Tipos de Clases en POO

Tomás Poveda Trujillo

Universidad de Cundinamarca (Chia)

Facultad de Ingeniería

Ingeniería de sistemas

Ingeniería de Software

Docente

William Alexander Matallana Porras

10/02026

Introducción

La Programación Orientada a Objetos (POO) es un modelo de programación que nos permite modelar sistemas a través de objetos que representan entidades del mundo real o conceptual. Estos objetos contienen atributos y métodos, lo que facilita la organización, reutilización y mantenimiento del código.

Dentro de la POO existen diferentes tipos de clases, cada una con una función especifica en el diseño de software. Comprenderlos es fundamental para el de desarrollo de aplicaciones estructuradas, escalables y eficientes, especial mente en áreas como el desarrollo de videojuegos, aplicaciones empresariales y sistemas complejos.

Tipos de Clases

1.Clase concreta

Es la clase más común. Puede ser instanciada, es decir se pueden crear objetos directamente a partir de ella.

Ejemplo:

```
Public class Jugador {
    private int vida;
    private String nombre;
    public Jugador (String nombre, int vida) {
        this.nombre = nombre;
        this.vida= vida;
    }
    public void atacar (){
        System. Out. Println(nombre + “está atacando.”);
    }
}
```

Uso:

```
Jugador  J1=new jugador(“Carlos”,100);
J1.atacar();
```

2.Clase Abstracta

No puede instanciarse directamente. Sirve como base para otras clases y puede contener métodos abstractos (sin implementación).

Ejemplo:

```
Public abstract class Personaje {
    Protected int vida;
    Public Personaje (int  vida) {
        This. Vida = vida;
    }
    Public abstract void atacar();
}
```

Subclase:

```
Public class Guerrero extends Personaje {
    Public Guerrero(int vida) {
        Super(vida) ;
    }
    @Override
    Public void atacar(){
        System.out.println(“El guerrero ataca con espada.”);
    }
}
```

Uso:

```
Guerrero g1 = new Guerrero (150);

g1.atacar();
```

3.interface

Define un conjunto de métodos que deben ser implementados por las clases que la utilicen. No contiene implementación (excepto métodos por defecto en versiones modernas).

Ejemplo:

```
Public interface volador {

    Void volar();

}
```

Clase que implementa la interfaz:

```
Public class Dragon implements volador {

    @Override

    Public void volar () {

        System.out.println("El dragón está volando.");

    }

}
```

Uso:

```
Dragon d1 = new Dragon ();

d1. volar ();
```

4.Super Clase (clase base)

Es la clase padre de la cual otras clases heredan atributos y métodos.

Ejemplo:

```
Public class Animal {
    Public void comer () {
        System.out.println("El animal está comiendo.");
    }
}
```

5.Subclase (Clase Derivada)

Hereda de una superclase.

```
Public class perro extends animal {
    Public void ladrar () {
        System. Out . println ("El perro ladra.");
    }
}
```

Uso:

```
Perro p1=new perro ();
p1.comer();   //heredado
p1.ladrar();    //propio
```

6.Clase interna (inner class)

Es una clase definida dentro de otra clase.

```
Public class Juego {
    Class Menu {
        Public void mostrar() {
            System. Out. Println("mostrando menú del juego.");
        }
 }}
```

7.Clase estática (Nested static class)

Es una clase interna declarada con la palabra clave [static].

```
Public class Juego {
    Static class Configuration {
        Int volumen = 50;
        Public void mostrarVolumen() {
            System. Out. Println("volumen: "  +volumen);
        }
    }
}
```

Uso:

```
Juego.Configuracion config = new Juego.Configuracion();
Config.mostrarVolumen();
```

8.Clase Singleton (patrón de diseño)

Permite que exista una única instancia de la clase en todo el programa.

```
Public class GameManager {
    Private static GameManager instancia;
    Private GameManager() {} //constructor vacío
    Public static   gamemanager getInstancia(){
        If (instancia == null) {
            Instancia =new GameManager();
        }
        Return instancia;
    }
    Public void iniciarJuego() {
        System.out.println("juego iniciado.");
```

