Preyecto Bimestral - POO

Integrantes

- JUAN TACURI
- ANTHONY VICENTE

EQUIPO DE TRABAJO

Sistema de gestión de admisiones en UTPL

Introduccion

La Universidad Técnica Particular de Loja (UTPL) ofrece 28 carreras presenciales para el período octubre 2024–febrero 2025, lo que implica un proceso complejo de inscripciones, exámenes y asignación de cupos. Para apoyar la toma de decisiones y optimizar la gestión, se desarrolla un prototipo de sistema de admisiones en Java que simula todo el flujo: registro de postulantes, aplicación de exámenes de admisión o diagnóstico, cálculo de puntos de mérito y asignación de plazas según puntajes y criterios adicionales. Este proyecto académico permite validar la lógica de negocio, garantizar la correcta priorización de aspirantes y ofrecer reportes estadísticos antes de implementar un sistema real.

Sistema de gestión de admisiones en UTPL

OBJETIVO

- Implementar un modelo de dominio (UML) que refleje postulantes, carreras, exámenes y resultados.
- Desarrollar la aplicación en Java, respetando MVC y usando serialización (.dat).
- Permitir registrar postulantes, asignar exámenes (admisión/diagnóstico), calcular méritos y asignar cupos.
- Generar reportes de estadísticas sobre ocupación de cupos y rechazados.

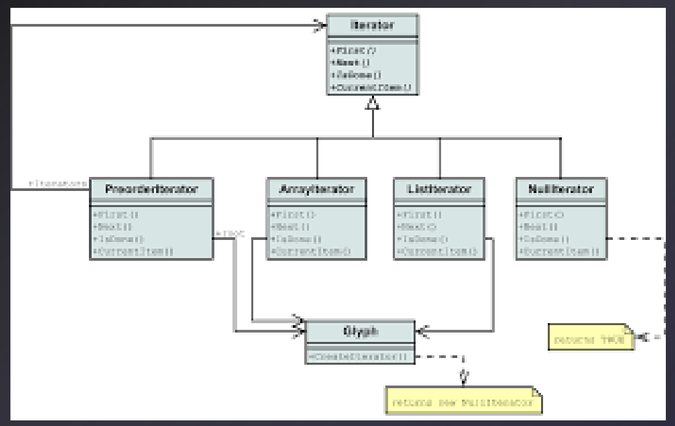


DIAGRAMA UML

Un diagrama UML es una forma de visualizar sistemas y software utilizando el Lenguaje Unificado de Modelado (UML). Los ingenieros de software crean diagramas UML online para comprender los diseños, la arquitectura del código y la implementación propuesta de sistemas de software complejos.

DIA-UML-MVC

Nombres de clases

M:
- Postulante
- Carrera
- Examen
- ExamenAdmision
- ExamenDiagnostico
- EstadoPostulacion
- Merito
- TipoExamen

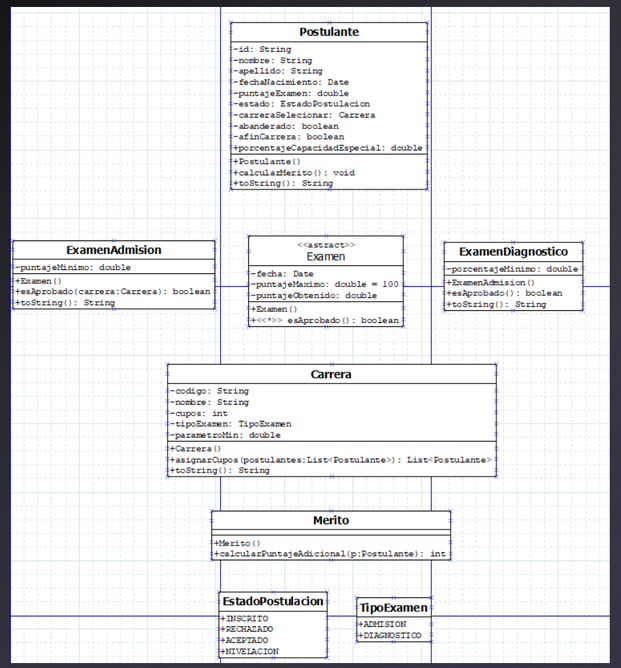
C:
- AdmisionController

V:
- Main

- M (Modelo): Entidades de dominio y lógica de negocio (Postulante, Carrera, Examen, etc.).
- C (Controlador): Orquesta la aplicación y llama al modelo (AdmisionController).
- V (Vista): Punto de entrada y manejo de consola (MainApp).

