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