Ejercicio 4: Salpicadero

March 17, 2024

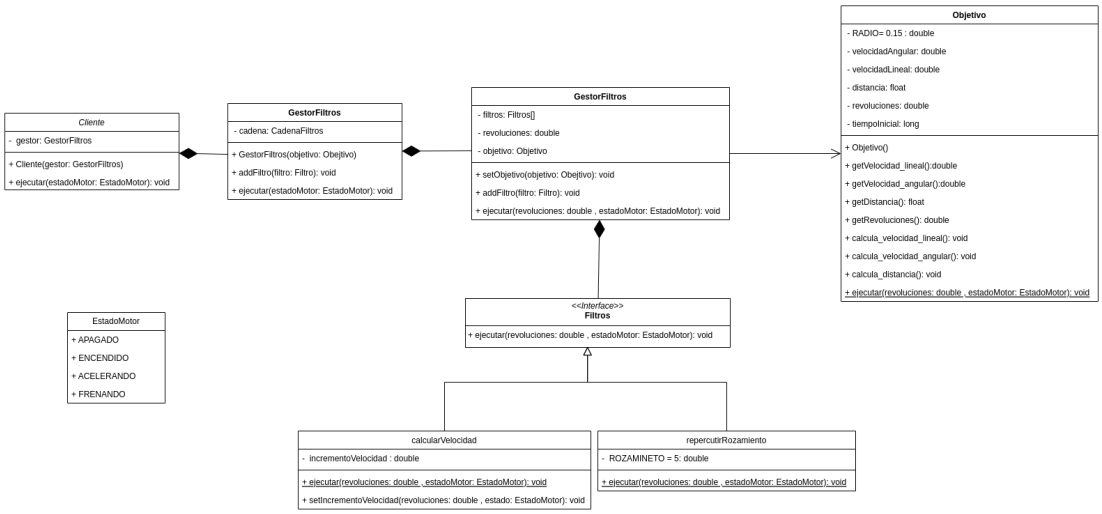

Figure 1: Diagrama de clase.

1 Clase Cliente

La clase `Cliente` representa un cliente que utiliza un `GestorFiltros` para establecer las revoluciones del motor apartir de un estado que se le pase.
Los atributos de la clase incluyen:

- `gestor:` el gestor de filtros utilizado por el cliente.

public Cliente(GestorFiltros g)

Constructor de la clase `Cliente` que recibe un `GestorFiltros` como parámetro y lo asigna al atributo `gestor`.

- **g:** el gestor de filtros a asignar.

public void ejecutar(EstadoMotor estadoMotor)
Método que ejecuta los filtros en el motor utilizando el gestor de filtros asignado.

- **estadoMotor:** el estado actual del motor.

2 Clase GestorFiltros

La clase `GestorFiltros` gestiona una cadena de filtros y proporciona métodos para agregar filtros y ejecutarlos en un motor.
Los atributos de la clase incluyen:

- `cadena:` la cadena de filtros administrada por el gestor.

public GestorFiltros(Objetivo o)
Constructor de la clase `GestorFiltros` que recibe un `Objetivo` como parámetro y que se lo pasa a `CadenaFiltros`

- **o:** el objetivo al que se aplicarán los filtros.

public void addFiltro(Filtro f)
Agrega un filtro a la cadena de filtros administrada por el gestor.

- **f:** el filtro que se añade a la cadena.

public void ejecutar(EstadoMotor estadoMotor)
Ejecuta la cadena de filtros en el motor.

- **estadoMotor:** el estado actual del motor.

3 Clase CadenaFiltros

La clase `CadenaFiltros` gestiona una lista de filtros y proporciona métodos para agregar filtros, establecer un objetivo y ejecutar los filtros en un motor.
Los atributos de la clase incluyen:

- `filtros:` una lista de filtros.
- `objetivo:` el objetivo al que se aplicarán los filtros.
- `revoluciones:` las revoluciones actuales del motor.

public void addFiltro(Filtro f)
Agrega un filtro a la lista de filtros.

- **f:** el filtro a agregar.

public void setObjetivo(Objetivo o)
Establece el objetivo al que se aplicarán los filtros.

- **o:** el objetivo a establecer.

public void ejecutar(EstadoMotor estadoMotor)
Ejecuta la lista de filtros en el motor y, opcionalmente, ejecuta el objetivo.

- **estadoMotor:** el estado actual del motor.

