



Escuela Especializada en Ingeniería ITCA-FEPADE

Escuela de Ingeniería en Computación

Diseño de arquitectura de sistemas

Estudiantes:

Carolina Raquel Velásquez Rauda-193820

Adriel Mauricio Mendez Reyes-105221

Docente:

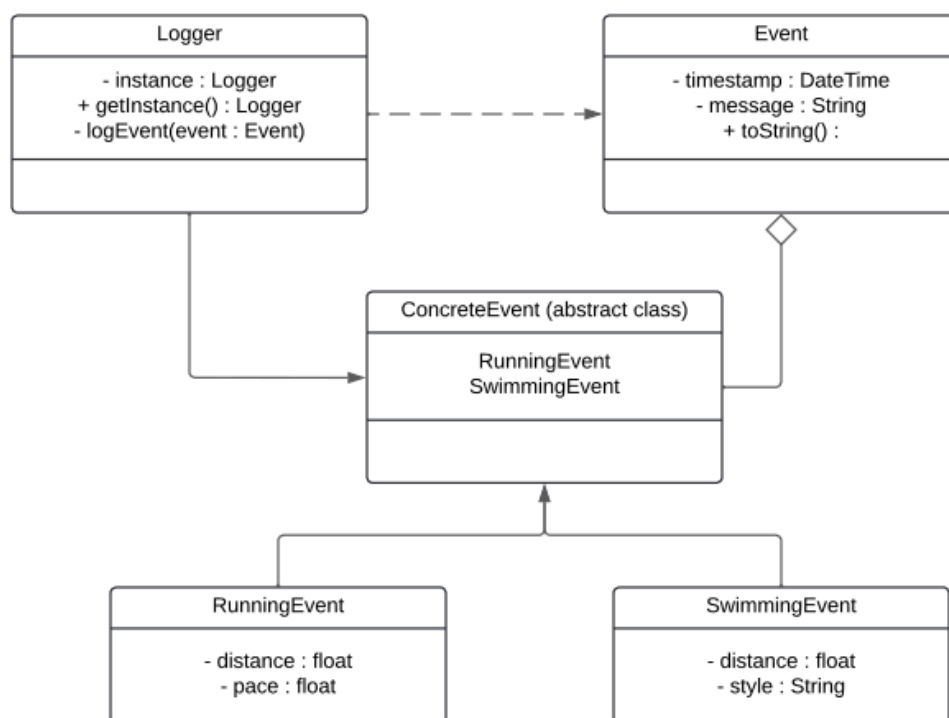
Jose Edgardo HenriqueZ Calderon

03/05/2024

Ejercicio 1

Supongamos que está desarrollando un sistema de registro de eventos para una aplicación de seguimiento de actividad física. Quiere asegurarse de que solo haya una instancia de la clase `Logger` en todo el sistema para evitar la creación excesiva de objetos y garantizar que todos los eventos se registren en un único lugar. Aplicar patrones de diseño Singleton para que la clase solo pueda tener una instancia y que esta instancia pueda ser accedida globalmente y Factory Method para crear un sistema de registro que maneje diferentes tipos de eventos de forma eficiente, detallar el patrón explicado a través de un diagrama UML y codificación.

Diagrama UML



Clases:

- **Logger**: Esta clase representa el registrador de eventos único del sistema.
- **Event**: Esta clase representa un evento genérico que se registra en el sistema.

- **ConcreteEvent:** Esta clase abstracta define la interfaz para eventos concretos.