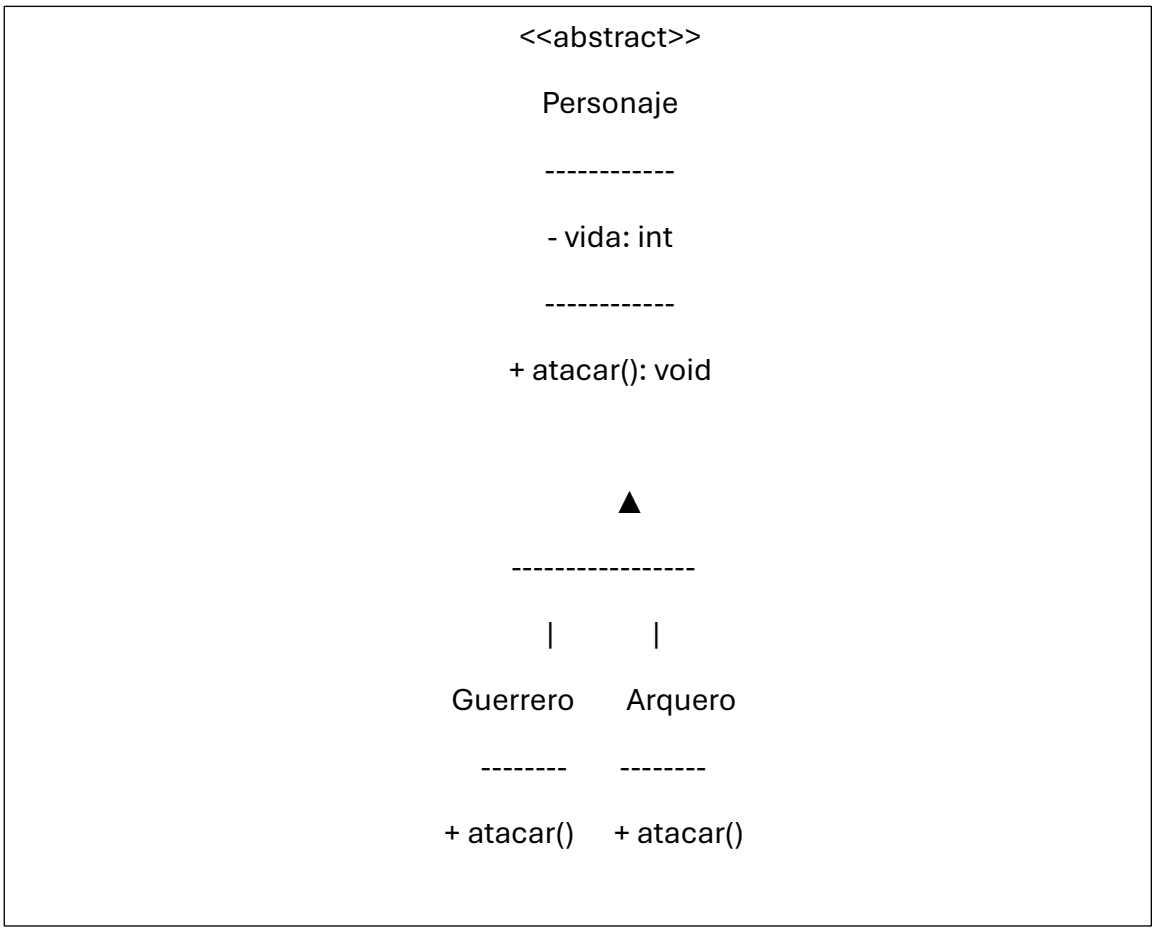
```
        }
    }
```

Uso:

