# Algoritmos y Estructura de Datos

## CLASES EN JAVA

DOCENTE: MARX DANLY LEÓN TRUJILLO
FACULTAD DE ING. INDUSTRIAL Y SISTEMAS
E.P. INGENIERÍA DE SISTEMAS

# CLASES

En programación orientada a objetos (POO), una clase es una plantilla o modelo que define un conjunto de atributos y métodos que serán compartidos por todos los objetos creados a partir de ella. Una clase puede ser considerada como un molde o plano a partir del cual se pueden crear múltiples instancias o ejemplares que comparten las mismas características y comportamientos.

Una clase se compone de dos partes principales: atributos y métodos. Los atributos son variables que definen las características o propiedades de un objeto, mientras que los métodos son las acciones que pueden ser realizadas por el objeto. Ambos componentes están estrechamente relacionados, ya que los métodos suelen actuar sobre los atributos de la clase.

# CLASES

```java
public class MiClase {
    // Atributos
    int numero;
    String texto;

    // Método constructor
    public MiClase(int numero, String texto) {
        this.numero = numero;
        this.texto = texto;
    }

    // Método
    public void imprimir() {
        System.out.println("Número: " + numero);
        System.out.println("Texto: " + texto);
    }
```

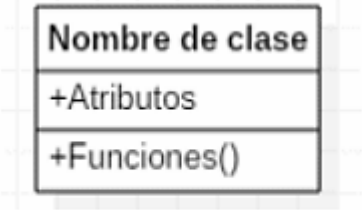
```java
    public static void main(String[] args) {
        // Crear objetos de la clase
        MiClase objeto1 = new MiClase(42, "Hola, mundo!");
        MiClase objeto2 = new MiClase(123, "Java es genial!");

        // Llamar a un método de objeto
        objeto1.imprimir();
        objeto2.imprimir();
    }
}
```

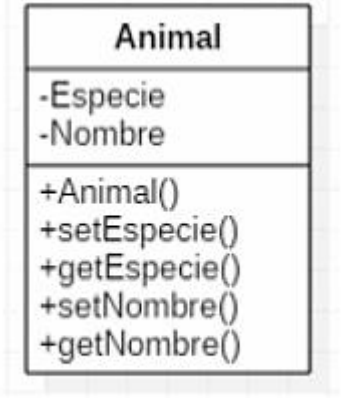
# CLASES

Una **CLASE** es una plantilla que define las propiedades (atributos) y comportamientos (métodos) de un objeto.

Un **OBJETO** es una instancia de una clase. Puedes crear múltiples objetos a partir de la misma clase.




Notación de una clase




Ejemplo de una clase

Java

# CLASES

Tanto los atributos como las funciones incluyen al principio de su descripción la visibilidad que tendrá. Esta visibilidad se identifica escribiendo un símbolo y podrá ser:

**(+) Pública.** Representa que se puede acceder al atributo o función desde cualquier lugar de la aplicación.

**(-) Privada.** Representa que se puede acceder al atributo o función únicamente desde la misma clase.

**(#) Protegida.** Representa que el atributo o función puede ser accedida únicamente desde la misma clase o desde las clases que hereden de ella (clases derivadas).

# CREACIÓN DE OBJETOS

En Java, la creación de un objeto implica la instanciación de una clase. Para instanciar un objeto, se utiliza la palabra clave new, seguida del nombre de la clase y los paréntesis vacíos. Siguiendo el ejemplo anterior, tenemos una clase Persona y podemos crear un objeto de la siguiente manera:

```
Persona persona = new Persona("Juan", 30);
```

En este ejemplo, estamos creando un objeto de tipo Persona y almacenándolo en la variable persona con el nombre «Juan» y la edad 30. Para acceder a los métodos de la clase Persona, podemos hacer lo siguiente:

```
persona.saludar();
```

# CONSTRUCTORES

En Java, un constructor es un método especial de una clase que se utiliza para inicializar los objetos de esa clase. El constructor se ejecuta automáticamente cuando se crea un objeto de esa clase y se utiliza para establecer valores iniciales para los atributos del objeto.

