



Universidad Tecnológica de Durango

Tecnologías de la Información

Programación Orientada a Objetos

Prácticas

“Evidencias de Prácticas”

Alumno:

- Edson Lomas Corral

3°B BIS

Docente:

- Ing. Dagoberto Fiscal Gurrola, M.T.I.

Octubre 2025

Tabla de Ilustraciones

Ilustración 1 Evidencia de GitHub de Programación Orientada a Objetos.	3
Ilustración 2 Implementar paradigmas Estructurados VS OO.	4
Ilustración 3 Modelar y Diagramar en POO Codificación.	5
Ilustración 4 Modelar y Diagramar en POO Diagrama.	6
Ilustración 5 Modelar y Diagramar un sistema de escuela Codificación Pte.1	7
Ilustración 6 Modelar y Diagramar un sistema de escuela Codificación Pte.2	8
Ilustración 7 Modelar y Diagramar un sistema de escuela Codificación Pte.3	9
Ilustración 8 Modelar y Diagramar un sistema de escuela Diagrama.	10
Ilustración 9 Diagramas UML del Sistema de Control Escolar (Clases y Objetos).	11
Ilustración 10 Diagramas UML del Sistema de Control Escolar (Casos de Uso).	12
Ilustración 11 Diagramas UML del Sistema de Control Escolar (Actividades).	13
Ilustración 12 Diagramas UML del Sistema de Control Escolar (Secuencias).	14
Ilustración 13 Diagramas UML del Sistema de Control Escolar (Estados).	15

Actividades GitHub

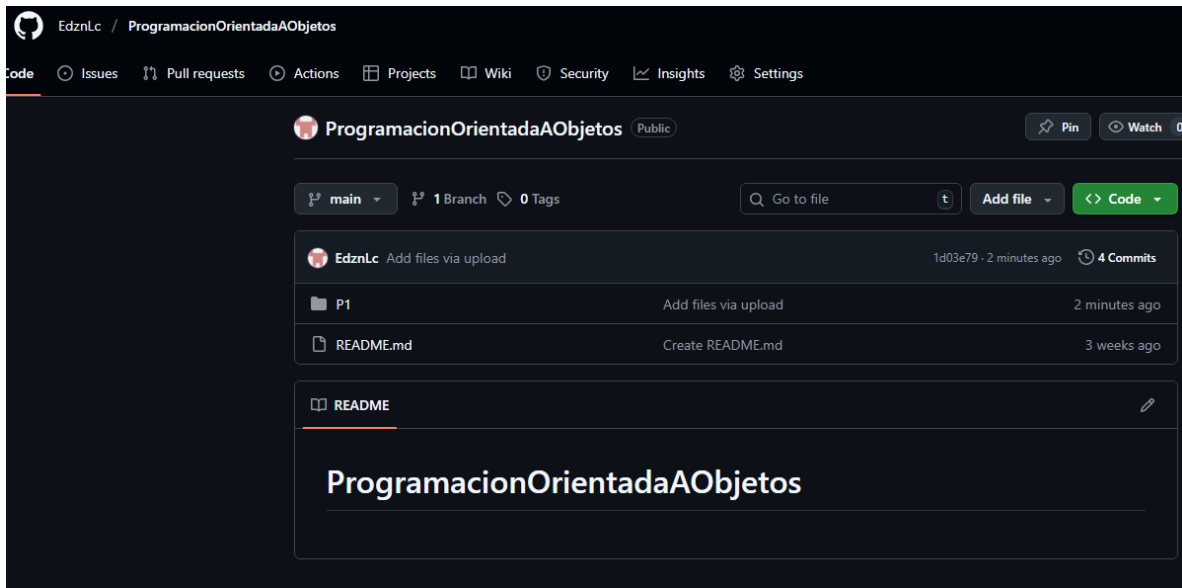
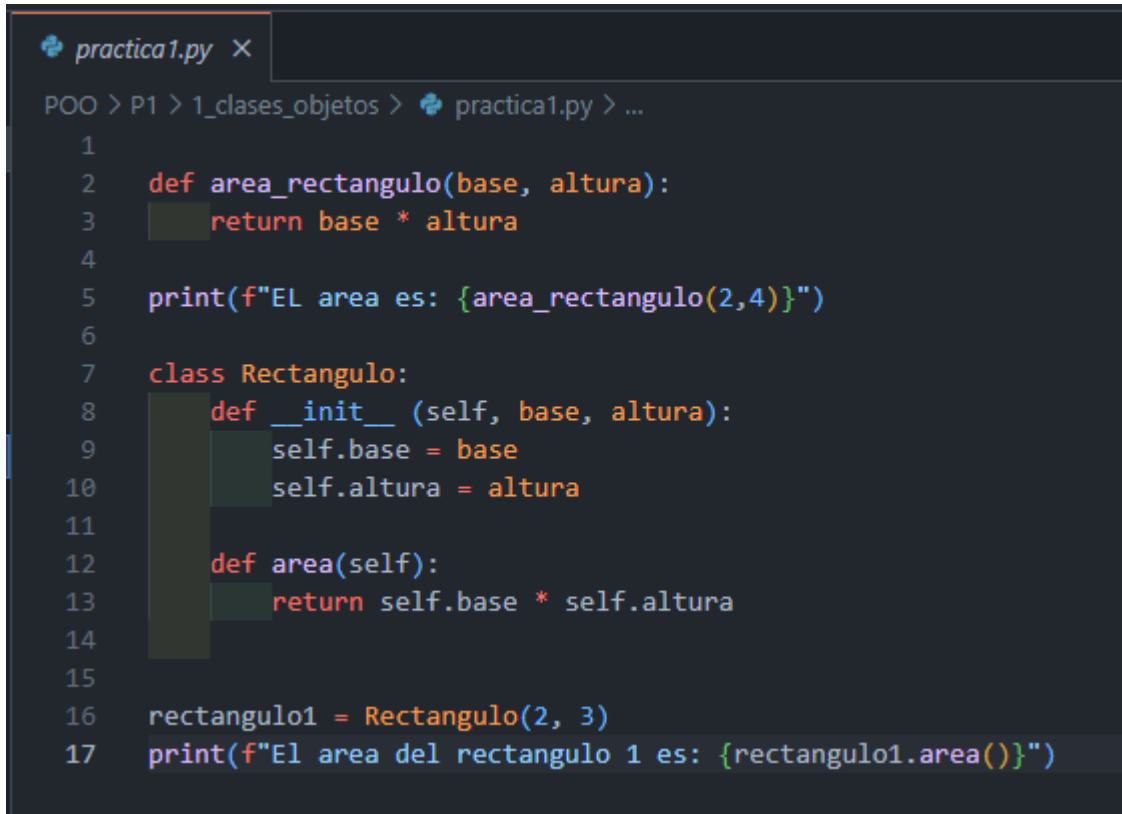


