UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA CAMPUS JUTIAPA, JUTIAPA

FACULTAD DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

CURSO: Emprendedores de Negocios

DOCENTE: Ingeniero Ruldin Efrain Ayala Ramos

Alumnos:

Nery Daniel Vásquez y Vásquez -0905-23-15654

Cristian Oliver Baten Itzep 0905-23-302

Odvin Alexis Méndez Lemus 0905-23-20562

Jostyn Manrrique Godoy Chinchilla 0905-23-2991

Leyvi Lorena Revolorio Revolorio

1. **Tareas Repetitivas y Encapsulamiento**

- El código ya está bastante bien modularizado usando la clase GestorEstudiantes, que encapsula las principales operaciones:
    - AgregarEstudiante()
    - MostrarEstudiantes()
    - CalcularPromedio()
    - MostrarMejorEstudiante()

Sin embargo, podríamos mejorar algunos aspectos:

- La validación de entrada podría extraerse a una función separada, ya que se usa en varios lugares:

csharp

Copy

```
private bool ValidarCalificacion(double calificacion)
{
   return calificacion >= 0 && calificacion <= 100;
}
```

2. **Variables Locales vs Globales**

- La única variable "global" (a nivel de clase) es el Dictionary<string, double> estudiantes, que está correctamente implementada como un campo privado de la clase GestorEstudiantes.
- Las demás variables (nombre, calificacion, opcion) son correctamente locales ya que solo se necesitan en sus respectivos métodos.

3. **Modularización Actual y Posibles Mejoras** El programa ya está modularizado, pero podríamos mejorar:

csharp

Copy

```
class GestorEstudiantes
{
```

```
    private Dictionary<string, double> estudiantes = new Dictionary<string, double>();

    // Nuevo método para validación de entrada
    private bool ValidarEntradaCalificacion(string input, out double calificacion)
    {
        return double.TryParse(input, out calificacion) &&
ValidarCalificacion(calificacion);
    }

    // Nuevo método para mostrar mensajes de error
    private void MostrarError(string mensaje)
    {
        Console.WriteLine($"Error: {mensaje}");
    }

    // Método para verificar si hay estudiantes
    private bool VerificarExistenciaEstudiantes()
    {
        if (estudiantes.Count == 0)
        {
            MostrarError("No hay estudiantes registrados.");
            return false;
        }
        return true;
    }
}
```

4. **Ventajas de la Modularización**

- **Mantenibilidad**: Cada función tiene una responsabilidad única
- **Reutilización**: Las funciones pueden ser llamadas desde diferentes partes del código
- **Pruebas**: Es más fácil probar funciones pequeñas y específicas
- **Legibilidad**: El código es más fácil de entender cuando está dividido en funciones con nombres descriptivos

5. **Importancia de Limitar Variables Globales** En este código:

- El Dictionary estudiantes está correctamente encapsulado como campo privado
- Evita problemas comunes de variables globales como:
    - Modificaciones no controladas
    - Dificultad para rastrear cambios
    - Dependencias ocultas

6. **Mejoras de Legibilidad** Podríamos mejorar el código así:

csharp

Copy

```
class GestorEstudiantes
{
  private const double CALIFICACION_MINIMA = 0;
  private const double CALIFICACION_MAXIMA = 100;
  private readonly Dictionary<string, double> _estudiantes;

  public GestorEstudiantes()
  {
    _estudiantes = new Dictionary<string, double>();
  }
```

```
    // Resto del código con nombres más descriptivos
    public void AgregarNuevoEstudiante()
    public void MostrarListaCompleta()
    public void CalcularPromedioGeneral()
    public void MostrarEstudianteDestacado()
}
```

7. **Sugerencias Adicionales**

- Agregar manejo de excepciones
- Implementar interfaces para mayor flexibilidad
- Agregar documentación XML para los métodos
- Considerar usar propiedades en lugar de campos directos
- Implementar logging para debug

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program
{
   static void Main()
   {
      var gestor = new GestorEstudiantes();
      Console.WriteLine("Bienvenido al sistema de gestión de estudiantes.");

      while (true)
      {
         Console.WriteLine("\n1. Agregar estudiante");
         Console.WriteLine("2. Mostrar lista de estudiantes");
         Console.WriteLine("3. Calcular promedio de calificaciones");
         Console.WriteLine("4. Mostrar estudiante con la calificación más alta");
         Console.WriteLine("5. Salir");
         Console.Write("Seleccione una opción: ");

         if (!int.TryParse(Console.ReadLine(), out int opcion))
         {
            Console.WriteLine("Por favor, ingrese un número válido.");
            continue;
         }
```

```
        switch (opcion)
        {
            case 1:
                gestor.AgregarEstudiante();
                break;
            case 2:
                gestor.MostrarEstudiantes();
                break;
            case 3:
                gestor.CalcularPromedio();
                break;
            case 4:
                gestor.MostrarMejorEstudiante();
                break;
            case 5:
                Console.WriteLine("Saliendo del sistema...");
                return;
            default:
                Console.WriteLine("Opción no válida. Intente de nuevo.");
                break;
        }
    }
  }
}

class GestorEstudiantes
```

```
{
    private Dictionary<string, double> estudiantes = new Dictionary<string, double>();

    public void AgregarEstudiante()
    {
        Console.Write("Ingrese el nombre del estudiante: ");
        string nombre = Console.ReadLine();

        if (estudiantes.ContainsKey(nombre))
        {
            Console.WriteLine("El estudiante ya está registrado.");
            return;
        }

        Console.Write("Ingrese la calificación del estudiante: ");
        if (!double.TryParse(Console.ReadLine(), out double calificacion) || calificacion < 0
|| calificacion > 100)
        {
            Console.WriteLine("Por favor, ingrese una calificación válida (0-100).");
            return;
        }

        estudiantes[nombre] = calificacion;
        Console.WriteLine("Estudiante agregado correctamente.");
    }
```

```
public void MostrarEstudiantes()
{
    if (estudiantes.Count == 0)
    {
        Console.WriteLine("No hay estudiantes registrados.");
        return;
    }

    Console.WriteLine("\nLista de estudiantes:");
    foreach (var estudiante in estudiantes)
    {
        Console.WriteLine($"{estudiante.Key} - Calificación: {estudiante.Value}");
    }
}

public void CalcularPromedio()
{
    if (estudiantes.Count == 0)
    {
        Console.WriteLine("No hay calificaciones registradas.");
        return;
    }

    double promedio = estudiantes.Values.Average();
    Console.WriteLine($"El promedio de calificaciones es: {promedio:F2}");
}
```

```
    public void MostrarMejorEstudiante()
    {
        if (estudiantes.Count == 0)
        {
            Console.WriteLine("No hay calificaciones registradas.");
            return;
        }

        var mejorEstudiante = estudiantes.OrderByDescending(e => e.Value).First();
        Console.WriteLine($"El estudiante con la calificación más alta es: 
{mejorEstudiante.Key} con {mejorEstudiante.Value}");
    }
}
```