Actividad Herencia y Polimorfismo

Para la siguiente actividad, debes crear una carpeta y en el crear un archivo por cada clase, el archivo debe llevar el mismo nombre que la clase, por ejemplo, clase se llama Auto, el archivo debe llamarse Auto.py (Respetar mayúsculas), además en la carpeta debe existir una archivo llamado Main.py.

Cuando finalice todos los ejercicios debe comprimir todos los .py en cualquiera de los siguientes formatos: .rar .zip .tar .tar.gz y debe adjuntar un archivo de texto REAME.txt donde estas todos los integrantes del grupo:

Desarrollo de un Sistema de Gestión Universitaria

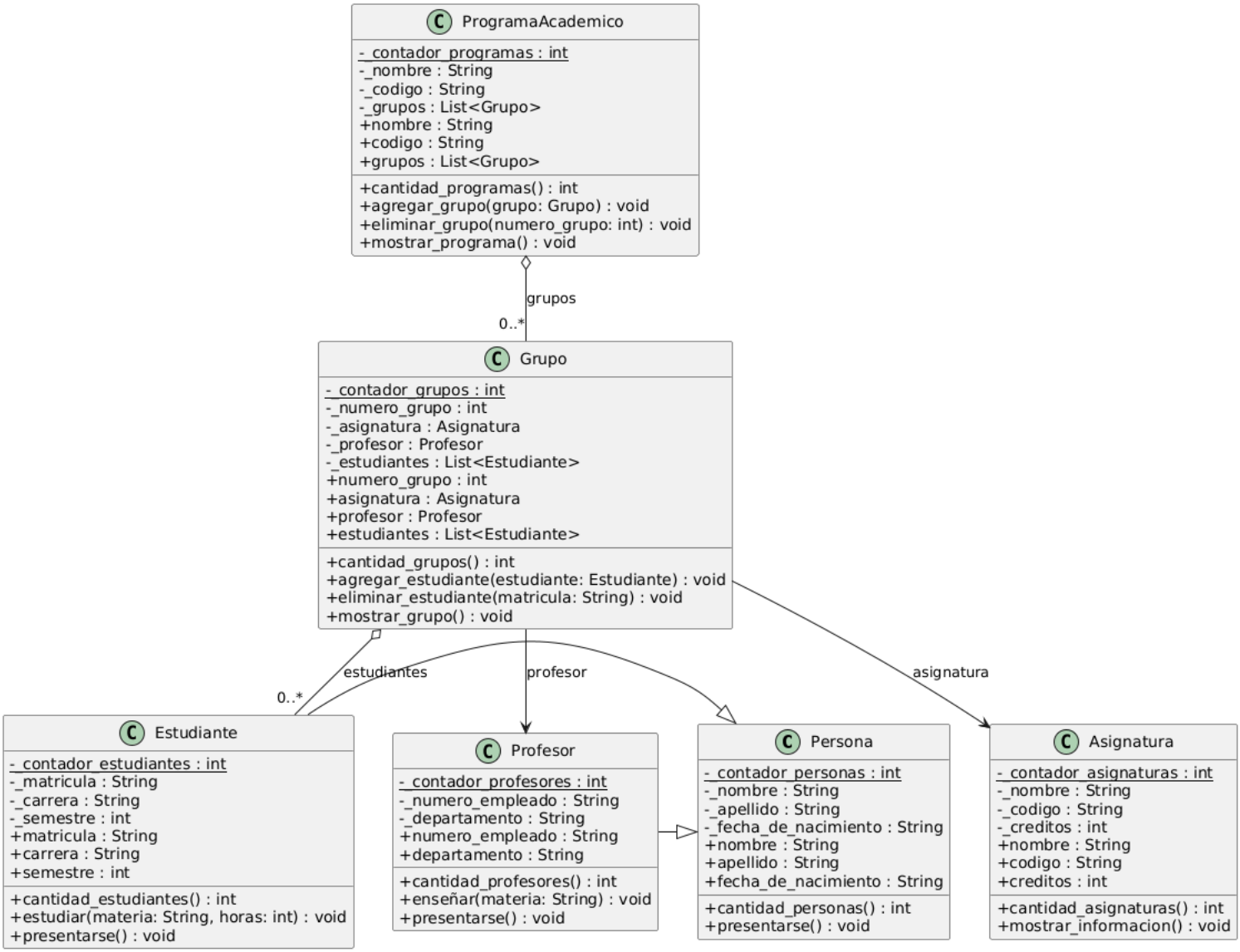
Objetivo

Desarrollar un sistema de gestión universitaria utilizando los principios de programación orientada a objetos (POO), incluyendo herencia, polimorfismo, encapsulamiento, métodos getters y setters, métodos privados, públicos y protegidos, así como el uso de memoria compartida mediante métodos de clase y atributos de clase y interfaz grafica

Descripción del Sistema

El sistema debe permitir gestionar las entidades principales de una universidad, tales como personas (estudiantes y profesores), asignaturas, grupos y programas académicos. Los estudiantes podrán interactuar con el sistema para realizar acciones como inscribirse en grupos y estudiar materias, mientras que los profesores podrán enseñar asignaturas.