Ilustración 1 Evidencia de GitHub de Programación Orientada a Objetos.

Da clic: [ENLACE PARA ACCEDER A GITHUB](#)

En la imagen anterior se muestra la evidencia del repositorio creado en GitHub con el nombre de Programación Orientada a Objetos y dentro de este se encuentra una carpeta con el nombre de P1 la cual contiene las prácticas de clases y objetos, getters y setters, múltiples objetos, constructor, encapsulamiento, herencia y polimorfismo realizadas en clase.

Práctica 1



```
practica1.py X
POO > P1 > 1_clases_objetos > practica1.py > ...
1
2  def area_rectangulo(base, altura):
3      return base * altura
4
5  print(f"EL area es: {area_rectangulo(2,4)}")
6
7  class Rectangulo:
8      def __init__(self, base, altura):
9          self.base = base
10         self.altura = altura
11
12         def area(self):
13             return self.base * self.altura
14
15
16  rectangulo1 = Rectangulo(2, 3)
17  print(f"El area del rectangulo 1 es: {rectangulo1.area()}")
```

Ilustración 2 Implementar paradigmas Estructurados VS OO.

En la imagen anterior se muestra la evidencia de la práctica 1, en la cual primeramente se puede observar cómo se codifica un algoritmo que calcula el área de un rectángulo con el paradigma Estructurado y luego con el Orientado a Objetos.

Práctica 2

```
practica2.py X
POO > P1 > 1_clases_objetos > practica2.py > ...
1  """
2  Ejercicio Practico #2 "Modelar y Diagramar en POO"
3
4  """
5
6  import os
7  os.system("cls")
8
9  class Coches:
10     def __init__(self, color, marca, velocidad):
11         self.color = color
12         self.marca = marca
13         self.velocidad = velocidad
14
15     def acelerar(self, incremento):
16         self.velocidad += incremento
17         return self.velocidad
18
19     def frenar(self, decremento):
20         self.velocidad -= decremento
21         return self.velocidad
22
23     def tocar_claxon(self):
24         return f"El coche de color {self.color} ha tocado el claxon"
25
26 #Instanciar objetos de la clase coches
27 coche1 = Coches("Blanco", "Toyota", 220)
28 coche2 = Coches("Amarillo", "Nissan", 180)
29
30
31 print(f"Los valores del objeto 1 son: {coche1.color}, {coche1.marca}, {coche1.velocidad}")
32 velocidad = print(f"La velocidad original del coche 1 es {coche1.velocidad} y cambio a {coche1.acelerar(50)}")
33 print(f"Los valores del objeto 2 son: {coche2.color}, {coche2.marca}, {coche2.velocidad}")
34 velocidad = print(f"La velocidad original del coche 2 es {coche2.velocidad} y cambio a {coche2.frenar(100)}")
```

Ilustración 3 Modelar y Diagramar en POO Codificación.

En la imagen anterior se muestra la evidencia de la codificación de la práctica 2, en la cual se puede observar la practica coches, donde primero se crea una clase con el mismo nombre y se inicializan sus atributos con un constructor, posteriormente se definen sus métodos y al final se crean los objetos y se imprimen los valores de sus atributos y métodos.