UML
MODELADO

DIAGRAMA

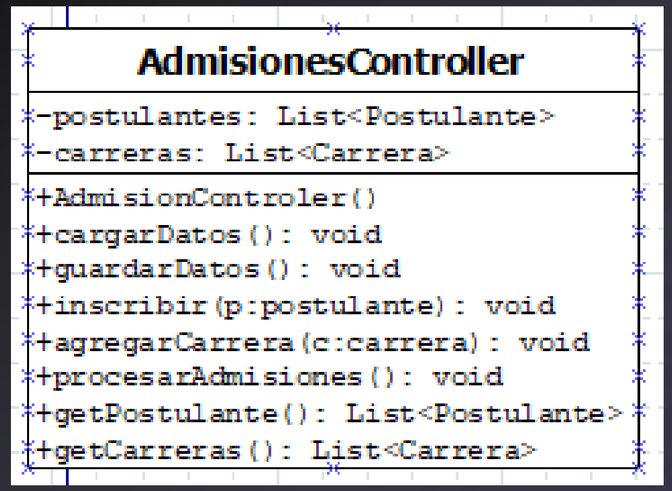


M:

- Postulante
- Carrera
- Examen
- ExamenAdmision
- ExamenDiagnostico
- EstadoPostulacion
- Merito
- TipoExamen

UML
MODELADO

DIAGRAMA

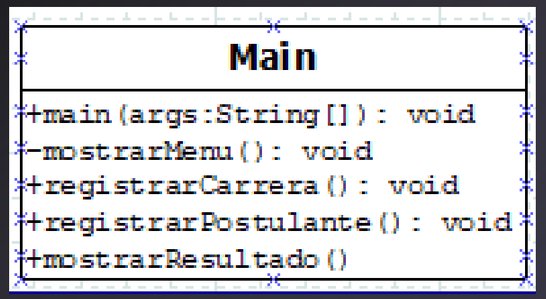


C:

- AdmisionController

UML
MODELADO

DIAGRAMA



V:

- Main

UML
MODELADO

DIAGRAMA

1. Usuario (consola)
2. ↓
3. MainApp (V)
4. ↓ Llama a
5. AdmisionController (C)
6. ↓ Usa / modifica
7. Clases de M
8. ↓ Puede invocar
9. GestorDatos (M)
10. ↔ Archivos .dat

- El usuario ingresa datos desde consola → Ejecutor.
- Ejecutor invoca a AdmisionController para procesar la acción solicitada.
- AdmisionController lee/actualiza objetos de las clases MC.
- Si es necesario, llama a GestorDatos para guardar/cargar en discos (.dat).
- Luego, Ejecutor recibe la respuesta (por ejemplo, lista de estadísticas) y la muestra.

