Recurso Educativo Abierto

POO en Java

Inicio

Programación de computadoras Código fuente Código de red portátil Editor de texto Software de computadora

Hola, bienvenid@ a este Recurso Educativo Abierto de la Programación Orientada a Objetos (POO), este recurso te va ayudar a adquirir conocimientos y habilidades para resolver problemas bajo este paradigma utilizando el Lenguaje

Java te invito a que navegues, explores y pruebes cada uno de los ejercicios e implementaciones en Java.

00:00 00:00

Bienvenida (CC BY)

Presentación

Nombre del curso: Programación II

Unidad: Programación Orienta a Objetos

Tema: Conceptos básicos del paradigma Orientado a Objetos

Clave: CCOS 010

Datos del profesor

Metadatos Profr. y REA: Carmen Cerón Garnica, Problemas y ejercitación
Perfil: Docente del Área de Programación
Orcid: 0000-0001-6480-6810
Nombre del REA: POO con Java
Otros datos importantes: Lenguaje Java

Introducción

Comenzamos

Es importante revisar los conocimientos previos al lenguaje java, las habilidades y pensamiento computacional parala solución de problemas mediante la programación.

Para poder realizar los ejercicios debes instalar software de edición de código y el compilador del lenguaje Java

Enlaces Web de Software

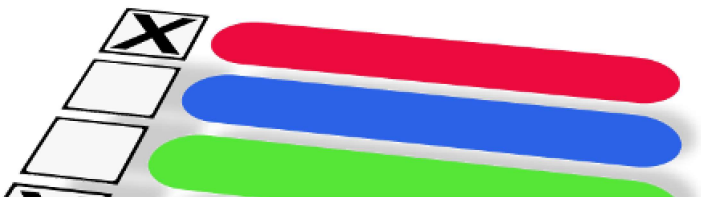
- Editores para Java:
  - Entorno de Desarrollo Netbeans: https://netbeans.org/
  - Editor eclipse: https://www.eclipse.org/downloads/
- Java Developer Center:
  - https://www.oracle.com/technetwork/es/java/javase/downloads/index.html

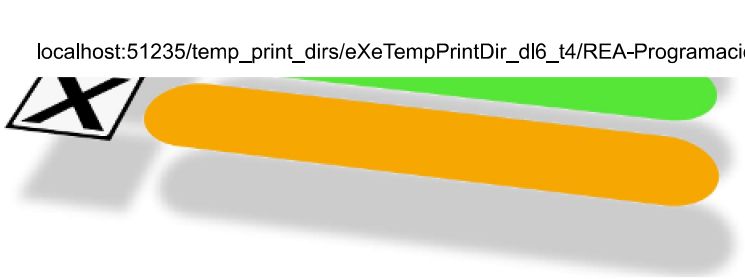
Enlaces a tutoriales de instalación de Eclipse y Java:

Repasemos un poco

Selecciona las respuestas correctas y pulsa sobre el botón ✓

10 0 0 0 00:00





Pulse aquí para jugar

Contenido

Temas

1. Conceptos Básicos POO

1.1 Clase y Objeto

1.2 Abstracción de datos

1.3 Encapsulamiento

1.4 Herencia

1.5 Polimorfismo

Ficheros adjuntos

Materiales Visuales:

- Clases y Objetos (Ventana nueva)
- Herencia (Ventana nueva)
- Polimorfismo (Ventana nueva)

1. Principios Básicos de la POO

Principios Básicos de la POO

Principio 1:Abstracción:

La abstracción se basa en obtener la información esencial de los objetos y cosas simples para representar la complejidad.

Por lo cual, estos objetos parte de un sistema representan código subyacente, ocultando los detalles complejos al usuario.

Principio 2. Encapsulamiento:

El encapsulamiento permite limitar el acceso a los datos e información de los objetos, protegiéndolos y ocultando detalle de los mismos datos.

Principio 3: Herencia

La herencia en POO permite que se defina una jerarquía entre clases y poder compartir atributos y métodos comunes puedan ser reutilizados.

Principio 4: Polimorfismo

El polimorfismo nos permite diseñar objetos que pueden tener varios comportamientos, lo que nos permite procesar objetos de diferentes maneras.

Los principios de la Programación Orientada a Objetos son: la Abstracción, el Encapsulamiento, la Herencia y el Polimorfismo

Beneficios de la POO

Ventajas de la Programación Orientada a Objetos

- Reutilización del código.
- Convierte cosas complejas en estructuras simples reproducibles.

- Evita la duplicación de código.
- Permite trabajar en equipo gracias al encapsulamiento ya que minimiza la posibilidad de duplicar funciones cuando varias personas trabajan sobre un mismo objeto al mismo tiempo.
- Al estar la clase bien estructurada permite la corrección de errores en varios lugares del código.
- Protege la información a través de la encapsulación, ya que solo se puede acceder a los datos del objeto a través de propiedades y métodos privados.
- La abstracción nos permite construir sistemas más complejos y de una forma más sencilla y organizada.

1.1 Clase y Objeto

Clase

Una clase es "un tipo definido por el usuario; son los bloques de construcción fundamentales de los programas orientados a objetos" (Joyanes & Zahonero, 2011, p. 192).

Es la unidad básica (o estructura) que encapsula toda la información de un Objeto.

Es un patrón para la definición de atributos y métodos para un tipo de objetos.

A través de ella podemos modelar (una Casa, un Auto, una Cuenta Corriente, etc.)

Objeto

Un objeto es “una entidad (algo) que tiene un estado, un comportamiento y una identidad” (Booch, como se citó en Joyanes & Zahonero, 2011). También recibe el nombre de "instancia"

Sus características son:

- Identidad: un identificador
- Estado: un conjunto de propiedades (atributos)
- Comportamiento: Un conjunto de operaciones (métodos)
- Un objeto se caracteriza por un numero de operaciones y un estado que recuerda el efecto de operaciones.
  El estado se representa: atributos y relaciones
- El comportamiento se representa: por sus operaciones y métodos.

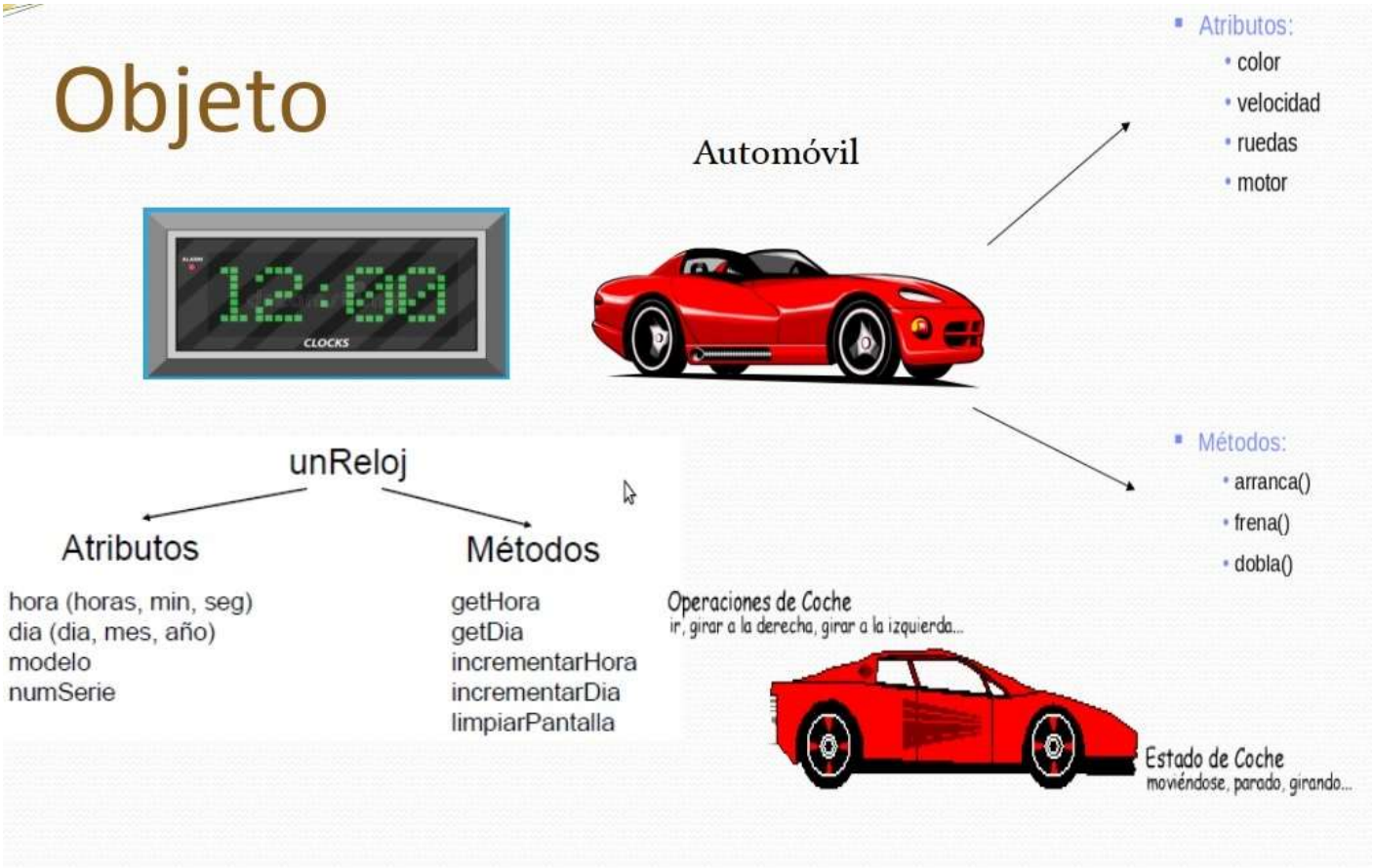
  Objeto= identidad + estado + comportamiento

Atributos y Métodos (operaciones)

- Atributo: Es una característica fundamental de cada objeto de una clase.
    - Una objeto puede tener varios atributos
    - Todos los atributos tienen un valor.
- Métodos (operaciones): Es una acción que se realiza sobre un objeto para consultar o modificar su estado

Esto quiere decir que una clase nos ayuda a describir a un nuevo grupo de objetos que comparten características dentro de un programa. Para poder definir estas clases se usan dos tipos elementos o miembros de elementos:

- **Atributos** - son los datos o la información que describen las características esenciales de los objetos o instancias de la clase.
- **Métodos** - son las operaciones que pueden realizar las instancias (objetos), es decir, sus acciones.



Referencias:

1. Joyanes, A. (2011). Programación en Java6. Algoritmos, programación orientada a objetos e interfaz gráfica de usuario (3ª ed). Mc Graw Hill.
2. James Martin, James J. (2007), Análisis y Diseño Orientado a Objetos (3ª. Ed.), Prentice Hall.

Lenguaje de Modelado Unificado UML

UML, es un Lenguaje de Modelado Unificado para representar relaciones entre elementos de un sistema de información.

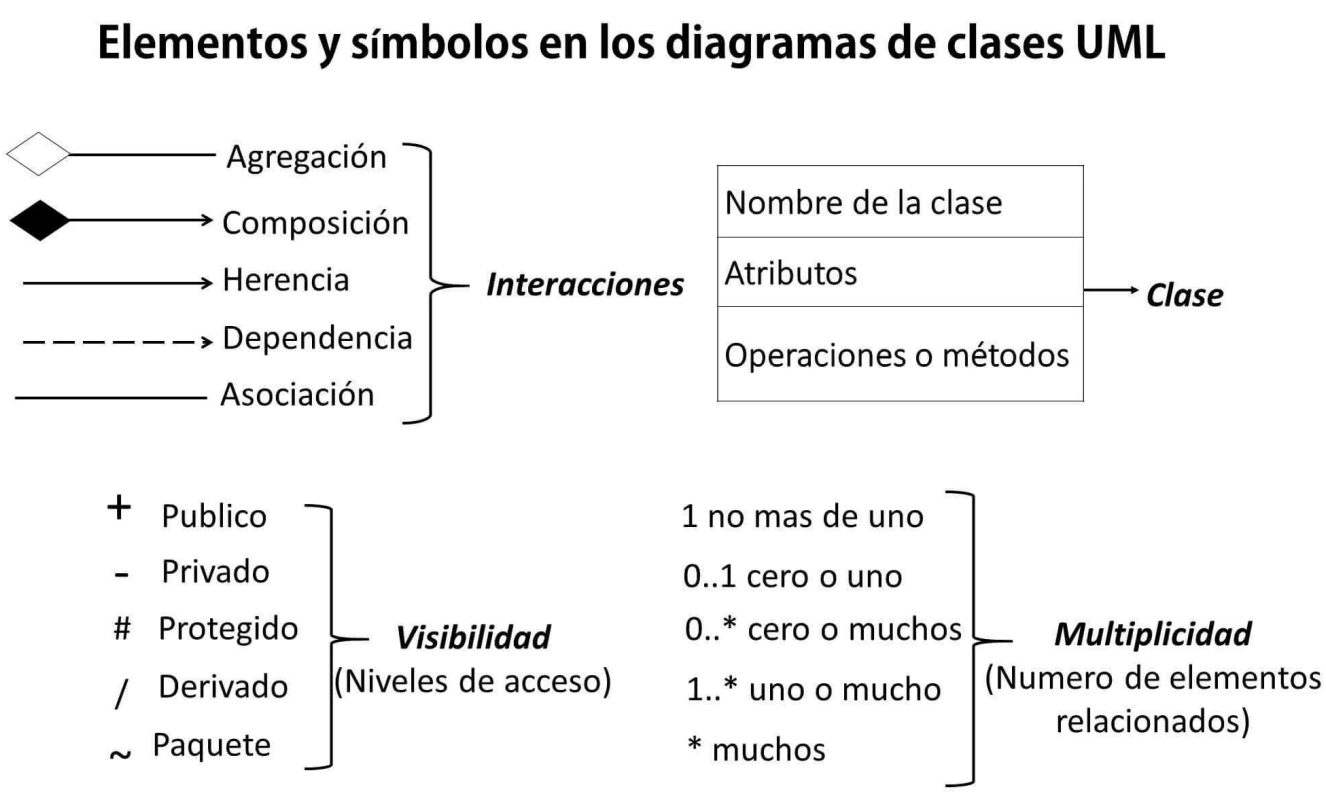
El modelado sirve para el diseño de sistemas, aun en aplicaciones de pequeño tamaño se obtienen beneficios de modelado, sin embargo es un hecho que entre más grande y más complejo es el sistema, más importante es el papel de que juega el modelado por una simple razón: "El hombre hace modelos de sistemas complejos porque no puede entenderlos en su totalidad".