Modelado:

- Clase: Coches
 - Atributos: marca, color, modelo, velocidad, caballaje, plazas
 - Métodos: acelerar(), frenar()

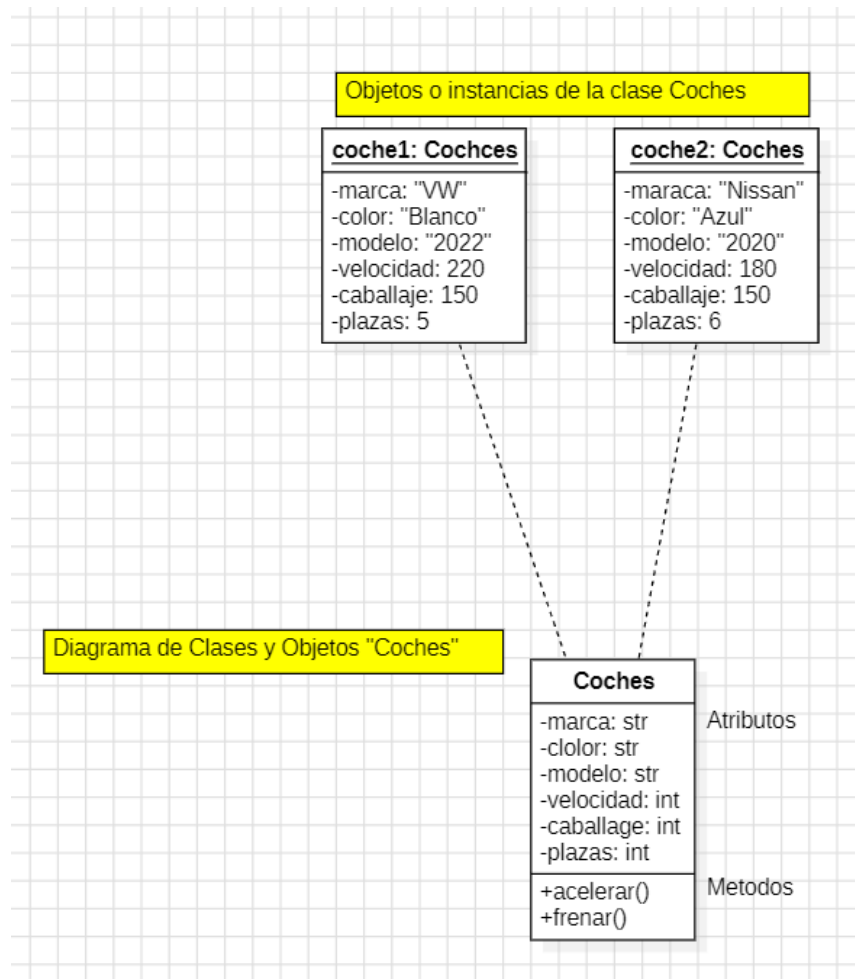


Ilustración 4 Modelar y Diagramar en POO Diagrama.

En la imagen anterior se muestra la evidencia del diagrama de la práctica 2, en la cual se pueden observar los diagramas de clases y objetos, en donde esta creada la clase Coches con atributos privados y tres métodos y dos objetos llamados coche1 y coche2 con el valor de los atributos de la clase.

Práctica 3

```
class Alumnos:
    def __init__(self,nombre,edad,matricula):
        self.__nombre=nombre
        self.__edad=edad
        self.__matricula=matricula

    def inscribirse(self):
        print(f"El alumno {self.__nombre} se inscribio")

    def estudiar(self):
        print(f"El alumno {self.__nombre} esta estudiando")

    def getNombre(self):
        return self.__nombre

    def setNombre(self,nuevo):
        self.__nombre=nuevo

    def getEdad(self):
        return self.__edad

    def setEdad(self,nuevo):
        self.__edad=nuevo

    def getMatricula(self):
        return self.__matricula

    def setMatricula(self,nuevo):
        self.__matricula=nuevo
```

Ilustración 5 Modelar y Diagramar un sistema de escuela Codificación Pte.1

En la imagen anterior se muestra la primera parte de la codificación de la práctica 3, en la cual se puede observar que esta creada una clase llamada Alumnos la cual inicializa sus atributos con un método constructor, define sus métodos y accede al valor de los atributos mediante getters y setters.

```
class Cursos:
    def __init__(self,nombre,codigo,creditos):
        self.__nombre=nombre
        self.__codigo=codigo
        self.__creditos=creditos

    def asignar(self):
        print(f"Se asigno el curso de {self.__nombre}")

    @property
    def nombre(self):
        return self.__nombre

    @nombre.setter
    def nombre(self,nuevo):
        self.__nombre=nuevo

    @property
    def codigo(self):
        return self.__codigo

    @codigo.setter
    def codigo(self,nuevo):
        self.__codigo=nuevo

    @property
    def credits(self):
        return self.__creditos

    @credits.setter
    def credits(self,nuevo):
        self.__creditos=nuevo
```