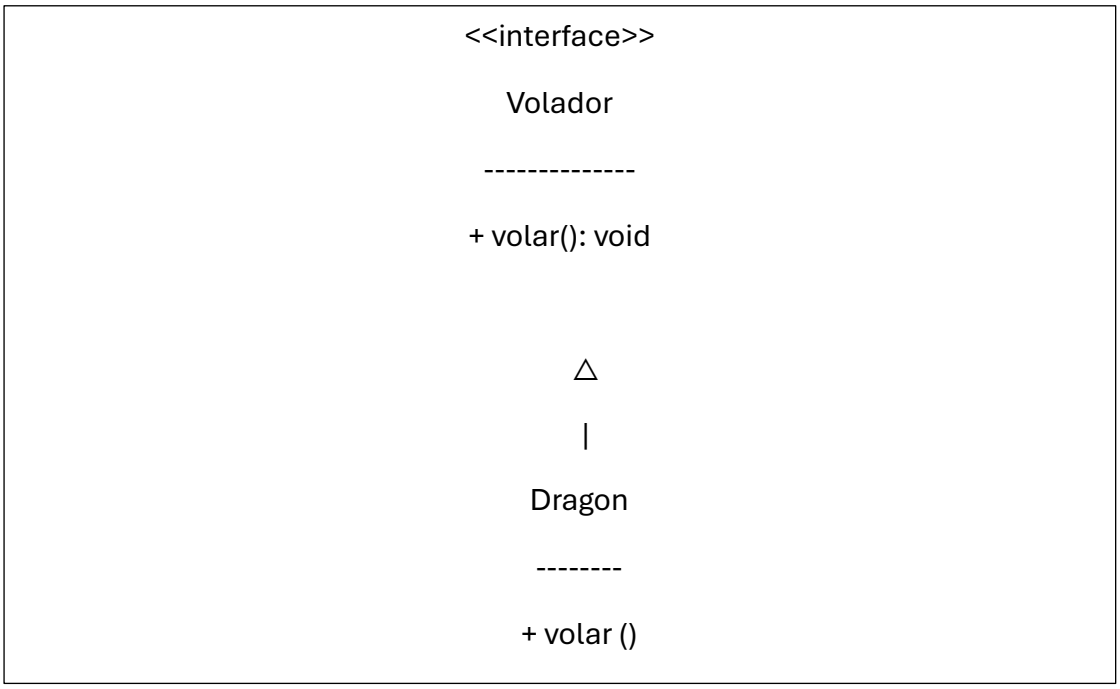
```
    GameManager gm= GameManager.getInstancia();
    gm.iniciarJuego();
```

Diagramas de las Clases

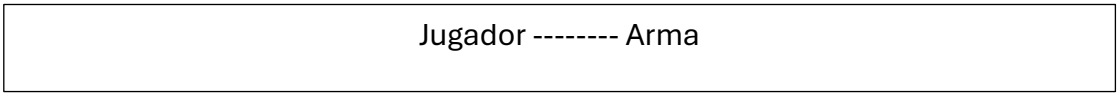
1.Diagrama de Herencia (Clase abstracta + Subclases)



2.Diagrama con Interface



3.Diagrama con Relación de asociación



Conclusiones

Los diferentes tipos de clases en la programación orientada a objetos permiten estructurar el software de manera organizada y eficiente. Las clases concretas permiten crear objetos funcionales, las clases abstractas y las interfaces promueven la reutilización y el polimorfismo, mientras que las relaciones de herencia facilitan la extensión del comportamiento. Además, estructuras como las clases internas y el patrón singleton contribuyen a una mejor organización del código.

El dominio de estos tipos de clases es fundamental para el desarrollo de software moderno, incluyendo aplicaciones empresariales, sistemas web y videojuegos.

EL patrón Singleton: El patrón de diseño Singleton es un patrón creacional que garantiza que una clase tenga solo una instancia y proporciona un punto de acceso global único a ella, ideal para controlar recursos compartidos como conexiones a

bases de datos o archivos de configuración, implementándose al hacer el constructor privado y ofreciendo un método estático para obtener la única instancia.