01-Practica09y10

DANIEL S ANGUIANO R

November 2025

1. Desarrollo

La aplicación desarrollada dentro del paquete `mx.unam.fi.poo.p910` permite el registro, cotización y la generación de reportes de vehículos ingresados al sistema implementando los conceptos de herencia, polimorfismo, encapsulamiento y `manejo de errores`. Con el propósito de explicar la lógica y las herramientas usadas en cada parte del código el desarrollo se divide en estas secciones:

1.1. Clases

Las clases de la aplicación definen los métodos que se ocuparán dentro del sistema y determinan el tipo de vehículos que se pueden registrar.

1.1.1. ServicioTaller

`ServicioTaller` es una interfaz que define dos métodos importantes que cada clase que represente a un vehículo se verá obligada a definir. Los métodos en cuestión son **calcularServicio()** y **generarReporteServicio()**.

1.1.2. Vehiculo

`Vehiculo` es una clase abstracta que implementa la interfaz `ServicioTaller` y es la superclase encargada de heredar a las clases que definen cada tipo de vehículo. Los atributos que incorpora son **marca**, **modelo** y **anio**, y al tener un encapsulamiento a nivel de archivo denotado por `"_"` usado como prefijo en los atributos, define sus getters y setters, haciendo uso del concepto de `manejo de errores` con **throw** para lanzar una excepción y detener el flujo del código implementando así validaciones en los setters que incluyen que **marca** y **modelo** no puedan ser vacíos y que **anio** no pueda ser menor a 1960, manteniendo así integridad y coherencia en los datos ingresados. Por último implementa le método **descripcion()** que devuelve los valores de los atributos ingresados.

1.1.3. Auto

`Auto` es una clase que extiende sus métodos y atributos de la clase **Vehiculo** añadiendo el atributo **_tieneAireAcondicionado** donde "_"marca que solo es accesible desde su archivo, por lo cual se crean getters y setters, además de sobreescribir el método **descripcion()** para agregar el nuevo atributo y definir los métodos **calcularServicio()** el cual calcula y regresa el total del monto a pagar sumando un precio base y un costo extra que depende de si tiene A/C y **generarReporteServicio()** que regresa en modo de cadena los atributos del vehículo y el monto a pagar.

1.1.4. Moto

`Moto` es una clase hija de **Vehiculo**, heredando así sus atributos y métodos, agregando un atributo de tipo entero **_Cilindrada** para el cual se contruyen su get y set, añadiendo en el set un **throw** para lanzar un error y manejar el caso en que se quiera ingresar una clindrada negativa al sistema.

Por otro lado los métodos que sobreescribe son **descripcion()** para agregar el nuevo atributo, **calcularServicio()** en el que se calcula y devuelve el monto total sumando una tarifa base y un costo extra que se suma en el caso de que la moto tenga arriba de 600 de cilindrada y **generarReporteServicio()** en el que se devuelven como String los atributos de la moto y el total a pagar.

1.1.5. Camion

`Camion` es una clase hija de **Vehiculo**, heredando sus métodos y atributos, añadiendo el atributo de tipo double **capacidadToneladas**, para el cual se designan sus respectivos get y set, incluyendo **throw** para arrojar un error en el caso de que se ingrese una capacidad negativa.

Por otra parte los métodos que sobreescribe, al igual que en los clases anteriores, son **descripcion()** para añadir le atributo **capacidad** a la descricpión del vehículo, **calcularServicio()** para calcular y regresar el total a pagar, sumando un costo base más un extra que depende de que la capacidad del camión sea mayor a 10 toneladas y **generarReporteServicio()** que devuelve en forma de String los atributos del vehículo y el total a pagar.

1.2. Funciones

En el código se implementaron nueve funciones con diferentes propósitos que permiten una ejecución más limpia y organizada desde el main:

Cuatro funciones se dedicaron a hacer más clara la lectura y captura de los distintos tipos de datos ingresados desde la interfaz de línea de comandos (CLI): **leerLinea()** que devuelve un String y en caso de ser NULL asigna un String vacío, **leerEntero()** se asegura de que se ingrese un entero convirtiendo la línea a entero

con `tryParse` y haciendo un bucle con while hasta que se ingrese un valor válido, **leerDouble()** es el mismo caso de **leerEntero()** pero con double y por último **leerBoolSN()** que hace minúsculas las líneas ingresadas siendo que si es "s"devuelve verdadero, si es "n"devuelve falso y repite el proceso hasta que se ingrese alguna de esas dos opciones válidas.

Por otra parte tres funciones tienen como objetivo la creación de objetos de los tres tipos de vehículos aceptados por el sistema: **crearAutoIntercativo()**, **crearMotoIntercativa()** y **crearCamionInteractivo()**, cada uno muestra la leyenda "Registro" y pide llenar los atributos correspondientes de cada tipo de vehículo, los captura con ayuda de las funciones anteriores y regresa un objeto de tipo vehículo con las especificaciones de cada objeto.

Por último, hay dos funciones que cumplen el propósito de mostrar un listado básico y los reportes de todos lo vehículos ingresados al sistema, estas son **mostrarListadoBasico()** y **mostrarReportesDetallados()**, ambos son de tipo void y recorren una lista que se les entrega como argumento, pero la primera funcion imprime una descripcion más el monto a pagar, mientras que la segunda genera y muestra un reporte de cada objeto contenido en la lista.

1.3. Main

El Main o la función principal es donde todo toma forma y las opciones que puede realizar el usuario se estructuran en un `switch`:

1)Registrar Auto
2)Registrar Moto
3)Registrar Camión
4)Ver flotilla (resumen)
5)Ver reportes detallados
0)Salir

Las primeras tres opciones implementan un **try/catch** para manejo de errores, cada uno, dentro del **try**, genera un nuevo objeto vehículo con la ayuda de su función crear respectiva y se agrega a la lista `flotilla`, en caso de ser exitoso se manda un mensaje de confirmación, pero en caso de haber algún error, este se almacena en la variable e que es lanzada y manejada por el **catch** imprimiendo el error que no permitió el registro.

Para las opciones 4) y 5) solo hacen uso de las funciones **mostrarListadoBasico()** y **mostrarReportesDetallados()**, implementando el concepto de polimorfismo pues dentro de estos métodos cada uno aplica su propia versión de **calcularServicio()** y **generarReporteServicio()**. Por último el programa termina una vez que el usuario decide presionar 0 en el menú para salir del sistema.