Sistema de matriculaciones para Ingeniería Biomédica v2.0

Trabajo Práctico del Módulo 3,4 y 5

Docentes: MSc. Bioing. Baldezzari Lucas e Ing. ALVAREZ Emiliano

Unidad Curricular: Programación Digital Avanzada

Fecha de entrega: 25/04/2025

Ingeniería Biomédica

Instituto Tecnológico Regional Sur-Oeste

Universidad Tecnológica del Uruguay

Índice

1. INTRODUCCIÓN

La gestión académica en instituciones de educación superior requiere herramientas que garanticen eficiencia, organización y cumplimiento de los planes de estudio. Uno de los procesos más sensibles es la inscripción a cursos y exámenes, especialmente cuando estos dependen de requisitos previos establecidos. En el caso de la carrera de Ingeniería Biomédica de la Universidad Tecnológica del Uruguay (UTEC), el sistema actual no verifica automáticamente dichas condiciones, lo que genera errores frecuentes y sobrecarga administrativa.

Con el objetivo de abordar esta problemática, se desarrolló un Sistema de Matriculaciones (SdM) utilizando el lenguaje Python y aplicando principios de Programación Orientada a Objetos (POO). El sistema permite gestionar la trayectoria académica del estudiante, controlando inscripciones, matrículas y aprobaciones, siempre en función de las previas definidas por el plan de estudios.

Este informe detalla el diseño, implementación y funcionamiento del sistema, así como las decisiones técnicas adoptadas durante su desarrollo. Además, se analiza la estructura del código, el uso de diagramas UML para visualizar las relaciones entre clases y la aplicación de buenas prácticas en manejo de archivos y validación de datos.

2. MARCO TEÓRICO

Los procesos administrativos en instituciones de educación superior ha adquirido una importancia creciente en los últimos años, tanto para optimizar la gestión interna como para mejorar la experiencia del estudiante. En particular, la matriculación y la inscripción a exámenes son actividades fundamentales que, en muchos casos, requieren cumplir requisitos previos definidos por el plan de estudio de cada carrera.

En la actualidad, dentro del sistema Universita XXI utilizado por la Universidad Tecnológica del Uruguay (UTEC), los y las estudiantes tienen la posibilidad de matricularse libremente a cualquier unidad curricular (UC) o examen final (EF), sin que el sistema verifique automáticamente si se cumplen las previaturas exigidas. Esta falta de control automatizado traslada la responsabilidad de validación al propio estudiante, en segunda instancia, al personal de secretaría o coordinación.

El resultado es una carga administrativa significativa y una posible pérdida de tiempo tanto para los estudiantes como para los docentes y el personal técnico administrativo. Es frecuente que las correcciones a inscripciones erróneas se realicen varias semanas después de comenzado el curso, lo que afecta el normal desarrollo de la actividad académica.

Frente a esta problemática, surge la necesidad de desarrollar un Sistema de Matriculación (SdM) que contemple automáticamente los requisitos del plan de estudios de la carrera de Ingeniería Biomédica (IBIO). Dicho sistema debe ser capaz de determinar, en base al historial académico del estudiante, si cumple o no con las condiciones necesarias para cursar o presentarse a examen en cada UC. Esto incluye la verificación de unidades curriculares previamente aprobadas o rendidas en modalidad de examen, de modo que se pueda evitar la inscripción indebida desde el inicio y garantizar un proceso más eficiente y justo.

3. OBJETIVOS

3.1 Objetivo General

- Implementar un Sistema de Matriculaciones -básico- a Unidades Curriculares y Exámenes para la carrera de Ingeniería Biomédica (plan 2019) utilizando Python y los conceptos de Programación Orientada a Objetos (POO) vistos hasta el momento.

3.2 Objetivos Específicos

- Reforzar los conceptos de Programación Orientada a Objetos (POO), Herencia, Abstracción, Encapsulamiento.
- Reforzar los conceptos de Clase y Objeto mediante la implementación del sistema de matriculaciones ya que deberá definir clases, objetos e interacciones entre estos.
- Reforzar los conceptos de diagramas UML.
- Implementar conceptos de Manejo de Excepciones y Manejo de Archivos en Python.