El constructor tiene el mismo nombre que la clase y no tiene tipo de retorno. Puede tener uno o más parámetros y puede ser sobrecargado. Si no se define ningún constructor en la clase, Java proporciona automáticamente un constructor predeterminado sin parámetros.

Aquí hay un ejemplo de un constructor de la clase

# CONSTRUCTORES

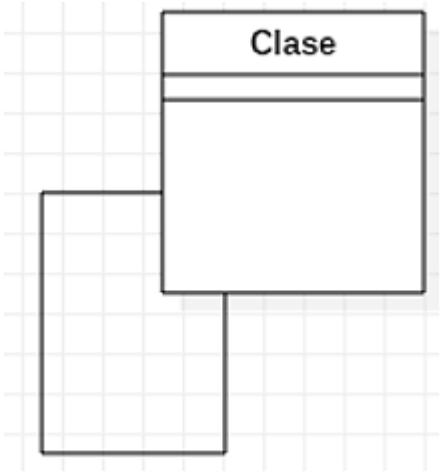
```java
public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // otros métodos de la clase Persona
}
```

# RELACIONES

Una relación identifica una dependencia. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina dependencia reflexiva. Las relaciones se representan con una linea que une las clases, esta línea variará dependiendo del tipo de relación




Relación reflexiva

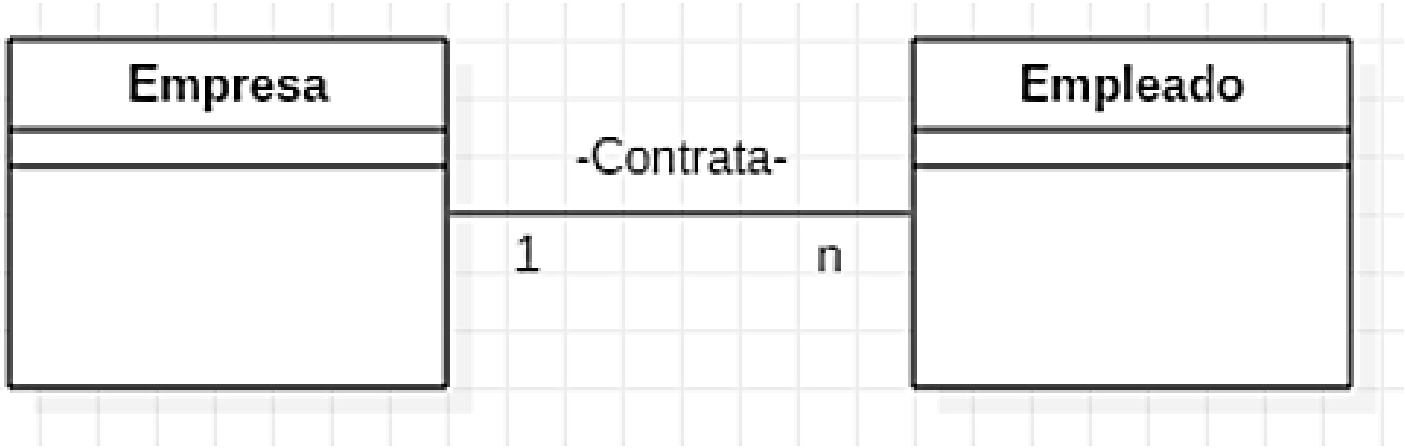
# RELACIONES

Las relaciones en el diagrama de clases tienen varias propiedades, que dependiendo la profundidad que se quiera dar al diagrama se representarán o no. Estas propiedades son las siguientes:

**Multiplicidad.** Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango… Se utiliza n o * para identificar un número cualquiera.

