



ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA



Universidad de las Fuerzas Armadas

“Creación de Objetos y UML”

Integrantes:

1. Brayan Stehp Mendoza Márquez

Curso:

1323

Asignatura De Programación orientada a objetos

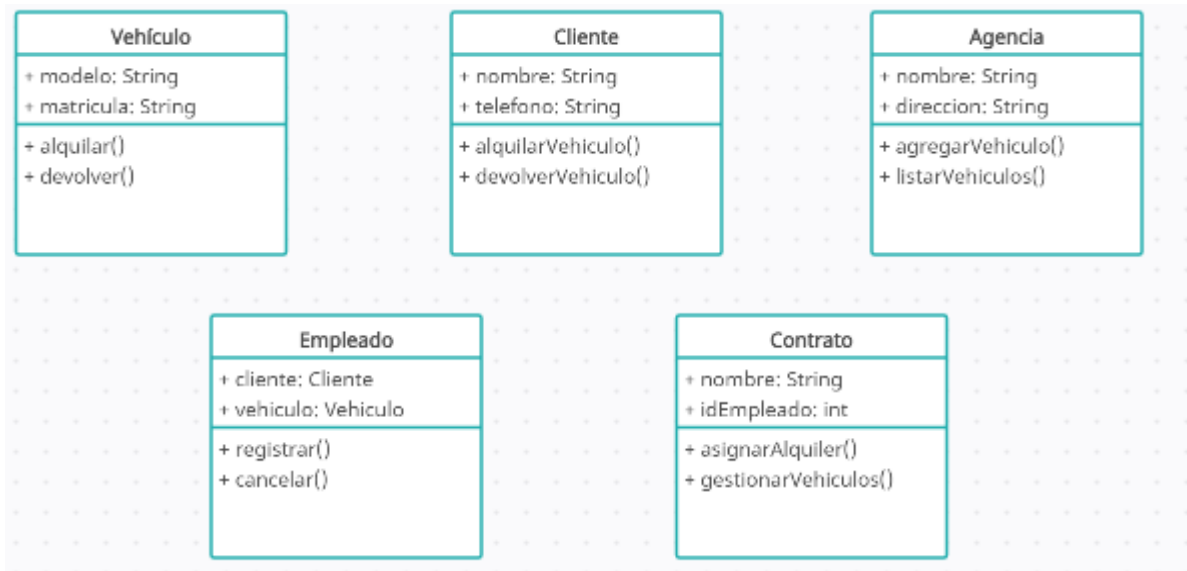
Docente:

LUIS ENRIQUE JARAMILLO MONTAÑO

05 de diciembre de 2024

1. Diseñe 5 objetos diferentes con su correspondiente diagrama UML, asegurándose de mostrar las relaciones entre ellos.

1.1 UML



1.2 Relaciones

Para realizar relaciones entre objetos e aprendido por diversas maneras, pero la que mas me gusto fue la forma en la que me encelaron en el instituto en que estaba estudiando ya que hay me hicieron ver que casi cualquier cosa puede tener una relación y con eso aprendí bastante rápido un ejemplo rápido aparte del que estoy haciendo en este deber seria:

Persona ↔ YouTube como seria esta relación seria así; **una persona ve videos en YouTube y YouTube es una aplicación de videos que permite ver videos a las personas.**

Vehículo ↔ Agencia

Relación: Asociación

Una Agencia tiene una lista de Vehículos que administra.

Cliente ↔ Vehículo

Relación: Composición

Un Cliente puede alquilar uno o más Vehículos, los cuales dependen del cliente.

Empleado ↔ Cliente y Vehículo

Relación: Asociación

Un Empleado gestiona clientes y vehículos.

Contrato ↔ Cliente y Vehículo

Relación: Agregación

Un Contrato usa instancias de Cliente y Vehículo, pero estas existen independientemente.

1.3 Código

```
IMPORT java.util.ArrayList
IMPORT java.util.List

class Vehiculo {

    public String modelo;
    public String matricula;

    public Vehiculo (String modelo, String matricula) {
        this.modelo = modelo;
        this.matricula = matricula;
    }

    public void mostrar () {
        System.out.println ("El vehiculo con matricula " + matricula + " no  

        sido alquilado.");
    }

    public void devolver () {
        System.out.println ("El vehiculo con matricula " + matricula + " ha sido  

        devuelto.");
    }

}

class Cliente {

    public String nombre;
    public String telefono;

    public Cliente (String nombre, String telefono) {
        this.nombre = nombre;
        this.telefono = telefono;
    }

}
```



```

PUBLIC VOID ADICIONARVEICULO (V) {
    SYSTEM.OUT.PRINTLN ("EL CLIENTE " + NOMBRE + " HA AGREGADO UN
    VEICULO");
}

```

```

}
PUBLIC VOID QUITARVEICULO (V) {
    SYSTEM.OUT.PRINTLN ("EL CLIENTE " + NOMBRE + " HA QUITADO
    SU VEICULO.");
}
}

