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.

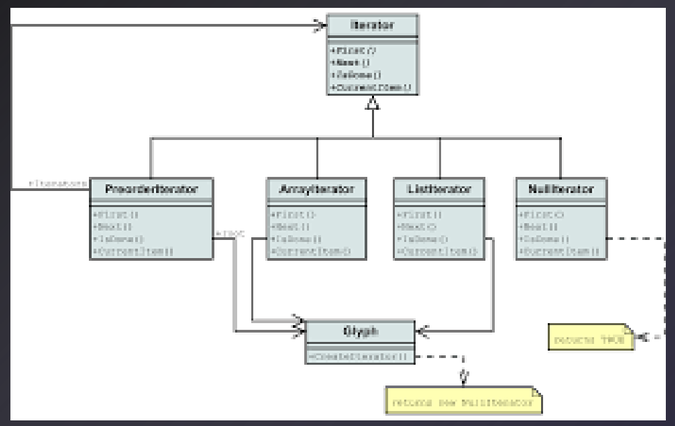


DIAGRAMA UML

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

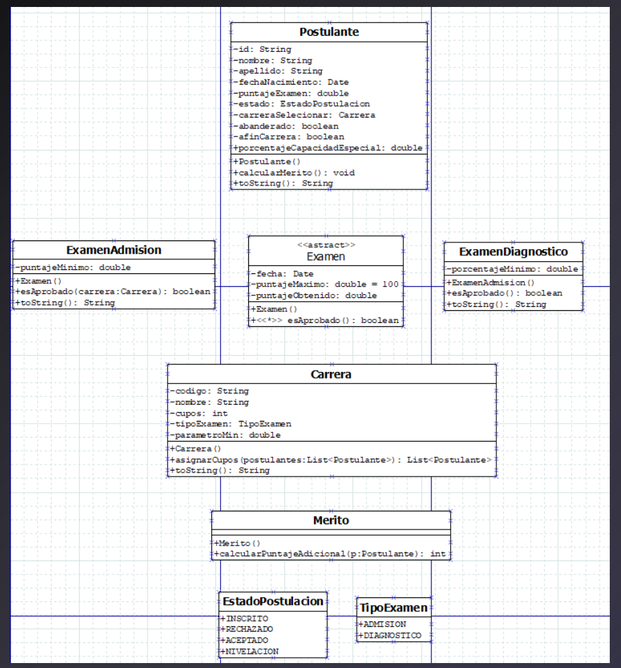
C:
- AdmisionController

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

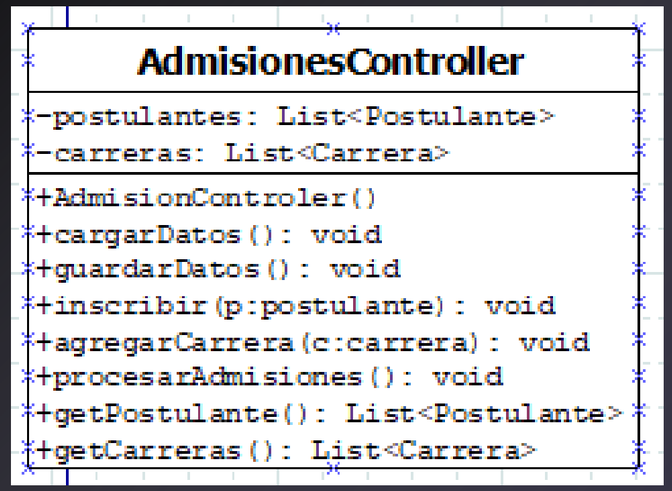


M:

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

UML
MODELADO

DIAGRAMA

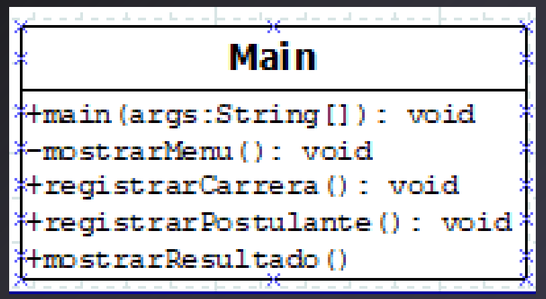


C:

- 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

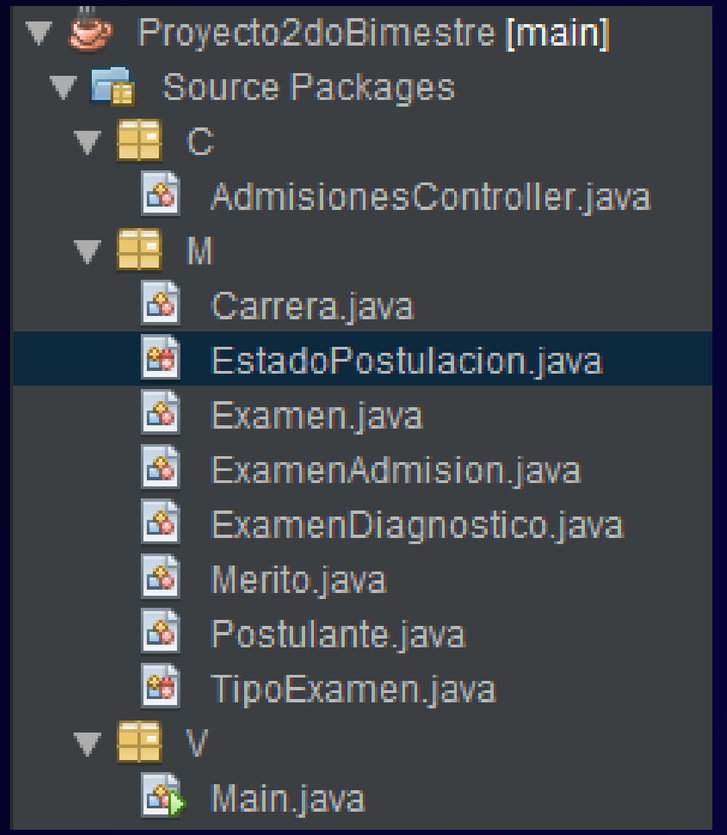
Java es un lenguaje de programación de alto nivel, versátil y ampliamente utilizado, especialmente en el desarrollo de aplicaciones empresariales y web, así como en la creación de software de escritorio y dispositivos móviles. Es conocido por su carácter multiplataforma, orientado a objetos y centrado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

MC – Clase Carrera

```java
public class Carrera implements Serializable {
    // Atributos
    private String idCarrera;
    private String nombre;
    private int cuposRegulares;
    private double puntajeMinimoAdmi;
    private double umbralDiagnostico;
    private boolean requiereDiagnostico;
    private ArrayList<Postulante> postulantes;

    // Constructor vacío
    public Carrera() {
        this.idCarrera = "";
        this.nombre = "";
        this.cuposRegulares = 0;
        this.puntajeMinimoAdmi = 0.0;
        this.umbralDiagnostico = 0.0;
        this.requiereDiagnostico = false;
        this.postulantes = new ArrayList<>();
    }

    // Constructor con todos los atributos
    public Carrera(String idCarrera, String nombre, int cuposRegulares,
                   double puntajeMinimoAdmi, double umbralDiagnostico,
                   boolean requiereDiagnostico, ArrayList<Postulante> postulantes) {
        this.idCarrera = idCarrera;
        this.nombre = nombre;
        this.cuposRegulares = cuposRegulares;
        this.puntajeMinimoAdmi = puntajeMinimoAdmi;
        this.umbralDiagnostico = umbralDiagnostico;
        this.requiereDiagnostico = requiereDiagnostico;
        this.postulantes = postulantes != null ? postulantes : new ArrayList<>();
    }

    // Getters y setters (un ejemplo)
    public String getIdCarrera() { return idCarrera; }
    public void setIdCarrera(String idCarrera) { this.idCarrera = idCarrera; }
    // … otros getters/setters …

    // Método clave: agrega un postulante a la lista
    public void agregarPostulante(Postulante p) {
        if (p != null) {
            postulantes.add(p);
        }
    }

    // Devuelve cuántos postulantes hay inscritos
    public int getCantidadPostulantes() {
        return postulantes.size();
    }

    @Override
    public String toString() {
        return "Carrera{" +
                "idCarrera='" + idCarrera + '\'' +
                ", nombre='" + nombre + '\'' +
                ", cuposRegulares=" + cuposRegulares +
                ", puntajeMinimoAdmi=" + puntajeMinimoAdmi +
                ", umbralDiagnostico=" + umbralDiagnostico +
                ", requiereDiagnostico=" + requiereDiagnostico +
                ", postulantesInscritos=" + postulantes.size() +
                '}';
    }
}
```

- 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
public class Inscripcion implements Serializable {
    // Atributos
    private String idInscripcion;
    private Postulante postulante;
    private Carrera carrera;
    private LocalDate fechaInscripcion;

    // Constructor vacío
    public Inscripcion() {
        this.idInscripcion = "";
        this.postulante = null;
        this.carrera = null;
        this.fechaInscripcion = null;
    }

    // Constructor con todos los atributos
    public Inscripcion(String idInscripcion, Postulante postulante,
                       Carrera carrera, LocalDate fechaInscripcion) {
        this.idInscripcion = idInscripcion;
        this.postulante = postulante;
        this.carrera = carrera;
        this.fechaInscripcion = fechaInscripcion;
    }

    // Getters y setters
    public String getIdInscripcion() { return idInscripcion; }
    public void setIdInscripcion(String idInscripcion) {
        this.idInscripcion = idInscripcion;
    }
    public Postulante getPostulante() { return postulante; }
    public void setPostulante(Postulante postulante) {
        this.postulante = postulante;
    }
    public Carrera getCarrera() { return carrera; }
    public void setCarrera(Carrera carrera) {
        this.carrera = carrera;
    }
    public LocalDate getFechaInscripcion() { return fechaInscripcion; }
    public void setFechaInscripcion(LocalDate fechaInscripcion) {
        this.fechaInscripcion = fechaInscripcion;
    }