**Nombre de la asociación.** En ocasiones se escriba una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como por ejemplo: «Una empresa contrata a n empleados»

# RELACIONES




Ejemplo de relación Empresa-Empleado

# RELACIONES

## Tipos de relaciones

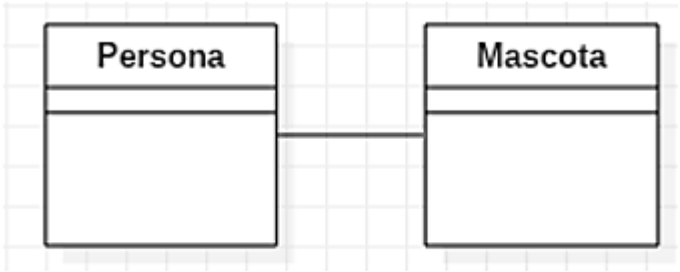
Un diagrama de clases incluye los siguientes tipos de relaciones:

- **Asociación.**
- **Agregación.**
- **Composición.**
- **Dependencia.**
- **Herencia.**

# ASOCIACIÓN

Este tipo de relación es el más común y se utiliza para representar dependencia semántica. Se representa con una simple linea continua que une las clases que están incluidas en la asociación.

Un ejemplo de asociación podría ser: «Una mascota pertenece a una persona».




Ejemplo de asociación

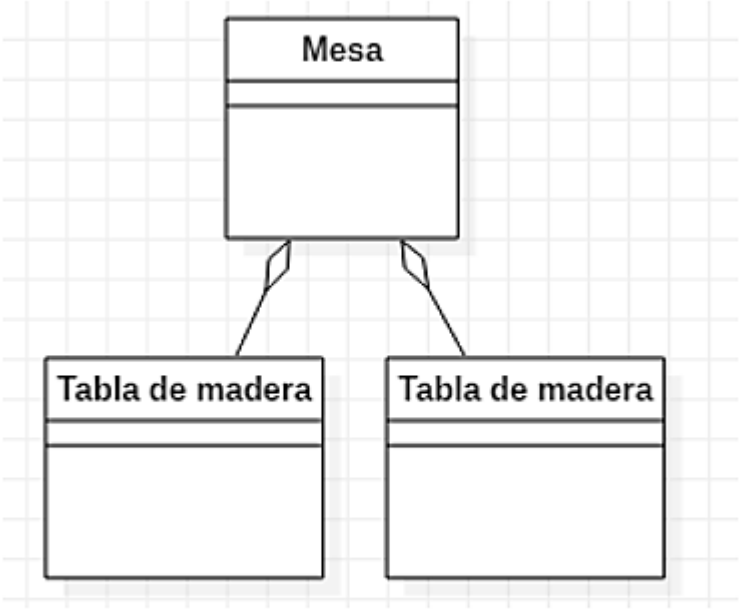
# AGREGACIÓN

Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que un objeto es parte de otro, pero aun así debe tener existencia en sí mismo.

Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

Un ejemplo de esta relación podría ser: «Las mesas están formadas por tablas de madera y tornillos o, dicho de otra manera, los tornillos y las tablas forman parte de una mesa». Como ves, el tornillo podría formar parte de más objetos, por lo que interesa especialmente su abstracción en otra clase.

# AGREGACIÓN



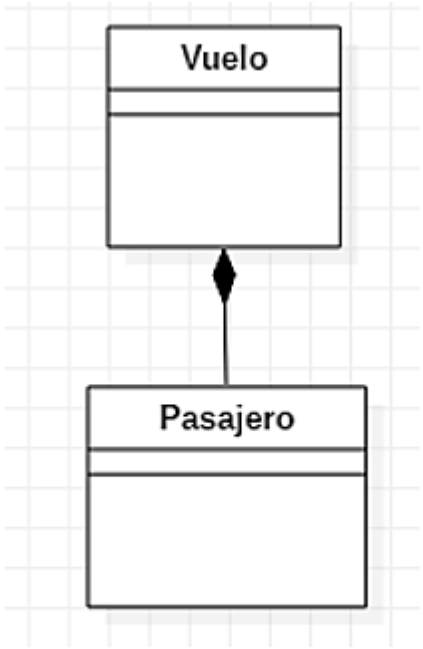

Ejemplo de agregación

## COMPOSICIÓN

La composición es similar a la agregación, representa una relación jerárquica entre un objeto y las partes que lo componen, pero de una forma más fuerte. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el primero no existe. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos sino que dependen del elemento que componen. Además, suelen tener los mismos tiempo de vida. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.