```

CLASS AGREGAR {

```

PUBLIC STATIC NOMBRES;
PUBLIC STATIC DIRECCION;

```

```

PUBLIC AGREGAR (STATIC NOMBRES, STATIC DIRECCION) {
    THIS.NOMBRES = NOMBRES;
    THIS.DIRECCION = DIRECCION;
    THIS.VEHICULO = NEW AGREGAR(5, 7, 1);
}

```

Edinacho

```

PUBLIC VOID ADELANTARVEICULO (VEICULO VEICULO) {
    VEICULO.ACC (VEICULO);
    SYSTEM.OUT.PRINTLN ("VEICULO " + VEICULO.MODELO + " ADELANTADO");
}

```

```

PUBLIC VOID LISTARVEICULO () {
    SYSTEM.OUT.PRINTLN ("VEICULOS DISPONIBLES EN LA AGENCIA
    + NOMBRE + ":");
    FOR (VEICULO VEICULO : VEICULOS) {
        SYSTEM.OUT.PRINTLN (" - MODELO " + VEICULO.MODELO + ", MODELO " +
        VEICULO.MODELO);
    }
}
}
}

```

CLASS EMERGEN

```

PUBLIC CLIENTE CLIENTE;
PUBLIC VEICULO VEICULO;

```

```

PUBLIC EMERGEN (CLIENTE CLIENTE, VEICULO VEICULO) {
    THIS.CLIENTE = CLIENTE;
    THIS.VEICULO = VEICULO;
}

```

```

PUBLIC VOID RESERVAR (V) {
    SYSTEM.OUT.PRINTLN ("EL CLIENTE " + CLIENTE.NOMBRE + " HA
    RESERVADO EL ALQUILER DEL VEICULO " + VEICULO.MODELO);
}

```



```

public class Cliente {
    public static void main (String [] args) {
        Vehiculo vehiculo1 = new Vehiculo (nombre Toyota, matricula, "ABC-123");
        Vehiculo vehiculo2 = new Vehiculo (nombre Honda, matricula, "XYZ-456");
        Cliente cliente1 = new Cliente (nombre Maria, telefono, "555-1234");
        Agencia agencia = new Agencia (nombre Agencia, direccion, Av. Libertad);
        agencia.agregarVehiculo (vehiculo1);
        agencia.agregarVehiculo (vehiculo2);
        agencia.consultarVehiculos ();
        cliente1.alquilarVehiculo (); //alquilar
        cliente1.devolverVehiculo (); //devolver
        empleado1.registrar (); //registrar
        empleado1.cancelar (); //cancelar
    }
}

```



Código en ejecución

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Vehículo Toyota Corolla agregado a la agencia.
Vehículo Honda Civic agregado a la agencia.
Vehículos disponibles en la agencia Agencia Central:
- Modelo: Toyota Corolla, Matrícula: ABC-123
- Modelo: Honda Civic, Matrícula: XYZ-456
El cliente María López ha alquilado un vehículo.
El cliente María López ha devuelto un vehículo.
El cliente María López ha registrado el alquiler del vehículo Toyota Corolla.
El cliente María López ha cancelado el alquiler del vehículo Toyota Corolla.
PS C:\Users\RVZEN\Github>

```

1.4 Resumen

Para hacer este código, seguimos varias ideas que aprendimos del libro del Ing. y aplicamos varios conceptos importantes de Java. Por ejemplo, usamos ArrayList para manejar listas dinámicas, como la flota de vehículos de la agencia, porque nos permite agregar y manejar elementos fácilmente sin preocuparnos por tamaños fijos, algo que el Ing. explicó súper claro en las clases.

Primero, diseñamos las clases según la UML (que básicamente es como un "mapa" para estructurar el programa). Creamos las clases principales (Vehiculo, Cliente, Agencia y Empleado) y les agregamos atributos y métodos que siguen

Luego, en la clase Agencia, usamos un **ArrayList** para manejar todos los vehículos que tiene la agencia. Esto nos sirvió para agregar vehículos y listar todos los disponibles, algo que es súper práctico porque podemos iterar fácilmente sobre la lista.

También trabajamos con la clase Empleado, donde definimos métodos para registrar y cancelar alquileres. Esto lo hicimos para simular cómo un empleado interactuaría con un cliente y los vehículos.

Finalmente, en el método main (), juntamos todo y simulamos un flujo completo: creamos una agencia, añadimos vehículos, un cliente alquila y devuelve un vehículo, y un empleado registra esa acción. Aquí usamos lo que el Ing. menciona mucho sobre cómo separar bien las responsabilidades entre las clases.

En resumen, usamos varias cosas del libro del Ing., como la importancia de estructurar bien el programa con la UML, el uso de ArrayList para manejar datos de forma flexible, y aplicamos POO para que cada clase tenga una responsabilidad clara. Así nos quedó un programa organizado y funcional.