Ilustración 6 Modelar y Diagramar un sistema de escuela Codificación Pte.2

En la imagen anterior se muestra la segunda parte de la codificación de la práctica 3, en la cual se puede observar que esta creada una clase llamada Cursos la cual inicializa sus atributos con un método constructor, define sus métodos y accede al valor de los atributos mediante propiedades.


```
class Profesores:
    def __init__(self,nombre,experiencia,num_profesor):
        self.__nombre=nombre
        self.__experiencia=experiencia
        self.__num_profesor=num_profesor
    def impartir(self):
        print(f"El profesor {self.__nombre} esta impartiendo un curso")
    def evaluar(self):
        print(f"El profesor {self.__nombre} esta evaluando")
    def getNombre(self):
        return self.__nombre
    def setNombre(self,nuevo):
        self.__nombre=nuevo
    def getExperiencia(self):
        return self.__experiencia
    def setExperiencia(self,nuevo):
        self.__experiencia=nuevo
    def getNum_Profesor(self):
        return self.__num_profesor
    def setNum_Profesor(self,nuevo):
        self.__num_profesor=nuevo

alumno1=Alumnos("Juan",19,3141240167)
alumno2=Alumnos("Diego",19,3141249632)
alumno1.inscribirse()

curso1=Cursos("matematicas","hrte65",2)
curso2=Cursos("historia","hrup75",3)
curso1.asignar()

profesor1=Profesores("Antonio",8,1289167)
profesor2=Profesores("Omar",10,1289198)
profesor1.impartir()
```

Ilustración 7 Modelar y Diagramar un sistema de escuela Codificación Pte.3

En la imagen anterior se muestra la tercera parte de la codificación de la práctica 3, en la cual se puede observar que esta creada una clase llamada Profesores la cual inicializa sus atributos con un método constructor, define sus métodos, accede al valor de los atributos mediante getters y setters y finalmente se crean los objetos y se usan sus métodos.

Modelado:

- Clase: Alumnos.
 - Atributos: nombre, edad, matricula.
 - Métodos: inscribirse(), estudiar.
- Clase: Cursos.
 - Atributos: nombre, código, créditos.
 - Métodos: asignar().
- Clase: Profesores.
 - Atributos: nombre, experiencia, num_profesor.
 - Métodos: impartir(), evaluar().

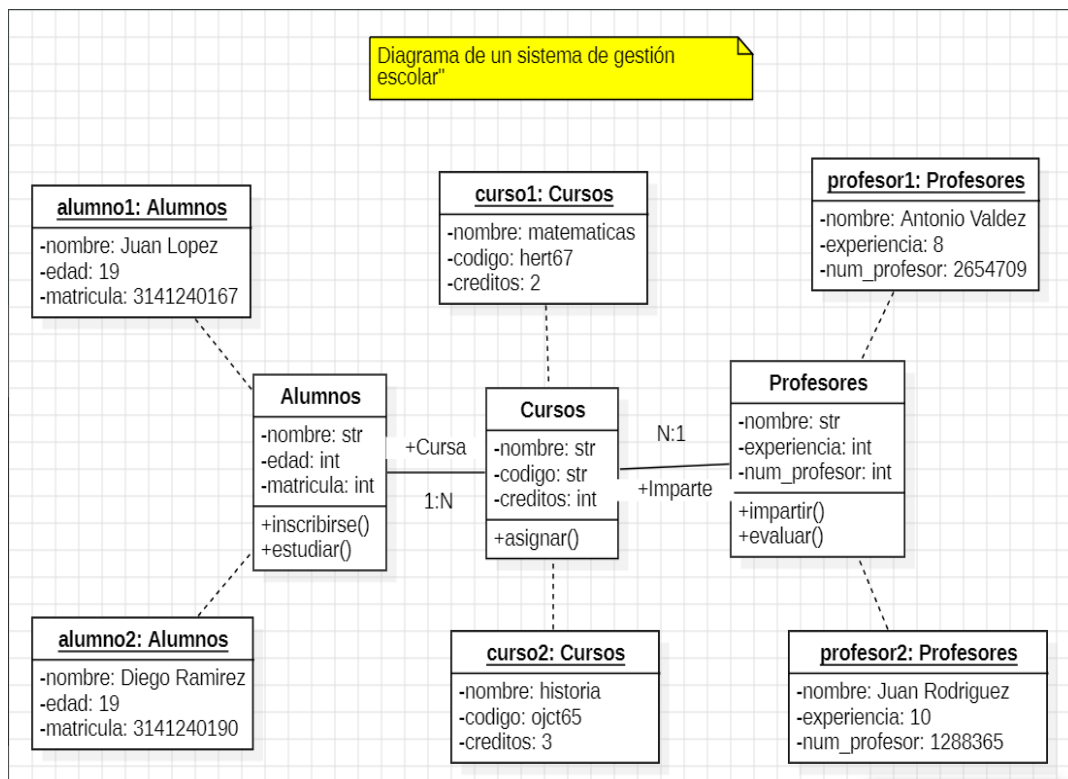


Ilustración 8 Modelar y Diagramar un sistema de escuela Diagrama.