4. MATERIALES

- Python 3.9 (o superior).
- Visual Studio (recomendado).
- Un repositorio en Github.com.
- Plan de estudios de la carrera de Ingeniería Biomédica.

5. PROCEDIMIENTO

1. El desarrollo del sistema de matriculación comenzó con la implementación de la clase PlanEstudio, encargada de cargar la estructura del plan de estudios desde un archivo JSON. Esta clase permite acceder a la información curricular organizada por semestre y constituye la base para validar los requisitos académicos de cada unidad curricular.
2. Posteriormente, se definió la clase UC (Unidad Curricular), que representa una materia del plan de estudios. Esta clase accede a los datos específicos de cada unidad, como

nombre, créditos y materias previas, y se utiliza como referencia central en los procesos de inscripción, matrícula y validación.

3. A continuación, se diseñó la clase GestorExpediente, cuya función es crear y gestionar los archivos académicos individuales de cada estudiante. Esta clase permite almacenar datos personales, registrar materias aprobadas y matriculadas, gestionar inscripciones a exámenes y acceder a la lista de inscriptos, todo ello a través de archivos JSON.
4. Con base en lo anterior, se desarrolló la clase Estudiante, que modela al alumno dentro del sistema. A partir del expediente gestionado por el GestorExpediente, esta clase permite realizar operaciones como inscribirse a exámenes, matricularse en unidades curriculares y validar requisitos previos para cada acción académica.
5. Finalmente, se implementó la clase Secretaria, la cual simula el rol administrativo. Esta clase carga estudiantes a partir de sus expedientes y permite ejecutar acciones como la inscripción a exámenes, el registro de materias aprobadas y la consulta de inscriptos, funcionando como una interfaz que centraliza las funciones administrativas.

6. RESULTADOS Y DISCUSIÓN

El sistema desarrollado permite gestionar la inscripción, matrícula y aprobación de las Unidades Curriculares (UCs) de la carrera de Ingeniería Biomédica. Incluye la validación de la aprobación de las previas (otras UCs que deben ser aprobadas previamente), así como el control de créditos académicos y el registro de datos personales y académicos de los estudiantes.

6.1 Clases y Roles del Código

A continuación se describen las clases implementadas en el sistema y los roles que cumple cada una dentro de la estructura del código:

Clase PlanEstudio

- Rol: Cargar el plan de estudio desde un archivo JSON.
- Constructor: def __init__(self, archivo="PlanEstudio2021.json")

  Inicializa el objeto PlanEstudio con el nombre del archivo de PlanEstudio2021. Permite establecer un archivo por defecto o personalizarlo al instanciar la clase.

- Método:
  - def cargarPlanEstudio(self):

    Abre y carga el archivo JSON utilizando la codificación UTF-8 (Unicode Transformation Format - 8 bits), la cual permite representar correctamente caracteres especiales como tildes, “ñ” y símbolos. Esta codificación es ampliamente utilizada para leer y escribir archivos de texto en computadoras. El método devuelve un diccionario que contiene toda la información del plan de estudios, organizada por semestres.

UC (Unidad Curricular)

- Rol: Representa una materia específica del plan.
- Constructor: def __init__(self, plan: PlanEstudio, semestre: str, codigo: str):

  Recibe un objeto PlanEstudio, un semestre (clave del diccionario) y el código de la materia. Carga internamente el diccionario del plan usando cargarPlanEstudio().

- Métodos:
  - def infoUC(self):

    Extrae la información asociada a la UC como nombre, previas y créditos. Devuelve True si lo hace correctamente y False si las credenciales ingresadas son incorrectas.

- def nombreMateria(self):

  Retorna el nombre de la materia si infoUC() fue ejecutado correctamente.

- def nombrePrevias(self):

  Devuelve la lista de previas.

- def getCreditos(self):

  Devuelve la cantidad de créditos de la UC.

