



ESCUELA DE
INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



Día, Fecha:

Sábado, 11/03/2023

Hora de inicio:


19:00

Introducción a la programación y computación 1 [D]

José Eduardo Morales García

Miembros estáticos (static) y miembros de instancia

- ▶ En Java, los miembros de una clase pueden ser estáticos o de instancia. Los miembros estáticos se asocian con la clase en sí, mientras que los miembros de instancia se asocian con las instancias (objetos) de la clase.
- ▶ Los miembros estáticos se definen utilizando la palabra clave "static" y pueden ser variables o métodos. Estos miembros se pueden acceder directamente a través del nombre de la clase sin necesidad de crear una instancia de la clase.



Por otro lado, los miembros de instancia se definen sin la palabra clave "static" y solo se pueden acceder a través de una instancia de la clase. Cada instancia tiene su propia copia de los miembros de instancia.

Es importante tener en cuenta que los miembros estáticos no pueden acceder a los miembros de instancia y viceversa. Además, los miembros estáticos no se pueden sobrescribir, mientras que los miembros de instancia se pueden sobrescribir en las subclases.

En general, los miembros estáticos se utilizan para variables y métodos que se aplican a la clase en su conjunto, mientras que los miembros de instancia se utilizan para variables y métodos que son específicos de cada instancia de la clase.

```
public class EjemploMiembros {

    // Miembro estático
    public static int contadorEstatico = 0;

    // Miembro de instancia
    public int contadorDeInstancia = 0;

    // Constructor
    public EjemploMiembros() {
        contadorEstatico++; // incrementa el contador estático
        contadorDeInstancia++; // incrementa el contador de instancia
    }

    // Método estático
    public static void imprimirContadorEstatico() {
        System.out.println("Contador estático: " + contadorEstatico);
    }

    // Método de instancia
    public void imprimirContadorDeInstancia() {
        System.out.println("Contador de instancia: " + contadorDeInstancia);
    }
}
```

los miembros estáticos son compartidos por todas las instancias de la clase, mientras que los miembros de instancia pertenecen a cada instancia individual. Los miembros estáticos se pueden acceder a través del nombre de la clase, mientras que los miembros de instancia solo se pueden acceder a través de una instancia de la clase.

Referencia "this" en Java

En Java, "this" es una referencia variable que se utiliza para hacer referencia al objeto actual de la clase. Se puede utilizar en cualquier lugar dentro de la clase y se refiere al objeto en el que se está trabajando actualmente.


```
public class Persona {  
    String nombre;  
  
    public Persona(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

► También se puede utilizar "this" para llamar a un constructor de la misma clase dentro de otro constructor de la misma clase.

```
public class Persona {  
    String nombre;  
    int edad;  
  
    public Persona(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public Persona(String nombre, int edad) {  
        this(nombre); // llamada al constructor Persona(String nombre)  
        this.edad = edad;  
    }  
}
```


Clases paramétricas (plantilla de clases)

- Las clases paramétricas, también conocidas como plantillas de clases, son una característica de programación orientada a objetos que permite definir una clase genérica que puede ser utilizada con diferentes tipos de datos. En Java, las clases paramétricas se implementan utilizando el mecanismo de tipos genéricos.



En este ejemplo, "T" es el parámetro de tipo que se utiliza para definir la clase genérica. "T" puede ser cualquier identificador válido y se utiliza dentro de la clase para representar el tipo de datos que se utilizará.

```
public class NombreClase<T> {  
    // Cuerpo de la clase  
}
```

Para utilizar una clase paramétrica en Java, se debe especificar el tipo de datos que se utilizará en lugar del parámetro de tipo. Esto se hace utilizando la sintaxis "<Tipo>" después del nombre de la clase.