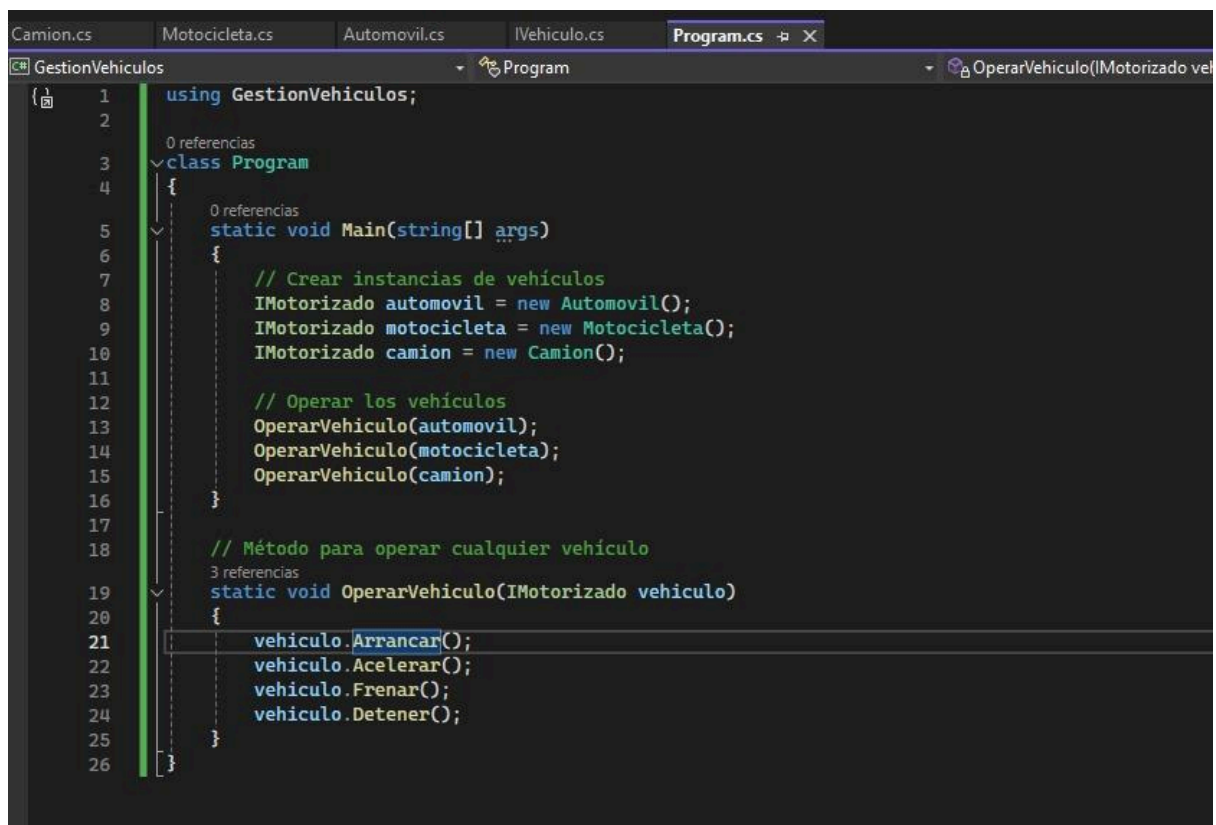
Relaciones:

- **Singleton:** La clase Logger sigue el patrón Singleton, lo que significa que solo puede existir una instancia de esta clase en todo el sistema.
- **Factory Method:** La clase Logger utiliza el patrón Factory Method para crear instancias de la clase Event concreta correspondiente. Esto permite al sistema manejar diferentes tipos de eventos de forma eficiente.

Ejercicio 2

Supongamos que estás desarrollando un sistema de gestión de vehículos. Los vehículos pueden ser de diferentes tipos, como automóviles, motocicletas y camiones. Utiliza los principios SOLID para diseñar e implementar las clases que representan estos vehículos. Anexar capturas de pantalla de la resolución del ejercicio. (Pruebas y funcionalidades).