UML es una técnica para la especificación sistemas en todas sus fases. Nació en 1994 cubriendo los aspectos principales de todos los métodos de diseño antecesores y, precisamente, los padres de UML son Grady Booch, autor del método Booch; James Rumbaugh, autor del método OMT e Ivar Jacobson, autor de los métodos OOSE y Objectory.

La versión 1.0 de UML fue liberada en Enero de 1997 y ha sido utilizado con éxito en sistemas construidos para toda clase de industrias alrededor del mundo: hospitales, bancos, comunicaciones, aeronáutica, finanzas, etc.

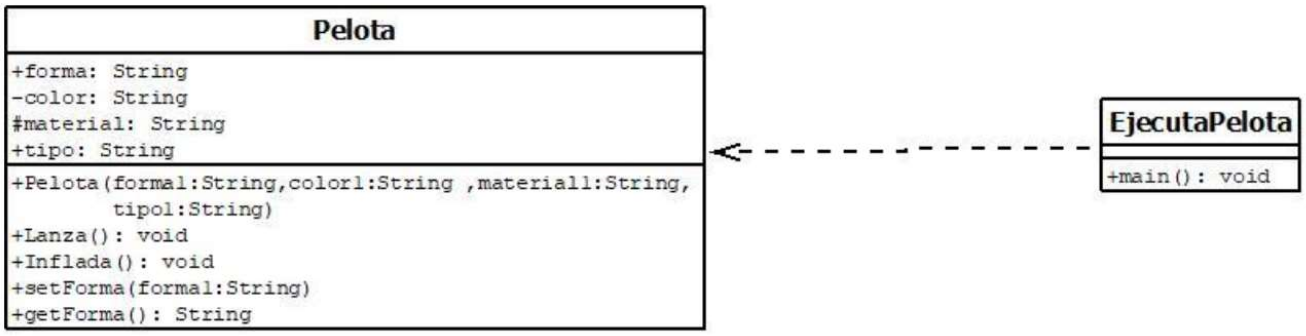
Los principales beneficios de UML son:

- Mejores tiempos totales de desarrollo (de 50 % o más).
- Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
- Establecer conceptos y artefactos ejecutables.
- Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.
- Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
- Mejor soporte a la planeación y al control de proyectos.
- Alta reutilización y minimización de costo



En la POO usamos UML para modelar nuestra solución en Java y diseñamos las clases y sus relaciones entre clases como son los diagramas de clases

Ejemplo de Diagrama de Clase



Referencias

Umbaugh, J., Jacobson, I., & Booch, G. (2005). El Lenguaje Unificado de Modelado Manual de Referencia Segunda Edición. Madrid: Pearson

1.2 Abstracción de datos.

Abstracción

Es la propiedad que se encarga de seleccionar los elementos más significativos de una situación o problema, para lo cual se quitan las propiedades y acciones de un objeto, hasta dejar lo mínimo necesario.

Esta es una etapa vital puesto que es aquí donde se representa la información en relación con la interfaz y el usuario. Las clase utilizan la abstracción para definir al usuario y combina representaciones de datos y métodos para usar, modificar y obtener esa información. Por cada abstracción clave solo debe haber una clase.

Ejemplificación de la Abstracción

Realizar las siguientes actividades:

1. Analizar los objetos las siguientes figuras

A)

B)

C)

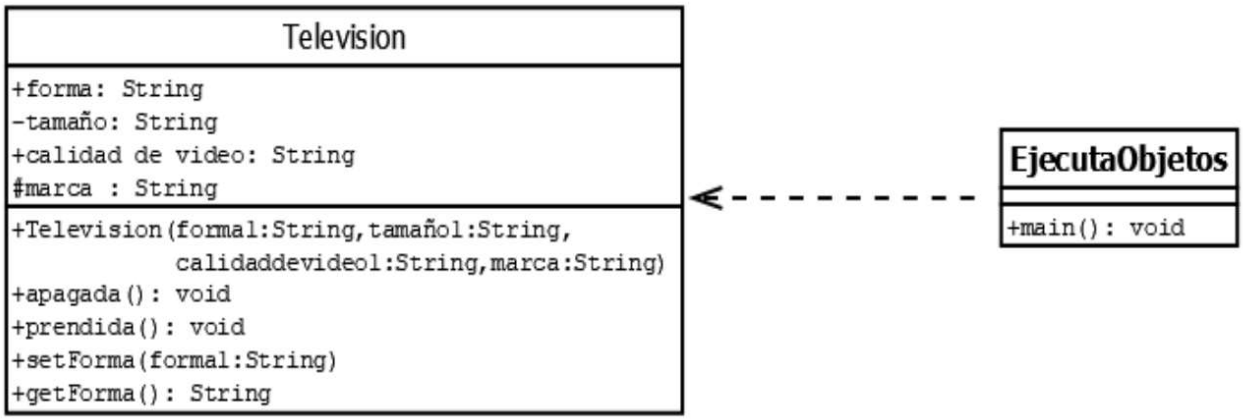
2. Determina sus características (atributos) y acciones (métodos) que cada objeto para lograr obtener la abstracción correspondiente

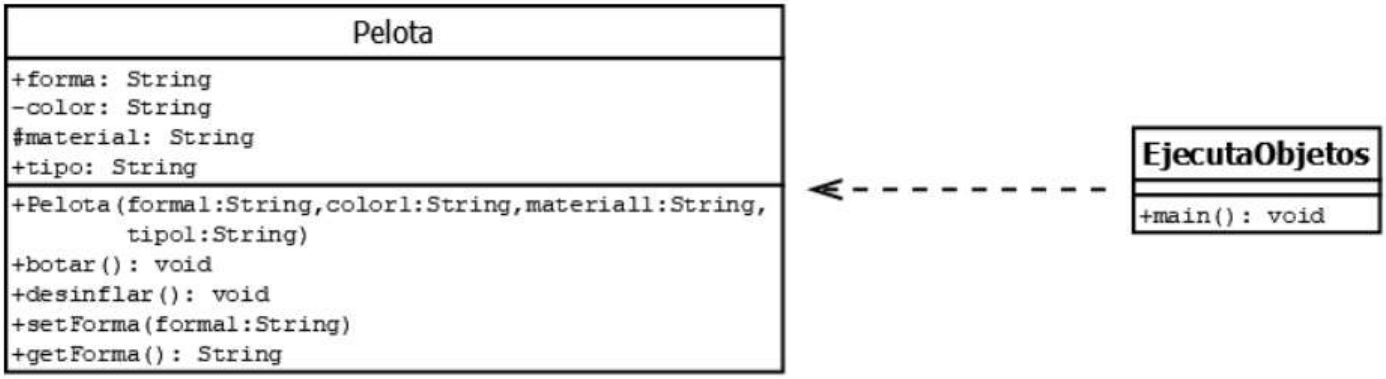
Objeto	Características	Acciones
Televisión	marca, modelo, tamaño	prender, apagar, subir volumen, cambiar canal
Pelota	color, tipo, forma, material	desinflar, inflar, botar
Avión	aerolínea, cantidad de motores, velocidad, capacidad	acelerar, elevarse

4. Realizar sus representaciones en UML utilizando el software de DIA

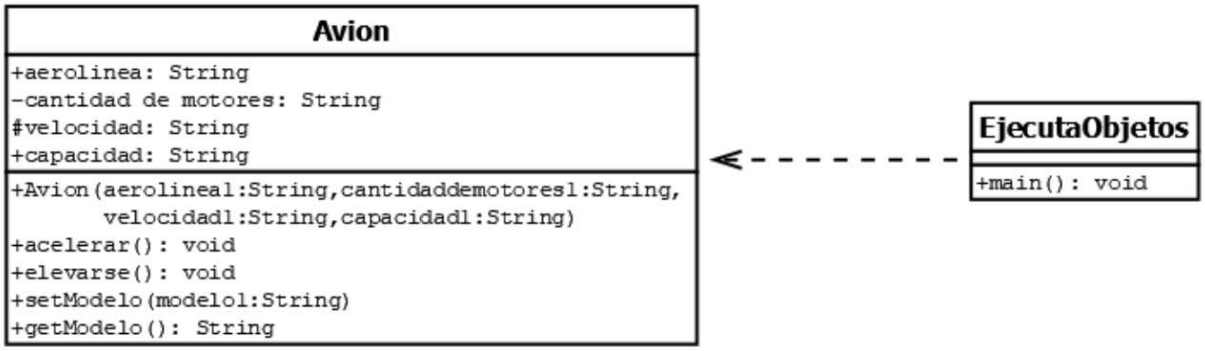
a)

b)





c)



6. Ahora crea su implementación en Java de cada objeto por medio de una clase con sus atributos (variables de instancia) y operaciones(métodos)

a) Televisión

```java
public class Television {
 // nombre de la clase

 // variables de instancia
 public String forma;
 private String tamaño;
 protected String calidaddevideo;
 public String marca;
 // variable de clase
 static int numero=0;

 public Television(String forma1,String tamaño1,String calidaddevideo1,String marca1)
 {
 // constructor inicializar variables
 forma=forma1;
 tamaño=tamaño1;
 calidaddevideo=calidaddevideo1;
 marca=marca1;
 System.out.println ("Construyo una television");
 numero++;
 }
```



```java
public void apagada()
{
System.out.println ("Television apagada");
}

public void prendida()
{
System.out.println ("Television encendida");
}


public void setForma(String forma1)
{
forma=forma1;
}
public String getForma()
{
return forma;
}
```

b) Pelota

```
public class Pelota {
    // nombre de la clase

// variables de instancia
public String forma;
private String color;
protected String material;
public String tipo;
// variable de clase
static int numero=0;

  public  Pelota(String  forma1,  String  color1,  String  material1,  String  tipo1)
{
// constructor inicializar variables
forma=forma1;
color=color1;
material=material1;
tipo=tipo1;
 System.out.println ("Construyo una pelota");
numero++;
}


public void Botar()
{
System.out.println ("La pelota está botando");
}

public void desinflar()
{
System.out.println ("Pelota ya se desinflo");
}


public void setForma(String forma1)
{
forma=forma1;
}

public String getForma()
{
return forma;
}
```

c) Avión

```
public class Avion {
 // nombre de la clase

 // variables de instancia
 public String aerolinea;
 private String cantidaddemotores;
 protected String velocidad;
 public String capacidad;
 // variable de clase
 static int numero=0;

 public Avion(String aerolinea1, String cantidaddemotores1, String velocidad1, String
capacidad1)
 {
 // constructor inicializar variables
 aerolinea=aerolinea1;
 cantidaddemotores=cantidaddemotores1;
 velocidad=velocidad1;
 capacidad=capacidad1;
 System.out.println ("Construyo un avion");
 numero++;

 }


 public void acelerar()
 {
 System.out.println ("El avion acelero");
 }

 public void elevarse()
 {
 System.out.println ("El avion se elevo");
 }


 public void setAerolinea(String aerolinea1)
 {
 aerolinea=aerolinea1;
 }

 public String getAerolinea()
 {
 return aerolinea;
 }
```