Diagrama de clases:



Requisitos del Sistema

1. Clases y Herencia

- Clase Persona (Clase Base)
  - Atributos Privados:
    - nombre (str)
    - apellido (str)
    - fecha_de_nacimiento (str)
  - Métodos Públicos:
    - presentarse(): Imprime una presentación genérica de la persona.

    - Propiedades (@property):
        - nombre: Permite obtener y establecer el nombre.
        - apellido: Permite obtener y establecer el apellido.
        - fecha_de_nacimiento: Permite obtener y establecer la fecha de nacimiento.
- Clase Estudiante (Hereda de Persona)
    - Atributos Privados:
        - matricula (str)
        - carrera (str)
        - semestre (int)
    - Métodos Públicos:
        - estudiar(materia: str, horas: int): Imprime un mensaje indicando que el estudiante ha estudiado una materia durante cierta cantidad de horas.
        - presentarse(): Sobrescribe el método de Persona para incluir información específica del estudiante.
    - **Propiedades (@property):**
        - matricula: Permite obtener y establecer la matrícula.
        - carrera: Permite obtener y establecer la carrera.
        - semestre: Permite obtener y establecer el semestre.
- **Clase Profesor (Hereda de Persona)**
    - **Atributos Privados:**
        - numero_empleado (str)
        - departamento (str)
    - **Métodos Públicos:**
        - enseñar(materia: str): Imprime un mensaje indicando que el profesor está enseñando una materia.
        - presentarse(): Sobrescribe el método de Persona para incluir información específica del profesor.
    - **Propiedades (@property):**
        - numero_empleado: Permite obtener y establecer el número de empleado.
        - departamento: Permite obtener y establecer el departamento.

2. Clases Adicionales

- **Clase Asignatura**
    - **Atributos Privados:**
        - nombre (str)
        - codigo (str)
        - creditos (int)

    - **Métodos Públicos:**
        - mostrar_informacion(): Imprime la información detallada de la asignatura.
    - **Propiedades (@property):**
        - nombre: Permite obtener y establecer el nombre de la asignatura.
        - codigo: Permite obtener y establecer el código de la asignatura.
        - creditos: Permite obtener y establecer los créditos de la asignatura.
- **Clase Grupo**
    - **Atributos Privados:**
        - numero_grupo (int)
        - asignatura (Asignatura)
        - profesor (Profesor)
        - estudiantes (Lista protegida de objetos Estudiante)
    - **Métodos Públicos:**
        - agregar_estudiante(estudiante: Estudiante): Agrega un estudiante al grupo con validación para evitar duplicados.
        - eliminar_estudiante(matricula: str): Elimina un estudiante del grupo por su matrícula, asegurando que exista.
        - mostrar_grupo(): Muestra la información completa del grupo, incluyendo asignatura, profesor y lista de estudiantes.
    - **Propiedades (@property):**
        - numero_grupo: Permite obtener y establecer el número del grupo.
        - asignatura: Permite obtener y establecer la asignatura del grupo.
        - profesor: Permite obtener y establecer el profesor asignado al grupo.
        - estudiantes: Solo permite obtener la lista de estudiantes (no se puede establecer directamente).
- **Clase ProgramaAcademico**
    - **Atributos Privados:**
        - nombre (str)
        - codigo (str)
        - grupos (Lista protegida de objetos Grupo)
    - **Métodos Públicos:**
        - agregar_grupo(grupo: Grupo): Agrega un grupo al programa académico con validación para evitar duplicados.
        - eliminar_grupo(numero_grupo: int): Elimina un grupo del programa académico por su número, asegurando que exista.
        - mostrar_programa(): Muestra la información completa del programa académico, incluyendo todos los grupos asociados.
    - **Propiedades (@property):**
        - **nombre:** Permite obtener y establecer el nombre del programa académico.

        - **codigo:** Permite obtener y establecer el código del programa académico.
        - **grupos:** Solo permite obtener la lista de grupos (no se puede establecer directamente).