Clase GestorExpediente

- Rol: Gestiona los archivos JSON por estudiante. Permite crear, cargar y modificar archivos JSON individuales para cada estudiante, facilitando el almacenamiento y la administración de su información personal y académica
- Constructor: def __init__(self, nombre: str, ci: int, añoIngreso: int):

  Inicializa los datos básicos del estudiante y prepara el nombre del archivo JSON personalizado por estudiante (usa nombre y cédula).

- Métodos:
  - def crearArchivo(self, indentacion: int = 4):

    Crea el archivo JSON con los datos iniciales del estudiante.

  - def cargarDatos(self):

    Carga el JSON del estudiante a memoria.

- def agregarExamenInscripto(self, materia :UC):

  Gestiona la inscripción a exámenes, permitiendo agregar o quitar una Unidad Curricular (UC) de la lista de Inscripción a Examen. Recibe como argumento un objeto UC correspondiente a la materia deseada y devuelve un valor booleano: True si la inscripción se realizó correctamente, o False si la UC ya estaba inscrita.

- def quitarUcExamen(self, materia :UC):

  Permite eliminar una Unidad Curricular (UC) de la lista de inscripción a examen del estudiante. Recibe como parámetro un objeto UC correspondiente a la materia que se desea quitar, y actualiza la lista eliminando dicha inscripción si está presente.

- def agregarUcsAprobadas(self, materia :UC):

  Permite modificar la lista de materias aprobadas del estudiante, agregando una nueva Unidad Curricular (UC) junto con la suma de sus créditos correspondientes. Recibe como parámetro un objeto UC que representa la materia aprobada y devuelve True si la operación se realizó correctamente, o False si la materia ya estaba registrada como aprobada.

- def quitarUcsAprobadas(self, materia :UC):

  Permite eliminar una Unidad Curricular (UC) de la lista de materias aprobadas del estudiante, descontando automáticamente sus créditos del total acumulado. Recibe como parámetro un objeto UC correspondiente a la materia que se desea remover

- def inscriptos(self, materia2: UC):

  Este método permite inscribir a un estudiante en una unidad curricular específica. Para ello, verifica si la materia ya figura en el archivo Registros.json. Si no existe una entrada para esa materia, la crea. Luego, comprueba si el nombre del estudiante ya está registrado en la lista de inscriptos; si no lo está, lo agrega, asegurando así que el registro de inscripciones se mantenga actualizado correctamente.

- def matricular(self, materia :UC):

  Permite inscribir o retirar una Unidad Curricular (UC) del listado de materias matriculadas del estudiante. En el caso de la inscripción, agrega la UC al registro académico si no se encuentra previamente registrada. Recibe como parámetro un objeto UC que representa la materia correspondiente a matricular o desmatricular.

- def desmatricular(self, materia :UC):

  Permite eliminar una Unidad Curricular (UC) de la lista de materias matriculadas del estudiante. Esta función actualiza el registro académico eliminando la materia correspondiente, siempre que se encuentre previamente matriculada. Recibe como parámetro un objeto UC que representa la materia que se desea desmatricular.

- def verListaInscriptos(self, registro = "Registros.json"):

  Este método permite obtener la lista completa de estudiantes inscriptos en cada unidad curricular, accediendo a la información almacenada en un archivo JSON, por defecto `Registros.json`. Para ello, abre el archivo en modo lectura, carga su contenido como un diccionario y lo retorna. Cada clave del diccionario representa una materia, y su valor es la lista de estudiantes inscriptos en ella. Esta funcionalidad resulta útil para visualizar o procesar el estado actual de las inscripciones.

Clase Estudiante

- Rol: Modela al estudiante y su trayectoria.
- Constructor: def __init__(self, usuario: GestorExpediente):

  Toma un objeto GestorExpediente, lo utiliza para cargar los datos desde archivo JSON, y expone esos datos como atributos.

- Métodos:
  - def infoEstudiante(self):

    Devuelve el nombre y la cédula de identidad del estudiante en forma de tupla, lo cual permite identificarlo de manera única dentro del sistema académico.

  - def ucsAprobadas(self):

    Retorna una lista de UCs aprobadas.