    @Override
    public String toString() {
        return "Inscripcion{" +
                "id='" + idInscripcion + '\'' +
                ", postulante=" + (postulante != null ? postulante.getCedula() : "-") +
                ", carrera=" + (carrera != null ? carrera.getIdCarrera() : "-") +
                ", fecha=" + fechaInscripcion +
                '}';
    }
}
```

MC - Clase Inscripcion

- Implementa Serializable para guardarse en inscripciones.dat.
- Atributos clave:
  - idInscripcion → código único de la inscripción.
  - postulante → objeto Postulante que se inscribe.
  - carrera → objeto Carrera a la que se inscribe.
  - fechaInscripcion → LocalDate en que se realizó la inscripción.
- Constructores:
  - Vacío: inicializa valores por defecto (cadenas vacías y null).
  - Completo: recibe id, referencia a postulante, referencia a carrera y fecha.
- Métodos:
  - Getters y setters para cada atributo.
  - toString(): imprime id de inscripción, cédula del postulante, id de carrera y fecha, útil al listar en consola.

```java
public class GestorDatos {
    private String rutaArchivoPostulantes;
    private String rutaArchivoCarreras;
    private String rutaArchivoInscripciones;
    private String rutaArchivoResultados;

    // Constructor vacío (rutas por defecto)
    public GestorDatos() {
        this.rutaArchivoPostulantes = "postulantes.dat";
        this.rutaArchivoCarreras    = "carreras.dat";
        this.rutaArchivoInscripciones = "inscripciones.dat";
        this.rutaArchivoResultados  = "resultados.dat";
    }

    // Constructor completo (rutas personalizadas)
    public GestorDatos(String rutaP, String rutaC, String rutaI, String rutaR) {
        this.rutaArchivoPostulantes  = rutaP;
        this.rutaArchivoCarreras     = rutaC;
        this.rutaArchivoInscripciones = rutaI;
        this.rutaArchivoResultados   = rutaR;
    }

    // Inicializa archivos (si no existen, los crea)
    public void inicializarDatos() {
        try {
            File f1 = new File(rutaArchivoPostulantes);
            if (!f1.exists()) {
                f1.createNewFile();
                guardarPostulantes(new ArrayList<>());
            }
            File f2 = new File(rutaArchivoCarreras);
            if (!f2.exists()) {
                f2.createNewFile();
                guardarCarreras(new ArrayList<>());
            }
            File f3 = new File(rutaArchivoInscripciones);
            if (!f3.exists()) {
                f3.createNewFile();
                guardarInscripciones(new ArrayList<>());
            }
            File f4 = new File(rutaArchivoResultados);
            if (!f4.exists()) {
                f4.createNewFile();
                guardarResultados(new ArrayList<>());
            }
        } catch (IOException e) {
            System.err.println("Error inicializando archivos: " + e.getMessage());
        }
    }

    // Guarda lista de postulantes en postulantes.dat
    public void guardarPostulantes(ArrayList<Postulante> lista) {
        try (ObjectOutputStream oos = new ObjectOutputStream(
                new FileOutputStream(rutaArchivoPostulantes))) {
            oos.writeObject(lista);
        } catch (IOException e) {
            System.err.println("Error al guardar postulantes: " + e.getMessage());
        }
    }

    // Carga lista de postulantes desde postulantes.dat
    @SuppressWarnings("unchecked")
    public ArrayList<Postulante> cargarPostulantes() {
        ArrayList<Postulante> lista = new ArrayList<>();
        File archivo = new File(rutaArchivoPostulantes);
        if (!archivo.exists() || archivo.length() == 0) {
            return lista;
        }
        try (ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream(rutaArchivoPostulantes))) {
            lista = (ArrayList<Postulante>) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
```

MC – Clase GestorDatos

- Su propósito: serializar y deserializar las listas de objetos en archivos .dat.
- Constructores:
  - Vacío: asigna nombres de archivos por defecto (postulantes.dat, carreras.dat, inscripciones.dat, resultados.dat).
  - Completo: permite personalizar rutas (e.g., "data/postulantes.dat").
- Método inicializarDatos():
  - Verifica si cada archivo existe.
  - Si no existe, lo crea y escribe una lista vacía (para evitar EOFException).
- Métodos guardarX y cargarX:
  - guardarX(ArrayList<Clase> lista): abre ObjectOutputStream para sobrescribir el archivo con la lista actual.
  - cargarX(): si el archivo existe y no está vacío, abre ObjectInputStream, lee el ArrayList<Clase> guardado y lo retorna; si está vacío o no existe, devuelve lista vacía.
  - Estos métodos existen para Postulante, Carrera, Inscripcion y Resultado.

```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