Programa principal



```

1  using GestionVehiculos;
2
3  0 referencias
4  class Program
5  {
6      0 referencias
7      static void Main(string[] args)
8      {
9          // Crear instancias de vehiculos
10         IMotorizado automovil = new Automovil();
11         IMotorizado motocicleta = new Motocicleta();
12         IMotorizado camion = new Camion();
13
14         // Operar los vehiculos
15         OperarVehiculo(automovil);
16         OperarVehiculo(motocicleta);
17         OperarVehiculo(camion);
18     }
19
20     // Método para operar cualquier vehículo
21     3 referencias
22     static void OperarVehiculo(IMotorizado vehiculo)
23     {
24         vehiculo.Arrancar();
25         vehiculo.Acelerar();
26         vehiculo.Frenar();
27         vehiculo.Detener();
28     }
29 }

```

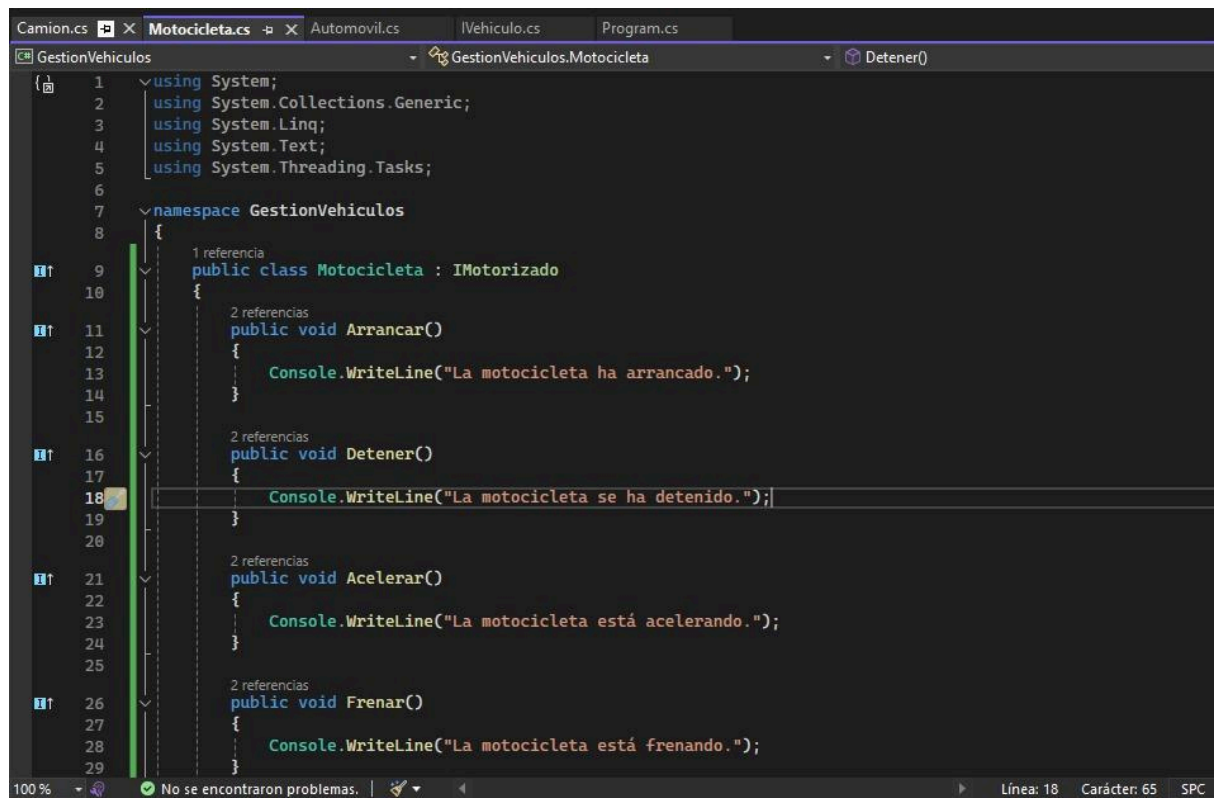
Clase IVehículo

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GestionVehiculos
8  {
9      1 referencia
10     public interface Ivehiculo
11     {
12         4 referencias
13         void Arrancar();
14         4 referencias
15         void Detener();
16     }
17
18     // Definición de la interfaz IMotorizado que hereda de Ivehiculo
19     7 referencias
20     public interface IMotorizado : Ivehiculo
21     {
22         4 referencias
23         void Acelerar();
24         4 referencias
25         void Frenar();
26     }
27 }
```

Clase Automovil

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GestionVehiculos
8  {
9      1 referencia
10     public class Automovil : IMotorizado
11     {
12         2 referencias