7. Crea una clase EjecutaObjetos e instancia las clases al menos con dos objetos que tengan diferentes estados

Television.java Pelota.java Avion.java EjecutarObjetos.java

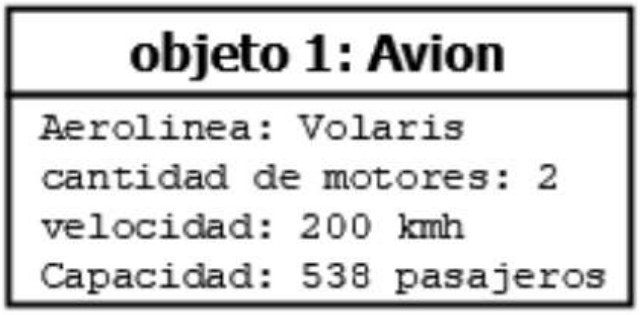
```
1
2  public class EjecutarObjetos {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6  Pelota objeto1=new Pelota("redonda","blanca y negra","cuero","soccer");
7          objeto1.Botar();
8          objeto1.desinflar();
9  Pelota objeto2=new Pelota("redonda","naranja y negra","caucho","baloncesto");
10         objeto2.Botar();
11         objeto2.desinflar();
12
13 Television objeto3=new Television("plana","42 pulgadas","4k a color","LG");
14         objeto3.apagada();
15         objeto3.prendida();
16 Television objeto4=new Television("curva","50 pulgadas","8k a color","Sony");
17         objeto4.apagada();
18         objeto4.prendida();
19
20 Avion objeto5=new Avion("Volaris","2","200 km/h","538 pasajeros");
21         objeto5.acelerar();
22         objeto5.elevarse();
23 Avion objeto6=new Avion("Aeromar","4","280 km/h","630 pasajeros");
24         objeto6.acelerar();
```

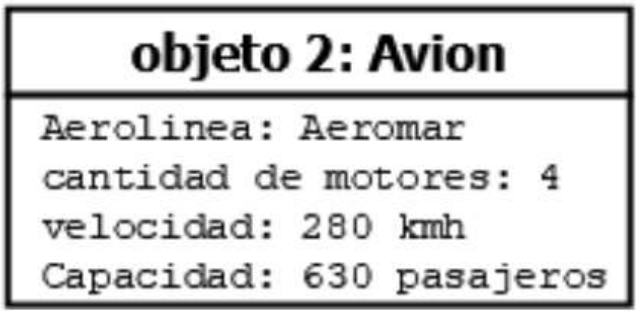
Console Problems Debug Shell

<terminated> EjecutarObjetos [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (23 ene. 2022 19:58:05 – 19:58:09)

```
Construyo una pelota
La pelota esta botando
Pelota ya se desinflo
Construyo una pelota
La pelota esta botando
Pelota ya se desinflo
Construyo una television
Television apagada
Television encendida
Construyo una television
Television apagada
Television encendida
Construyo un avion
El avion acelero
El avion se elevo
```

Representación de objetos en UML





Referencias

Umbaugh, J., Jacobson, I., & Booch, G. (2005). El Lenguaje Unificado de Modelado Manual de Referencia Segunda Edición. Madrid: Pearson

1.3 Encapsulamiento

Encapsulamiento (ocultación de datos)

Significa combinar datos y comportamientos en un paquete, ocultando los detalles de la implementación del usuario del objeto.

El encapsulamiento reúne en una estructura los elementos que pertenecen a una misma entidad, agrupando de datos y operaciones y aumentando la cohesión de los componentes del sistema.

Esta propiedad es útil para el usuario puesto que no necesita conocer el funcionamiento interno para satisfacer su necesidad, los usuarios ven la interfaz sin conocer la implementación. Además sirven como puntos de control, y hace a las respuestas más eficientes a los cambios.

Para poder realizar el encapsulamiento en Java utilizamos los modificadores de visibilidad para controlar el acceso a la información de los objetos como son:--

- **private:** Cuando un método o atributo (variable) es declarada como private, su uso queda restringido solo al interior la clase y no puede acceder a estos desde ninguna clase

#protected: Cuando un método o atributo es visible para las clases que se encuentren en el mismo paquete y para cualquier subclase de esta aunque este en otro paquete. Se utiliza en la herencia entre clases.

+public: Cuando los atributos y métodos tienen la a máxima visibilidad, desde cualquier clase, y paquetes diferente

Visibilidad	Public	Protected	Default	Private
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase del mismo Paquete	SI	SI	SI	NO
Desde una SubClase fuera del mismo Paquete	SI	SI, a través de la herencia	NO	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

Fuente: Abellán, B. (2012). *Modificadores de acceso.* Programacion Orientada a Objetos. Recuperado el 20 de Junio de 2022 de http://picarcodigo.blogspot.com/2012/05/modificadores-de-acceso.html

Referencias

Umbaugh, J., Jacobson, I., & Booch, G. (2005). El Lenguaje Unificado de Modelado Manual de Referencia Segunda Edición. Madrid: Pearson

1.4 Herencia

Herencia

Esta propiedad permite la creación de nuevas clases a partir de una ya existente. La **base** es la clase existente que sirve de modelo y hereda sus características a las **derivadas**, las nuevas clases formadas.

Tipos de herencia

Herencia simple

Una clase posee una sola superclase directa. El gráfico de herencia es un árbol

Herencia múltiple

Una clase posee varias superclases directas. El gráfico de herencia no es un árbol

Mecanismos de herencia:

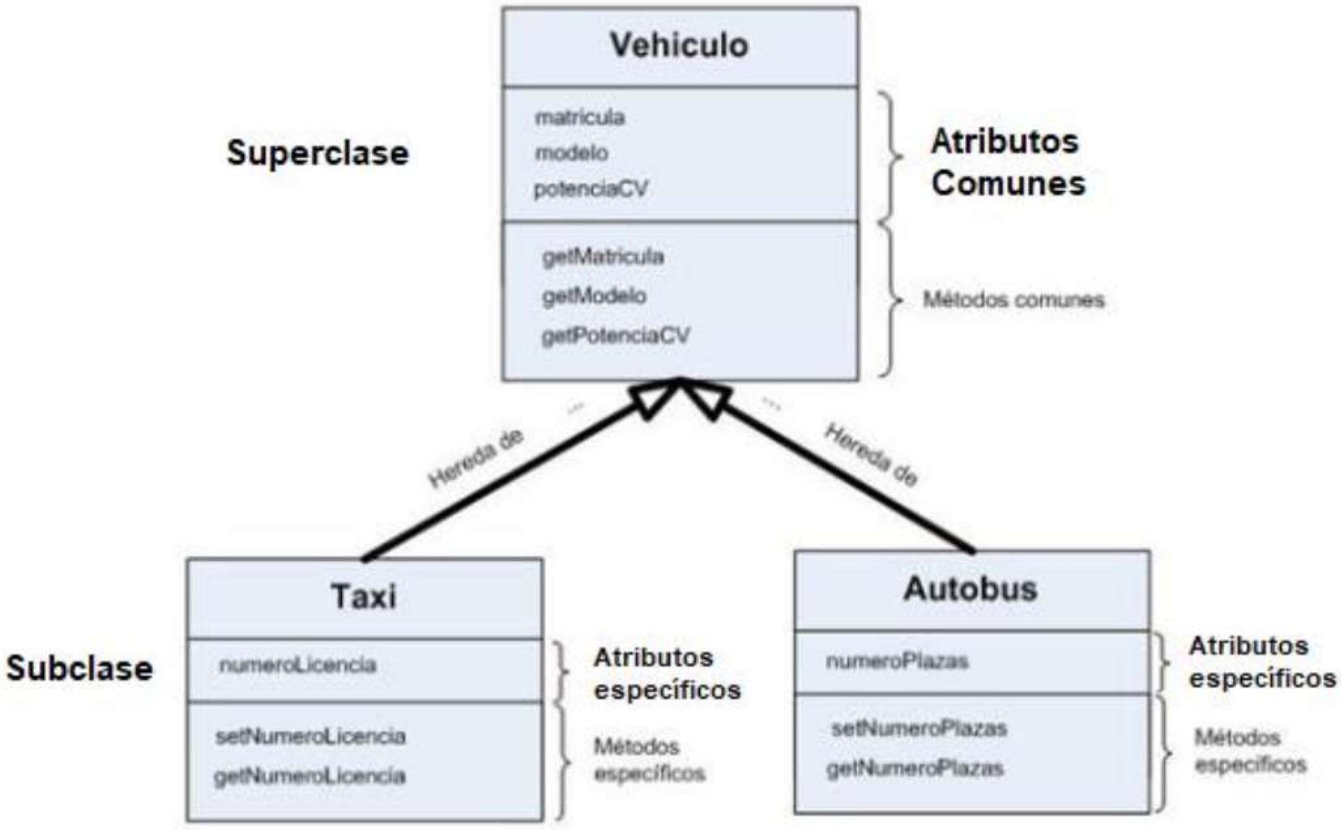
- Enriquecimiento: Se añaden variables y/o métodos
- Substitución: Un método heredado recibe una nueva definición (la antigua no es adecuada al nuevo conjunto de objetos descritos por la superclase)

Visibilidad:

- Pública (public)
- Privada (private)
- Protegida (protected)

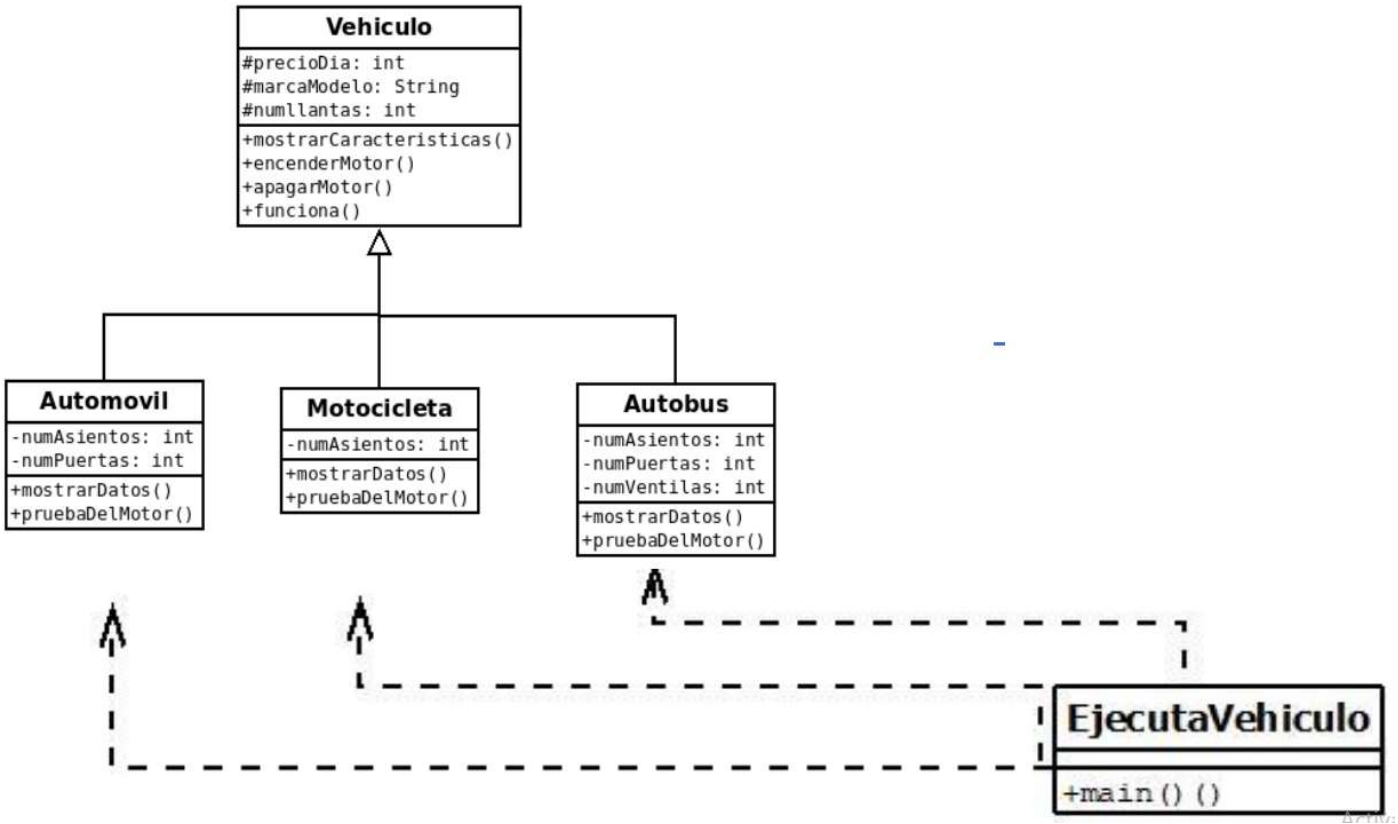
Ejemplificación de la Herencia

La herencia es una relación entre dos clases donde una denominada clase base o general (padre) y la otra
derivada o subclase (hija)
La herencia se conoce como generalización o especialización incluso se observa una jerarquía entre las clases,
en ejemplo:
La clase Vehículo es la superclase (clase padre) y clase taxi (clase hija o derivada) es una subclase.
Así también
la clase Autobús (clase hija o derivada) es una subclase



Programa de Herencia de Vehículos y tipos de vehículos

a) Realiza el modelado de las clases y agrega otro tipo de vehículo al diagrama



b) Posteriormente, prueba el código en Java y agrega el código de nuevo tipo de vehículo

Vehículo:

```
public class Vehiculo{
 // variables de instancia con visibilidad  protegidas
   protected int precioDia;
   protected String marcaModelo;
   protected int numllantas;

   // constructor
   public Vehiculo(String marcaModelo1, int precioDia1, int numllantas1) {
      marcaModelo = marcaModelo1;
      precioDia = precioDia1;
      numllantas = numllantas1;
      System.out.println("construyo el vehículo");
   }

   // metodos
   public void mostrarCaracteristicas(){
      System.out.println("Marca: " + marcaModelo);
      System.out.println("Precio: " + precioDia);
      System.out.println("Numero de llantas: " + numllantas);
   }
   public void encenderMotor(){
      System.out.println("el motor se encendio");
   }
   public void apagarMotor(){
      System.out.println("el motor se apago");
   }
```

```
    }
    public void funciona(){
        System.out.println("el motor de " + marcaModelo + " funciona correctamente");
    }
}
```

