual posee como atributos un número identificador, el precio total, una lista de los vehículos a alquilar, un objeto de la clase Detalle y datos del cliente, como el documento, un teléfono, y el nombre y apellido. La clase Detalle previamente mencionada contiene la cantidad de vehículos que se alquilaron, la fecha en que se retiraron los vehículos y la cantidad de días que dura el alquiler.

Por último, también se creó una clase Sucursal, la cual contiene un numero de sucursal, la dirección de la sucursal, una lista de los vehículos que posee y una lista de los alquileres realizados. Se utilizaron métodos en cada clase para establecer y obtener los valores de sus atributos. Se crearon constructores para inicializar las estancias de las clases con valores ingresados por el usuario.

Para la interacción con el usuario, se creó un menú que ofrece diversas opciones como el alquiler de uno o más vehículos, para este se implementó un proceso de entrada de datos detallados, solicitando al usuario que ingrese el número del vehículo que se alquilará, el documento, teléfono, nombre y apellido del cliente, y la cantidad de días que dura el alquiler; o también la opción de añadir vehículos a la sucursal, lo cual pedirá los datos necesarios para registrar el nuevo vehículo, como la matricula, la marca, el color, la capacidad del tanque, el precio de alquiler diario, los km por litro, el tipo de vehículo y el dato adicional dependiendo del tipo.

Otras opciones del menú son ver los vehículos de la sucursal, lo cual desplegara un submenú en el cual se da a elegir al usuario si quiere ver todos los vehículos y su respectiva información, solo los vehículos alquilados o solo los vehículos sin alquilar; también ver los alquileres, opción que despliega una lista de los alquileres de la sucursal para elegir de cual se desean ver los datos; y una opción para ver los clientes de la sucursal, la cual despliega una lista con los nombres de los clientes y al ingresar uno de los nombres, muestra los alquileres realizados por ese cliente.

El menú también cuenta con la opción de salir del programa, lo cual finaliza la ejecución del programa.

Índice

1. Introducción

1.1. Clases y Objetos

En la programación orientada a objetos (POO), una clase actúa como un patrón o un modelo para la creación de objetos. Establece un conjunto de atributos y métodos que pueden ser aplicados a cualquier objeto derivado de dicha clase. Un objeto representa una instancia de una clase que tiene un estado, es decir, los valores de sus atributos, y un comportamiento, que son las acciones que puede llevar a cabo mediante sus métodos.

Las clases y los objetos son componentes esenciales de la POO, y se emplean para simular y representar objetos del mundo real y sus interacciones en el software. Al definir una clase, se está definiendo una plantilla para la creación de objetos que comparten propiedades y funcionalidades comunes. Los objetos se crean a partir de estas clases y pueden colaborar entre sí y con otros objetos del sistema para ejecutar tareas específicas.

Las clases en C# son plantillas o modelos que se utilizan para crear objetos. Una clase define la estructura, los campos, las propiedades, los métodos y los eventos que pertenecen a un objeto en particular. En otras palabras, una clase proporciona una descripción de cómo se comportará un objeto.

Por lo tanto, cuando se crea un objeto a partir de una clase, se está creando una instancia de esa clase. La instancia es un objeto único que tiene sus propios valores para los campos y propiedades definidos en la clase. [1]

1.2. Herencia

La herencia en C# es un mecanismo que posibilita la creación de nuevas clases tomando como base una clase preexistente, que se denomina clase base o superclase. A la nueva clase generada se le llama clase derivada o subclase.

La clase derivada hereda todos los elementos de la clase base y tiene la capacidad de agregar nuevos elementos o modificar los elementos heredados.

Cuando se hereda de una clase, la subclase adquiere todos los campos, métodos y propiedades de la clase base, pero tiene la flexibilidad de proporcionar implementaciones diferentes de estos elementos si es necesario.

Además de permitir la reutilización de código, la herencia también tiene el beneficio de reducir la cantidad de código que se necesita escribir al aprovechar los miembros y comportamientos de una clase existente en lugar de crearlos desde cero. [2]

1.3. Polimorfismo

El polimorfismo es un concepto clave en la programación orientada a objetos que se refiere a la capacidad de una clase hija (o subclase) de proporcionar una implementación específica de un método que ha sido definido en su clase padre (o superclase). Esto permite que un mismo método tenga múltiples comportamientos dependiendo del contexto en el que se llame.

El polimorfismo se puede clasificar en dos tipos: estático y dinámico. El polimorfismo estático se refiere a la sobrecarga de métodos, que permite que una clase tenga varios métodos con el mismo nombre, pero con diferentes parámetros. Por otro lado, el polimorfismo dinámico se refiere a la sobreescritura de métodos, que permite que una clase hija proporcione una implementación específica de un método que ha sido definido en su clase padre.

En otras palabras, el polimorfismo nos permite definir un método en la clase base y luego sobrescribir ese mismo método en una clase derivada, de manera que ambas clases tengan un método con el mismo nombre y firma, pero con implementaciones diferentes. [3]

1.4. Composición

La composición es un concepto clave en la programación orientada a objetos que se refiere a la relación entre dos clases: una clase contenedora y una o más clases contenidas. La clase contenida es una parte importante de la clase contenedora, y si la clase contenedora desaparece, la clase contenida también debe desaparecer. Esta relación se llama “relación fuerte”.

En otras palabras, la composición es un tipo de relación de alto grado de dependencia entre la clase contenedora y las clases que la componen. Cuando se crea una instancia de la clase contenedora, deben crearse, como parte de su conformación, instancias de los objetos que la componen. [4][5]

1.5. Agregación

La agregación es un concepto clave en la programación orientada a objetos que se refiere a la relación entre dos clases: una clase contenedora y una o más clases contenidas. En la agregación, los elementos se agregan al contenedor con el tiempo, pero si el contenedor desaparece, los elementos aún pueden existir. Esta relación también se conoce como una “relación débil”.

La agregación se utiliza para construir objetos complejos a partir de objetos más simples. Por ejemplo, un automóvil puede estar compuesto por un motor, una transmisión, un chasis y otros componentes. Cada uno de estos componentes es una clase contenida que se combina para formar la clase contenedora, que es el automóvil completo.

En otras palabras, la agregación es un tipo de relación con un bajo grado de dependencia. Por ejemplo, una instancia de una clase Persona puede tener o no un atributo de una clase Ropa durante su tiempo de vida, pero esto no afecta su propia existencia. Al mismo tiempo, un objeto de la clase Ropa podría existir independientemente de si es agregado a una Persona o a un Maniquí. [4][5]

1.6. Asociación

La asociación es un concepto clave en la programación orientada a objetos que se refiere a la relación entre dos clases: una clase contenedora y una o más clases contenidas. Esta relación implica la existencia de una relación estructural entre objetos de esas clases. Los dos objetos pueden existir independientemente, no es una condición de que los objetos se creen a la vez. En otras palabras, son dos objetos que en un momento dado se unen para trabajar juntos.