13         public void Arrancar()
14         {
15             Console.WriteLine("El automóvil ha arrancado.");
16         }
17
18         2 referencias
19         public void Detener()
20         {
21             Console.WriteLine("El automóvil se ha detenido.");
22         }
23
24         2 referencias
25         public void Acelerar()
26         {
27             Console.WriteLine("El automóvil está acelerando.");
28         }
29
30         2 referencias
31         public void Frenar()
32         {
33             Console.WriteLine("El automóvil está frenando.");
34         }
35     }
36 }
```

Clase motocicleta

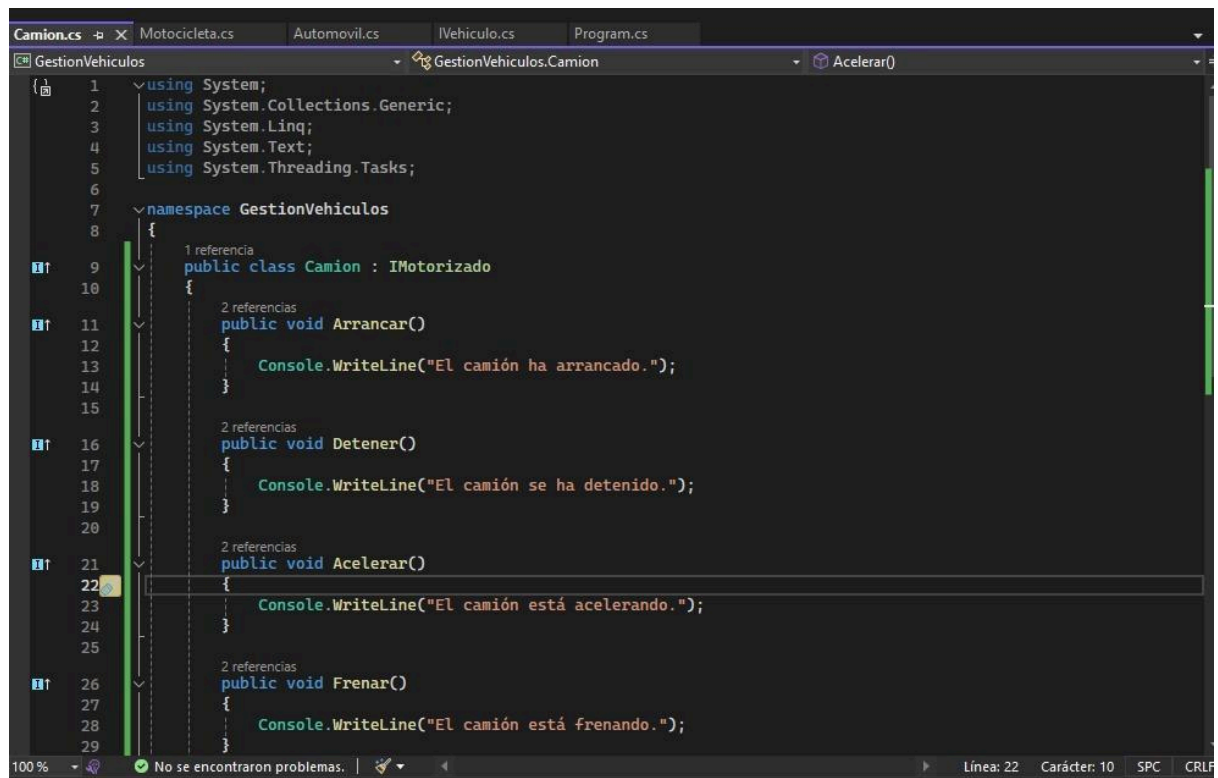


```
Camion.cs | X | Motocicleta.cs | Automovil.cs | IVehiculo.cs | Program.cs
GestionVehiculos | GestionVehiculos.Motocicleta | Detener()

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GestionVehiculos
8  {
9      1 referencia
10     public class Motocicleta : IMotorizado
11     {
12         2 referencias
13         public void Arrancar()
14         {
15             Console.WriteLine("La motocicleta ha arrancado.");
16         }
17
18         2 referencias
19         public void Detener()
20         {
21             Console.WriteLine("La motocicleta se ha detenido.");
22         }
23
24         2 referencias
25         public void Acelerar()
26         {
27             Console.WriteLine("La motocicleta está acelerando.");
28         }
29
30         2 referencias
31         public void Frenar()
32         {
33             Console.WriteLine("La motocicleta está frenando.");
34         }
35     }
36 }
```