UML
MODELADO

CODIGO JAVA

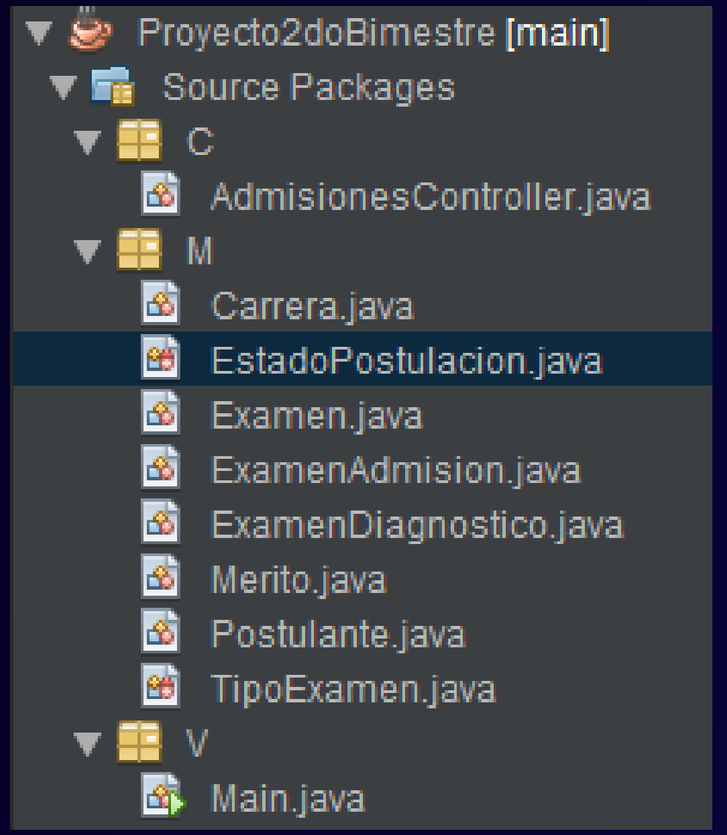
Java es un lenguaje de programación de alto nivel, versátil y ampliamente utilizado, especialmente en el desarrollo de aplicaciones empresariales y web, así como en la creación de software de escritorio y dispositivos móviles. Es conocido por su carácter multiplataforma, orientado a objetos y centrado en la red

.JAVA-Serializacion-
CODIGO- SOLID

JAVA CODIGO

- M (Modelo)
- • Clases: Postulante, Carrera, Examen (+ subclases), CriterioMerito, Inscripcion, Resultado, GestorDatos.
- C (Controlador)
- • Clase: AdmisionController (gestiona lógica de admisiones).
- V (Vista/Ejecutor)
- • Clase: MainApp (maneja entrada/salida por consola y despliega menú).

Estructura del Proyecto Java



M – Clase Postulante

```java
package M;
import java.io.Serializable;
import java.util.Date;

public class Postulante implements Serializable {
    private String id;
    private String nombre;
    private String apellido;
    private Date fechaNacimiento;
    private double puntajeExamen;
    private EstadoPostulacion estado;
    private Carrera carreraSeleccionada;
    private boolean abanderado;
    private boolean afinCarrera;
    private double porcentajeCapacidadEspecial;

    public Postulante(String id, String nombre, String apellido, Date fechaNacimiento,
                      double puntajeExamen, Carrera carreraSeleccionada,
                      boolean abanderado, boolean afinCarrera, double porcentajeCapacidadEspecial) {
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
        this.fechaNacimiento = fechaNacimiento;
        this.puntajeExamen = puntajeExamen;
        this.estado = EstadoPostulacion.INSCRITO;
        this.carreraSeleccionada = carreraSeleccionada;
        this.abanderado = abanderado;
        this.afinCarrera = afinCarrera;
        this.porcentajeCapacidadEspecial = porcentajeCapacidadEspecial;
    }

    public double calcularMerito() {
        Merito m = new Merito();
        return m.calcularPuntajeAdicional(this);
    }

    // Getters
    public String getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public Date getFechaNacimiento() {
        return fechaNacimiento;
    }

    public double getPuntajeExamen() {
        return puntajeExamen;
    }

    public EstadoPostulacion getEstado() {
        return estado;
    }

    public Carrera getCarreraSeleccionada() {
        return carreraSeleccionada;
    }

    public boolean isAbanderado() {
        return abanderado;
    }

    public boolean isAfinCarrera() {
        return afinCarrera;
    }
```

- Implementa Serializable para guardarse en postulantes.dat.
- Atributos clave:
  - cedula, nombre, apellido, fechaNacimiento → datos personales.
  - carreraPostulada → referencia a la carrera elegida.
  - puntajeExamen y puntajeAdicional → puntajes de admisión/diagnóstico y mérito.
  - Banderas de mérito: esAbanderado, afinidadBachillerato, porcentajeCapacidadEspecial.
  - tipoExamen y aprobado → para saber si rindió y pasó su examen.
- Constructores:
- Vació (para deserialización y creación genérica).
- Completo (inicializa todos los atributos de un postulante).
-
-