La multiplicidad es un concepto relacionado con la asociación que determina la cantidad de objetos que pueden estar involucrados en una asociación. La multiplicidad se expresa mediante un rango numérico que indica cuántos objetos pueden estar involucrados en una asociación. Por ejemplo, una multiplicidad de 1…* indica que hay al menos un objeto involucrado en la asociación, pero puede haber muchos más.

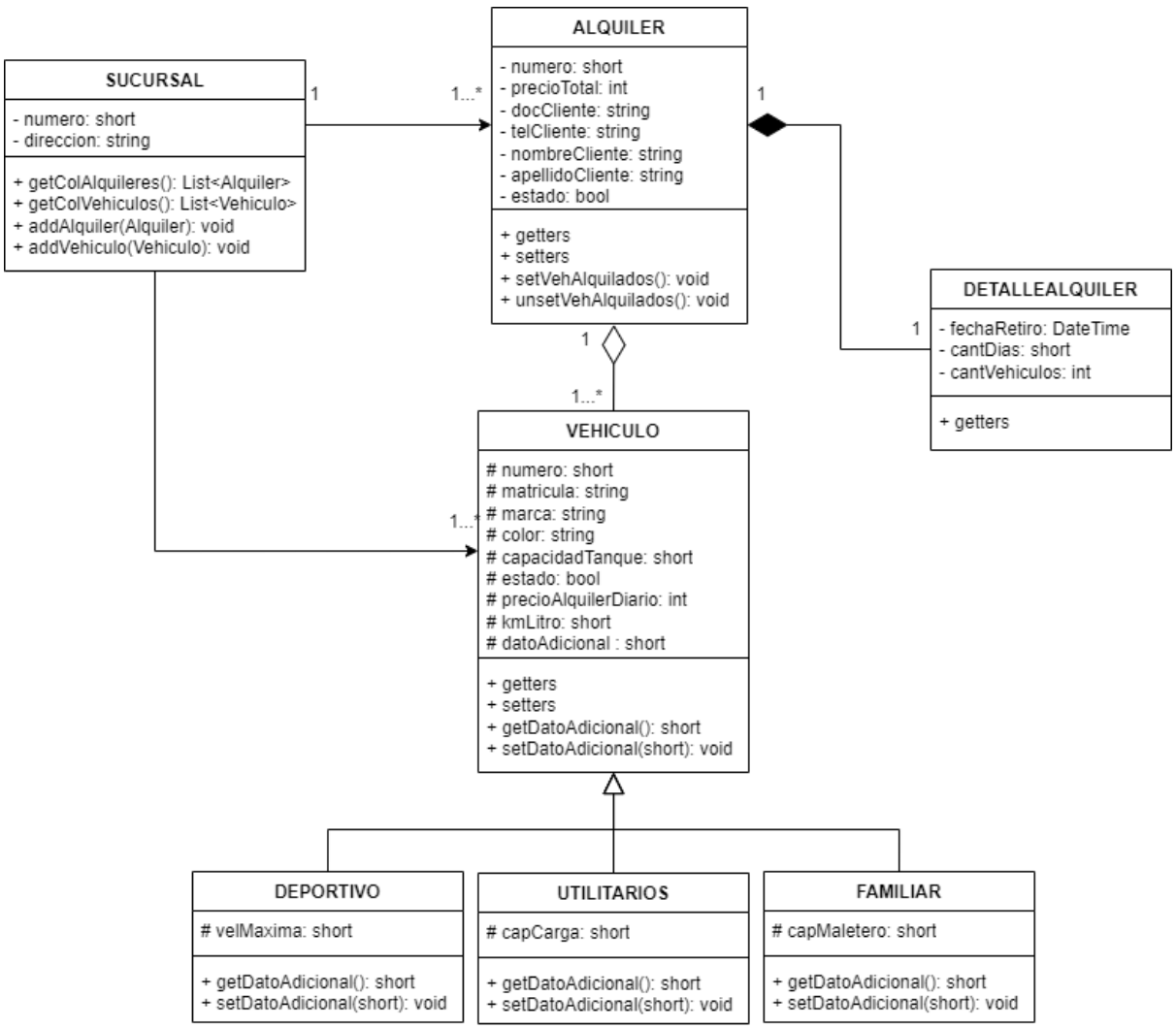
La navegabilidad es otro concepto relacionado con la asociación que determina el sentido de la asociación. En otras palabras, la navegabilidad indica si una clase puede acceder a otra clase a través de la asociación. Hay dos tipos de navegabilidad: unidireccional y bidireccional. En una asociación unidireccional, una clase envía información a otra clase, mientras que, en una asociación bidireccional, ambas clases pueden enviar y recibir información. [6][7]

2. Metodología

Para crear un programa de un servicio de alquiler de vehículos con POO se aplicaron los diferentes conceptos de esta. Las clases se utilizaron para crear los objetos que se necesitaban con sus respectivos atributos. Estuvieron presentes otros conceptos como el de herencia, utilizado para crear una clase Padre llamada Vehículo, esta contenía todos los atributos (matricula, marca, color, etc.) y métodos en común (getters y setters de los atributos) que comparten las clases Familiar, Utilitario y Deportivo. Se hace uso del Polimorfismo para crear métodos únicos para cada tipo de vehículo, como el método getDatoAdicional() que sirve para saber el dato adicional de ese vehículo dependiendo de su tipo y el método setDatoAdicional() que sirve para darle un valor a ese dato adicional.

También se crea la clase Alquiler, que contenía como atributos: datos del cliente, una lista de los vehículos que se alquilaron, el precio total del alquiler y un objeto de la clase Detalle que se instancia en la creación del alquiler.

Se crea además una clase Sucursal, que contendrá una lista de vehículos y de alquileres además de un número de sucursal y la dirección, y métodos para añadir vehículos o alquileres a sus respectivas listas.



Link GitHub: https://github.com/LoisMorales/Practico2

2.1. Clase Vehículo

2.1.1. Atributos

Se definen atributos para guardar los datos de los objetos que estamos creando. Para este programa se requerían los siguientes:

numero (short)

matricula (string)

marca (string)

color (string)

capTanque (short)

estado (bool)

precioAlquilerDiario (int)

kmLitro (short)

datoAdicional (short)

2.1.2. Constructores

En los constructores utilizamos el concepto de sobrecarga por lo que se llaman igual que la clase, pero con atributos diferentes.