```
NombreClase<Integer> objeto1 = new NombreClase<Integer>();  
NombreClase<String> objeto2 = new NombreClase<String>();
```

En este ejemplo, se crean dos objetos de la clase NombreClase, uno con tipo Integer y otro con tipo String. El tipo de dato especificado se utiliza para reemplazar el parámetro de tipo "T" en la definición de la clase.

Ventajas de las clases paramétricas

- Reutilización de código: Las clases paramétricas permiten definir una clase genérica que se puede utilizar con diferentes tipos de datos, lo que reduce la necesidad de crear clases similares para diferentes tipos de datos.
- Seguridad de tipos: El uso de tipos genéricos en las clases paramétricas ayuda a garantizar la seguridad de tipos en tiempo de compilación, lo que ayuda a evitar errores de ejecución.
- Legibilidad del código: Las clases paramétricas hacen que el código sea más legible y fácil de entender, ya que los nombres de los tipos de datos utilizados son más descriptivos que los nombres de variables arbitrarios.

diagrama de clases

- El diagrama de clases es una herramienta visual utilizada en programación orientada a objetos para representar las clases y sus relaciones en un sistema. Es una forma útil de representar la estructura de un sistema de software y las relaciones entre sus componentes.

Componentes de un diagrama de clases

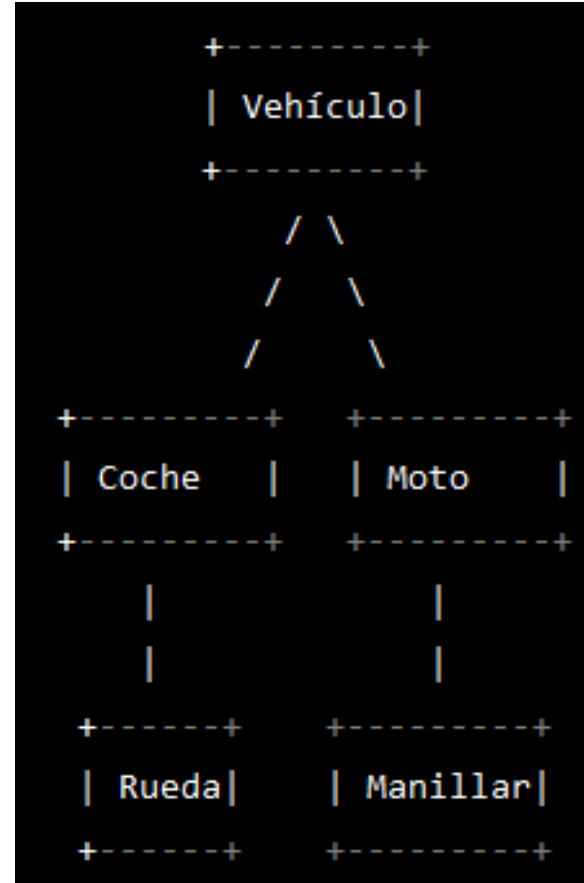
1. Clases: representan los objetos o entidades en el sistema.
2. Relaciones: representan cómo se relacionan las clases en el sistema. Hay varios tipos de relaciones, como la relación de herencia, la relación de agregación y la relación de composición.
3. Atributos: son las características de una clase, como su nombre, tipo de datos y valores predeterminados.

Tipos de relaciones que se pueden representar en un diagrama de clases

- Herencia: representa una relación en la que una clase es una versión especializada de otra clase. Se representa con una flecha que apunta desde la subclase a la superclase.
- Agregación: representa una relación en la que una clase tiene una relación de "tiene un" con otra clase. Se representa con una línea con un rombo en el extremo de la clase que tiene la relación "tiene un".
- Composición: es una forma especializada de agregación en la que una clase es responsable de crear y destruir otra clase. Se representa con una línea con un rombo lleno en el extremo de la clase que crea la otra clase.

diagrama de clases que representa una relación de herencia y una relación de agregación

La clase Vehículo es la superclase de las clases Coche y Moto, lo que se representa con una flecha que apunta desde Coche y Moto hacia Vehículo. Además, la clase Coche tiene una relación de "tiene un" con la clase Rueda, mientras que la clase Moto tiene una relación de "tiene un" con la clase Manillar. Estas relaciones se representan con líneas con un rombo en el extremo de la clase que tiene la relación "tiene un".



Ámbito de las propiedades, Métodos

En este ejemplo, la clase Persona tiene dos propiedades, "nombre" y "edad". La propiedad "nombre" es pública, lo que significa que puede ser accedida desde cualquier parte del sistema. La propiedad "edad" es privada, lo que significa que solo puede ser accedida desde la clase Persona.