En la imagen anterior se muestra la evidencia del diagrama de la práctica 3, en la cual se pueden observar los diagramas de clases y objetos, en donde están creadas las clases Alumnos, Cursos y Profesores con atributos privados y métodos públicos y dos objetos por cada clase con los valores de sus atributos además de su asociación y cardinalidad.

Práctica 4

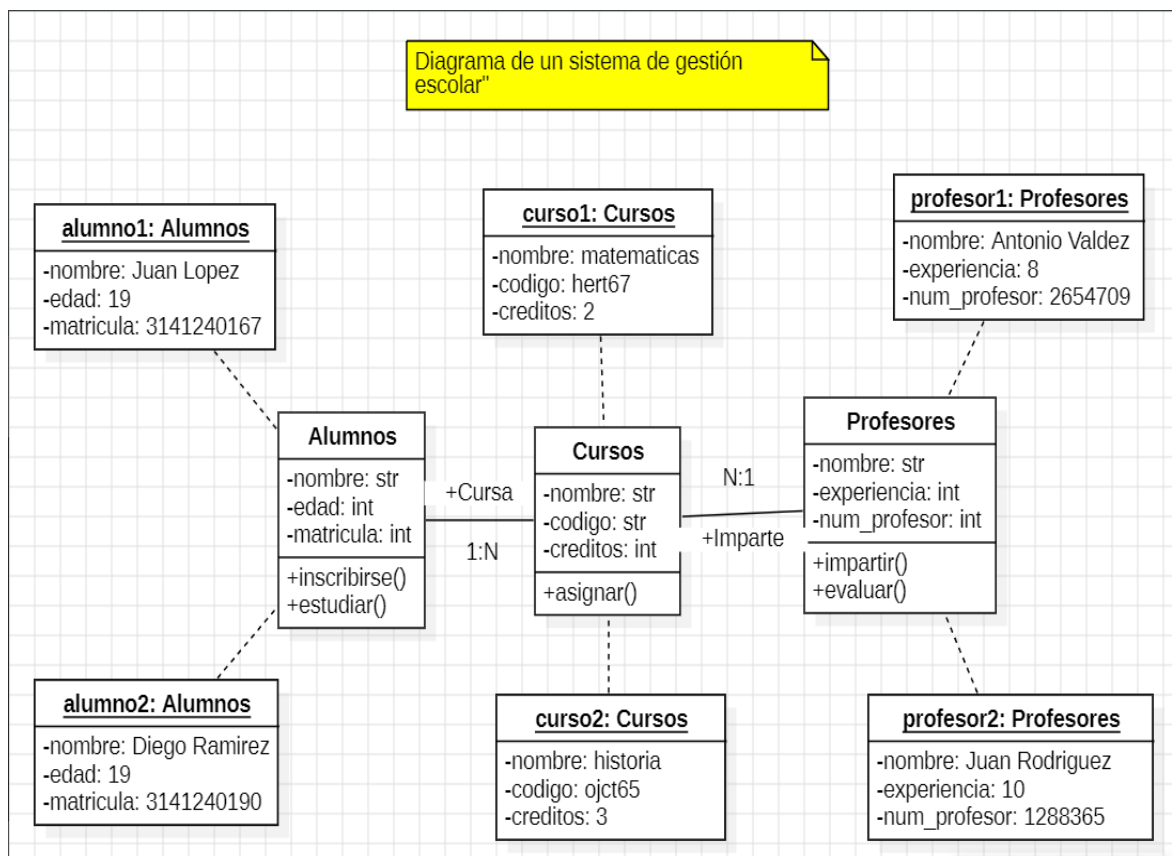


Ilustración 9 Diagramas UML del Sistema de Control Escolar (Clases y Objetos).

En la imagen anterior se muestra la evidencia de los diagramas de clases y objetos de la práctica 4, en donde están creadas las clases Alumnos, Cursos y Profesores con atributos privados y métodos públicos y dos objetos por cada clase con los valores de sus atributos además de su asociación y cardinalidad.