Automóvil:

```
public class Automovil extends Vehiculo{
    // variables
    private int numAsientos = 5;
    private int numPuertas;

    public Automovil(String marcaModelo,int precioDia,int numllantas, int asientos, int puertas){
        super(marcaModelo,precioDia,numllantas);
        numAsientos = asientos;
        numPuertas = puertas;
        System.out.println("ha creado un auto");
    }
    public void mostrarDatos(){
        super.mostrarCaracteristicas();
        System.out.println( "Asientos: " + numAsientos);
        System.out.println( "Puertas: " + numPuertas);
    }
    public void pruebaDelMotor(){
        super.encenderMotor();
        super.apagarMotor();
        super.funciona();
    }
}
```

Motocicleta:

```
public class Motocicleta extends Vehiculo{
        //variables
        private int numAsientos = 1;

        public Motocicleta(String marcaModelo, int precioDia, int numllantas, int asientos) {
                super(marcaModelo,precioDia,numllantas);
                numAsientos = asientos;
                System.out.println("Ha creado una motocicleta");
        }
        public void mostrarDatos() {
                super.mostrarCaracteristicas();
                System.out.println("Asientos: "+ numAsientos);

        }
        public void pruebaDelMotor() {
                super.encenderMotor();
                super.apagarMotor();
                super.funciona();
}
```

Autobús:

```
public class Autobus extends Vehiculo{
        //variables de instancia privadas
```

```
        private int numAsientos = 41;
        private int numPuertas;
        private int numVentilas;
         // constructor con parámetros
        public Autobus(String marcaModelo, int precioDia, int numllantas, int asientos, int puertas,
int ventilas) {
                //con super estamos llamando a los atributos de la clase padre o superclase.
                super(marcaModelo,precioDia,numllantas);
                numAsientos = asientos;
                numPuertas = puertas;
                numVentilas = ventilas;
                System.out.println("Ha creado un autobús");
        }
        public void mostrarDatos() {
                super.mostrarCaracteristicas();
                System.out.println("Asientos: " + numAsientos);
                System.out.println("Puertas: " + numPuertas);
                System.out.println("Ventilas: "+ numVentilas);
        }
        public void pruebaDelMotor()
        {
            //con super estamos llamando los métodos de la clase padre o superclase.
              super.encenderMotor();
              super.apagarMotor();
              super.funciona();
       }
```

c) Crea la clase EjecutaVehículo

Ejecuta Vehiculo:

```
public class EjecutaVehiculo {
    public static void main(String[] args){
        // Automovil
        Automovil v1 = new Automovil("volvo 550",120,4,5,4);
        v1.mostrarDatos();
        v1.pruebaDelMotor();

        // Moto
        Motocicleta m1 = new Motocicleta("Italika",120,2,2);
        m1.mostrarDatos();
        m1.pruebaDelMotor();

        //Autobús
        Autobus b1 = new Autobus("Mercedez",300,8,42,2,2);
        b1.mostrarDatos();
        b1.pruebaDelMotor();

        Autobus b2 = new Autobus("Mercedez smart",250,6,25,1,1);
        b2.mostrarDatos();
        b2.pruebaDelMotor();
    }
}
```

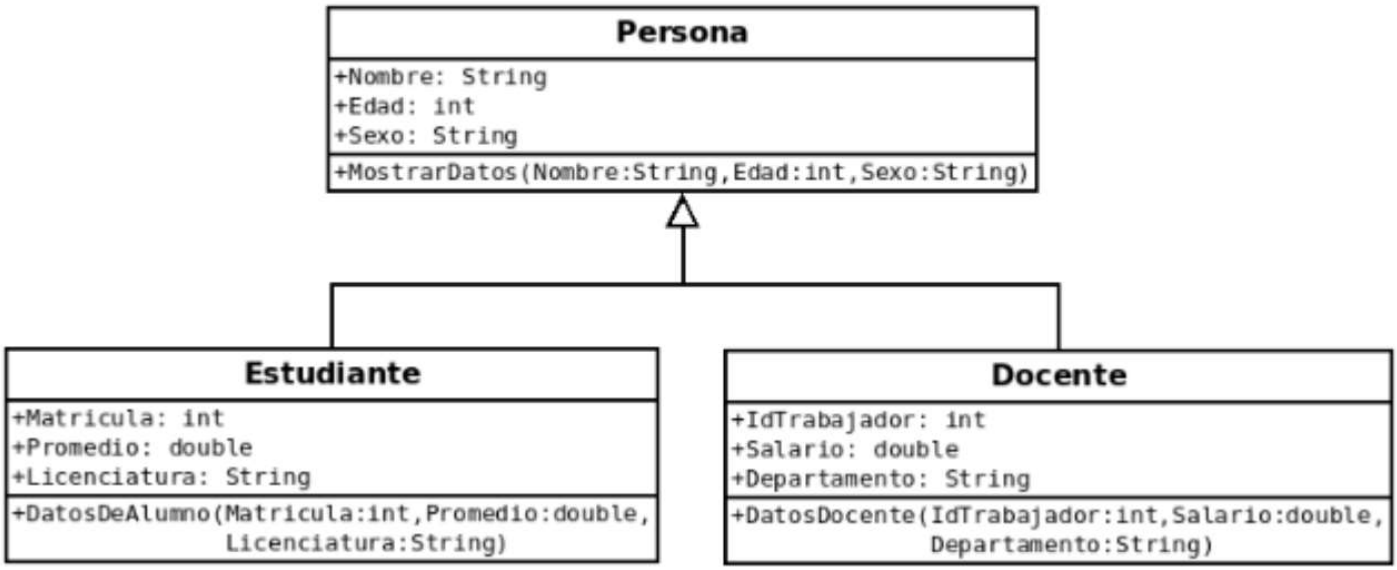
HAZ AHORA:

Programa de herencia Persona y tipos de personas

a) Diseña el diagrama de clases Personas y tres tipos de Personas junto con la relación de dependencia con la clase EjecutaPersonas.

b) Diseña la implementación y solución en Java

c) Genera tres objetos de cada clase en la clase EjecutaPersonas



Referencias

Umbaugh, J., Jacobson, I., & Booch, G. (2005). El Lenguaje Unificado de Modelado Manual de Referencia Segunda Edición. Madrid: Pearson

1.5 Polimorfismo

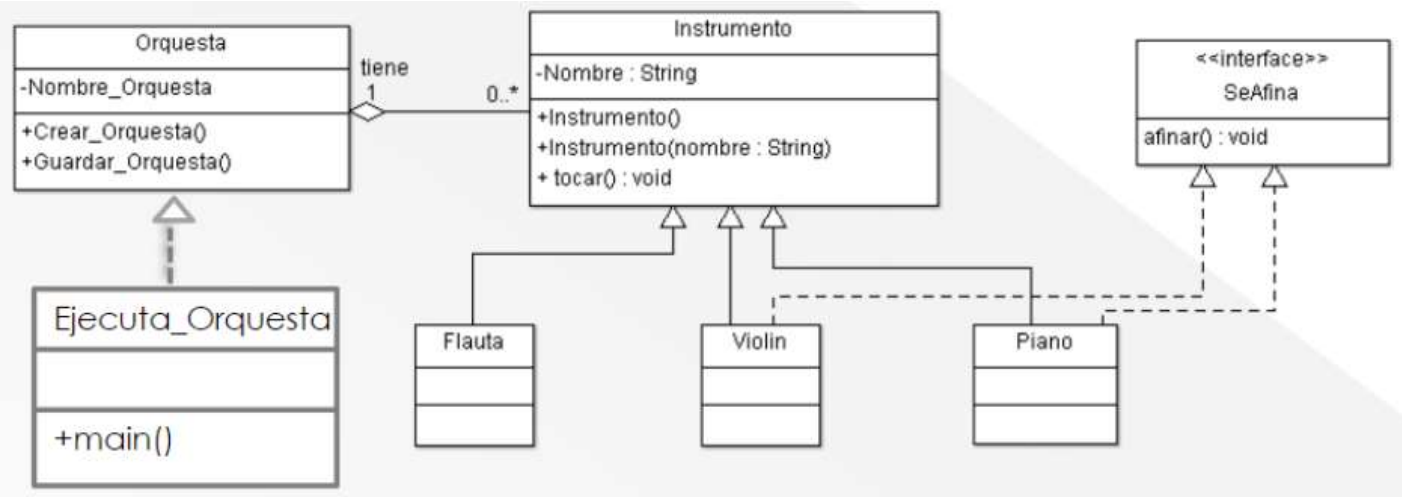
Polimorfismo

Este permite a una operación o función tener el mismo nombre en clases diferentes y actuar de modo distinto en cada una de ellas.

El polimorfismo implica la capacidad de una operación de ser interpretada sólo por el propio objeto que lo invoca. Es decir a pesar de que la función comparta el mismo nombre, tendrá un significado diferente para cada clase y por ende podrá hacer las operaciones que le fueron indicadas en su definición a pesar de ser diferentes.

Ejemplificación de el Polimorfismo

Una orquesta tiene varios instrumentos y cada instrumento toca a su manera. Las orquestas primero afinan los instrumentos afinables y luego tocan, como se demuestra en el diagrama de clases.



```
public class Violin extends Instrumento implements SeAfina
{


public Violin(String nombre)
{
        super(nombre);

}
public void afinar()
{
System.out.println("Afinando instrumento de
cuerda"+super.nombre);
}

}
```

```
public interface SeAfina
{
    public void afinar();

}
```

```
public class Orquesta
{
String Nombre_Orquesta;
public Orquesta(String NombreOrquesta)
{
        Nombre_Orquesta=NombreOrquesta
}

public void Crear_Orquesta()
{
        Piano piano=new Piano ("Piano de C
        piano.afinar();
        Flauta flauta=new Flauta ("flautín");
        flauta.tocar();
}

public void Guardar_Orquesta()
{       System.out.println("se ha guardado la
orquesta "+Nombre_Orquesta);
}
```

```
public class Ejecuta_Orquesta
{


public static void main(String
a[])
{

Orquesta objeto1=new
Orquesta("Bellas Artes");
objeto1.Crear_Orquesta();
objeto1.Guardar_Orquesta();
}
}
```

```
Tocando instrumento Piano de Cola
Afinando instrumento Piano de Cola
Tocando instrumento flautín
se ha guardado la orquesta Bellas Artes
```

```
public class Piano extends Instrumento implements SeAfina{

   public Piano(String nombre1)
   {
                  super(nombre1);
                  super.tocar();
   }

   public void afinar()
         {
         System.out.println("Afinando instrumento "+ super.nombre);
         }



}
```

CÓDIGO

```
public class Orquesta {
      String Nombre_Orquesta;
      public Orquesta(String NombreOrquesta)
      {
            Nombre_Orquesta=NombreOrquesta;

      }

      public void Crear_Orquesta()
      {
            Piano piano=new Piano ("Piano de Cola");           piano.afinar();
            Flauta flauta=new Flauta ("flautin");
flauta.tocar();
            Violin violin=new Violin("Violin electrico");
violin.afinar();         violin.Mantenimiento();
        Tuba tuba= new Tuba ("Tuba bajo");         tuba.Mantenimiento();
        tuba.afinar();
```

```
        }

        public void Guardar_Orquesta()
        {
                System.out.println("se ha guardado la orquesta "+Nombre_Orquesta);
        }     }
```

INTRUMENTO

```
public class Instrumento {
        String nombre;

        public Instrumento (String nombre )
        {
         this.nombre=nombre;

        }
        public void tocar()
        {
        System.out.println("Tocando instrumento "+ nombre);
        }

          }
public interface DarMantenimiento {

        public void Mantenimiento();
} public interface SeAfina {   public
void afinar();
}
```

TUBA

```
public class Tuba extends Instrumento implements DarMantenimiento,SeAfina {
public Tuba (String nombre1) {
         super(nombre1);
         super.tocar();
      }

      @Override
      public void Mantenimiento() {
            System.out.println("Dando Mantenimiento al instrumento de viento
"+super.nombre);
      }
      public void afinar() {
            System.out.println("Se afino el instrumento de viento
"+super.nombre);
      }     }
```

VIOLIN

```
public class Violin extends Instrumento implements SeAfina, DarMantenimiento{

public Violin(String nombre1)
{
      super(nombre1);
    super.tocar();
}
```

```
public void afinar()
	{
	System.out.println("Afinando instrumento de cuerda "+super.nombre);
	}
public void Mantenimiento() {
	System.out.println("Dando Mantenimiento al instrumento de cuerda
"+super.nombre);
} }
```

FLAUTA

```
public class Flauta  extends Instrumento implements DarMantenimiento {

		public Flauta(String nombre)
		{
				super(nombre);
super.tocar();
		}
	public void Mantenimiento() {
	 System.out.println("Dando mantenimiento al instrumento de viento
"+super.nombre);
	}
	}
```