M – Clase Carrera

```java
package M;
import java.io.Serializable;
import java.util.List;
import java.util.stream.Collectors;

public class Carrera implements Serializable {
    private String codigo;
    private String nombre;
    private int cupos;
    private TipoExamen tipoExamen;
    private double parametroMin;

    public Carrera(String codigo, String nombre, int cupos, TipoExamen tipoExamen, double parametroMin) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.cupos = cupos;
        this.tipoExamen = tipoExamen;
        this.parametroMin = parametroMin;
    }

    public List<Postulante> asignarCupos(List<Postulante> postulantes) {
        List<Postulante> seleccionados;
        if (tipoExamen == TipoExamen.DIAGNOSTICO) {
            seleccionados = postulantes.stream()
                    .filter(p -> p.getCarreraSeleccionada().equals(this))
                    .limit(cupos)
                    .collect(Collectors.toList());
        } else {
            seleccionados = postulantes.stream()
                    .filter(p -> p.getCarreraSeleccionada().equals(this))
                    .filter(p -> p.getPuntajeExamen() + p.calcularMerito() >= parametroMin)
                    .sorted((a, b) -> Double.compare(b.getPuntajeExamen() + b.calcularMerito(),
                            a.getPuntajeExamen() + a.calcularMerito()))
                    .limit(cupos)
                    .collect(Collectors.toList());
        }
        return seleccionados;
    }

    public String getNombre() { return nombre; }
    public TipoExamen getTipoExamen() { return tipoExamen; }
    public double getParametroMin() { return parametroMin; }

    @Override
    public String toString() {
        return String.format("Carrera[codigo=%s, nombre=%s, cupos=%d, tipo=%s, min=%.2f]",
                codigo, nombre, cupos, tipoExamen, parametroMin);
    }
}
```

- Implementa Serializable para guardarse en carreras.dat.
- Atributos clave:
  - idCarrera, nombre → identificador y nombre legible de la carrera.
  - cuposRegulares → número de plazas disponibles.
  - puntajeMinimoAdmi → puntaje mínimo requerido en el examen de admisión.
  - umbralDiagnostico → porcentaje mínimo en el examen diagnóstico para evitar nivelación.
  - requiereDiagnostico → si esta carrera usa examen diagnóstico en lugar de puntaje mínimo.
  - postulantes → lista de Postulante inscrito a esta carrera.
- Constructores:
  - Vacío: inicializa valores por defecto y lista vacía de postulantes.
  - Completo: recibe todos los parámetros, incluida la lista inicial (puede partir vacía).
- Métodos principales:
  - agregarPostulante(Postulante p): añade un aspirante a la lista interna.
  - getCantidadPostulantes(): devuelve el tamaño de la lista de postulantes.
  - toString(): muestra, en una línea, todos los datos esenciales (ID, nombre, cupos, puntajes y cantidad de inscritos).

MC – Examen y Subclases

```java
public abstract class Examen implements Serializable {
    private String idExamen;
    private LocalDate fechaExamen;
    private double puntajeObtenido;

    public Examen() { /*…*/ }
    public Examen(String idExamen, LocalDate fechaExamen, double puntajeObtenido) {
        this.idExamen = idExamen;
        this.fechaExamen = fechaExamen;
        this.puntajeObtenido = puntajeObtenido;
    }

    public String getIdExamen() { return idExamen; }
    public void setIdExamen(String idExamen) { this.idExamen = idExamen; }
    public LocalDate getFechaExamen() { return fechaExamen; }
    public void setFechaExamen(LocalDate fechaExamen) { this.fechaExamen = fechaExamen; }
    public double getPuntajeObtenido() { return puntajeObtenido; }
    public void setPuntajeObtenido(double puntajeObtenido) { this.puntajeObtenido = puntajeObtenido; }

    // Método abstracto que cada subclase implementa
    public abstract boolean calcularAprobado(double requisito);

    @Override
    public String toString() { /*…*/ }
}
```

```java
public class ExamenAdmision extends Examen {
    private double puntajeMinimo;

    public ExamenAdmision() { super(); this.puntajeMinimo = 0.0; }
    public ExamenAdmision(String idExamen, LocalDate fechaExamen,
                          double puntajeObtenido, double puntajeMinimo) {
        super(idExamen, fechaExamen, puntajeObtenido);
        this.puntajeMinimo = puntajeMinimo;
    }

    public double getPuntajeMinimo() { return puntajeMinimo; }
    public void setPuntajeMinimo(double puntajeMinimo) { this.puntajeMinimo = puntajeMinimo; }

    @Override
    public boolean calcularAprobado(double requisito) {
        // Para admisión se compara con puntajeMinimo interno
        return getPuntajeObtenido() >= puntajeMinimo;
    }

    @Override
    public String toString() { /*…*/ }
}
```

```java
public class ExamenDiagnostico extends Examen {
    private double umbralNivelacion;

    public ExamenDiagnostico() { super(); this.umbralNivelacion = 0.0; }
    public ExamenDiagnostico(String idExamen, LocalDate fechaExamen,
                             double puntajeObtenido, double umbralNivelacion) {
        super(idExamen, fechaExamen, puntajeObtenido);
        this.umbralNivelacion = umbralNivelacion;
    }

    public double getUmbralNivelacion() { return umbralNivelacion; }
    public void setUmbralNivelacion(double umbralNivelacion) { this.umbralNivelacion = umbralNivelacion; }

    @Override
    public boolean calcularAprobado(double requisito) {
        // Para diagnóstico se compara con umbralNivelacion interno
        return getPuntajeObtenido() >= umbralNivelacion;
    }

    @Override
    public String toString() { /*…*/ }
}
```