Se representa con una linea continua con un rombo relleno en la clase que es compuesta.

# COMPOSICIÓN

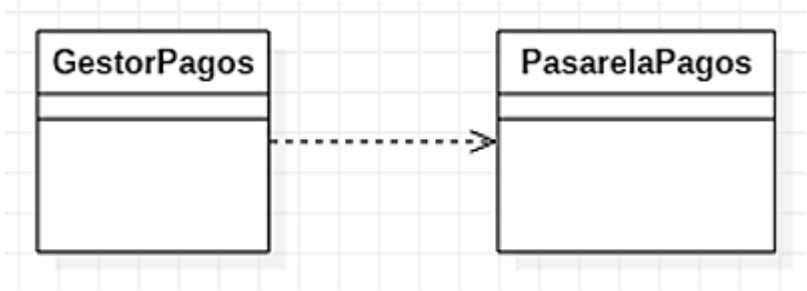



Ejemplo de composición

## DEPENDENCIA

Se utiliza este tipo de relación para representar que una clase requiere de otra para ofrecer sus funcionalidades. Es muy sencilla y se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.

Un ejemplo de esta relación podría ser la siguiente:

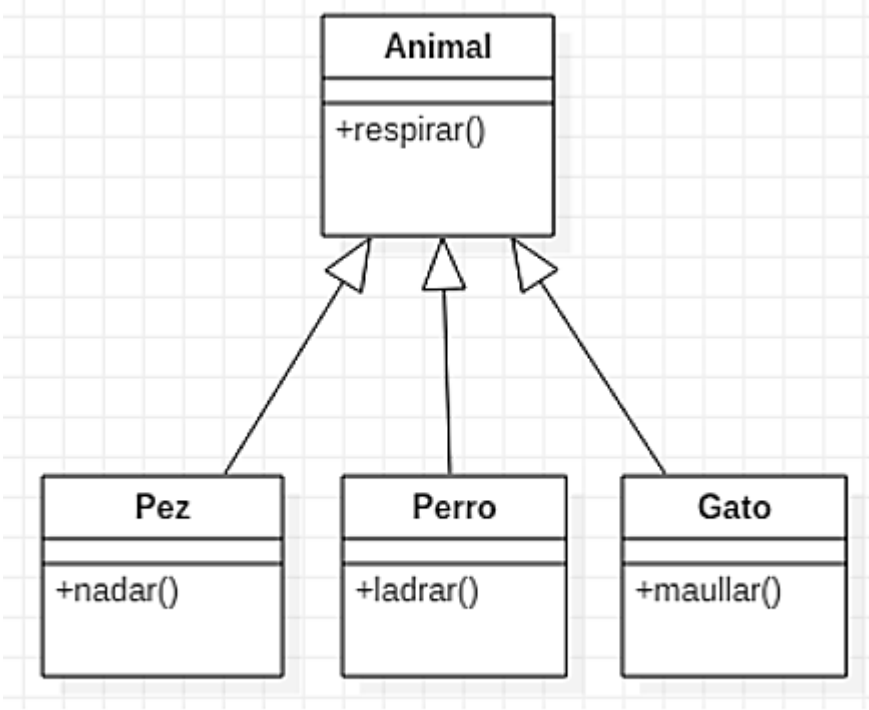