4 Interfaz Filtro

La interfaz `Filtro` define un método para ejecutar filtros en un motor.
public double ejecutar(double revoluciones, EstadoMotor estadoMotor)
Método que ejecuta un filtro en el motor.

- **revoluciones:** las revoluciones actuales del motor.
- **estadoMotor:** el estado actual del motor.
- **Return:** el resultado de aplicar el filtro.

5 Clase FiltroCalcularVelocidad

La clase `FiltroCalcularVelocidad` implementa la interfaz `Filtro` y proporciona métodos para calcular la velocidad de un motor.
Los atributos de la clase incluyen:

- `incrementoVelocidad:` que indica el cambio en la velocidad.
- `MAX_REVOLUCIONES`, que representa el máximo de revoluciones permitidas antes de ajustar la velocidad.

private void setIncrementoVelocidad(double revoluciones, EstadoMotor estado)
Establece el incremento de velocidad según las revoluciones y el estado del motor, en caso de que las revoluciones sean $\geq$ `MAX_REVOLUCIONES` hace que el `incrementoVelocidad` $=$ 0.

- **revoluciones:** Las revoluciones actuales del motor.
- **estadoMotor:** El estado actual del motor.

public double ejecutar(double revoluciones, EstadoMotor estadoMotor)
Calcula la velocidad del motor en función de las revoluciones y el estado del motor, y devuelve nuevo valor para las revoluciones del motor. En caso de ser menor que 0 devuelve `revoluciones = 0`.

- **revoluciones:** Las revoluciones actuales del motor.
- **estadoMotor:** El estado actual del motor.

6 Clase FiltroRepercutirRozamiento

La clase `FiltroRepercutirRozamiento` implementa la interfaz `Filtro` y proporciona métodos para calcular el rozamiento.
Los atributos de la clase incluyen:

- `ROZAMIENTO`, es una constante que representa el rozamiento y tiene un valor de 5.

public double ejecutar(double revoluciones, EstadoMotor estadoMotor)
Devuelve las revoluciones quitando una cantidad fija(`ROZAMIENTO`) debido al rozamiento. En caso de ser menor que 0 devuelve `revoluciones = 0`.

- **revoluciones:** Las revoluciones actuales del motor.
- **estadoMotor:** El estado actual del motor.

7 Clase Objetivo

La clase `Objetivo` representa un objetivo al que se le pueden aplicar filtros para calcular su velocidad, distancia, etc.
Los atributos de la clase incluyen:

- `velocidad_lineal:` la velocidad lineal del objetivo.
- `velocidad_angular:` la velocidad angular del objetivo.
- `distancia:` la distancia recorrida por el objetivo.
- `revoluciones:` las revoluciones actuales del objetivo.
- `RADIO:` constante que define el radio del objetivo = 0.15.
- `tiempoInicial:` el tiempo inicial del objetivo.

public Objetivo()
Constructor de la clase `Objetivo` que inicializa los atributos a 0 y establece el tiempo inicial.
public double getVelocidad_lineal()
Devuelve la velocidad lineal del objetivo calculada en la función `calcula_velocidad_lineal()`.
public double getVelocidad_angular()
Devuelve la velocidad angular del objetivo calculada en la función `calcula_velocidad_angular()`
public double getDistancia()

Devuelve la distancia recorrida por el objetivo calculada en la función `calcula_distancia()`

public double getRevoluciones()

Devuelve las revoluciones del objetivo.

public void calcula_velocidad_lineal()

Calcula la velocidad lineal del objetivo a partir de la formula `2.0 * pi * RADIO * this.revoluciones * (60 / 1000)` ;

public void calcula_velocidad_angular()

Calcula la velocidad angular del objetivo a partir de la formula `2.0 * pi * 60` ;

public void calcula_distancia()

Calcula la distancia recorrida por el objetivo a patir de la diferencia entre el `tiempoInicial` y el instante actual ,y esta diferencia multiplicada por la `velocidad_lineal` previamente calculada.

public void ejecutar(double revoluciones, EstadoMotor estadoMotor)

Establece las `revoluciones` y ejecuta los cálculos de la `velocidad_lineal`, la `velocidad_angular` y la `distancia`.

- **revoluciones:** Las revoluciones actuales del motor.
- **estadoMotor:** El estado actual del motor.

8 Enum EstadoMotor

La enumeracion `EstadoMotor` define los estados en los que está el motor:

- **APAGADO**
- **ENCENDIDO**
- **ACELERANDO**
- **FRENANDO**