Programación Orientada a Objetos

Desarrollo de Sistemas Web - Back End

Versión 1.0 F. Act 05/09/2024

Programación orientada a objetos

- La **Programación Orientada a Objetos** (POO, o en inglés OOP) es un paradigma de programación que organiza el código en torno a *objetos*, que son instancias de *clases*. Este enfoque es ampliamente utilizado debido a su capacidad para modelar conceptos del mundo real y facilitar la planificación, la organización y la escalabilidad del código en aplicaciones complejas.

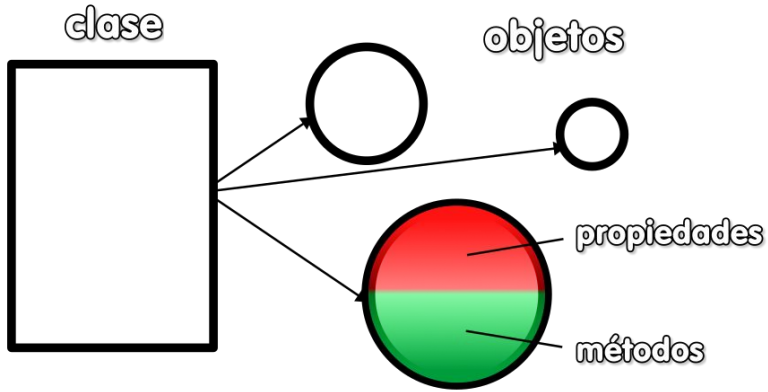
Conceptos Clave de la POO

1. **Clase**:
    - **Definición:** Una *clase* es una plantilla o un esquema que define un tipo de objeto. Es una estructura que agrupa datos (variables) y métodos (funciones) que operan sobre esos datos.
    - **Ejemplo:** En un juego, podríamos tener una clase Héroe que define propiedades como vidas, fuerza, y métodos como atacar() o curar().
2. **Objeto**:
    - **Definición:** Un *objeto* es una instancia concreta de una clase. Representa una entidad específica creada a partir de una clase con sus propias propiedades y comportamientos.
    - **Ejemplo:** Un objeto héroe1 podría ser un Héroe específico con 3 vidas, 10 de fuerza, y habilidades específicas como disparar().

Diferencia entre Clase y Objeto

Diferencia entre Clase y Objeto

- **Clase:** Es un concepto abstracto que define un tipo de objeto. Actúa como un plano o plantilla.
- **Objeto:** Es una instancia concreta de una clase con valores específicos para sus propiedades.



Beneficios de la POO

1. **Organización**:
   - **Descripción:** La POO ayuda a organizar el código al agrupar variables y funciones relacionadas dentro de una clase. Esto evita el caos que se puede producir al tener muchas variables y funciones sueltas en el código.
   - **Ejemplo:** En lugar de tener variables como vidas, fuerza, y funciones como atacar() esparcidas por el código, estas se agrupan en la clase Héroe.
2. **Modularidad**:
   - **Descripción:** La POO permite crear módulos independientes (clases) que pueden ser reutilizados y mantenidos fácilmente.
   - **Ejemplo:** La clase Héroe puede ser reutilizada para diferentes personajes en el juego, cada uno con sus propias instancias.
3. **Escalabilidad**:
   - **Descripción:** Las clases pueden ser extendidas para crear nuevas clases con propiedades adicionales o modificadas, facilitando la expansión del código.
   - **Ejemplo:** Podríamos extender la clase Héroe para crear una clase Mago que hereda todas las características del Héroe y añade nuevas habilidades mágicas.

Una clase

Propiedad: Variable que existe dentro de una clase. Puede ser pública o privada.

- Propiedad pública: Propiedad a la que se puede acceder desde fuera de la clase.
- Propiedad privada: Propiedad a la que no se puede acceder desde fuera de la clase.
- Propiedad computada: Función para acceder a una propiedad con modificaciones (getter/setter).

Método: Función que existe dentro de una clase. Puede ser pública o privada..

- Método público: Método que se puede ejecutar desde dentro y fuera de la clase.
- Método privado : Método que sólo se puede ejecutar desde dentro de la clase.
- Constructor: Método especial que se ejecuta automáticamente cuando se crea una instancia.
- Método estático: Método que se ejecuta directamente desde la clase, no desde la instancia.
- Inicializador estático: Bloque de código que se ejecuta al definir la clase, sin necesidad de instancia.

```
class Persona {
  // Propiedad pública
  nombre;

  // Propiedad privada
  #edad;

  constructor(nombre, edad) {
    this.nombre = nombre;
    this.#edad = edad;
  }
}
```

Propiedad Pública:

Una propiedad pública es accesible desde fuera de la clase.

```
class Car {
    constructor(brand) {
        this.brand = brand; // Propiedad pública
    }
}

const myCar = new Car('Toyota');
console.log(myCar.brand); // Acceso a propiedad pública
```

Propiedad Privada

Una propiedad privada no se puede acceder directamente desde fuera de la clase. Se define con un # delante del nombre de la propiedad.

```
class Persona {
  #edad;

  constructor(edad) {
    this.#edad = edad;
  }

  obtenerEdad() {
    return this.#edad;
  }
}

const persona = new Persona(30);
console.log(persona.obtenerEdad()); // 30
console.log(persona.#edad); // Error: Property '#edad' is not accessible outside class
```

Propiedad Conmutada

Una propiedad computada se accede mediante funciones getter y setter.

```
class Persona {
  #nombre;

  constructor(nombre) {
    this.#nombre = nombre;
  }

  get nombre() {
    return this.#nombre.toUpperCase(); // Modificación
  }

  set nombre(nuevoNombre) {
    this.#nombre = nuevoNombre;
  }
}

const persona = new Persona('ana');
console.log(persona.nombre); // "ANA"
persona.nombre = 'luis';
console.log(persona.nombre); // "LUIS"
```