UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE PROGRAMACIÓN DE COMPUTADORAS

Facilitador(a): Migdalia Testa Asignatura: Desarrollo IV

Estudiante: _____ Fecha: 09 de octubre de 2024

A. TÍTULO DE LA EXPERIENCIA: Herencia y Polimorfismo

B. TEMAS:

- Revisar el manejo de Herencia
- Implementar el uso de Polimorfismo

C. OBJETIVO(S):

- Utilizar arreglos dinámicos en los programas.

D. METODOLOGÍA: Desarrollar el programa que se solicita a continuación.

E. PROCEDIMIENTO DE LA EXPERIENCIA

Pasemos a crear una solución con C# para consola orientado a objetos para la siguiente situación

Queremos guardar los nombres y las edades de los alumnos de un curso.
Realiza un programa que permita introducir el nombre y la edad de cada alumno en listas.

Al finalizar se mostrará los siguientes datos:
a. Todos los alumnos mayores de edad.
b. Obtener la edad mayor y listar los alumnos con esta edad.

Cree una clase Persona

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HerenciaconPolimorfismo
{
    using System;

    public class Persona
    {
        public int Id { get; set; }
        public string Nombre { get; set; }

        public virtual void MostrarInformacion()
        {
            Console.WriteLine($"ID: {Id}, Nombre: {Nombre}");
```

```
        }
    }

    public class Cliente : Persona
    {
        public string Categoria { get; set; }

        public override void MostrarInformacion()
        {
            base.MostrarInformacion();
            Console.WriteLine($"Categoría: {Categoria}");
        }
    }

    public class Empleado : Persona
    {
        public string Puesto { get; set; }

        public override void MostrarInformacion()
        {
            base.MostrarInformacion();
            Console.WriteLine($"Puesto: {Puesto}");
        }
    }

}
```

Crear la clase GestorPersonas

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HerenciaconPolimorfismo
{
    using System;
    using System.Collections.Generic;

    public class GestorPersonas
    {
        private List<Persona> personas = new List<Persona>();

        public void CrearPersona(Persona persona)
        {
            personas.Add(persona);
        }

        public void LeerPersonas()
        {
```

```
            foreach (var persona in personas)
            {
                persona.MostrarInformacion();
            }
        }

        public void ActualizarPersona(int id, Persona nuevaPersona)
        {
            var persona = personas.Find(p => p.Id == id);
            if (persona != null)
            {
                persona.Nombre = nuevaPersona.Nombre;
                if (persona is Cliente cliente && nuevaPersona is Cliente
nuevoCliente)
                {
                    cliente.Categoria = nuevoCliente.Categoria;
                }
                else if (persona is Empleado empleado && nuevaPersona is Empleado
nuevoEmpleado)
                {
                    empleado.Puesto = nuevoEmpleado.Puesto;
                }
            }
        }

        public void EliminarPersona(int id)
        {
            var persona = personas.Find(p => p.Id == id);
            if (persona != null)
            {
                personas.Remove(persona);
            }
        }
    }

}
```

Crear la clase Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HerenciaconPolimorfismo
{
    using System;
    using System.Collections.Generic;
```

```
    public class GestorPersonas
    {
        private List<Persona> personas = new List<Persona>();

        public void CrearPersona(Persona persona)
        {
            personas.Add(persona);
        }

        public void LeerPersonas()
        {
            foreach (var persona in personas)
            {
                persona.MostrarInformacion();
            }
        }

        public void ActualizarPersona(int id, Persona nuevaPersona)
        {
            var persona = personas.Find(p => p.Id == id);
            if (persona != null)
            {
                persona.Nombre = nuevaPersona.Nombre;
                if (persona is Cliente cliente && nuevaPersona is Cliente
nuevoCliente)
                {
                    cliente.Categoria = nuevoCliente.Categoria;
                }
                else if (persona is Empleado empleado && nuevaPersona is Empleado
nuevoEmpleado)
                {
                    empleado.Puesto = nuevoEmpleado.Puesto;
                }
            }
        }

        public void EliminarPersona(int id)
        {
            var persona = personas.Find(p => p.Id == id);
            if (persona != null)
            {
                personas.Remove(persona);
            }
        }
    }

}
```

F. ENUNCIADO DE LA EXPERIENCIA:

Gestión de un Zoológico.

A continuación, se presenta la jerarquía de clases que representa los animales de un posible zoológico:

La clase Animal es una clase abstracta con cuatro atributos miembros protected:

a) Una cadena indicando la especie (león, águila, abeja),

b) Una cadena indicando el nombre del animal concreto

c) Un dato numérico real indicando el peso en kg.

d) Un dato numérico entero indicando el número de jaula que se asigna al animal.

Además, la clase Animal declara un método virtual DefinirClaseDeAnimalEres(): string que habrá que sobrescribir en las clases derivadas.

La clase Mamifero no añade nuevos atributos, aunque deberá implementar el método DefinirClaseDeAnimalEres().

La clase Ave tiene dos nuevos atributos protected:

a) Una cadena colorPlumaje indicando el color predominante y

b) Un dato numérico real indicando la alturaMaximaVuelo.

La clase Insecto tiene un nuevo atributo miembro protected de tipo booleano llamado vuela que indica si el insecto vuela o no.

Para realizar este ejercicio se pide lo siguiente:

I. Crear las cuatro clases indicadas, con los correspondientes constructores y sobrecarga de constructores, Como ayuda, se indica que el orden de los argumentos en el constructor parametrizado de la clase base es:

   public Animal(string especie, string nombre, double peso, int jaula) {...}

II. Definir los métodos llamados DefinirClaseDeAnimalEres( ) en cada una de las clases derivadas de Animal. Este método devuelve una cadena (es string) pero no recibe parámetros o argumentos.

Debe ser capaz de mostrar por la pantalla la información correspondiente al animal de que se trate (ver el ejemplo), utilizando para ello la información almacenada en las variables miembro.

Ejemplo: Soy un mamífero llamado: xxxxxxxx

de la especie: xxxxxxxx

Peso en Kg: xxx

Estoy en la jaula: xx

G. RECURSOS:

ArrayList
Internet

H. CONSIDERACIONES FINALES:

Una vez finalizados los programas publicarlos en E-Campus.

I. BIBLIOGRAFIA:

J. RÚBRICAS:

En este trabajo será evaluado como una práctica.