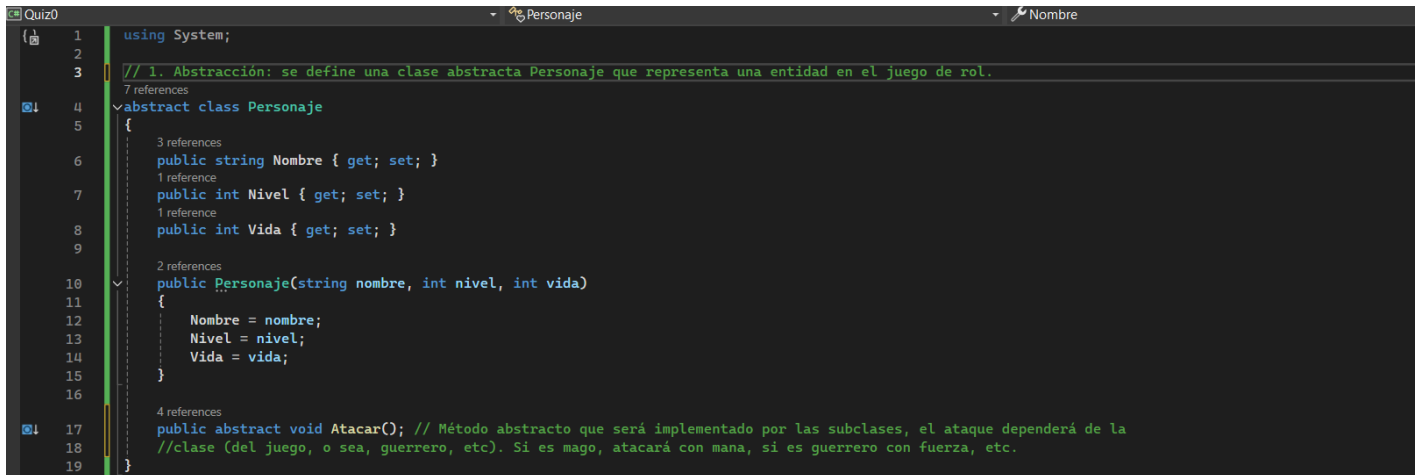


## Curso: Algoritmos y Estructura de Datos 1

## Quiz 0

Para el primer principio que menciona el folleto (principio de Abstracción) se usa una clase abstracta (Personaje) para representar cualquier tipo de personaje en el juego de rol. De esta forma nos va a permitir definir ciertos atributos (nombre, nivel y vida) y métodos (atacar) sin que haga falta de especificarlo cómo se va a implementar para cada personaje. La *figura 1* muestra la creación de la clase Personaje, así como sus atributos y métodos.



```
1 using System;
2
3 // 1. Abstracción: se define una clase abstracta Personaje que representa una entidad en el juego de rol.
4 abstract class Personaje
5 {
6     3 references
7     public string Nombre { get; set; }
8     1 reference
9     public int Nivel { get; set; }
10    1 reference
11    public int Vida { get; set; }
12
13    2 references
14    public Personaje(string nombre, int nivel, int vida)
15    {
16        Nombre = nombre;
17        Nivel = nivel;
18        Vida = vida;
19    }
20
21    4 references
22    public abstract void Atacar(); // Método abstracto que será implementado por las subclases, el ataque dependerá de la
23    //clase (del juego, o sea, guerrero, etc). Si es mago, atacará con mana, si es guerrero con fuerza, etc.
24 }
```

Figura 1. Implementación en C# del principio de abstracción

Para el principio de encapsulamiento, se aplica un encapsulamiento del atributo fuerza del guerrero (que es herencia de la clase Personaje) como lo muestra la *figura 2*. Esto protege los datos de un objeto, permitiendo ocultar su implementación y exponer solo lo necesario mediante modificadores de acceso (que en este caso es con `private`).

```
21 // 2. Encapsulamiento: la clase guerrero tiene atributos privados y métodos publicos para interactuar con ellos. Sus atributos privados
22 // (fuerza) están protegidos para evitar modificaciones.
23 class Guerrero : Personaje
24 {
25     private int fuerza;
26
27     1 reference
28     public Guerrero(string nombre, int nivel, int vida, int fuerza) : base(nombre, nivel, vida)
29     {
30         this.fuerza = fuerza;
31     }
32
33     0 references
34     public int GetFuerza()
35     {
36         return fuerza;
37     }
38
39     0 references
40     public void SetFuerza(int nuevaFuerza)
41     {
42         if (nuevaFuerza > 0)
43             fuerza = nuevaFuerza;
44     }
45
46     3 references
47     public override void Atacar()
48     {
49         Console.WriteLine($"{Nombre} ataca con su espada con fuerza {fuerza}!");
50     }
51 }
```

Figura 2. Implementación en C# del principio de encapsulamiento.

En el caso del principio de herencia, se establece una jerarquía donde guerrero y mago son tipos específicos de Personaje (herendan de la clase Personaje), reutilizando código y añadiendo su propio comportamiento. La *figura 3* muestra la implementación del principio de herencia en el código (el cual ya se aplicó para la clase guerrero en la *figura 2*).

```
49 // 3. Herencia: la clase mago hereda de Personaje y añade un atributo específico, que en su caso es mana.
50 2 references
51 class Mago : Personaje
52 {
53     2 references
54     private int Mana { get; set; }
55
56     1 reference
57     public Mago(string nombre, int nivel, int vida, int mana) : base(nombre, nivel, vida)
58     {
59         Mana = mana;
60     }
61
62     3 references
63     public override void Atacar()
64     {
65         Console.WriteLine($"{Nombre} lanza un hechizo usando {Mana} de mana!");
66     }
67 }
```

Figura 3. Implementación en C# del principio de herencia.

Finalmente, se implementó un polimorfismo en tiempo de ejecución el cual ocurre cuando una subclase sobrescribe un método de la clase base. En este caso, las subclases mago y guerrero sobrescriben (haciendo uso de un override) el método Atacar de la clase base Personaje. Como se observa en la *figura 1*, el método Atacar está definido en Personaje, pero cada subclase (guerrero y mago) lo implementa de forma diferente, como lo muestra la *figura 3* y la *figura 4*.

```
65 // 4. Polimorfismo: permite tratar diferentes tipos de personajes de manera uniforme y sin replicaciones.  
66 0 references  
67 class Program  
68 {  
69     0 references  
70     static void Main()  
71     {  
72         Personaje guerrero = new Guerrero("Garrosh", 10, 100, 20);  
73         Personaje mago = new Mago("Khadgar", 12, 80, 50);  
74         guerrero.Atacar(); // Polimorfismo en tiempo de ejecución (override)  
75         mago.Atacar();  
76         Console.ReadLine();  
77     }  
78 }
```

Figura 4. Implementación y código para prueba de clases.

La *figura 5* muestra el comportamiento de la prueba de clases de la *figura 4*. Se observa que los personajes atacan de forma diferente según la clase a la que pertenezcan (guerrero o mago).



```
C:\Users\Javier\OneDrive\Escriba aquí...
Garrosh ataca con su espada con fuerza 20!
Khadgar lanza un hechizo usando 50 de mana!
```

**Figura 5.** Print de la terminal al correr el programa.