100 % | No se encontraron problemas. | Línea: 18 | Carácter: 65 | SPC

Clase Camion



```
Camion.cs | X | Motocicleta.cs | Automovil.cs | IVehiculo.cs | Program.cs
GestionVehiculos | GestionVehiculos.Camion | Acelerar()

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GestionVehiculos
8  {
9      1 referencia
10     public class Camion : IMotorizado
11     {
12         2 referencias
13         public void Arrancar()
14         {
15             Console.WriteLine("El camión ha arrancado.");
16         }
17
18         2 referencias
19         public void Detener()
20         {
21             Console.WriteLine("El camión se ha detenido.");
22         }
23
24         2 referencias
25         public void Acelerar()
26         {
27             Console.WriteLine("El camión está acelerando.");
28         }
29
30         2 referencias
31         public void Frenar()
32         {
33             Console.WriteLine("El camión está frenando.");
34         }
35     }
36 }
```

100 % | No se encontraron problemas. | Línea: 22 | Carácter: 10 | SPC | CRLF

Explicación

1.Principio de Responsabilidad Única (SRP):

Este principio se refiere a que una clase debe tener una única razón para cambiar. En nuestro diseño:

Las interfaces IVehiculo e IMotorizado tienen una única responsabilidad: definir el comportamiento básico de un vehículo y el comportamiento específico de un vehículo motorizado, respectivamente.

Las clases concretas Automovil, Motocicleta y Camion implementan estas interfaces y se centran únicamente en proporcionar la implementación de los métodos definidos en las interfaces. No tienen la responsabilidad de cambiar si cambia la forma en que se comportan los vehículos.

2. Principio de Abierto/Cerrado (OCP):

Este principio establece que una clase debe estar abierta para la extensión pero cerrada para la modificación. En nuestro diseño:

Las interfaces IVehiculo e IMotorizado están abiertas para la extensión. Podemos crear nuevas interfaces para diferentes tipos de vehículos sin modificar estas interfaces existentes.

Las clases concretas Automovil, Motocicleta y Camion están cerradas para la modificación. Podemos agregar nuevos comportamientos específicos de vehículos creando nuevas clases que implementen las interfaces existentes, sin necesidad de cambiar el código de estas clases.

3. Principio de Sustitución de Liskov (LSP):

Este principio establece que los objetos de un programa deben ser reemplazables por instancias de sus subtipos sin afectar la corrección del programa. En nuestro diseño:

Todas las clases concretas (Automovil, Motocicleta y Camion) implementan la interfaz IMotorizado, por lo que pueden ser reemplazadas por instancias de IMotorizado sin afectar el comportamiento del programa.

4. Principio de Segregación de la Interfaz (ISP):

Este principio sugiere que los clientes no deben verse obligados a depender de interfaces que no utilizan. En nuestro diseño:

Hemos definido interfaces específicas (IVehiculo e IMotorizado) que contienen solo los métodos necesarios para el comportamiento de los vehículos.

Las clases concretas implementan solo las interfaces que necesitan. Por ejemplo, la clase Motocicleta implementa la interfaz IMotorizado, ya que es un vehículo motorizado, pero no implementa ninguna otra interfaz que no sea necesaria.

