Gestión de Estudiantes

1. Introducción

El presente proyecto corresponde al desarrollo de un sistema de gestión de estudiantes utilizando C# con Windows Forms, aplicando los principios de Programación Orientada a Objetos (POO) y arquitectura modular para mantener la separación de responsabilidades.

El sistema permite:

- Registrar estudiantes.
- Editar información.
- Diferenciar estudiantes de pregrado y posgrado.
- Listar registros en pantalla.
- Buscar estudiantes.
- Eliminar registros.

La solución fue diseñada con enfoque en:

- Encapsulamiento
- Herencia
- Polimorfismo
- Abstracción
- Factory Pattern (Patrón Fábrica)
- Repositorio en memoria

2. Objetivo General

Desarrollar una aplicación de escritorio que gestione información académica de estudiantes mediante formularios y componentes visuales.

3. Objetivos Específicos

✓ Aplicar POO en clases modelo.
✓ Implementar herencia para categorías de estudiantes.
✓ Usar patrones de diseño para creación de entidades.
✓ Manipular datos utilizando repositorios.
✓ Construir formularios intuitivos.
✓ Aplicar validaciones básicas.

4. Arquitectura del Proyecto

El proyecto usa una estructura separada por capas

SGEstudiantes

```
├─ Models
│   ├─ Student.cs
│   ├─ Undergrad.cs
│   ├─ Postgrad.cs
├─ Factories
│   └─ StudentFactory.cs
├─ Repositories
│   ├─ IStudentRepository.cs
│   └─ StudentRepository.cs
├─ Forms
│   ├─ MainForm.cs
│   ├─ FormStudents.cs
│   └─ FormStudentDetails.cs
```

5. Diseño Orientado a Objetos

5.1 Clase base Student

```
public abstract class Student
{
   public int StudentId { get; set; }
   public string Nombre { get; set; }
   public string Identificacion { get; set; }
   public string Carrera { get; set; }
   public DateTime FechaIngreso { get; set; }
   public abstract string Tipo { get; }
}
```

5.2 Estudiante de pregrado

```
public class Undergrad : Student
{
   public int Semestre { get; set; }
   public override string Tipo => "Pregrado";
}
```

5.3 Estudiante de posgrado

```
public class Postgrad : Student
{
   public string Programa { get; set; }
   public override string Tipo => "Posgrado";
}
```

6. Patrón Factory

```
public class StudentFactory
{
   public Student CreateStudent(string tipo)
   {
      return tipo.ToLower() switch
      {
         "pregrado" => new Undergrad(),
         "posgrado" => new Postgrad(),
         _ => null
      };
   }
}
```

7. Repositorio (CRUD en memoria)

```
public class StudentRepository : IStudentRepository
{
   private readonly List<Student> _data = new();
   private int _nextId = 1;
```

```
public StudentRepository()
{
    SeedSampleData();
}

public List<Student> GetAll() => _data.ToList();

public Student GetById(int id) => _data.FirstOrDefault(s => s.StudentId == id);

public int Add(Student s)
{
    s.StudentId = _nextId++;
    _data.Add(s);
    return s.StudentId;
}

public bool Delete(int id)
{
    var s = GetById(id);
    return s != null && _data.Remove(s);
}

public bool Update(Student s)
{
    var e = GetById(s.StudentId);
    if (e == null) return false;

    e.Nombre = s.Nombre;
    e.Identificacion = s.Identificacion;
    e.Carrera = s.Carrera;
    e.FechaIngreso = s.FechaIngreso;

    if (e is Undergrad ug && s is Undergrad u2)
        ug.Semestre = u2.Semestre;

    if (e is Postgrad pg && s is Postgrad p2)
        pg.Programa = p2.Programa;

    return true;
}

public void SeedSampleData()
{
```

```
        Add(new Undergrad { Nombre="Laura", Identificacion="112", Carrera="Sistemas",
FechaIngreso=DateTime.Now, Semestre=4 });
        Add(new Undergrad { Nombre="Pablo", Identificacion="113", Carrera="Economía",
FechaIngreso=DateTime.Now, Semestre=6 });
        Add(new Postgrad { Nombre="María", Identificacion="114", Carrera="MBA",
FechaIngreso=DateTime.Now, Programa="Gerencia" });
    }
}
```

8. Pantallas del Sistema

8.1 MainForm

- Botón para abrir el módulo de estudiantes.

Código:

```
private void btnGestionEstudiantes_Click(object sender, EventArgs e)
{
    var frm = new FormStudents();
    frm.ShowDialog();
}
```

8.2 Formulario de Lista — FormStudents

Funciones:
✓ Visualizar tabla
✓ Buscar
✓ Crear
✓ Editar
✓ Eliminar

```
private void LoadStudents()
{
    dgvStudents.DataSource = _repo.GetAll()
        .Select(s => new
        {
            s.StudentId,
            s.Nombre,
```

```
            s.Identificacion,
            s.Carrera,
            Tipo = s.Tipo,
            Extra = s is Undergrad ug ? $"Sem: {ug.Semestre}" :
                  s is Postgrad pg ? $"Prog: {pg.Programa}" : ""
        }).ToList();
}
```

8.3 Formulario de Detalles — FormStudentDetails

- Combobox cboTipo
- TextBox txtNombre
- TextBox txtIdentificacion
- TextBox txtCarrera
- DatePicker dtpFecha
- TextBox txtExtra

9. Validaciones

✓ Campos requeridos
✓ Identificación no vacía
✓ Semestre > 0
✓ Programa obligatorio en posgrado

10. Conclusiones

El desarrollo permitió aplicar correctamente los principios de programación orientada a objetos y organizar la solución con capas separadas.
El patrón Factory simplifica la creación de entidades.
El repositorio desacopla los formularios de los datos.
El programa puede evolucionar fácilmente migrando a bases de datos reales.