Vehiculo(short numero, string matricula, string marca, string color, short capTanque, bool estado, int precioAlquilerDiario, short kmLitro)

Vehiculo(): vacío para que podamos crear objetos a partir del ingreso del usuario.

2.1.3. Métodos

Se utilizan métodos getters y setters para todos los atributos, estos nos permiten obtener (getters) y modificar (setters) cada atributo de la clase por separado.

El polimorfismo fue utilizado en estos 2 métodos: getDatoAdicional() y setDatoAdicional(). Estos se declaran en la clase padre Vehiculo para después en las clases hijos hacer lo que se llama override para que los métodos actúen diferente dependiendo la clase que los llame.

2.2. Clase Familiar, Clase Utilitario y Clase Deportivo

2.2.1. Atributos

Las clases Familiar, Utilitario y Deportivo heredan de la clase Vehiculo todos sus atributos y se les agrega uno llamado capMaletero en la clase Familiar, uno llamado capCarga en la clase Utilitario y uno llamado velMaxima en la clase Deportivo, que son únicos de cada clase.

2.2.2. Constructores

En este caso, al haber utilizado herencia, el constructor de las clases hijas está llamando al constructor de la clase padre Vehiculo y pasándole los parámetros que se han recibido en el constructor de las hijas. Esto se hace para inicializar adecuadamente los atributos de la clase padre con los valores dados al crear un objeto de las clases Familiar, Utilitario y Deportivo.

- Familiar(short numero, string matricula, string marca, string color, short capTanque, int precioAlquilerDiario, short kmLitro, short capMaletero):base(numero, matricula, marca, color, capTanque, precioAlquilerDiario, kmLitro)

- Utilitario(short numero, string matricula, string marca, string color, short capTanque, int precioAlquilerDiario, short kmLitro, short capCarga):base(numero, matricula, marca, color, capTanque, precioAlquilerDiario, kmLitro)

- Deportivo(short numero, string matricula, string marca, string color, short capTanque, int precioAlquilerDiario, short kmLitro, short velMaxima):base(numero, matricula, marca, color, capTanque, precioAlquilerDiario, kmLitro)

2.2.3. Métodos

En estas clases existen los métodos polimórficos ya mencionados que permiten realizar diferentes acciones dependiendo que clase los utilice, estos son: getDatoAdicional() y setDatoAdicional(); los cuales modifican los atributos capMaletero en la clase Familiar, capCargo en la clase Utilitario, y velMaxima en la clase Deportivo.

2.3. Clase Alquiler

La clase Alquiler se utiliza para registrar todos los datos pertinentes al alquiler de uno o más vehículos.

2.3.1. Atributos

Los datos del alquiler se guardan en los siguientes atributos:

numero (short)

precioTotal (int)

docCliente (string)

telCliente (string)

nombreCliente (string)

apellidoCliente (string)

detalle (Detalle)

estado (bool)

colVehiculos (List<Vehiculo>)

2.3.2. Constructores

Alquiler(short numero, string docCliente, string telCliente, string nombreCliente, string apellidoCliente, List<Vehiculo> colVehiculos, short cantDias): En el constructor se incluye el parámetro cantDias, porque al crearse el objeto Detalle dentro del objeto Alquiler (composición), es necesario pasarle este dato a su constructor.

Alquiler(): vacío para que podamos crear objetos a partir del ingreso del usuario.

2.3.3. Métodos

Se utilizan métodos getters y setters para todos los atributos, estos nos permiten obtener (getters) y modificar (setters) cada atributo de la clase por separado.

El setter del atributo precioTotal es distinto al de los otros atributos, esto se debe a que debe realizar una cuenta en base al precio del alquiler diario de cada vehículo alquilado y la cantidad de días que dura el alquiler. El setter del atributo detalle también es distinto, ya que a este se le pasan los datos necesarios para crear el objeto Detalle del alquiler cuando este último se crea a partir del ingreso del usuario.

También se crearon otros dos métodos: setVehAlquilados() y unsetVehAlquilados(), estos se utilizan para marcar o desmarcar los vehículos dentro del alquiler como alquilados o sin alquilar respectivamente dependiendo de si el alquiler ha finalizado o no.

2.4. Clase Detalle

La clase Detalle se utiliza para registrar los detalles del alquiler en el que se crea el objeto de esta clase.

2.4.1. Atributos

Esta clase posee los siguientes atributos:

fechaRetiro (DateTime)

cantDias (short)

cantVehiculos (int)

2.4.2. Constructor

Detalle(int cantVehiculos, short cantDias): al constructor no se le pasa ningún dato para guardar en el atributo fechaRetiro ya que este atributo toma automáticamente el valor de la fecha en la que se crea.

2.4.3. Métodos

Se utilizan métodos getters para todos los atributos.

2.5. Clase Sucursal

La clase Sucursal es la entidad que contiene dos listas que actúan como un almacén para mantener un registro de los vehículos y los alquileres. Estas listas se configuran inicialmente mediante el constructor de la clase y se utilizan a través de varios métodos para llevar a cabo diferentes tareas.

2.5.1. Atributos

La clase Sucursal guarda los datos propios de la sucursal en sus atributos:

numero (short)

dirección (string)

colAlquileres (List<Alquiler>)

colVehiculos (List<Vehiculos>)

2.5.2. Constructor

Sucursal(short numero, string direccion, List<Alquiler> colAlquileres, List<Vehiculos> colVehiculos)

2.5.3. Métodos

En esta clase se hicieron diferentes métodos que acceden y editan las listas creadas:

addAlquiler(Alquiler alquiler): añade un nuevo alquiler a la lista de alquileres.

addVehiculo(Vehiculo vehiculo): añade un nuevo vehículo a la lista de vehículos.

2.6. Menú y Control de Excepciones

Se crea un menú para realizar las diferentes necesidades planteadas, las cuales involucran a las clases anteriormente mencionadas, como la de alquilar uno o más vehículos, agregar vehículos a la sucursal, listar los vehículos, listar los alquileres y también listar los clientes de la sucursal.

Para validar que las opciones elegidas fueran correctas, se utilizó la función try-catch buscando las excepciones FormatException y OverflowException, y también se utilizó Exception para controlar alguna excepción no esperada.

3. Resultados y Discusión

3.1. Programa por Consola

El programa presenta un menú principal para que el usuario elija que quiere hacer (ver Figura 3.2.1).

La opción 1 permite al usuario añadir un nuevo alquiler a la lista de alquileres, al seleccionar esta opción, se muestra un submenú que pregunta al usuario si desea añadir un vehiculo al alquiler o continuar (ver Figura 3.2.2), al elegir la primera opción al usuario se le muestra una lista con los vehículos actuales de la sucursal y se le pide seleccionar uno para añadir (ver Figura 3.2.3), si el vehículo seleccionado está disponible para alquilar se muestra un mensaje de que se ha añadido y el submenú previo (ver Figura 3.2.4), pero si el vehiculo ya esta alquilado se muestra un mensaje al usuario que lo advierte de esto y de nuevo el submenú previo (ver Figura 3.2.5), luego de esto, cuando el usuario elija continuar, el programa comienza a pedir los datos necesarios para el registro del alquiler (ver Figura 3.2.6).