Examen (abstracta):

- Atributos: idExamen (String), fechaExamen (LocalDate), puntajeObtenido (double).
- Incluye constructor vacío y completo.
- Define el método abstracto calcularAprobado(double requisito) que las subclases deben implementar.
- Implementa toString() para mostrar datos en consola.

ExamenAdmision:

- Hereda de Examen.
- Atributo adicional: puntajeMinimo (double).
- En calcularAprobado(…), compara puntajeObtenido >= puntajeMinimo.
- El parámetro requisito del método se ignora porque usa su propio puntajeMinimo.

ExamenDiagnostico:

- Hereda de Examen.
- Atributo adicional: umbralNivelacion (double).
- En calcularAprobado(…), compara puntajeObtenido >= umbralNivelacion.
- Marca “aprobado” si alcanza el umbral; de lo contrario, el postulante requiere nivelación.

```java
package C;
import M.Carrera;
import M.Postulante;
import M.EstadoPostulacion;
import java.util.*;
import java.io.*;

public class AdmisionesController {
    private List<Postulante> postulantes;
    private List<Carrera> carreras;

    public AdmisionesController() {
        postulantes = new ArrayList<>();
        carreras = new ArrayList<>();
    }

    @SuppressWarnings("unchecked")
    public void cargarDatos() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("postulantes.dat"))) {
            postulantes = (List<Postulante>) ois.readObject();
        } catch (Exception e) {}
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("carreras.dat"))) {
            carreras = (List<Carrera>) ois.readObject();
        } catch (Exception e) {}
    }

    public void guardarDatos() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("postulantes.dat"))) {
            oos.writeObject(postulantes);
        } catch (IOException e) { e.printStackTrace(); }
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("carreras.dat"))) {
            oos.writeObject(carreras);
        } catch (IOException e) { e.printStackTrace(); }
    }

    public void inscribir(Postulante p) {
        postulantes.add(p);
    }

    public void agregarCarrera(Carrera c) {
        carreras.add(c);
    }

    public void procesarAdmisiones() {
        for (Carrera c : carreras) {
            List<Postulante> admitidos = c.asignarCupos(postulantes);
            for (Postulante p : postulantes) {
                if (p.getCarreraSeleccionada().equals(c)) {
                    if (admitidos.contains(p)) p.setEstado(EstadoPostulacion.ACEPTADO);
                    else p.setEstado(EstadoPostulacion.RECHAZADO);
                }
            }
        }
    }

    public List<Postulante> getPostulantes() {
        return postulantes;
    }

    public List<Carrera> getCarreras() {
        return carreras;
    }
}
```