```
+-----+  
|  Persona  |  
+-----+  
| +nombre  |  
| -edad    |  
+-----+
```

Ámbito de las propiedades, Métodos

En este ejemplo, la clase Persona tiene dos métodos, "getNombre()" y "setEdad()". El método "getNombre()" es público, lo que significa que puede ser llamado desde cualquier parte del sistema. El método "setEdad()" es privado, lo que significa que solo puede ser llamado desde la clase Persona.

```
+-----+  
|  Persona  |  
+-----+  
| +getNombre() |  
| -setEdad()   |  
+-----+
```


Asociaciones y Restricciones en un Diagrama de Clases

- ▶ Las asociaciones son relaciones entre objetos que nos permiten modelar cómo interactúan los distintos elementos de un sistema. En un diagrama de clases, las asociaciones se representan mediante líneas que conectan las clases. Estas líneas pueden tener restricciones que indican detalles importantes sobre la relación.

Tipos de asociaciones

1. Asociación unidireccional: la asociación unidireccional es aquella en la que la relación entre dos objetos solo fluye en una dirección.
2. Asociación bidireccional: la asociación bidireccional es aquella en la que la relación entre dos objetos fluye en ambas direcciones.

Clases de asociaciones

1. Asociación simple: es la asociación más común y representa una conexión directa entre dos clases.
2. Asociación de agregación: representa una relación de "parte de". Una clase A es parte de otra clase B.
3. Asociación de composición: representa una relación de "todo". Una clase A es el todo y la clase B es una parte fundamental de ella.

Restricciones de multiplicidad

1. Uno a Uno: solo existe una relación entre los objetos.
2. Uno a muchos: un objeto puede estar relacionado con muchos objetos en el otro extremo de la asociación.
3. Muchos a muchos: varios objetos pueden estar relacionados con varios objetos en el otro extremo de la asociación.

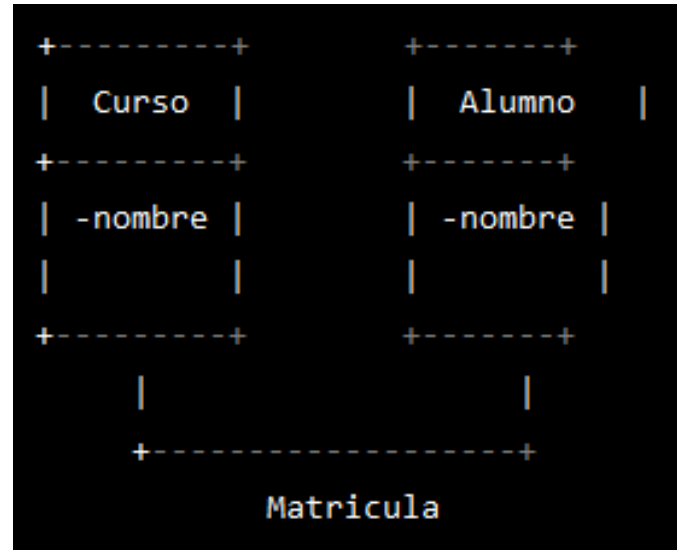
Dependencia

La dependencia representa una relación temporal entre dos clases. Cuando una clase depende de otra, significa que la segunda es necesaria para el correcto funcionamiento de la primera.

```
+-----+           +-----+
|  Persona  |       |  Coche  |
+-----+           +-----+
| -nombre  |       | -modelo  |
| -edad    | <----- | -color  |
+-----+           +-----+
```

Relaciones Múltiples (Asociativas)

- Una relación múltiple es aquella en la que una clase tiene más de una asociación con otra clase. Esto puede suceder en situaciones donde necesitamos representar una relación más compleja entre objetos.



Relaciones Reflexivas

- Una relación reflexiva es aquella en la que una clase se relaciona consigo misma. Esto sucede cuando necesitamos modelar una relación entre objetos de la misma clase.

En este ejemplo, la clase Persona tiene una relación reflexiva, ya que necesitamos relacionar una persona con sus padres (que también son personas). La propiedad padre y madre son de tipo Persona.

```
+-----+  
| Persona |  
+-----+  
| -nombre |  
| -edad   |  
| -padre  |  
| -madre  |  
+-----+
```



Parte práctica