La opción 2 permite al usuario añadir un nuevo vehiculo a la lista de vehículos, al seleccionar esta opción, al usuario se le comienzan a pedir los datos del vehículo (ver Figura 3.2.7), luego al usuario se le presenta un submenú que le pide elegir el tipo de vehiculo que está ingresando y por último le pide ingresar el dato adicional especifico del tipo elegido (ver Figura 3.2.8, Figura 3.2.9, y Figura 3.2.10).

La opción 3 permite al usuario visualizar una lista de los vehículos de la sucursal, cuando este elige esta opción, se le muestra un submenú con tres opciones, listar todos los vehículos, listar los vehículos alquilados o listar los vehículos sin alquilar (ver Figura 3.2.11), si elige la primera opción, se muestra una lista de todos los vehículos con sus números, marcas y matriculas para su identificación y se le da a elegir por su número (ver Figura 3.2.12) para así ver los detalles del vehículo seleccionado (ver Figura 3.2.13); si en cambio el usuario elige la segunda o tercer opción, se le muestra una lista de los vehículos alquilados (ver Figura 3.2.14) y los no alquilados (ver Figura 3.2.15) respectivamente.

La opción 4 permite al usuario visualizar una lista de los alquileres de la sucursal, al seleccionar esta opción, se muestran todos los alquileres seguidos de su respectivo número y la opción de seleccionar uno según este último (ver Figura 3.2.16), cuando el usuario elige un alquiler, se muestran sus respectivos datos, como el cliente, el costo, etc. y se le pregunta al usuario si el alquiler seleccionado ya ha finalizado y la opción de elegir “si” o “no” (ver Figura 3.2.17), si el usuario elige “si”, al volver a ingresar a los datos del mismo alquiler se mostrará un mensaje indicando que ya finalizó (ver Figura 3.2.18), además, todos los vehículos que se hallaban en el alquiler, volverán a poder ser alquilados; en cambio, si elige “no”, al volver a ingresar al mismo alquiler se le volverá a preguntar.

La opción 5 permite al usuario visualizar una lista de los clientes de la sucursal, cuando el usuario elige esta opción, se muestran los nombres de todos los clientes de la sucursal y se le da a elegir un cliente (ver Figura 3.2.19), cuando el usuario

ingresa el nombre del cliente, se le muestra una lista de los alquileres realizados por este (ver Figura 3.2.20).

La opción 0 permite al usuario finalizar el programa.