C - Clase AdmisionController

- Responsabilidad principal: orquestar la lógica de negocio y coordinar modelo y persistencia.
- Atributos:
  - listaCarreras, listaPostulantes, listaInscripciones, listaResultados (todas ArrayList<>).
  - gestor (instancia de GestorDatos).
- Métodos clave:
  - cargarDatosIniciales():
  - guardarDatos():
  - registrarPostulante(Postulante p):
  - inscribirPostulanteACarrera(String cedulaP, String idCarrera, LocalDate fecha):
  - asignarExamenAInscripcion(String idInscripcion, Examen examen):
  - procesarAdmisionesPorCarrera(String idCarrera):
  - generarEstadisticas():

```java
package V;
import C.AdmisionesController;
import M.Postulante;
import M.Carrera;
import M.TipoExamen;

import java.text.SimpleDateFormat;
import java.util.*;

public class Main {
    private static final Scanner scanner = new Scanner(System.in);
    private static final AdmisionesController ctrl = new AdmisionesController();
    private static final SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");

    public static void main(String[] args) {
        ctrl.cargarDatos();
        boolean exit = false;
        while (!exit) {
            mostrarMenu();
            int op = Integer.parseInt(scanner.nextLine());
            switch (op) {
                case 1: registrarCarrera(); break;
                case 2: registrarPostulante(); break;
                case 3:
                    ctrl.procesarAdmisiones();
                    System.out.println("Admisiones procesadas.");
                    break;
                case 4: mostrarResultados(); break;
                case 0:
                    exit = true;
                    ctrl.guardarDatos();
                    break;
                default: System.out.println("Opción inválida.");
            }
        }
        scanner.close();
    }

    private static void mostrarMenu() {
        System.out.println("\n=== Gestor de Admisiones UTPL ===");
        System.out.println("=== BY JUAN TACURI & ANTHONY VICENTE ===");
        System.out.println("1. Registrar carrera");
        System.out.println("2. Inscribir postulante");
        System.out.println("3. Procesar admisiones");
        System.out.println("4. Mostrar resultados");
        System.out.println("0. Salir");
        System.out.print("Seleccione opción: ");
    }

    private static void registrarCarrera() {
        try {
            System.out.print("Código Carrera: "); String codigo = scanner.nextLine();
            System.out.print("Nombre Carrera: "); String nombre = scanner.nextLine();
            System.out.print("Cupos: "); int cupos = Integer.parseInt(scanner.nextLine());
            System.out.print("Tipo de Examen (ADMISION/DIAGNOSTICO): ");
            TipoExamen tipo = TipoExamen.valueOf(scanner.nextLine().toUpperCase());
            System.out.print("Valor mínimo (puntaje o %): "); double param = Double.parseDouble(scanner.nextLine());
            Carrera c = new Carrera(codigo, nombre, cupos, tipo, param);
            ctrl.agregarCarrera(c);
            ctrl.guardarDatos();
            System.out.println("Carrera registrada y guardada en carreras.dat.");
        } catch (Exception e) {
            System.out.println("Error en datos de carrera.");
        }
    }

    private static void registrarPostulante() {
        try {
            List<Carrera> lista = ctrl.getCarreras();
            if (lista.isEmpty()) {
                System.out.println("Debe registrar al menos una carrera primero.");
```

V – Clase MainApp

- Propósito: único punto donde ocurre interacción con el usuario (consola).
- Atributos:
  - AdmisionController sistema → instancia del controlador.
  - Scanner sc → para leer datos del usuario.
  - DateTimeFormatter fmt → formatea LocalDate como “yyyy-MM-dd”.
- main(String[] args):
  a. Crea MainApp.
  b. Llama a sistema.cargarDatosIniciales() para cargar/crear archivos .dat.
  c. Invoca mostrarMenuPrincipal().
- mostrarMenuPrincipal():
- Opciones principales (resumen):

1. Registrar nuevo postulante → opcionRegistrarPostulante().
2. Listar carreras → opcionListarCarreras().
3. Inscribir postulante → opcionInscribirACarrera().
4. Registrar examen → opcionRegistrarResultadoExamen().
5. Procesar admisiones → opcionProcesarAdmisiones().

Mostrar estadísticas → opcionMostrarEstadisticas().

CONSOLA

with
NetBeans

JAVA
CODIGO

```
run:

=== Gestor de Admisiones UTPL ===
=== BY JUAN TACURI & ANTHONY VICENTE ===
1. Registrar carrera
2. Inscribir postulante
3. Procesar admisiones
4. Mostrar resultados
0. Salir
Seleccione opci◆n:
```

Thank You