  - def inscribirseExamen(self, materia :UC):

    Permite al estudiante inscribirse a un examen correspondiente a una Unidad Curricular (UC), siempre y cuando haya aprobado las materias previas requeridas. Recibe como parámetro un objeto UC que contiene la información de la materia y devuelve un valor booleano: True si se cumplen las condiciones de inscripción, o False en caso contrario.

  - def quitatExamen(self, materia :UC):

    Elimina una inscripción a examen.

- def matricularUC(self, materia :UC):

  Permite inscribir o eliminar una Unidad Curricular (UC) del listado de materias matriculadas del estudiante, según el cumplimiento de las previas requeridas. Recibe como parámetro un objeto UC que representa la materia a cursar y verifica que las materias previas estén aprobadas o en proceso de examen. Devuelve True si la inscripción se realiza correctamente, o False si no se cumplen los requisitos previos.

- def desmatricularUC(self, materia :UC):

  Elimina una Unidad Curricular (UC) del listado de materias matriculadas del estudiante. Recibe como parámetro un objeto UC que representa la materia a desmatricular y actualiza el registro académico eliminando dicha inscripción.

- def _str_(self):

  Devuelve una representación en formato de cadena (string) del estudiante, incluyendo información básica como su nombre y cédula, para facilitar su identificación dentro del sistema

Clase Secretaria

- *Rol:* Simula el rol administrativo de secretaria o coordinadora.
- *Constructor:* def __init__(self, nombre: str):
- Almacena el nombre y prepara la clase para cargar un estudiante.
- *Métodos:*
  - def cargarEstudiante(self, expediente: GestorExpediente):

    Carga los datos de un estudiante a partir de su expediente digital. Recibe como parámetro un objeto GestorExpediente, el cual contiene la información académica y administrativa del estudiante.

- def inscribirEstExamen():

  Permite inscribir al estudiante a un examen correspondiente a una Unidad Curricular (UC). Recibe como parámetro un objeto UC que representa la materia en la que se desea registrar la inscripción.

- def quitarExamenEstudiante():

  Elimina la inscripción del estudiante a un examen correspondiente a una Unidad Curricular (UC). Recibe como parámetro un objeto UC que representa la materia de la cual se desea retirar la inscripción.

- def agregar_uc_aprobada():

  Agrega una materia aprobada al expediente del estudiante, registrándola junto con sus créditos correspondientes. Recibe como parámetro un objeto UC que representa la materia aprobada y devuelve True si la materia fue agregada correctamente, o False si ya se encontraba registrada previamente.

- def quitar_uc():

  Elimina una Unidad Curricular (UC) del listado de materias aprobadas en el expediente del estudiante. Recibe como parámetro un objeto UC que representa la materia a eliminar y devuelve True si la eliminación se realizó correctamente.

- def matricularEstudiante():

  Matrícula al estudiante en una Unidad Curricular (UC), siempre y cuando cumpla con los requisitos establecidos, como la aprobación de las materias previas. Recibe como parámetro un objeto UC que representa la materia a la que se desea inscribir.

- def desmatricularEstudiante():

Elimina la inscripción del estudiante a una Unidad Curricular (UC) previamente matriculada, actualizando su registro académico en consecuencia. Recibe como parámetro un objeto UC que representa la materia que se desea desmatricular.

- def verInscriptos(self):

  Este método permite obtener el registro completo de estudiantes inscriptos por unidad curricular, accediendo al expediente del sistema. Devuelve un diccionario en el que las claves corresponden a las materias y los valores son las listas de estudiantes inscriptos en cada una, lo que facilita la consulta centralizada de las inscripciones actuales.