3.2. Anexo Consola

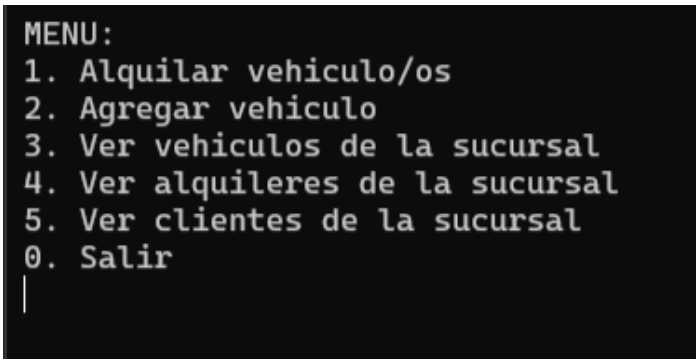


Figura 3.2.1. Menú Principal

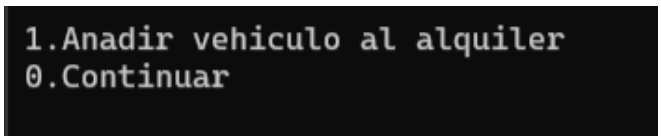


Figura 3.2.2. Opción 1: Submenú

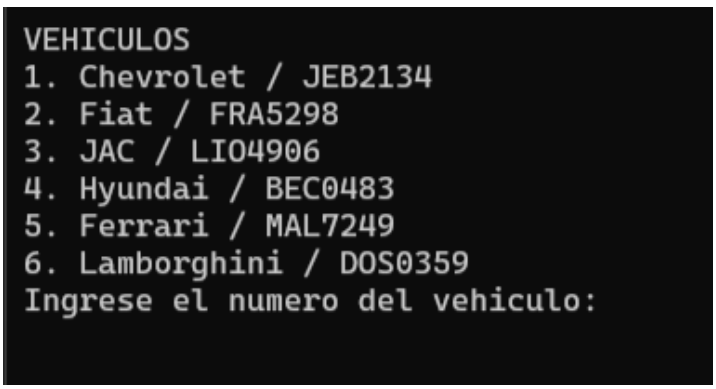


Figura 3.2.3. Opción 1: Lista de vehículos para alquilar

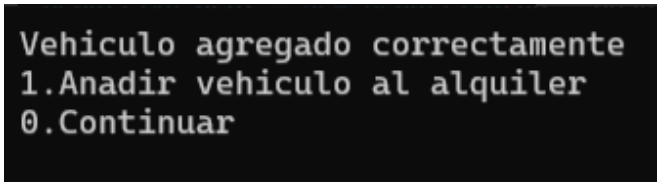


Figura 3.2.4. Opción 1: Vehículo agregado

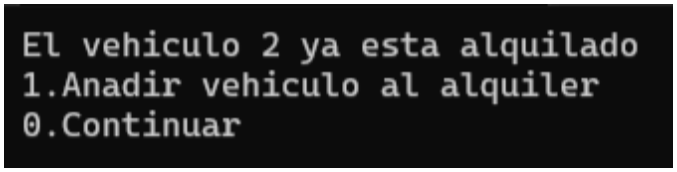


Figura 3.2.5. Opción 1: Vehículo ya alquilado

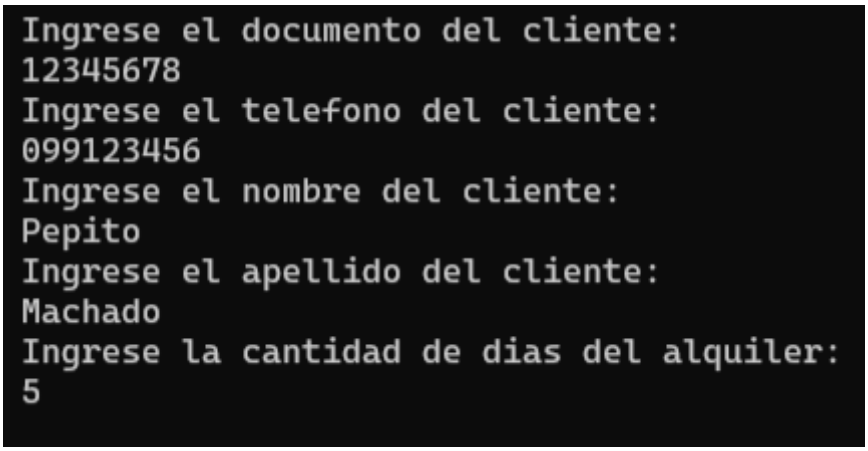


Figura 3.2.6. Opción 1: Ingreso de datos del alquiler

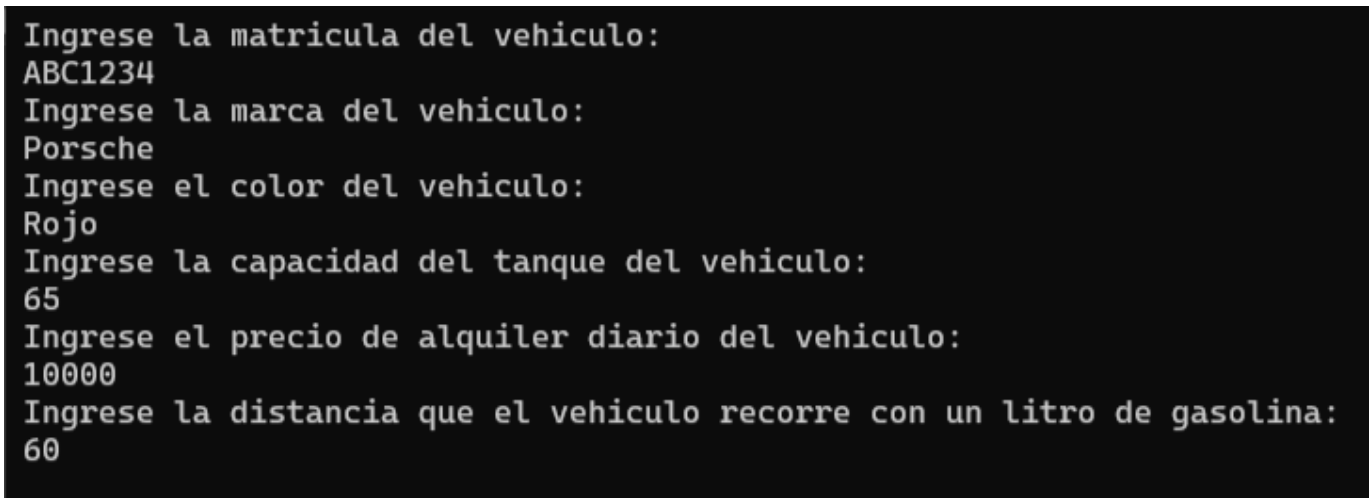


Figura 3.2.7. Opción 2: Ingreso de datos del vehículo

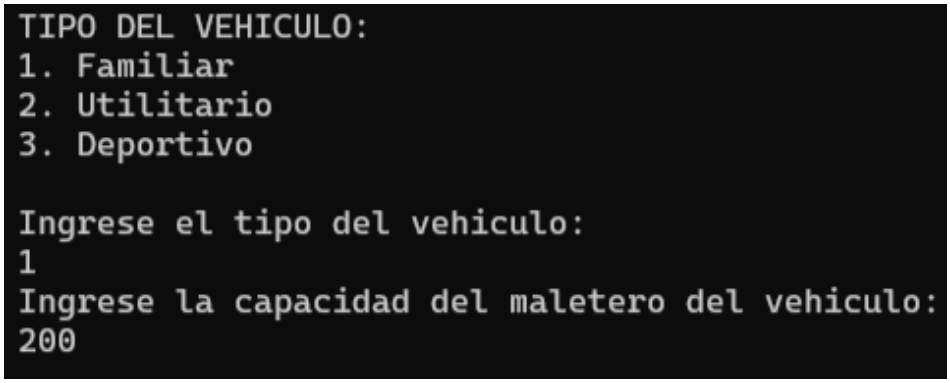


Figura 3.2.8. Opción 2: Submenú de tipo de vehículo (Familiar)

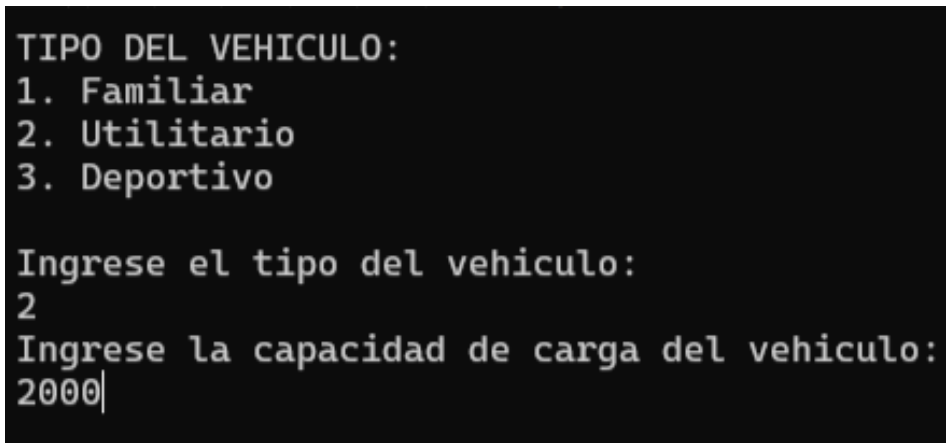


Figura 3.2.9. Opción 2: Submenú de tipo de vehículo (Utilitario)

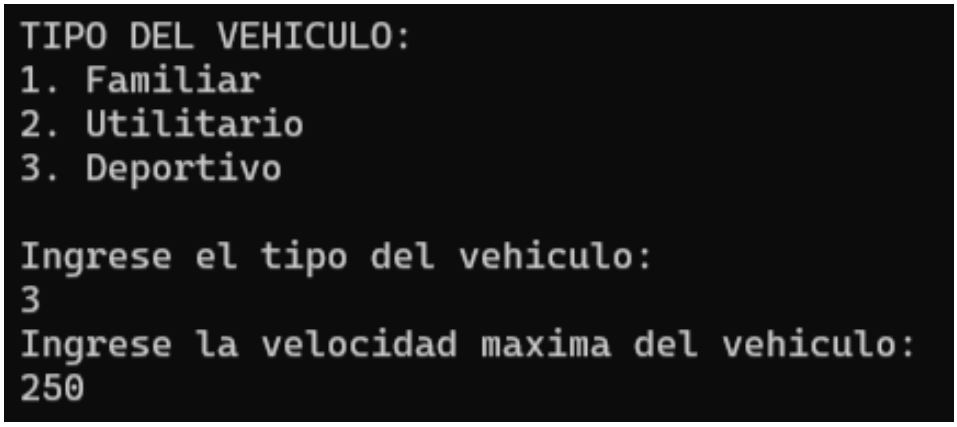


Figura 3.2.10. Opción 2: Submenú de tipo de vehículo (Deportivo)