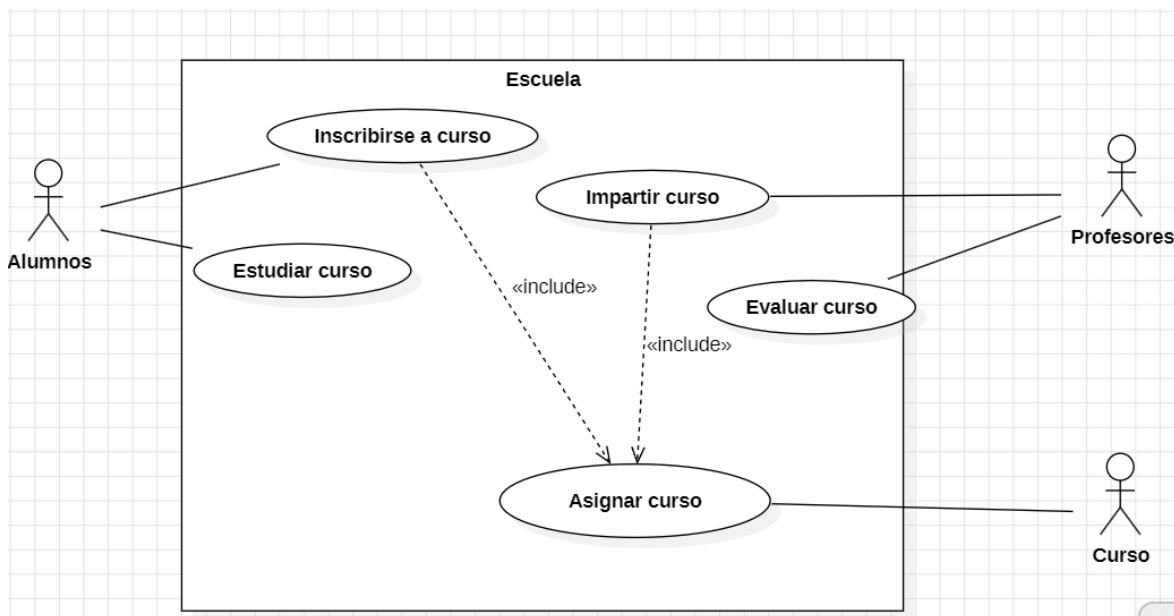


Ilustración 10 Diagramas UML del Sistema de Control Escolar (Casos de Uso).

En la imagen anterior se muestra la evidencia del diagrama de Casos de Uso de la práctica 4, donde se representan los actores Alumnos, Profesores y Curso donde podemos apreciar los distintos casos de uso que pueden realizar a lo largo del sistema.

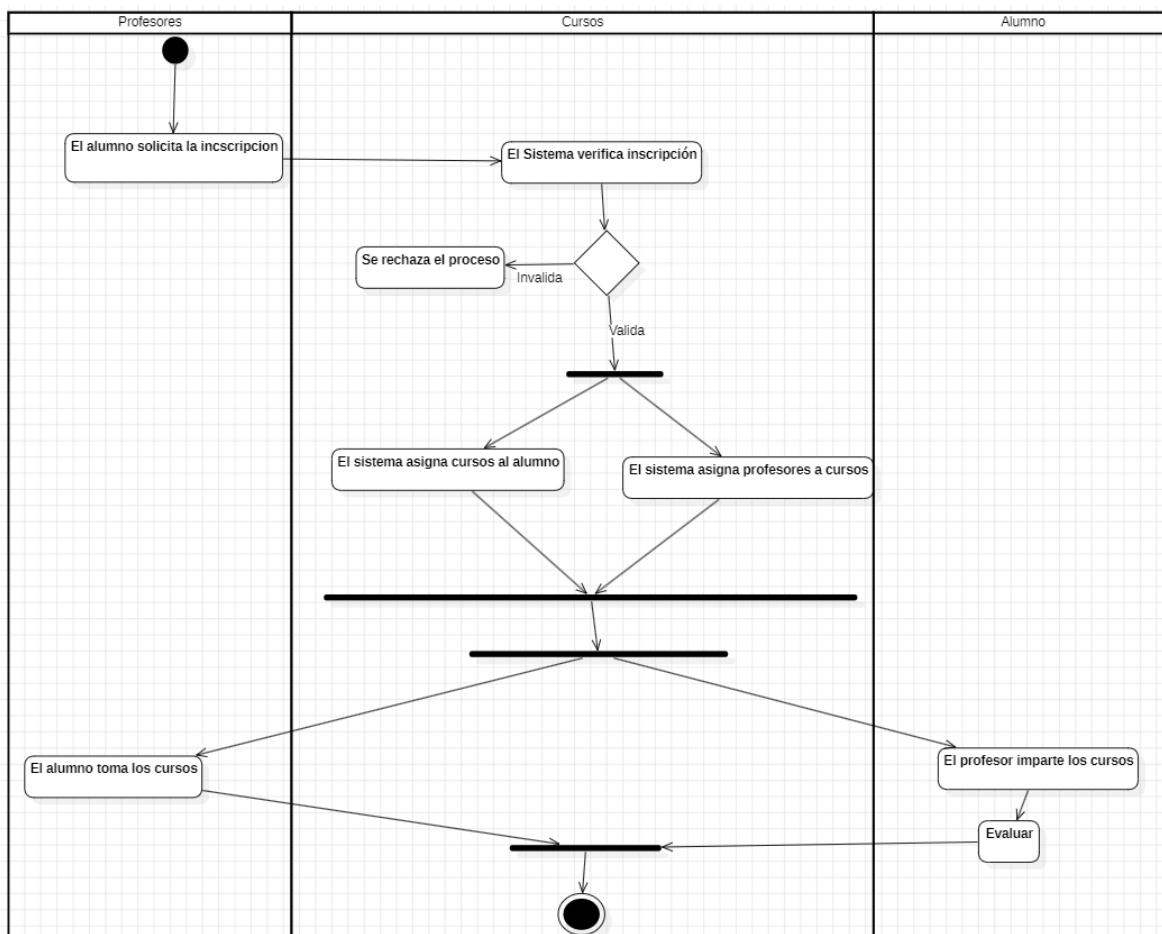


Ilustración 11 Diagramas UML del Sistema de Control Escolar (Actividades).

