Evaluación del 1er Parcial

(Parte Práctica).

Nombre: Edwin Matthew Salazar Encalada **Nrc:** 14949

Fecha: 13/12/2023 **Docente:** DIEGO MEDARDO SAAVEDRA GARCIA

Asignatura: Web Avanzado

Ejercicio 1: Variables y Operadores en C#

1. Declarar dos variables, **`numero1`** y **`numero2`**, e inicialízalas con valores numéricos.
2. Calcula la suma de estas dos variables y almacena el resultado en una tercera variable llamada **`resultado`**.
3. Imprime en la consola el valor de `resultado`.

Ejercicio 2: Estructuras de Control en C#

1. Declara una variable `edad` e inicialízala con un valor numérico.
2. Utiliza una estructura **`if`** para determinar si la persona es mayor de edad (mayor o igual a 18).
3. Imprime en la consola un mensaje indicando si la persona es mayor de edad o no.

Ejercicio 3: Programación Orientada a Objetos - Clases y Objetos

1. Crea una clase llamada **`Estudiante`** con propiedades como **`Nombre`**, **`Edad`** y **`Calificacion`**.
2. Crea un objeto de tipo **`Estudiante`** llamado **`estudiante1`** e inicializa sus propiedades con valores ficticios.
3. Imprime en la consola la información del estudiante.

Ejercicio 4: Programación Orientada a Objetos - Métodos

1. Amplía la clase `Estudiante` con un método llamado **`MostrarInformacion`** que imprima en la consola los detalles del estudiante.
2. Llama al método **`MostrarInformacion`** para el objeto **`estudiante1`** y observa la salida.

Ejercicio 5: Programación Orientada a Objetos - Herencia

1. Crea una nueva clase llamada **`EstudianteGraduado`** que herede de la clase **`Estudiante`**.
2. Añade una nueva propiedad a **`EstudianteGraduado`** llamada **`Titulo`** que almacene el título obtenido.

3. Crea un objeto de tipo **`EstudianteGraduado`** llamado **`graduado1`** e inicializa sus propiedades.
4. Utiliza el método **`MostrarInformacion`** de la clase base para mostrar la información del estudiante graduado.

Preguntas de Reflexión:

1. **¿Cuál es la diferencia entre una variable y una propiedad en C#?**

En C#, tanto las variables como las propiedades son elementos fundamentales utilizados para almacenar y manipular datos, pero tienen diferencias clave en su uso y propósito.

Variable:
Almacenamiento de Datos: Una variable es una ubicación de almacenamiento en la memoria que contiene un valor específico.

Acceso Directo: Puedes acceder directamente a una variable para leer o modificar su valor.

Visibilidad: La visibilidad de una variable puede depender de su alcance (por ejemplo, si está declarada dentro de un método, solo es visible en ese método).

Sintaxis de Declaración: Se declara usando el tipo de dato seguido por el nombre de la variable, como por ejemplo: int numero = 5;.

Propiedad:
Encapsulamiento: Una propiedad es una especie de método especial que proporciona un mecanismo para acceder y modificar valores privados de una clase. Proporciona un nivel de encapsulamiento al ocultar los detalles internos de implementación y permitir un control más granular sobre el acceso a los datos.

Métodos de Acceso: Una propiedad generalmente tiene métodos de acceso llamados get y set. El método get se utiliza para obtener el valor de la propiedad, mientras que el método set se utiliza para establecer el valor de la propiedad.

Sintaxis de Declaración: Se declara utilizando una sintaxis específica con get y set,

2. **Explica cómo funciona la estructura `if` y por qué es útil en programación.**

La estructura if en programación es una instrucción de control de flujo que permite ejecutar cierto bloque de código si una condición dada es evaluada como verdadera. Su estructura básica en C# es la siguiente:

```
if (condicion)
{

}
```

Donde condicion es una expresión booleana. Si la condición es verdadera, el bloque de código dentro del if se ejecutará; de lo contrario, se omitirá.

Funcionamiento:
Evaluación de la Condición: La expresión dentro del paréntesis (condicion) se evalúa

como verdadera o falsa. Esta condición puede ser una comparación, una operación lógica, o cualquier expresión que devuelva un valor booleano.

Ejecución del Bloque de Código: Si la condición es verdadera, el bloque de código dentro de las llaves {} asociado al if se ejecutará. Si la condición es falsa, el bloque se omitirá y la ejecución continuará con la siguiente instrucción después del bloque if.