Ejemplo de dependencia

# HERENCIA

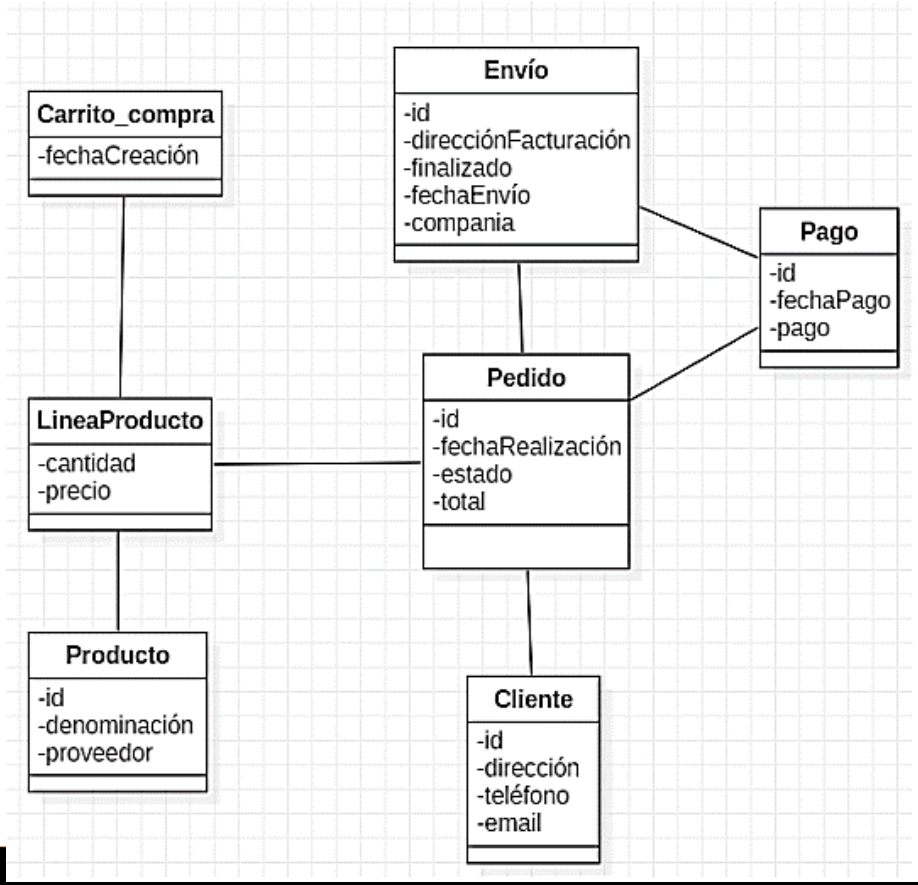
Otra relación muy común en el diagrama de clases es la herencia. Este tipo de relaciones permiten que una clase (clase hija o subclase) reciba los atributos y métodos de otra clase (clase padre o superclase). Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma. Se utiliza en relaciones «es un».

Un ejemplo de esta relación podría ser la siguiente: Un pez, un perro y un gato son animales.