En la imagen anterior se muestra la evidencia del diagrama de Actividades de la práctica 4, donde podemos observar los tres diferentes objetos y las diferentes actividades que pueden realizar a lo largo del sistema contando con un inicio y fin.

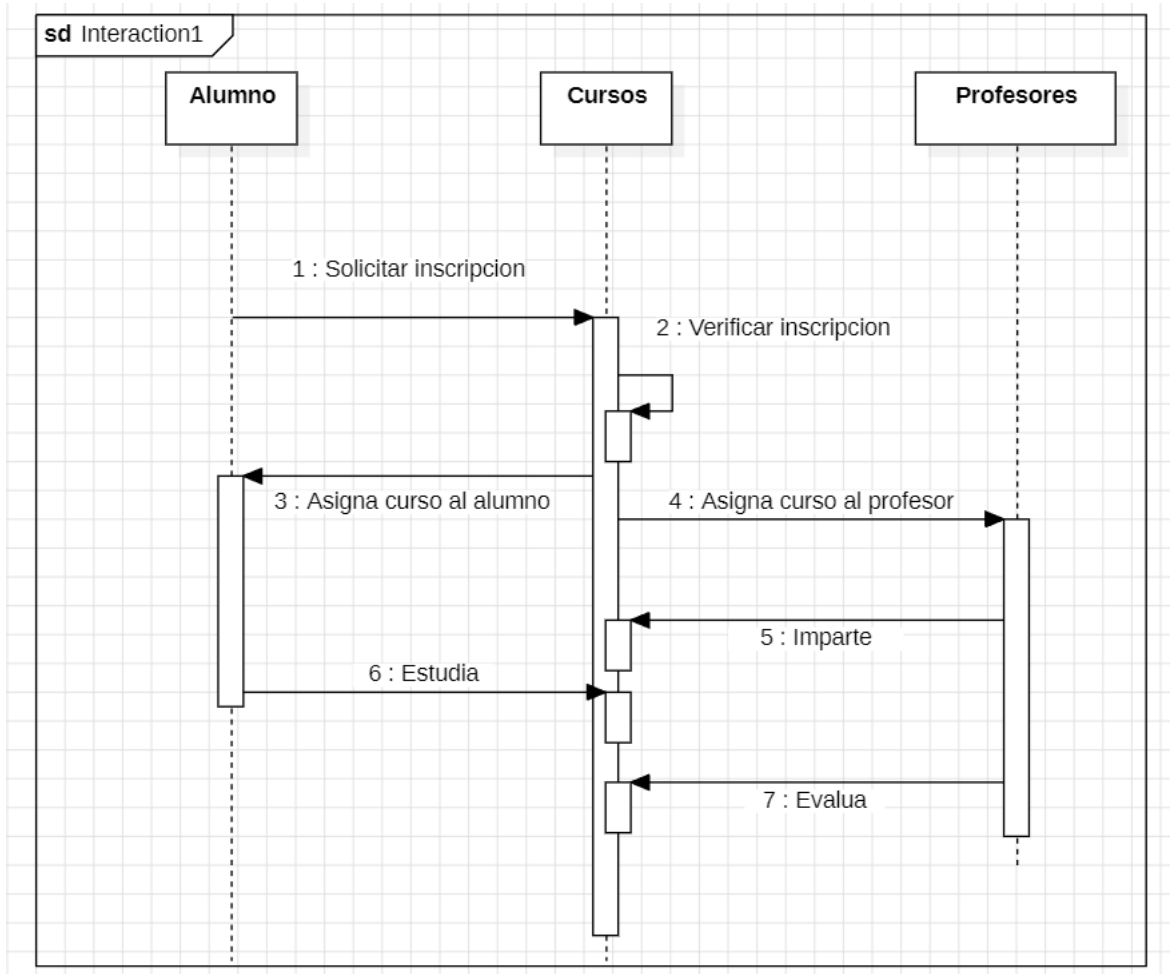


Ilustración 12 Diagramas UML del Sistema de Control Escolar (Secuencias).

En la imagen anterior se muestra la evidencia del diagrama de Secuencias de la práctica 4, que muestra el tiempo de vida y acciones que los objetos pueden realizar a lo largo del programa, integrando las posibles relaciones entre objetos para el correcto funcionamiento del sistema.