PIANO

```
public class Piano extends Instrumento implements SeAfina{
     public Piano(String nombre1)
    {
		super(nombre1);
```

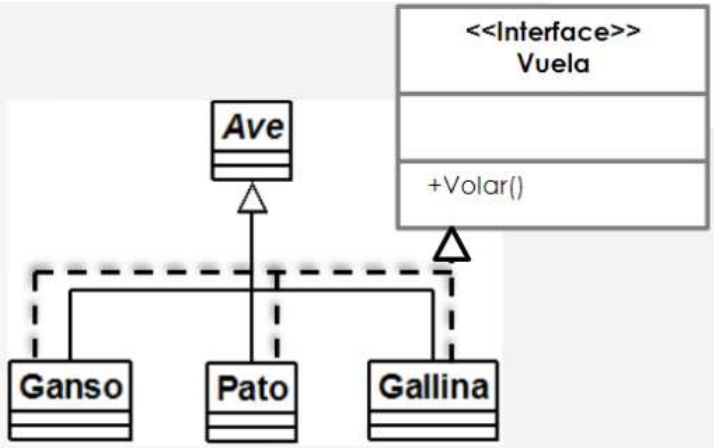
```
            super.tocar();
    }
    public void afinar()        {
System.out.println("Afinando
instrumento "+ super.nombre);
      }
} public class EjecutaOrqueta {
      public static void main(String a[])
      {
            Orquesta objeto1=new Orquesta("Bellas Artes");
            objeto1.Crear_Orquesta();
            objeto1.Guardar_Orquesta();
      }
        }
```

EJECUCIÓN DEL PROGRAMA

```
Tocando instrumento Piano de Cola
Afinando instrumento Piano de Cola
Tocando instrumento flautin
Tocando instrumento flautin
Tocando instrumento Violin electrico
Afinando instrumento de cuerda Violin electrico
Dando Mantenimiento al instrumento de cuerda Violin electrico
Tocando instrumento Trombon bajo
Dando Mantenimiento al instrumento de viento Trombon bajo
Se afino el instrumento de viento Trombon bajo
se ha guardado la orquesta Bellas Artes
```

HAZ AHORA:

Diseñar una clase de ave voladora de acuerdo al siguiente diagrama y la información:

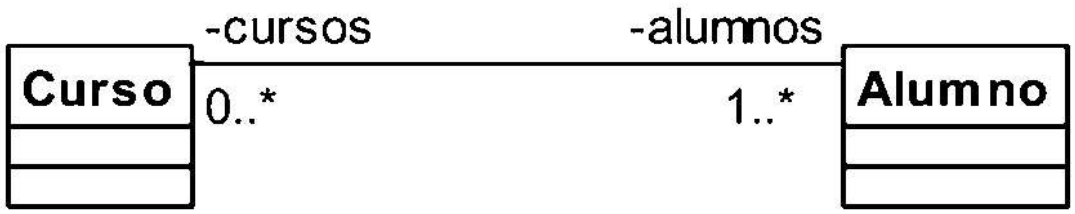


- Toda Ave tiene plumas, alas, pico y realiza un sonido
- Ganso:- graznean
- Pato: cua-cua
- Gallina: cocoroque

1.6 Relaciones

Asociación

- Una asociación representa la relación entre dos o más clases.
- Una asociación binaria es una relación entre dos clases.
- Existe una asociación si un objeto de una clase requiere un objeto de otra clase para hacer su trabajo
- En UML, una asociación binaria está representada por una línea sólida que conecta dos clases.



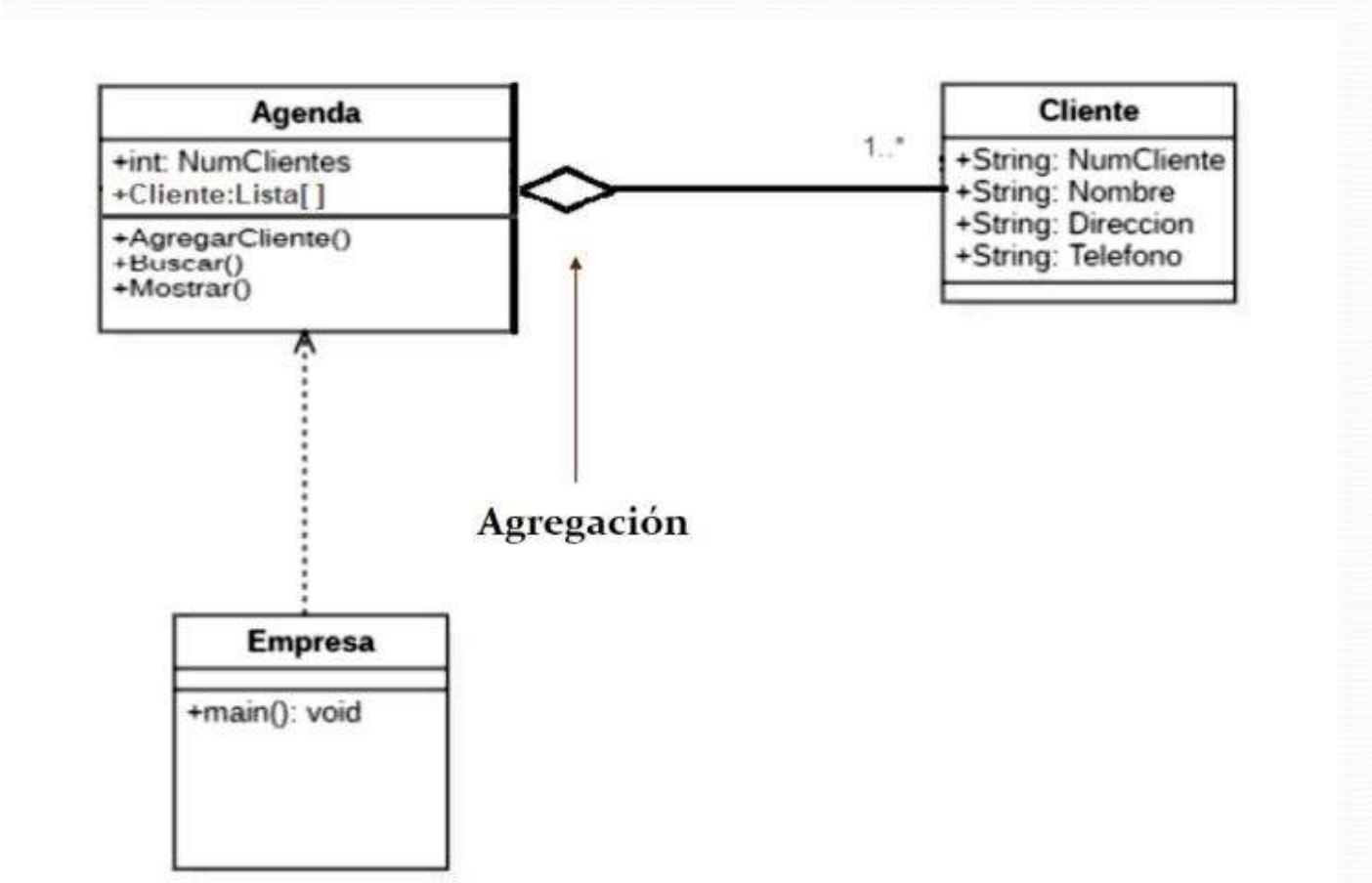
Agregación

La agregación es una forma especial de asociación y se conoce como relación débil. Una agregación es una asociación entre las clases A y B, donde cada instancia de la clase A contiene instancias
de la clase B

- En este sentido, una instancia de la clase B es parte de una instancia de la clase A.
- A la instancia de la clase A se le conoce como agregada (aggregate) y a la instancia de la clase B se le conoce como componente agregado, que será un atributo en la Clase A

Por ejemplo:

En una empresa se requiere un sistema para llevar la agenda de los clientes, de los cuales solo se requiere su número de cliente, nombre, dirección y teléfono. Además el sistema debe almacenar y mostrar la información de la agenda o buscar por el número de cliente.

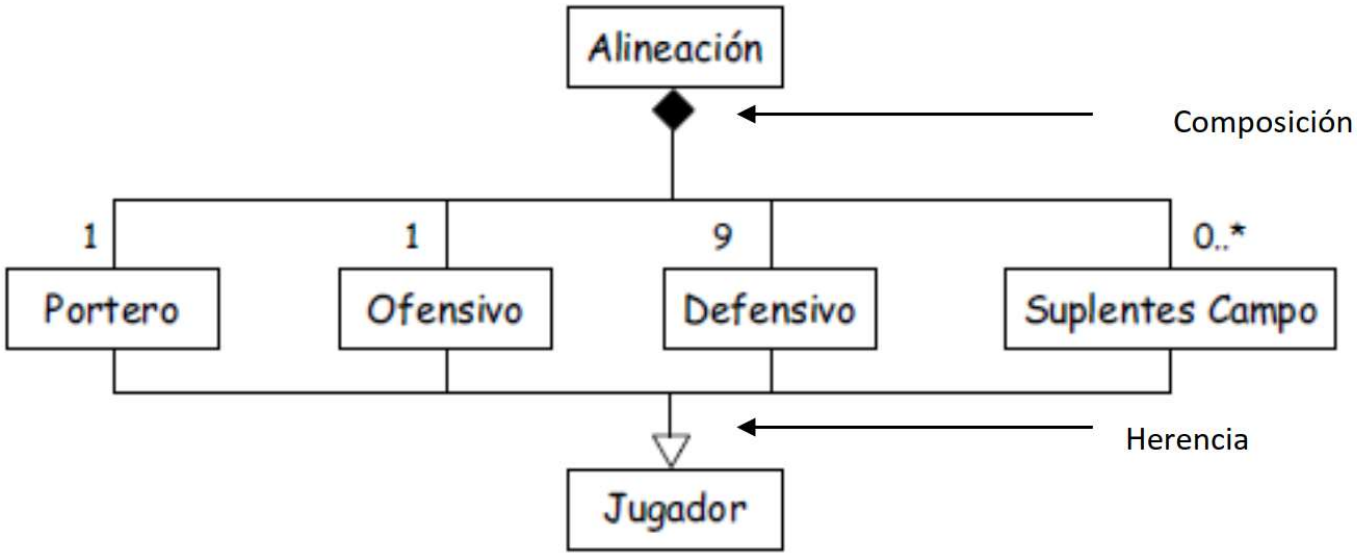


¿Quién es la clase que es componente agregado y quien la clase a quien le vamos agregar?

Composición

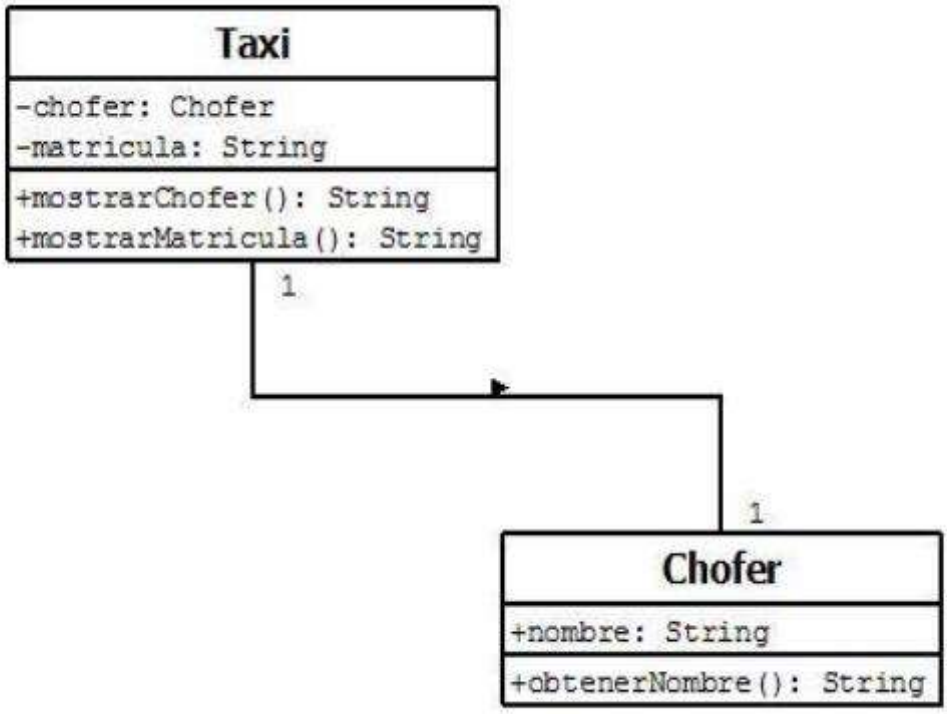
- Composición (relación fuerte), en la cual los objetos como atributos no tienen sentido fuera del objeto resultante.
- También se puede entender la composición como una relación en la que, los objetos, deben dejar de existir cuando lo hace el objeto compuesto.

En este ejemplo del Equipo de Futbol, la alineación se compone de 1 portero, 1 ofensivo o delantero, 9 defensivos y 0 hasta muchos suplentes en la banca. El siguiente diagrama también presenta una herencia de la Clase Jugador a los demás que son tipos de jugadores.