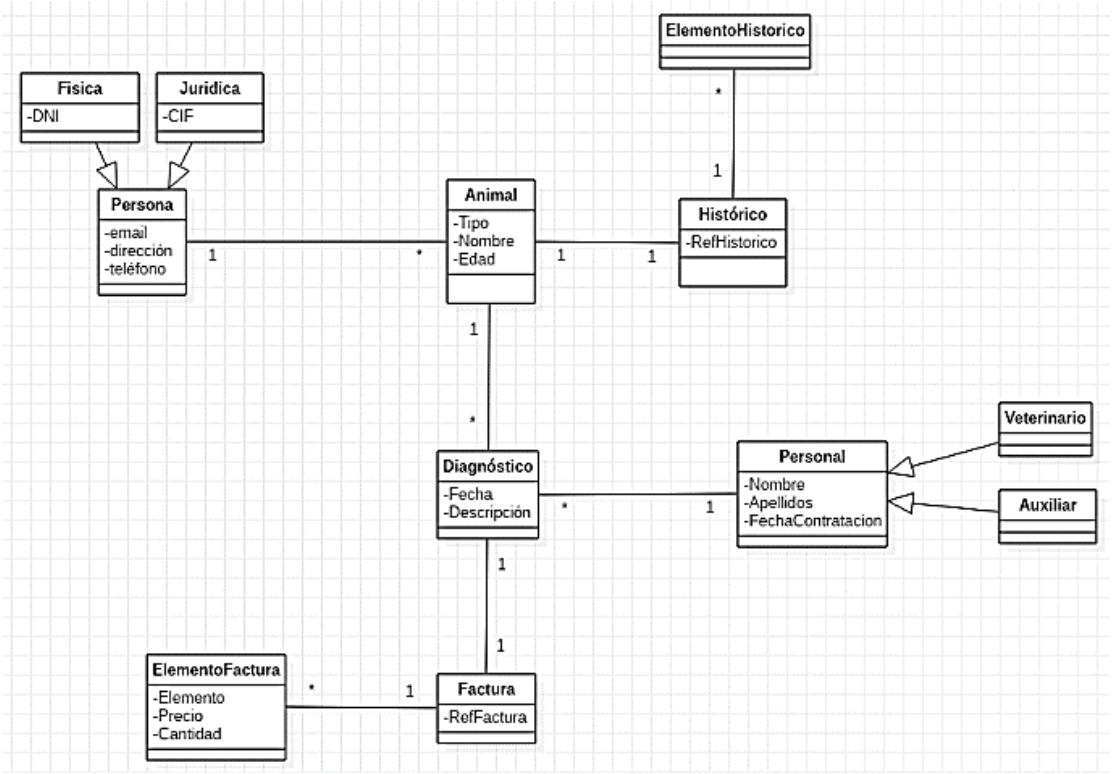



Ejemplo de herencia

# HERENCIA


Carrito_compra
-fechaCreación
Envío
-id
-direcciónFacturación
-finalizado
-fechaEnvío
-compania
Pago
-id
-fechaPago
-pago
Pedido
-id
-fechaRealización
-estado
-total
LineaProducto
-cantidad
-precio
Producto
-id
-denominación
-proveedor
Cliente
-id
-dirección
-teléfono
-email

# HERENCIA


ElementoHistorico
Fisica
-DNI
Juridica
-CIF
Persona
-email
-dirección
-teléfono
Animal
-Tipo
-Nombre
-Edad
Histórico
-RefHistorico
Veterinario
Personal
-Nombre
-Apellidos
-FechaContratacion
Auxiliar
Diagnóstico
-Fecha
-Descripción
ElementoFactura
-Elemento
-Precio
-Cantidad
Factura
-RefFactura
1
*

# EJERCICIOS

```
public class Jugador {

    //ATRIBUTOS
    String tamaño;
    int numero;
    String color;
    double velocidad;
    String poder;

    //METODO CONSTRUCTOR
    public Jugador (String tamaño, int numero, String color, double velocidad, String poder)
    {
        this.color=color;
        this.numero=numero;
        this.poder=poder;
        this.velocidad=velocidad;
        this.tamaño=tamaño;
    }
```

# EJERCICIOS

```
//METODOS        GET-SET-MOSTRAR
public String getColor()
{
    return color;
}

public void setColor (String color)
{
    this.color=color;
}

public int getNumero()
{
    return numero;
}

public void setNumero (int numero)
{
    this.numero=numero;
}
```

```
public String getPoder()
{
    return poder;
}

public void setPoder (String poder)
{
    this.poder=poder;
}

public double getVelocidad()
{
    return velocidad;
}

public void setVelocidad (double velocidad)
{
    this.velocidad=velocidad;
}
```

# EJERCICIOS

```
public String getTamaño()
{
    return tamaño;
}

public void setTamaño (String tamaño)
{
    this.tamaño=tamaño;
}

public void mostrarJugador(){
    System.out.println("Jugador "+getNumero() );
}
public static void main(String[] args) {
    // TODO code application logic here
    Jugador gamer1 = new Jugador("pequeño",1,"blue",3.7,"agilidad");
    Jugador gamer2 = new Jugador("pequeño",2,"red",1.5,"fuerza");
    Jugador gamer3 = new Jugador("mediano",3,"orange",2.0,"fuerza");
    Jugador gamer4 = new Jugador("mediano",4,"brown",4.1,"inteligencia");
    Jugador gamer5 = new Jugador("grande",5,"gray",2.7,"fuerza");

    gamer2.mostrarJugador();
}
```