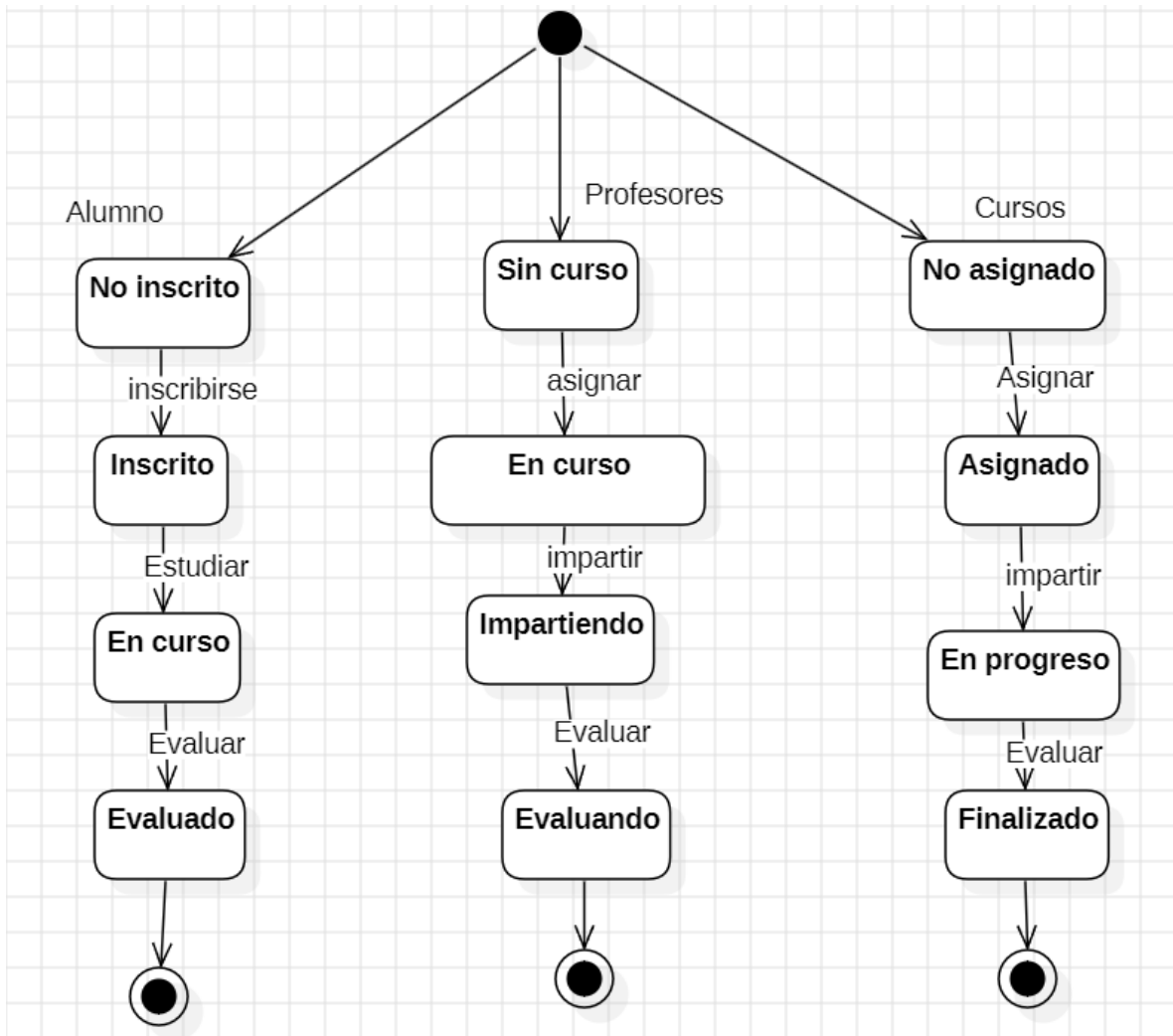


Ilustración 13 Diagramas UML del Sistema de Control Escolar (Estados).

En la imagen anterior se muestra la evidencia del diagrama de estados de la práctica 4, donde podemos observar a los objetos de Alumno, Profesores y Cursos, para después representas los posibles estados con los que pueden contar cada uno de ellos.

Retroalimentación

Gracias al estudio de los temas y la realización de las prácticas de Programación Orientada a Objetos (POO) en Python, he adquirido la habilidad para comprender y aplicar sus conceptos esenciales. Ahora soy capaz de manejar clases y objetos, así como de controlar el acceso a los atributos mediante getters, setters y el método constructor.

La implementación de la herencia y el polimorfismo me ha permitido entender cómo reutilizar código y ampliar funcionalidades de forma eficaz, lo que se traduce en un software más adaptable y fácil de mantener. Además, la elaboración de diagramas UML, como los de clases, objetos, casos de uso, secuencia, estados y actividades, me ha proporcionado una perspectiva más nítida sobre la interacción entre los distintos componentes de un sistema, simplificando las fases de diseño y documentación.

En conclusión, he asimilado que la aplicación de los principios de la POO —incluyendo el uso de clases, objetos, atributos, métodos y sus cuatro pilares fundamentales— es clave para organizar el código de manera lógica y coherente. Esto permite que cada parte del sistema tenga una responsabilidad bien definida, dando como resultado la creación de aplicaciones más modulares y mejor estructuradas.