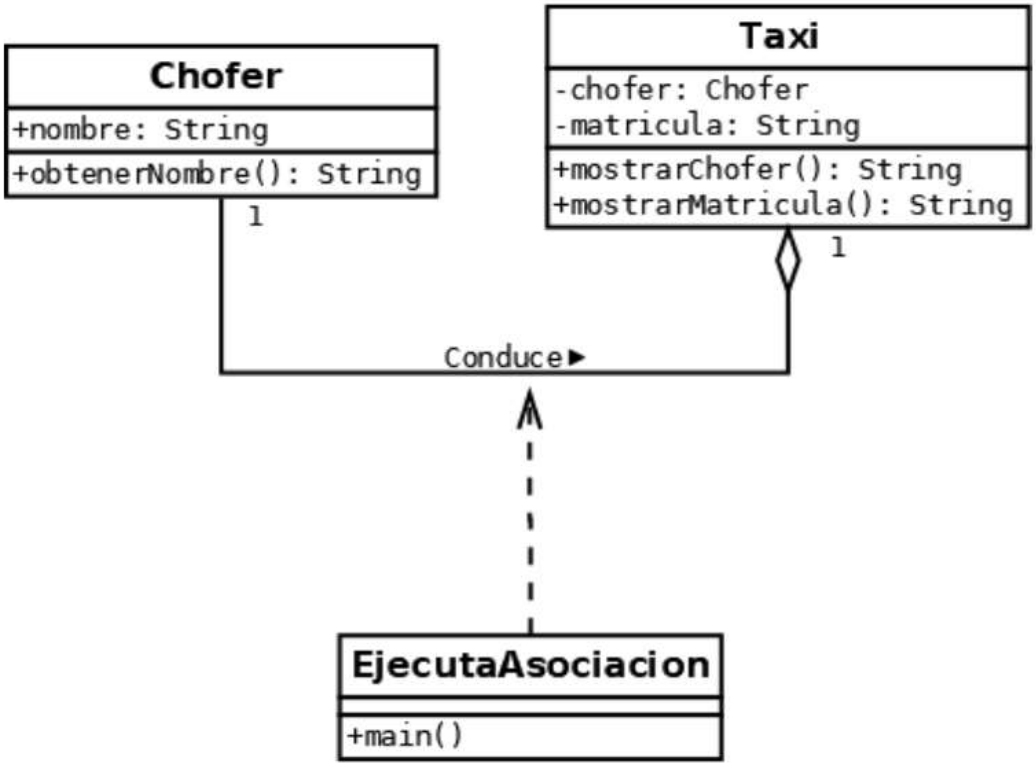
Ejemplificación de la Asociación

a) Realiza el programa de Asociación entre las dos clases Chofer y Taxi. ¿Un Taxi tiene un Chofer? Sí y uno. ¿Chofer tiene un Taxi? No, necesariamente. El atributo de Chofer es parte de la Clase Taxi



¿Cuál es clase requiere de un objeto de otra clase para hacer su trabajo?

b) Diseña el diagrama de UML y agrega la clase EjecutaAsociacion y la relación de dependecia.



//Código (Chofer)

```
public class Chofer {
       private String nombre;
       public Chofer(){ nombre="";}
     public Chofer(String nom){ nombre=nom;}
     public String obtenerNombre(){ return nombre;}
}
```

//Código (Taxi)

```
public class Taxi {
       private String matricula;
       private Chofer chofer;
       public Taxi(){matricula=""; chofer=null;}
       public Taxi(String mat,Chofer chfr){matricula=mat;
       chofer=chfr;}
       public void mostrarChofer(){
        System.out.println("Nombre Chofer:"+ chofer.obtenerNombre());
        }
        public void mostrarMatricula(){
        System.out.println("Matricula del Taxi:"+ matricula);
        }
}
```

//Código (EjecutaAsociacion)

```
public class EjecutaAsociacion {

       public static void main(String[] args) {
       Chofer chofer=new Chofer("Juan Armando Muros");
       Taxi TaxiYaris= new Taxi("WLU-94-69",chofer);
       TaxiYaris.mostrarMatricula();
    System.out.println(" lo maneja el chofer en curso...");
       TaxiYaris.mostrarChofer();
       }
}
}
```

Ejemplificación de la Agregación

a) Realiza el siguiente ejercicio con su diagrama UML de la Agenda y diseña tres agendas: Amigos, Familiares y Universidad.

b) Diseña el diagrama de UML incluye el EjecutaAgenda y la relación de dependecia.

c) Agrega cinco objetos a cada agenda y posteriormente, realiza un menú para visualizar solo la agenda que se quiera ver. 1. Amigos 2. Familiares 3. Universidad (datos de directores, estudiantes, etc.)

La relación de agregación se define como una conexión entre la clase agregada y una clase componente

Una asesora de seguros requiere tener una agenda de contactos de las personas para las que trabaja. Por lo cual se requiere rediseñar el siguiente diagrama de clases con datos de nombre, sexo, edad, teléfono y dirección. Podemos dar de alta, o registrar contactos, buscar datos, modificar los datos de los contactos y eliminar contactos.

1. Identificar las clases: sujetos o sustantivos. (color)

2. Identificar los atributos: adjetivos, características de los objetos. (color)

3. Identificar las operaciones: verbos, acciones u operaciones con los objetos. (color)

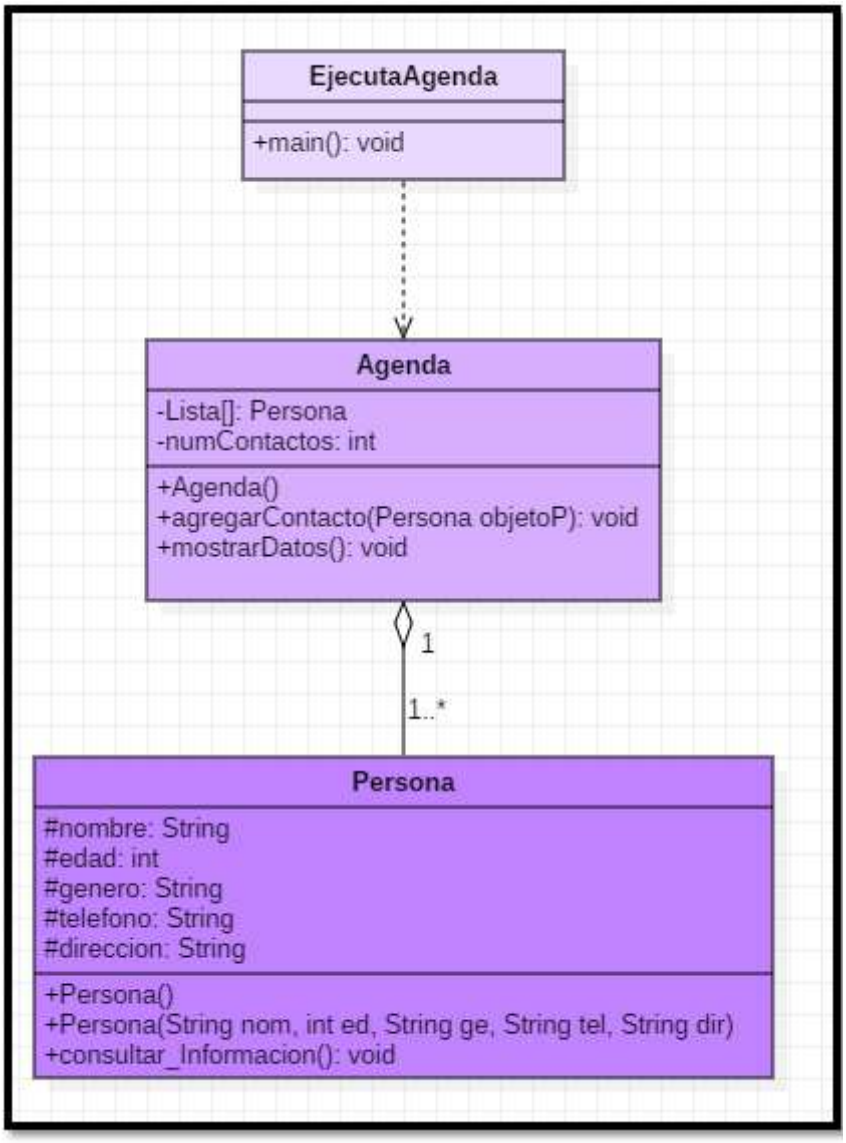
4. Identificar las relaciones: identificar la frase relacionada “tener” asociación. (color)

5. Identificar otras relaciones: “requiere”, “necesita” o “usa” dependencia.

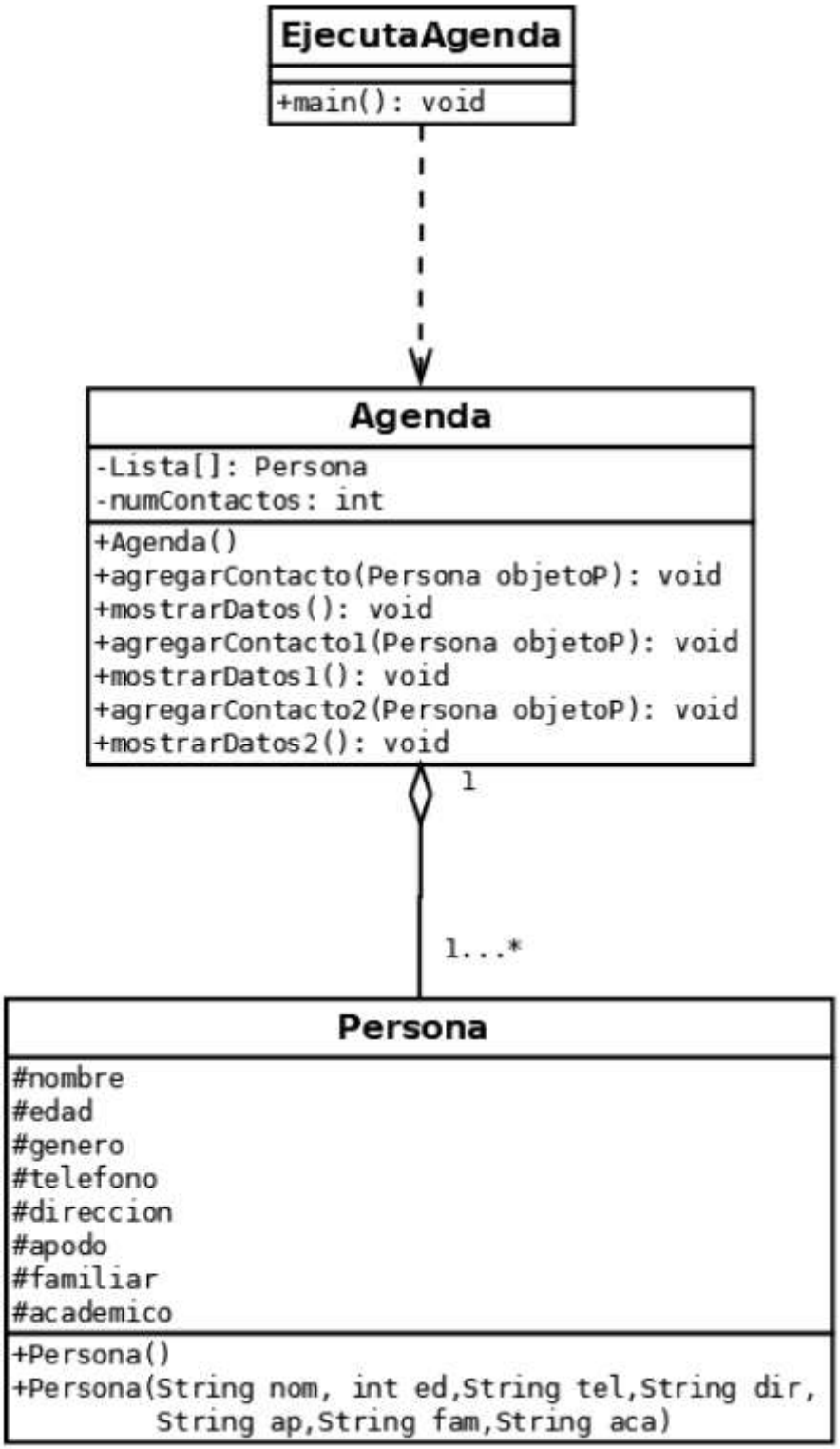
6. Realiza el análisis y diseño de las clases, los tipos de relación y su diagrama UML.