3. **¿Qué ventajas ofrece la programación orientada a objetos en comparación con otrosparadigmas de programación?**

La programación orientada a objetos (POO) es un paradigma de programación que utiliza objetos, clases y otros conceptos relacionados para organizar y estructurar el código. Comparada con otros paradigmas de programación, la POO ofrece varias ventajas:

- Reutilización de Código (Reusabilidad): La POO permite la reutilización de código a través de conceptos como herencia y composición. Puedes crear nuevas clases basadas en clases existentes, lo que facilita la reutilización de funcionalidades ya implementadas.

- Modularidad: La POO favorece la creación de módulos y componentes bien encapsulados. Las clases actúan como unidades independientes, y los cambios internos a una clase no afectan directamente a otras partes del programa, lo que facilita el mantenimiento y la evolución del software.

- Encapsulamiento: La encapsulación oculta los detalles internos de implementación de una clase y expone solo la interfaz necesaria para interactuar con ella. Esto mejora la seguridad y facilita el control del acceso a los datos y funciones.

- Abstracción: La abstracción permite representar conceptos del mundo real en el código de una manera más cercana a cómo los entendemos conceptualmente. Esto facilita la modelización y comprensión del software.

4. **¿Cuándo usarías la herencia en un diseño de clases?**

La herencia es un concepto fundamental en la programación orientada a objetos que permite la creación de nuevas clases basadas en clases existentes, aprovechando y extendiendo sus características y comportamientos. Se utiliza en diversos escenarios y contextos, y aquí te proporciono algunas situaciones comunes en las que podrías optar por utilizar la herencia en el diseño de clases:

- Cuando Existe una Relación de "Es Un": La herencia es especialmente útil cuando existe una relación natural de "es un" entre las clases. Por ejemplo, si estás modelando vehículos, puedes tener una clase base "Vehiculo" y clases derivadas como "Coche", "Camión" y "Motocicleta".

- Para Promover la Reutilización del Código: La herencia permite la reutilización de código al heredar características y comportamientos de una clase base. Si varias clases comparten funcionalidades comunes, puedes colocar esas funcionalidades en una clase base y derivar otras clases de ella.

- Para Establecer una Jerarquía de Clases: La herencia facilita la organización de clases en una jerarquía. Puedes tener una clase base general y clases derivadas más específicas, creando una estructura jerárquica que refleje la relación entre los diferentes tipos de objetos en tu sistema.

- Cuando Necesitas Polimorfismo: La herencia permite el uso del polimorfismo, que es la

capacidad de tratar objetos de clases derivadas como objetos de la clase base. Esto es útil cuando necesitas flexibilidad en el manejo de diferentes tipos de objetos a través de una interfaz común.

5. **¿Por qué es importante la encapsulación en programación orientada a objetos?**

- Ocultamiento de Detalles Internos:

  Beneficio: La encapsulación permite ocultar los detalles internos de la implementación de una clase, exponiendo solo la interfaz pública necesaria para interactuar con ella.
  Ventaja: Al ocultar los detalles de implementación, se reduce la complejidad y se simplifica el uso de las clases. Los usuarios de la clase pueden centrarse en cómo interactuar con ella en lugar de preocuparse por su implementación interna.

- Control de Acceso a Datos:

  Beneficio: La encapsulación permite controlar el acceso a los datos de una clase.
  Ventaja: Al definir propiedades privadas y proporcionar métodos públicos para acceder y modificar esos datos, puedes aplicar lógica adicional (validación, cálculos, etc.) y garantizar un acceso controlado y seguro a los datos internos.

- Facilita el Mantenimiento:

  Beneficio: Los cambios internos a una clase (modificaciones en la implementación) no afectan directamente a los usuarios externos de la clase.
  Ventaja: Esto facilita el mantenimiento del código porque puedes realizar cambios en la implementación sin afectar a otras partes del programa que dependen de la clase. La encapsulación reduce las dependencias y aumenta la independencia entre módulos.

- Mejora la Modularidad:

  Beneficio: La encapsulación contribuye a la modularidad al dividir el código en unidades más pequeñas e independientes.
  Ventaja: Las clases encapsuladas actúan como módulos autónomos, y puedes cambiar la implementación de una clase sin afectar a otras partes del sistema. Esto mejora la escalabilidad y facilita la colaboración entre diferentes partes del código.

Repositorio:
https://github.com/salazarmatthew/MatthewSalazar_PruebaP1_WebAvanzado