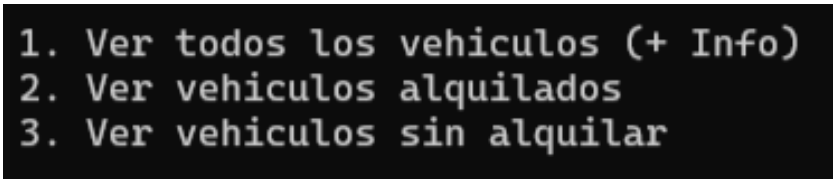


Figura 3.2.11. Opción 3: Submenú de listas de vehículos

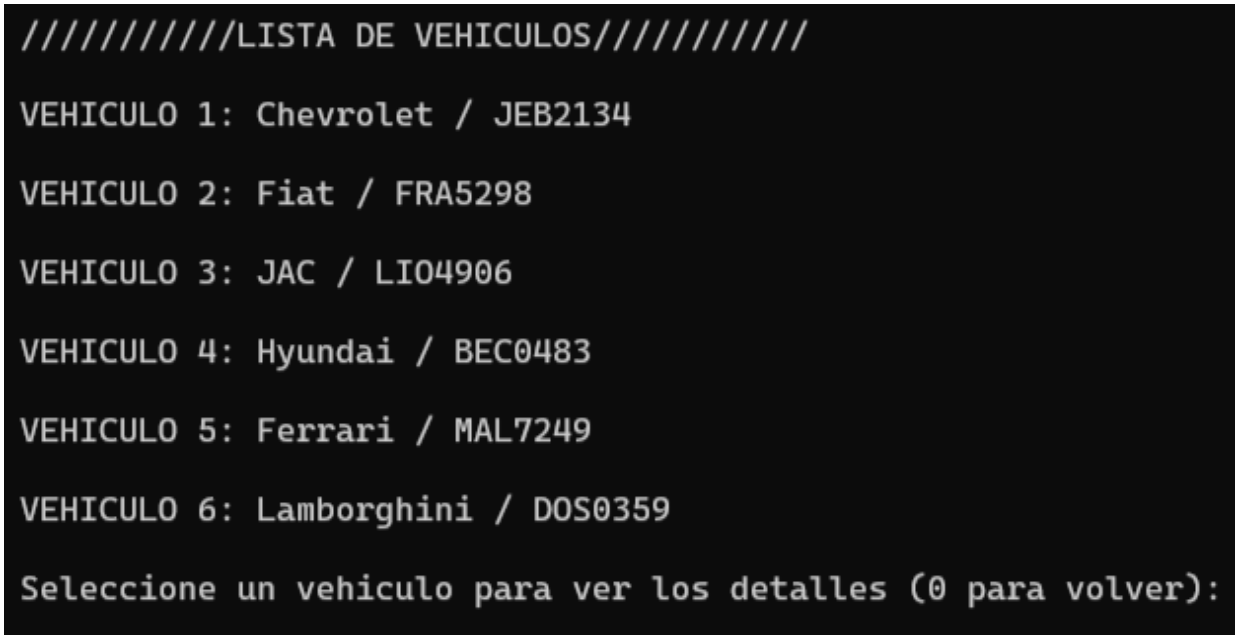


Figura 3.2.12. Opción 3: Opción 1: Lista de todos los vehículos

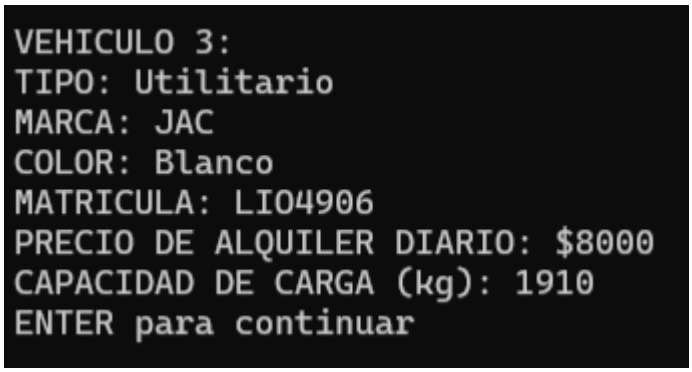


Figura 3.2.13. Opción 3: Opción 1: Detalles del vehículo seleccionado

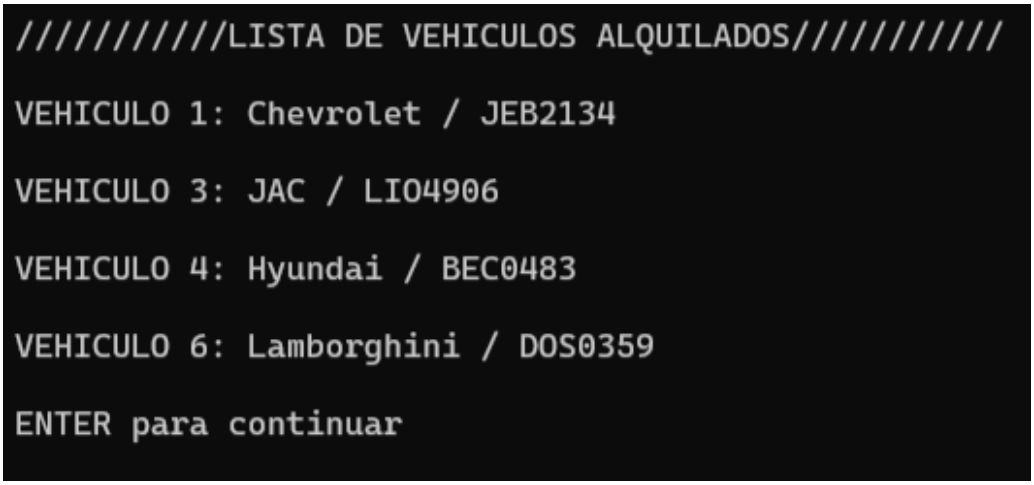


Figura 3.2.14. Opción 3: Opción 2: Lista de vehículos alquilados

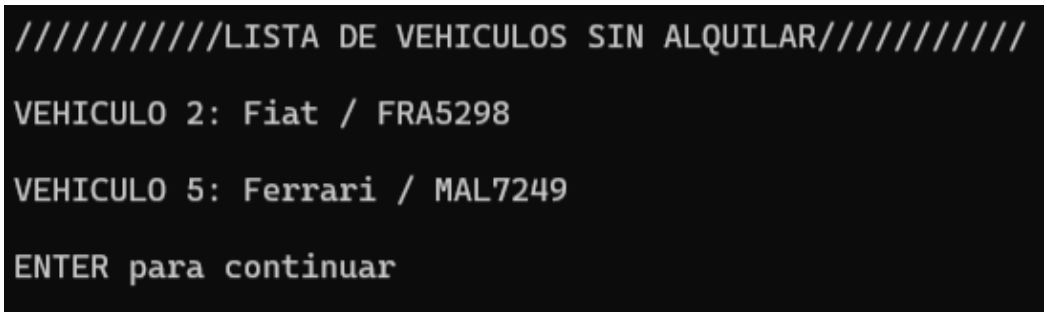


Figura 3.2.15. Opción 3: Opción 3: Lista de vehículos sin alquilar

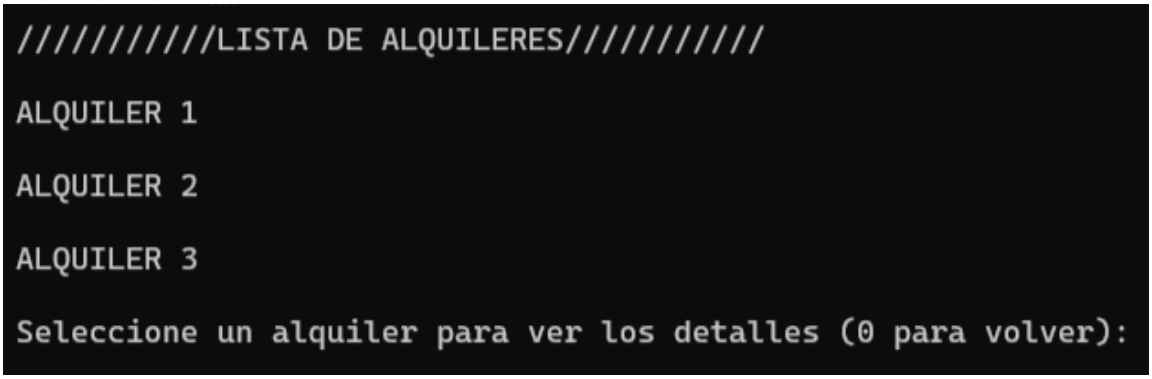


Figura 3.2.16. Opción 4: Lista de alquileres

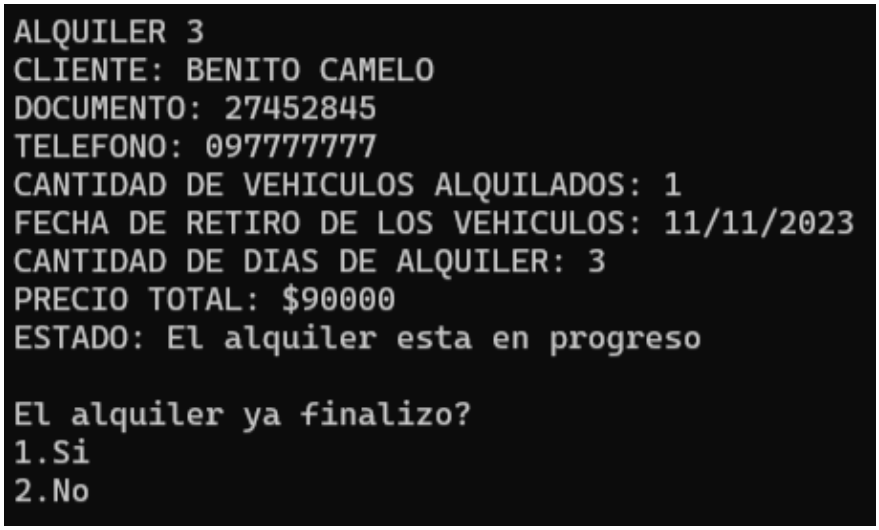


Figura 3.2.17. Opción 4: Detalles de alquiler seleccionado

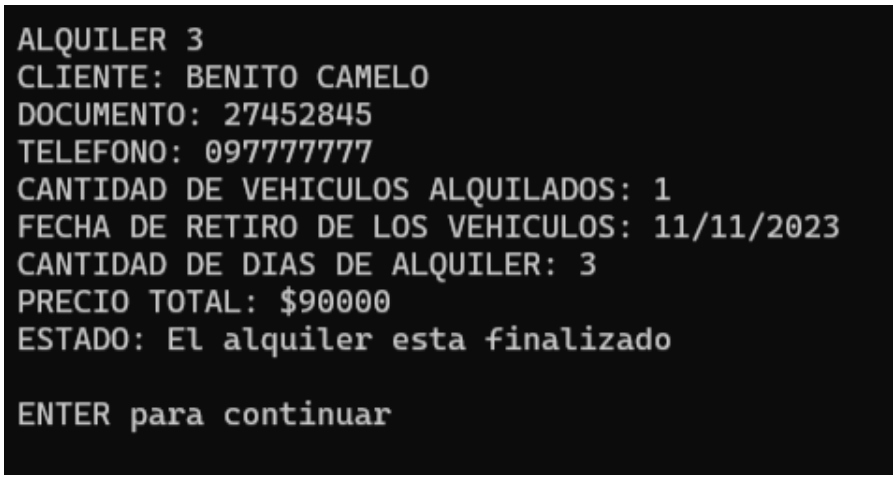


Figura 3.2.18. Opción 4: Detalles de alquiler luego de finalizado

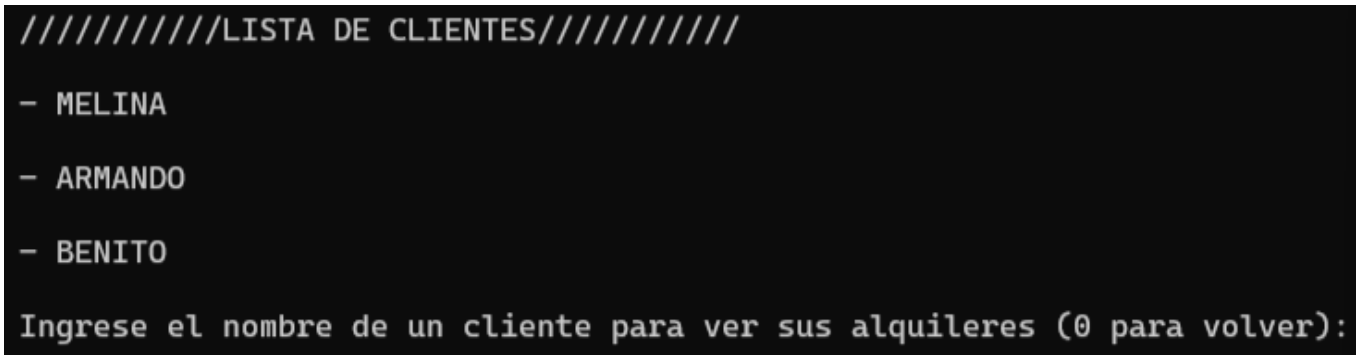


Figura 3.2.19. Opción 5: Lista de clientes

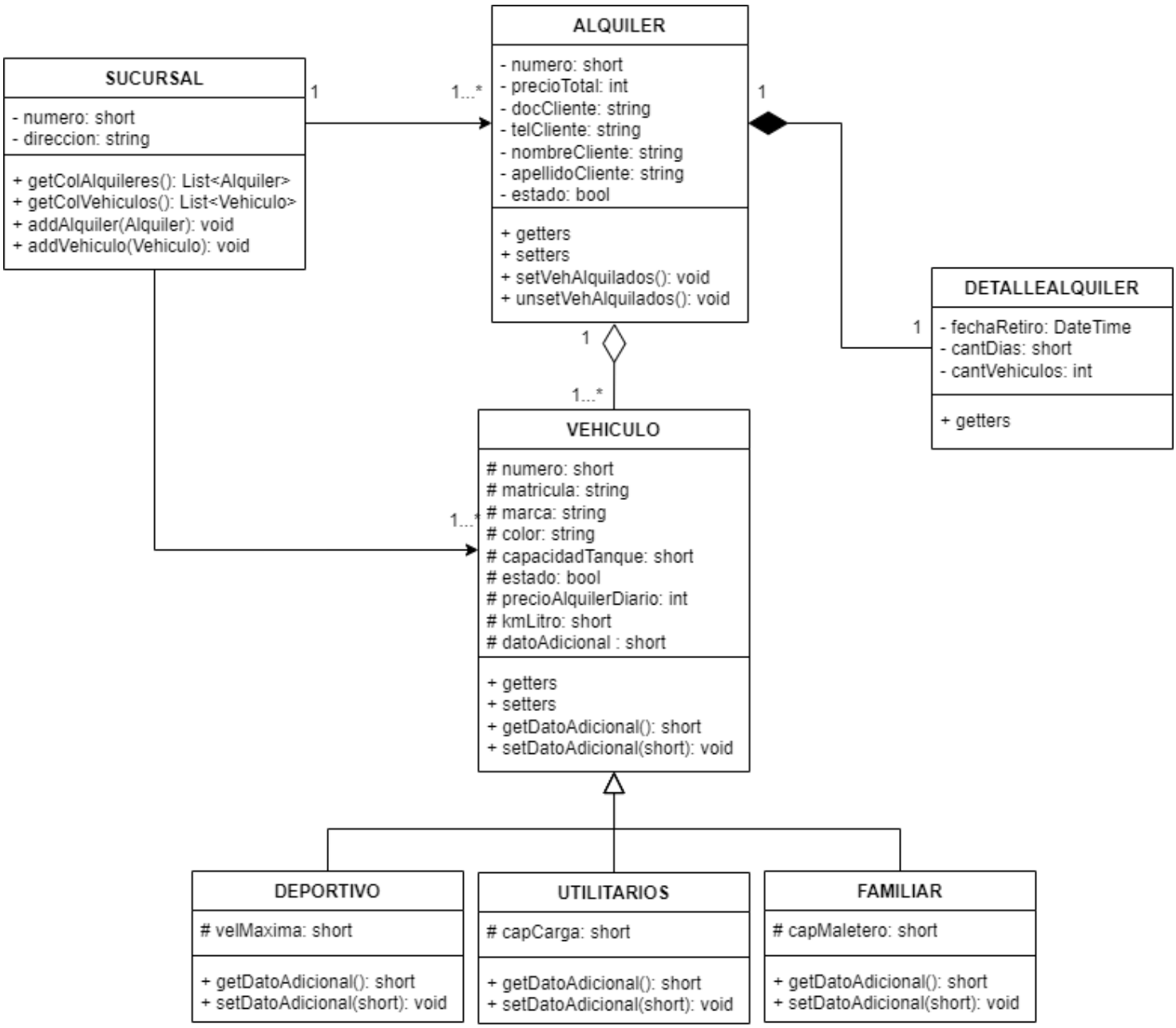
```
El cliente MELINA hizo el/los alquiler/es:

- Alquiler 1

ENTER para continuar
```

Figura 3.2.20. Opción 5: Alquileres realizados por cliente seleccionado

3.3. Diagrama de Clases



4. Conclusiones

Los métodos polimórficos setDatoAdicional y getDatoAdicional ayudaron a que cada clase de tipo de vehículo (Familiar, Utilitario y Deportivo) pueda guardar y mostrar sus respectivos datos adicionales al haber sido declarados en la clase padre Vehiculo.

La validación de selección de opciones en el menú y los submenús, se logró utilizando la función try-catch, la cual permitió el control de excepciones al intentar (try) una parte de código y atrapar (catch) una excepción si esta ocurría. Se utilizaron 2 catch para evitar las excepciones FormatException, el formato de un argumento no es válido o cuando una cadena de formato compuesto no tiene el formato correcto (por ejemplo, guardar un string en una variable tipo int), y OverflowException, la cual surge cuando una operación aritmética o de conversión de un contexto protegido provoca un desbordamiento (por ejemplo, cuando se quiere guardar un numero mayor al limite de la variable elegida); y también se utilizó un tercer catch con la excepción Exception, el cual evita cualquier excepción no esperada.

La implementación de la herencia en la clase padre Vehiculo y las clases hijas Familiar, Utilitario y Deportivo, logró simplificar el manejo y la creación de estas clases y sus atributos.

La creación de un menú interactivo mejoró la comunicación entre el usuario y el programa y sus funcionalidades, permitiendo así, su correcto manejo y su utilidad a la hora de su empleo.

5. Referencias

[1] https://oregoom.com/c-sharp/poo/

[2] https://oregoom.com/c-sharp/herencias/

[3] https://www.netmentor.es/entrada/polimorfismo-poo

[4] https://www.enfocate.doneber.dev/poo/11

[5] https://fundamentospoorrr.blogspot.com/2019/03/composicion-agregacion-y-asociacion.html

[6] https://logaligroup.com/asociacion/

[7] https://hsanchez903.wordpress.com/2015/02/22/clases-asociacion-y-generalizacion-en-staruml/