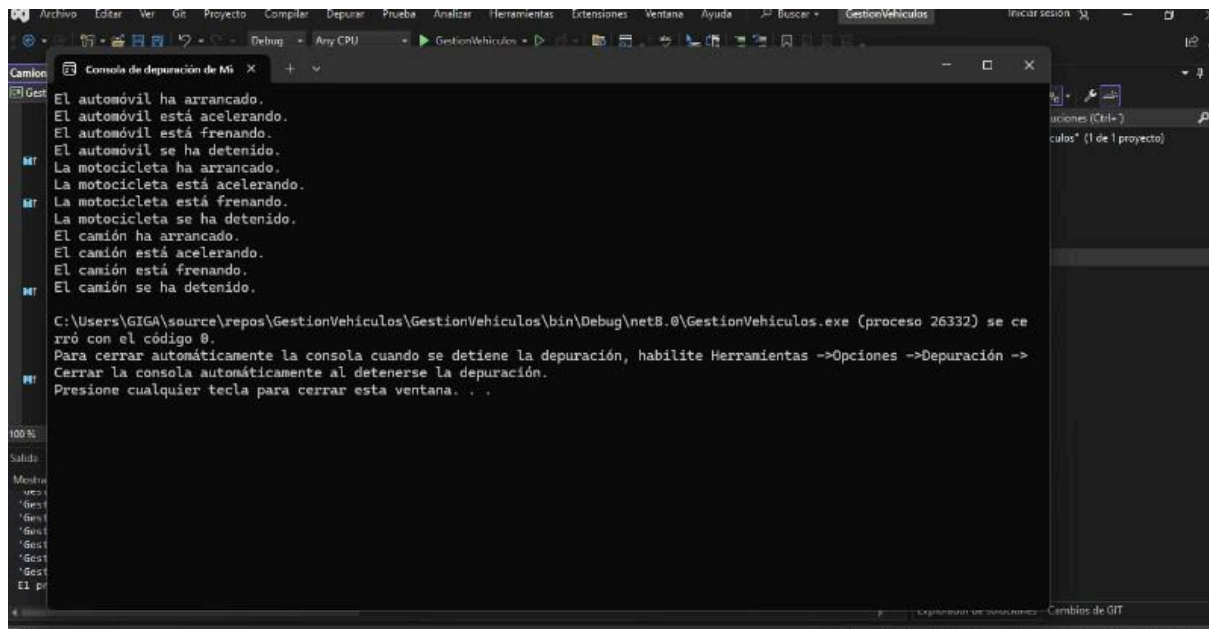
5. Principio de Inversión de Dependencia (DIP):

Este principio establece que los módulos de alto nivel no deben depender de módulos de bajo nivel, sino de abstracciones. En nuestro diseño:

El método OperarVehiculo en la clase Program depende de la abstracción IMotorizado, que es una interfaz de alto nivel. No depende de las clases concretas Automovil, Motocicleta o Camion.

Las clases concretas dependen de las interfaces (IVehiculo e IMotorizado) en lugar de depender de otras clases concretas.

Prueba de Funcionamiento



```
Archivo Editar Ver Git Proyecto Compiler Depurar Prueba Analizar Herramientas Extensiones Ventana Ayuda GestionVehiculos Iniciar sesión
Debug Any CPU GestionVehiculos
Camion
El automóvil ha arrancado.
El automóvil está acelerando.
El automóvil está frenando.
El automóvil se ha detenido.
La motocicleta ha arrancado.
La motocicleta está acelerando.
La motocicleta está frenando.
La motocicleta se ha detenido.
El camión ha arrancado.
El camión está acelerando.
El camión está frenando.
El camión se ha detenido.
C:\Users\GIGA\source\repos\GestionVehiculos\GestionVehiculos\bin\Debug\net8.0\GestionVehiculos.exe (proceso 26332) se cerró con el código 8.
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración -> Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```