7. Programa un sistema que represente la relación de Agenda-Persona-Contacto.

- **¿Una agenda tiene personas?** Sí
- **¿Una persona tiene agenda?** No
- **¿Una asesora de seguros requiere, necesita o usa una agenda de contactos?** Sí
- **¿Una agenda de contactos requiere, necesita o usa una asesora?** No
- Si es asociación y se requiere conocer el subtipo debe preguntar el número de elementos
- Si es un número exacto de elementos = Composición
- Si no es un número exacto o indefinido = Agregación


EjecutaAgenda
+main(): void
Agenda
-Lista[]: Persona
-numContactos: int
+Agenda()
+agregarContacto(Persona objetoP): void
+mostrarDatos(): void
1
1..*
Persona
#nombre: String
#edad: int
#genero: String
#telefono: String
#direccion: String
+Persona()
+Persona(String nom, int ed, String ge, String tel, String dir)
+consultar_Informacion(): void

Solución:



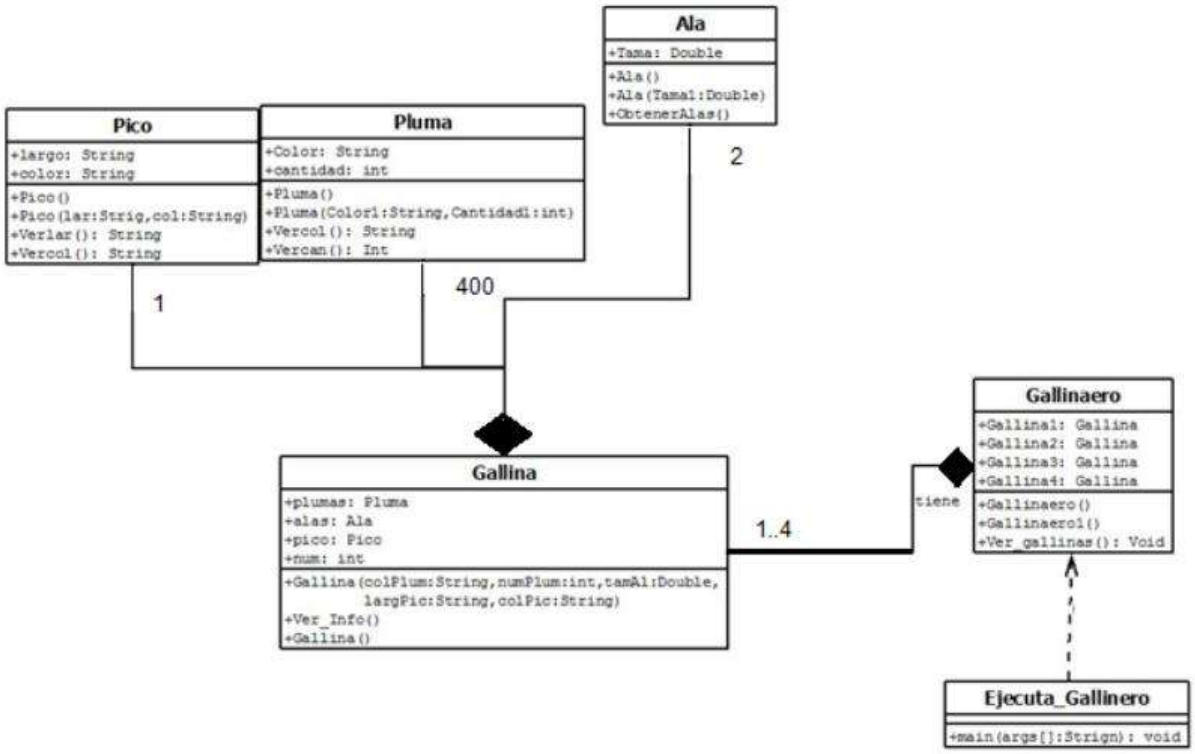
Código (zip - 18202 B) En caso de no poder abrir, se encuentra al final en los Ficheros adjuntos.

Ejemplificación de la Composición

La composición es un tipo de relación “fuerte “entre dos clases. Se identifica también con la frase “tiene”, “está compuesta de“. Generalmente se especifica el número de elementos que tiene o esta
compuesta.

Planteamiento

Una gallina está formada por un pico, plumas y dos alas. Todo Gallinero tiene un nombre y un tamaño de ancho de 2.5 por 5 de largo y puede almacenar gallinas.



Una gallina está formada por un pico, plumas y dos alas. Todo Gallinero tiene un nombre y un tamaño de ancho de 2.5 por 5 de largo y puede almacenar gallinas siendo cuatro gallinas.

1) Identificar las clases: sujetos o sustantivos. (color)

2) Identificar los atributos: adjetivos, características de los objetos. (color)

3) Identificar las operaciones: verbos, acciones u operaciones con los objetos. (color)

4) Identificar las relaciones: identificar la frase relacionada “tener” asociación. (color)

5) Identificar otras relaciones: “requiere”, “necesita” o “usa” dependencia.

6) Realiza el análisis y diseño de las clases, los tipos de relación y su diagrama UML.

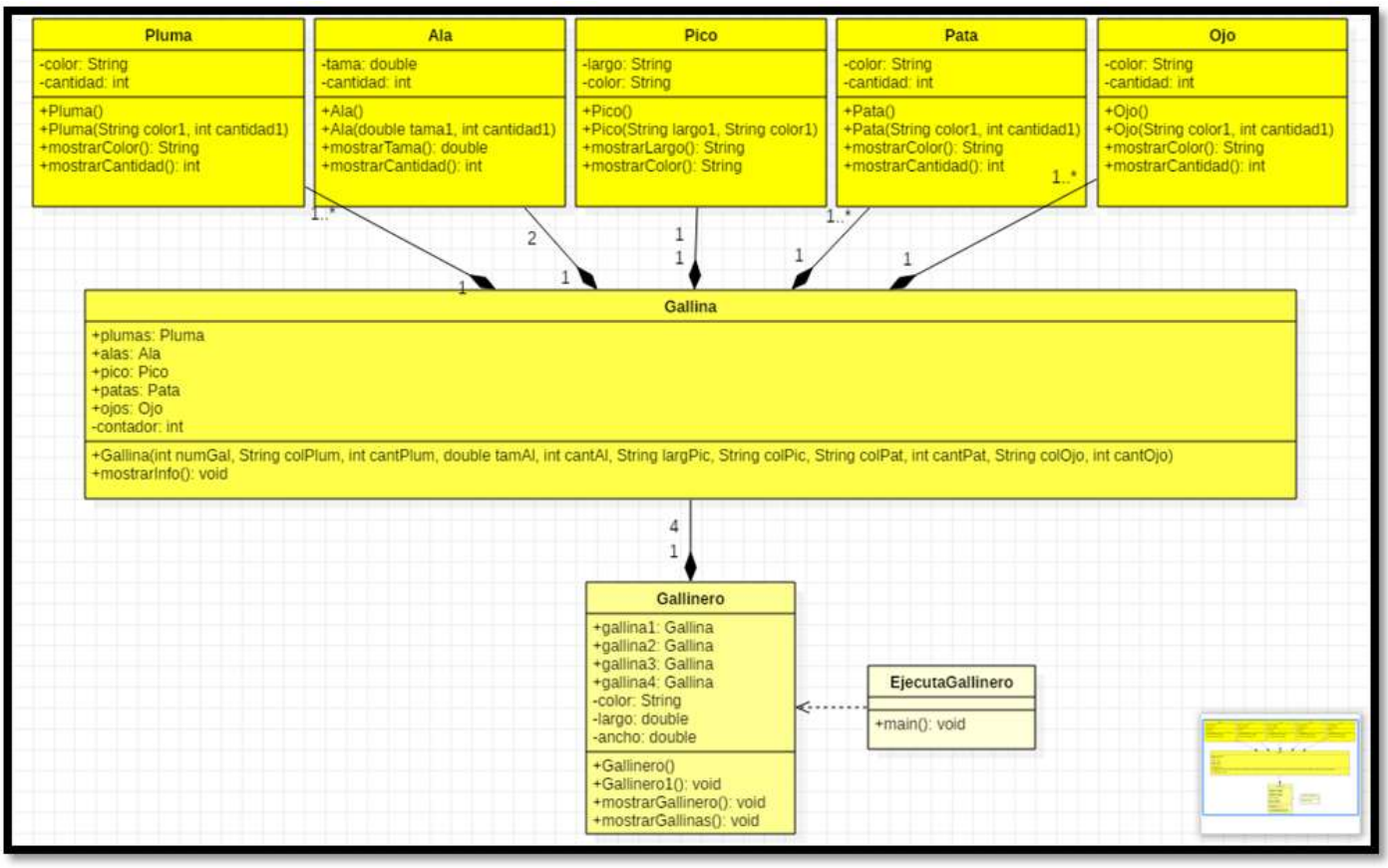
1. **¿Un gallinero tiene gallinas?** Sí, entonces hay asociación de **¿qué tipo Composición o Agregación?, ¿Cuántas tiene?** 4, entonces es composición
2. **¿Una gallina tienen gallineros?** No
3. **¿Una gallina tiene plumas?** Sí, entonces hay asociación de **¿qué tipo Composición o Agregación?, ¿Cuántas tiene?** 400, entonces es composición
4. **¿Una pluma tiene una gallina?** No
5. **¿Una gallina tiene alas?** Sí, entonces hay asociación de **¿qué tipo Composición o Agregación?, ¿Cuántas tiene?** 2, entonces es composición
6. **¿Un ala tiene una gallina?** No
7. **¿Una gallina tiene un pico?** Sí, Sí, entonces hay asociación de **¿qué tipo Composición o Agregación?, ¿Cuántas tiene?** 1, entonces es composición
8. **¿Un pico tiene una gallina?** No
9. **¿Una gallina tiene patas?** Sí, entonces hay asociación de **¿qué tipo Composición o Agregación?, ¿Cuántas tiene?** 2, entonces es composición
10. **¿Una pata tiene una gallina?** No
11. **¿Una gallina tiene ojos?** Sí, entonces hay asociación de **¿qué tipo Composición o Agregación?**, **¿Cuántas tiene?** 2, entonces es composición
12. **¿Un ojo tiene una gallina?** No

¿Cuáles son los tipos de relaciones entre las clases identificadas?

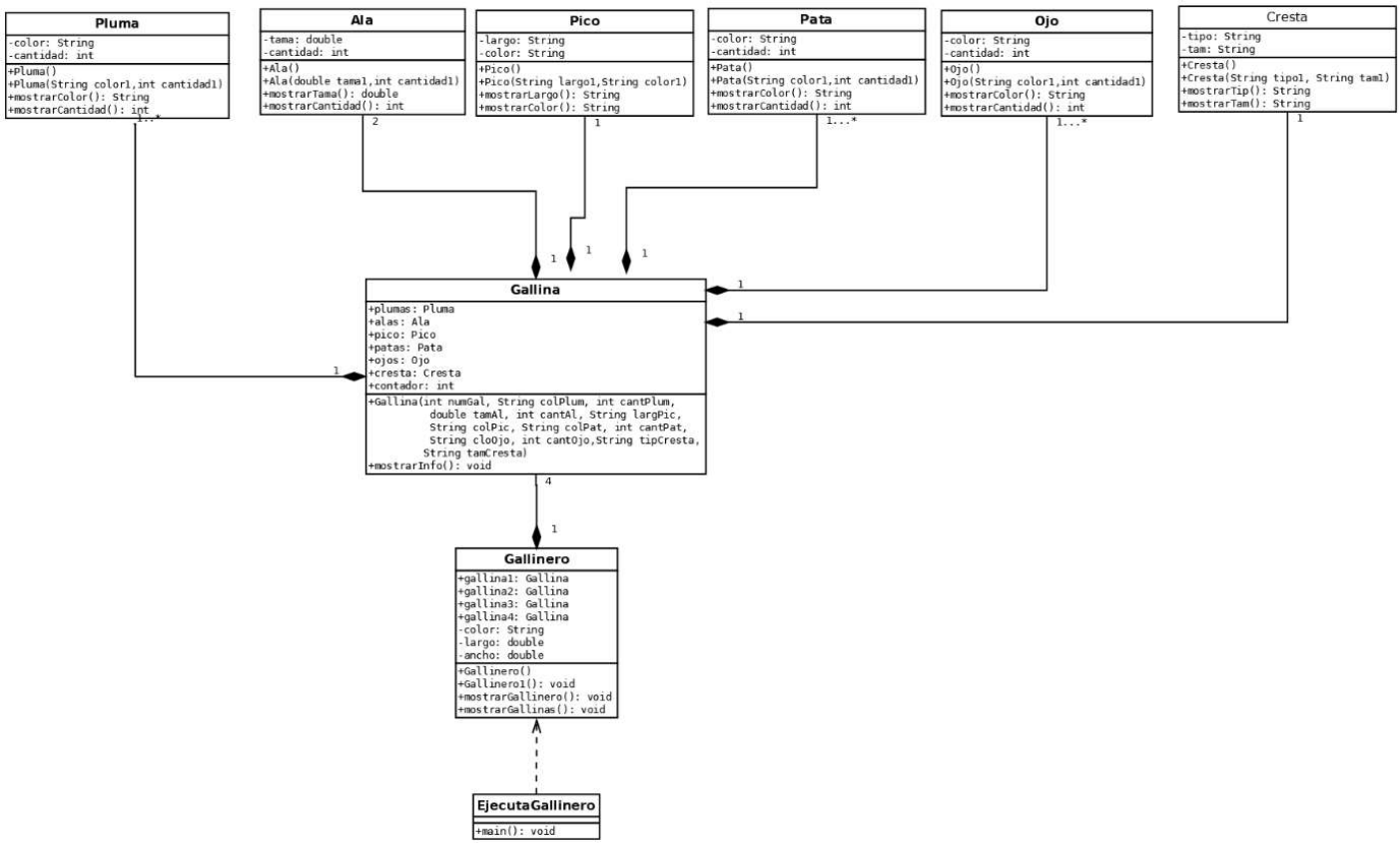
- Gallinero-Gallina: Composición porque tiene 4 Gallinas pero también puede ser de Agregación en caso que no se especifique numero de gallinas.
- Gallina-Pluma:Composición
- Gallina-Ala:Composición
- Gallina-Pico:Composición
- Gallina-Pata:Composición
- Gallina-ojo: Composición

Agrega otro componente a la gallina: cejas, oreja, cabello, etc.