3. Interacción y Validaciones

- Validaciones Requeridas:
    - **Agregar Estudiante a Grupo:**
        - Verificar que el estudiante no esté ya inscrito en el grupo para evitar duplicados.
    - **Eliminar Estudiante de Grupo:**
        - Asegurar que la matrícula del estudiante exista en el grupo antes de eliminarlo.
    - **Agregar Grupo al Programa Académico:**
        - Verificar que el grupo no esté ya incluido en el programa académico para evitar duplicados.
    - **Eliminar Grupo del Programa Académico:**
        - Asegurar que el número del grupo exista en el programa académico antes de eliminarlo.
- **Interfaz Textual:**
    - Implementar una interfaz de usuario basada en texto que permita realizar las siguientes acciones:
        - **Crear y Gestionar Entidades:**
            - Crear y gestionar profesores, estudiantes, asignaturas y grupos.
        - **Asignar Estudiantes a Grupos:**
            - Inscribir estudiantes en grupos específicos, asegurando que no se dupliquen.
        - **Agregar o Eliminar Grupos del Programa Académico:**
            - Gestionar la inclusión y exclusión de grupos dentro de un programa académico.
        - **Mostrar Información:**
            - Visualizar información detallada de programas académicos y sus componentes, incluyendo grupos, asignaturas, profesores y estudiantes.
- Documentación:
    - Incluir comentarios y docstrings en cada clase y método para explicar su propósito y funcionamiento. Esto facilitará la comprensión del código y su mantenimiento posterior.

4. Consideraciones de Diseño

- **Encapsulamiento y Protección de Atributos:**
  - Todos los atributos de las clases deben estar protegidos (por ejemplo, utilizando un prefijo _) para evitar accesos directos desde fuera de la clase.
  - Utilizar propiedades (@property) y setters para controlar el acceso y modificación de los atributos, garantizando así el encapsulamiento.
- **Herencia y Polimorfismo:**
  - Las clases Estudiante y Profesor deben heredar de la clase base Persona, permitiendo la reutilización de código y la extensión de funcionalidades específicas.
  - Sobrescribir métodos como presentarse() en las clases derivadas para proporcionar presentaciones más detalladas, demostrando así el polimorfismo.
- **Manejo de Memoria Compartida:**
  - Utilizar atributos de clase y métodos de clase cuando sea necesario para manejar información compartida entre todas las instancias de una clase.
  - Por ejemplo, si se requiere llevar un registro de todas las personas creadas, se puede implementar un atributo de clase en Persona que almacene estas instancias.
- **Validaciones y Manejo de Errores:**
  - Implementar validaciones en los métodos para asegurar la integridad de los datos.
  - Manejar posibles errores o entradas inválidas de manera adecuada, proporcionando mensajes informativos al usuario.

5. Flujo de Trabajo Sugerido

1. **Definición de Clases:**
   - Comenzar definiendo la clase base Persona y sus métodos.
   - Extender la clase Persona para crear las clases Estudiante y Profesor, añadiendo sus atributos y métodos específicos.
   - Definir las clases adicionales Asignatura, Grupo y ProgramaAcademico con sus respectivas relaciones y funcionalidades.
2. **Implementación de Relaciones:**
   - Asegurar que las clases se relacionen correctamente, por ejemplo, que un Grupo esté asociado a una Asignatura y un Profesor, y que contenga una lista de Estudiantes.
3. **Incorporación de Validaciones:**
   - Implementar las validaciones necesarias en los métodos de las clases para evitar inconsistencias, como duplicados o eliminaciones de entidades

inexistentes.

4. **Desarrollo de la Interfaz Textual:**
    - Crear una interfaz de usuario sencilla basada en texto que permita interactuar con el sistema, realizando operaciones como agregar o eliminar grupos, inscribir estudiantes, etc.
5. **Documentación:**
    - Añadir comentarios y docstrings en cada clase y método para explicar su propósito y funcionamiento, facilitando así la comprensión del código.
6. **Pruebas y Validación:**
    - Realizar pruebas creando instancias de las clases y ejecutando los métodos para asegurar que el sistema funciona correctamente y cumple con los requisitos establecidos.

Consideraciones Adicionales

- **Encapsulamiento y Protección de Atributos:**
    - Todos los atributos de las clases están protegidos (prefijados con _) para evitar accesos directos desde fuera de la clase.
    - Se utilizan propiedades (@property) y setters para controlar el acceso y modificación de los atributos.
- **Herencia y Polimorfismo:**
    - Las clases Estudiante y Profesor heredan de la clase base Persona, permitiendo la reutilización de código y la extensión de funcionalidades específicas.
    - El método presentarse es sobrescrito en las clases derivadas para proporcionar presentaciones más detalladas, demostrando así el polimorfismo.
- **Validaciones:**
    - Al agregar o eliminar estudiantes y grupos, se realizan verificaciones para evitar duplicados y asegurar la integridad de los datos.
- **Interfaz de Usuario:**
    - La interfaz textual proporciona una forma interactiva de gestionar el sistema, permitiendo a los usuarios realizar operaciones básicas de manera sencilla.
- **Documentación:**
    - Se incluyen docstrings en cada clase y método para facilitar la comprensión del código y su mantenimiento.

Índice