6.2 Relaciones entre Clases

El siguiente diagrama UML representa gráficamente la estructura del sistema de matriculación, mostrando las clases definidas, sus atributos, métodos y relaciones. Este recurso facilita la comprensión del diseño orientado a objetos y la interacción entre componentes del sistema:

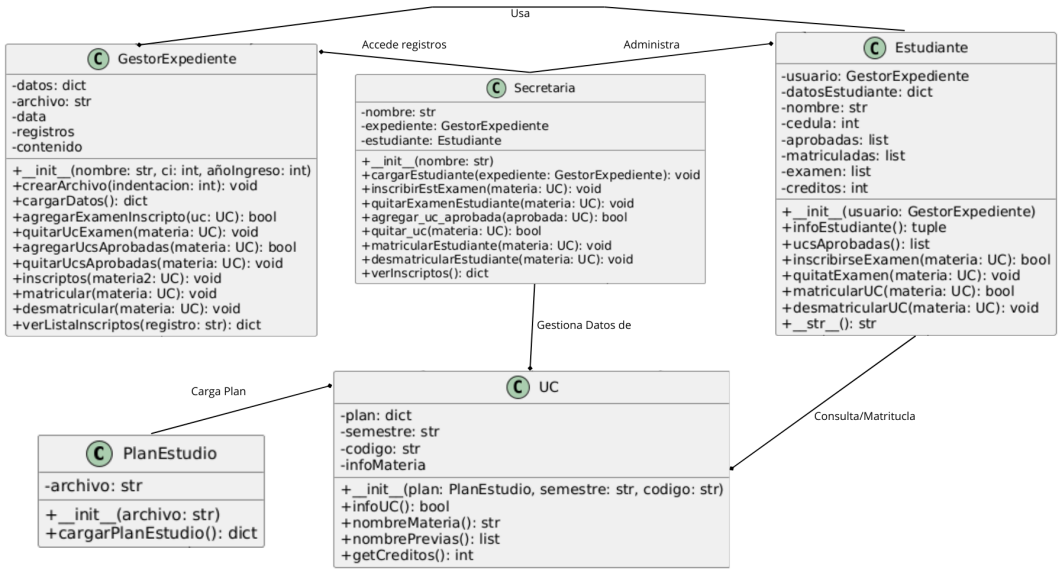

Figura 1. Diagrama UML

Se identifican distintos tipos de relaciones entre clases, como composición, agregación y asociaciones simples, representando diferentes grados de dependencia entre los objetos:

- PlanEstudio y UC mantienen una relación de composición. Cada objeto UC requiere obligatoriamente una instancia de PlanEstudio para existir, ya que extrae su información directamente del archivo del plan. Sin un plan de estudio válido, una unidad curricular no puede operar.
- La clase UC funciona como entidad base para múltiples operaciones en otras clases. Las clases Estudiante, GestorExpediente y Secretaria interactúan con objetos UC para consultar información, realizar inscripciones o gestionar aprobaciones. Estas son asociaciones simples, ya que utilizan UC como argumento o consulta, pero no conservan su instancia como atributo permanente.
- La clase GestorExpediente posee una relación de composición fuerte con el archivo JSON del estudiante. Es responsable de crear, modificar y mantener toda la información académica del alumno, encapsulando su persistencia y manipulación.
- La clase Estudiante tiene una relación de agregación con GestorExpediente, ya que depende de él para acceder y actualizar sus datos académicos, pero no es responsable directa de su gestión estructural o almacenamiento.
- Finalmente, la clase Secretaria encapsula tanto a un objeto Estudiante como a su GestorExpediente, estableciendo una composición funcional. Todas sus operaciones dependen de estos componentes para interactuar correctamente con el sistema.

CONCLUSIONES

Se logró implementar un Sistema de Matriculaciones básico para la carrera de Ingeniería Biomédica, utilizando Python y los principios de Programación Orientada a Objetos. El sistema permite controlar la inscripción a unidades curriculares y exámenes en función del plan de estudios, evitando errores comunes de matriculación.

Durante el desarrollo se reforzaron los conceptos de clases, objetos, abstracción, encapsulamiento y relaciones como composición y agregación. Cada clase tiene una función clara dentro del sistema: desde la carga del plan de estudio hasta la gestión de inscripciones, historial académico y validaciones.

También se aplicaron técnicas de manejo de archivos y excepciones en Python para asegurar la persistencia y consistencia de los datos. Además, se utilizó un diagrama UML para representar visualmente la estructura del sistema y las relaciones entre clases.

En resumen, el proyecto permitió integrar los contenidos trabajados en clase en una solución funcional y cercana a una problemática real de gestión académica.