Modifica el siguiente diagrama y agrega la clase del componente que seleccionaste:



Solución:



Código (zip - 20897 B) En caso de no poder abrir, se encuentra al final en los Ficheros adjuntos.

HAZ AHORA:

Realizar el ejercicio de Equipo de Futbol y su alineación, diseña con al menos dos atributos para las clases. Jugador tiene nombre y número de jugador.

Ficheros adjuntos

- Gallinas.zip (Ventana nueva)
- Agendas.zip (Ventana nueva)

Referencias

Umbaugh, J., Jacobson, I., & Booch, G. (2005). El Lenguaje Unificado de Modelado Manual de Referencia Segunda Edición. Madrid: Pearson

2. Ejercicios

打工仔. *programador PNG* (Todos los derechos reservados)

A continuación encontrarás una serie de ejercicios para entender mejor la Programación Orientada a Objetos.

Realizar la solución y su representación en UML y su implementación en Java

Problema guiado

Elabore el Diagrama de Clases y sus relaciones del siguiente problema

Un centro de instalaciones deportivas quiere hacer una aplicación de reservas. En el centro existen instalaciones deportivas (piscinas, frontones, gimnasios y pistas de tesis). El centro en cuestión tiene socios, de los cuales se almacenan su nombre, dirección, ciudad, provincia, teléfono y cuota. Además, existen una serie de artículos que se pueden reservar si el socio lo requiere (balones, redes y raquetas).
Cada instalación es reservada por un socio en una fecha dada desde una hora de inicio hasta una hora de fin. Cada reserva puede tener asociada uno o varios artículos deportivos que se alquilan aparte.
Por ejemplo, si yo quiero hacer una reserva para jugar al tenis, tengo que reservar una instalación polideportiva y si lo necesito, las raquetas.

Para resolver este problema utilizaremos la siguiente metodología:

1. Identificar clases
2. Identificar atributos
3. Identificar métodos
4. Identificar relaciones
5. Dibujar el diagrama

1. Clases

Un centro de instalaciones deportivas quiere hacer una aplicación de reservas. En el centro existen instalaciones deportivas (piscinas, frontones, gimnasios y pistas de tesis).

El centro en cuestión tiene
socios, de los cuales se almacenan su nombre, dirección, ciudad, provincia, teléfono y cuota. Además, existen una serie de artículos que se pueden reservar si el socio lo requiere (balones, redes y raquetas). Cada instalación es reservada por un socio en una fecha dada desde una hora de inicio hasta una hora de fin. Cada reserva puede tener asociada uno o varios artículos deportivos que se alquilan aparte. Por ejemplo, si yo quiero hacer una reserva para jugar al tenis, tengo que reservar una
instalación polideportiva y si lo necesito, las raquetas.

Clases: centro de instalaciones deportivas, instalaciones deportivas (subclases: piscinas, frontones, gimnasios y pistas de tesis), socios, artículos (subclases: balones, redes y raquetas), reservación.

2. Atributos

Un centro de instalaciones deportivas quiere hacer una aplicación de reservas. En el centro existen instalaciones deportivas (piscinas, frontones, gimnasios y pistas de tesis).
El centro en cuestión tiene
socios, de los cuales se almacenan su nombre, dirección, ciudad, provincia, teléfono y cuota. Además, existen una serie de artículos que se pueden reservar si el socio lo requiere (balones, redes y raquetas). Cada instalación es reservada por un socio en una fecha dada desde una hora de inicio hasta una hora de fin. Cada reserva puede tener asociada uno o varios artículos deportivos que se alquilan aparte. Por ejemplo, si yo quiero hacer una reserva para jugar al tenis, tengo que reservar una
instalación polideportiva y si lo necesito, las raquetas.

Atributos: socio - nombre, dirección, ciudad, provincia, teléfono y cuota, reservación - fecha, hora de inicio y de fin. Las intalaciones deportivas y los artículos también tiene un número que delimita cuántos se encuentran disponibles.

3. Métodos

Un centro de instalaciones deportivas quiere hacer una aplicación de reservas. En el centro existen instalaciones deportivas (piscinas, frontones, gimnasios y pistas de tesis). El centro en cuestión tiene
socios, de los cuales se almacenan su nombre, dirección, ciudad, provincia, teléfono y cuota. Además, existen una serie de artículos que se pueden reservar si el socio lo requiere (balones, redes y raquetas). Cada instalación es reservada por un socio en una fecha dada desde una hora de inicio hasta una hora de fin. Cada reserva puede tener asociada uno o varios artículos deportivos que se alquilan aparte. Por ejemplo, si yo quiero hacer una reserva para jugar al tenis, tengo que reservar una
instalación polideportiva y si lo necesito, las raquetas.

Métodos: reservación - asociar un artículo y una instalación.

4. Relaciones

La instalación deportiva tiene subclases las cuales son: piscinas, frontones, gimnasios y pistas de tesis.

Los artículos tienen subclases: balones, redes y raquetas.

Cada vez que estos se reservan se debes restar al número de objetos disponibles.

5. Diagrama

¡Es tu turno! Diseña el Diagrama con la información previa.

Problema 1

Proponga el diagrama de clases para una aplicación de software que gestione las escenas filmadas para la realización de una película.

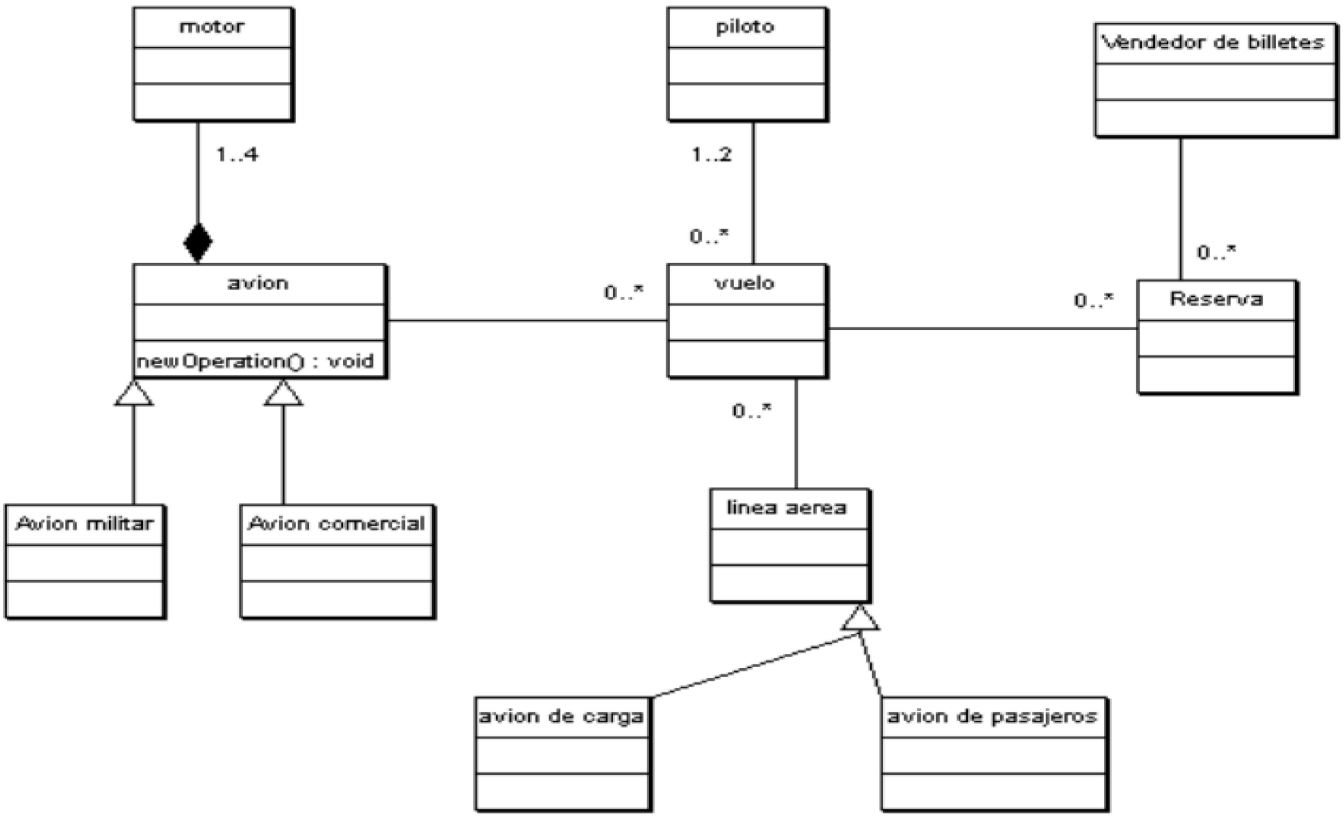
Cada escena se identifica por un código y una descripción. Cada escena se filma desde diferentes posiciones (al menos una), cada una de éstas se denomina escenario. Cada escenario se caracteriza por un código y una descripción, donde los parámetros fotográficos son capturados (por ejemplo: apertura, exposición, longitud focal, filtros, etc.).

Considerar que un escenario está relacionado a una única escena.
Para cada escenario, varias tomas pueden ser filmadas (al menos una). Cada toma se caracteriza por un número de toma, un número real que representa los metros de película que se han utilizado para
el rodaje de la toma, y el código de la bobina donde se almacena la película. Considere que una toma está asociada a un sólo escenario.

Las escenas se dividen en internas aquellas que se filman en un teatro, y externas las que se filman en una locación y pueden ser una "escena de día" o una "escena nocturna". Las locaciones se caracterizan por un código, dirección y una descripción.

Problema 2

Identifique los tipos de relaciones entre clases y describa el siguiente Diagrama de Clases



Problema 3

Diseñar el diagrama de clases sobre un “Grupo de cafeterías”.

De cada cafetería, se considerará el nombre, licencia fiscal, domicilio, fecha de apertura, horario. Cada cafetería tendrá uno o varios titulares, de los que se almacena sus datos personales: nombre,
RFC, teléfono y domicilio.

Respecto a los empleados de cada cafetería, considerar que un empleado puede trabajar en varios sitios. Para cada empleado, se tendrá sus datos personales: número de empleado, nombre, RFC y
domicilio; también se anotará la función que desempeña, que puede ser distinta en cada cafetería.

Para cada negocio, se llevará un inventario de existencias, con el nombre del artículo, su código, la cantidad y el precio de coste. A efectos de contabilidad, se llevará una relación de los pedidos, con un número de pedido, fecha, proveedor, código y nombre de los artículos suministrados y cantidad.

Problema 4

El departamento de Informática de un hospital está realizando un nuevo registro de datos de todas las personas que tienen relación con esa institución, que son: los empleados, los médicos y los pacientes, realizar el diagrama de clases correspondiente.

Los empleados son categorizados en función de si son contratados por Planilla o de forma Eventual y son los encargados de los procesos administrativos.

Los médicos (que podrían considerarse un tipo especial de empleados contratados por Planilla) se encargan de las atenciones de las consultas médicas.

Para solicitar una cita, el paciente es atendido por un empleado. El paciente indica el servicio en el cual quiere pasar consulta y el empleado le indica el nombre del médico, la fecha y la hora de la cita.

Los atributos de cada uno de ellos se indican a continuación:

- Persona: número de DNI, nombre, apellido, fecha de nacimiento, dirección, ciudad de procedencia.
- Paciente: número de historia clínica, sexo, grupo sanguíneo, lista de medicamentos a los que es
  alérgico.
- Empleado: código de Empleado, número de horas extras, fecha de ingreso, área, cargo.
- Empleado por Planilla: salario mensual, porcentaje adicional por hora extra.
- Empleado Eventual: honorarios por hora, número de horas totales (normales + extras) trabajadas, fecha de término del contrato.

- Médico: especialidad (cirujano, oftalmólogo, etc.), servicio (cirugía, oftalmología, etc.), número de
  consultorio.

Las operaciones que involucren algún cálculo, deben desarrollarse a través de interfaces. La aplicación a desarrollar debe permitir:

- Registrar los datos de los empleados, los pacientes y los médicos.
- Registrar los datos de una cita médica.
- Listar los datos de los médicos ordenados en forma descendente por la especialidad.
- Listar los datos (nombres y apellidos) de los pacientes atendidos por un médico determinado (ingresando su código).

Evaluación

Actividad

El estudiante tiene que elaborar algún producto que demuestre la competencia adquirida, esta actividad puede ser a través de alguna herramienta digital como Puzzles, Juegos interactivos, etc.

Condiciones: Debe estar ligada a una actividad dentro del LMS de cada profesor para que pueda ponderarse con un porcentaje dentro de los criterios de evaluación de la unidad.

Selecciona

Selecciona las respuestas correctas y pulsa sobre el botón "responder"

10 0 0 0 00:00

Pulse aquí para jugar