

Instituto Tecnológico Costa Rica

Escuela de Ingeniería en Computadores

Algoritmos Y Estructuras De Datos I (CE1103)

Profesor: Jose Isaac Ramirez Herrera

Quiz 0: Programación Orientada a Objetos (POO)

Estudiante: Adrián Monge Mairena

Carné: 2023800088

Ciclo: I Semestre 2025

Contenidos

Introducción.....	3
Clases y Objetos	3
Abstracción	4
Encapsulamiento.....	4
Herencia.....	5
Polimorfismo	5
Ejecución.....	6
Conclusión	7
Bibliografía	7

Introducción

Este documento, confeccionado por Adrián Monge Mairena de carnet 2023800088, durante su preparación profesional en el Tecnológico de Costa Rica en la carrera de Licenciatura en Ingeniería en Computadores, presenta una explicación de los principales conceptos de Programación Orientada a Objetos (POO) junto con ejemplos en C# .NET. Cada principio es descrito brevemente y acompañado de un fragmento de código y su respectiva captura de pantalla de ejecución.

Clases y Objetos

Una **clase** es una plantilla o modelo que define la estructura y el comportamiento de un objeto. Un **objeto** es una instancia de una clase, que contiene atributos (datos) y métodos (acciones). La abstracción permite modelar los objetos ocultando los detalles de implementación y exponiendo solo lo esencial. Se implementa mediante clases abstractas o interfaces.

```
// Quiz 0 //  
// Adrian Monge Mairena | 2023800088 //  
// Algoritmo y Estructura de Datos I | 1S 2025 | Jose Isaac Ramirez Herrera //  
using System;  
  
// 1. CLASE: Es una plantilla que define la estructura y el comportamiento de los objetos.  
2 references  
class Auto  
{  
    // 2. OBJETO: Es una instancia de una clase. Contiene atributos (datos) y métodos (acciones).  
    2 references  
    public string Marca { get; set; }  
    3 references  
    public int Velocidad { get; set; }  
  
    1 reference  
    public void Acelerar()  
    {  
        Velocidad += 10;  
        Console.WriteLine($"El auto {Marca} aceleró a {Velocidad} km/h");  
    }  
}
```

Figura 1: Ejemplo de una Clase con un objeto en su instancia

Abstracción

La abstracción permite modelar los objetos ocultando los detalles de implementación y exponiendo solo lo esencial. Se implementa mediante clases abstractas o interfaces.

```
// 3. ABSTRACCIÓN: Se ocultan los detalles innecesarios y se muestran solo las características esenciales.
2 references
abstract class Figura
{
    2 references
    public abstract void Dibujar(); // Método abstracto obliga a las clases derivadas a implementarlo
}

1 reference
class Circulo : Figura
{
    2 references
    public override void Dibujar()
    {
        Console.WriteLine("Dibujando un círculo");
    }
}
```

Figura 2: Ejemplo de Abstracción

Encapsulamiento

El encapsulamiento restringe el acceso a los datos de una clase y solo permite su modificación a través de métodos definidos.

```
// 4. ENCAPSULAMIENTO: Se ocultan los detalles internos de una clase y solo se exponen métodos controlados.
2 references
class CuentaBancaria
{
    private decimal saldo; // La variable es privada, no puede modificarse directamente desde fuera

    1 reference
    public void Depositar(decimal cantidad)
    {
        if (cantidad > 0)
        {
            saldo += cantidad;
            Console.WriteLine($"Depósito exitoso. Saldo actual: {saldo}");
        }
    }

    1 reference
    public decimal ObtenerSaldo() // Método para acceder al saldo
    {
        return saldo;
    }
}
```

Figura 3: Ejemplo de Encapsulamiento sobre la variable Saldo

Herencia

La herencia permite que una clase reutilice atributos y métodos de otra clase. Se establece una relación padre-hijo entre clases.

```
// 5. HERENCIA: Permite que una clase derive de otra, reutilizando código.
1 reference
class Animal
{
    1 reference
    public void Comer()
    {
        Console.WriteLine("El animal está comiendo");
    }
}

2 references
class Perro : Animal // Perro hereda de Animal
{
    1 reference
    public void Ladrar()
    {
        Console.WriteLine("El perro está ladrando");
    }
}
```

Figura 4: Definición de la clase Animal y Perro

Polimorfismo

El polimorfismo permite que un mismo método tenga diferentes comportamientos según la clase que lo implemente.

```
// 6. POLIMORFISMO: Permite que un mismo método se comporte de diferentes maneras según el objeto.
2 references
class FiguraGeometrica
{
    2 references
    public virtual void Dibujar()
    {
        Console.WriteLine("Dibujando una figura");
    }
}

1 reference
class Cuadrado : FiguraGeometrica
{
    2 references
    public override void Dibujar()
    {
        Console.WriteLine("Dibujando un cuadrado");
    }
}
```

Figura 5: Definición del método Dibujar en la clase FiguraGeometrica que cambia su comportamiento cuando es llamado por la Clase Cuadrado

Ejecución

```
0 references
class Program
{
    0 references
    static void Main()
    {
        // Ejemplo de Objeto y Clase
        Auto miAuto = new Auto { Marca = "Toyota", Velocidad = 0 };
        miAuto.Acelerar();

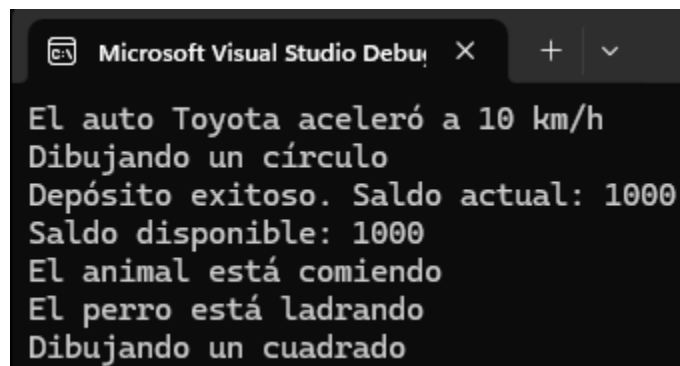
        // Ejemplo de Abstracción
        Figura figura = new Circulo();
        figura.Dibujar();

        // Ejemplo de Encapsulamiento
        CuentaBancaria cuenta = new CuentaBancaria();
        cuenta.Depositar(1000);
        Console.WriteLine("Saldo disponible: " + cuenta.ObtenerSaldo());

        // Ejemplo de Herencia
        Perro perro = new Perro();
        perro.Comer(); // Método heredado de Animal
        perro.Ladrar(); // Método propio de Perro

        // Ejemplo de Polimorfismo
        FiguraGeometrica figura1 = new Cuadrado();
        figura1.Dibujar(); // Llama al método específico de Cuadrado
    }
}
```

Figura 6: Fragmento de Código que ejecuta los ejemplos de cada Concepto



```
Microsoft Visual Studio Debug Console
El auto Toyota aceleró a 10 km/h
Dibujando un círculo
Depósito exitoso. Saldo actual: 1000
Saldo disponible: 1000
El animal está comiendo
El perro está ladrando
Dibujando un cuadrado
```

Figura 7: Resultado de las ejecución de cada ejemplo

Conclusión

Se ha demostrado la aplicación de los principios fundamentales de POO en C# mediante ejemplos prácticos y explicaciones claras. La comprensión de estos conceptos es clave para el desarrollo eficiente de software estructurado y mantenible.

Bibliografía

- Ramírez Herrera, J. I. (2025). *Algoritmos y estructuras de datos* (Versión 0.3).
Obtenido el 19 de febrero de 2025, de [este enlace](#).