Relaciones entre clases

Alumno: Puma Huanca Anthony Rusbel
Docente: Ing. Coyla Idme Leonel
Lenguajes de programación II – FINESI
Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Relaciones entre clases

En la POO en Python, las relaciones entre clases permiten modelar cómo los objetos interactúan entre sí en un sistema. Existen tres relaciones fundamentales que definen cómo una clase puede relacionarse con otra.

Asociación

Es una relación general entre dos clases, donde una clase utiliza a otra. Esta relación no implica propiedad ni dependencia fuerte.

Características:

Los objetos están relacionados pero son independientes.

Pueden ser unidireccional o bidireccional.

Se basa en el uso de instancias de otra clase.

Ejemplo:

Crear las clases `Profesor` y `Curso`, con sus respectivos atributos y establecer el tipo de relación.

Clase: Profesor.
Atributo: nombre.
Objeto: `prof = Profesor("Dr. Morillos")`

Clase: Curso.
Atributos: nombre, profesor.
Objeto: `curso = Curso("Muestreo", prof)`

```
class Profesor:
    def __init__(self, nombre):
        self.nombre = nombre

class Curso:
    def __init__(self, nombre, profesor):
```

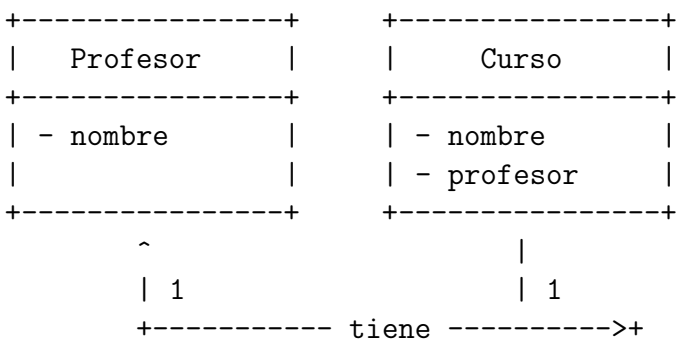
```
        self.nombre = nombre
        self.profesor = profesor

prof = Profesor("Dr. Morillos")
curso = Curso("Muestreo", prof)
print(curso.profesor.nombre)
```

Listing 1: Código Asociación

Ejecución:

```
Dr. Morillos
```

Relación (diagrama textual):



Interpretación:

Cada `Curso` tiene un solo `Profesor` (relación 1 a 1).

El atributo `profesor` dentro de `Curso` establece esta relación.

`Profesor` no depende directamente de `Curso` (no hay referencia inversa).

Ejemplo:

Crear las clases `Departamento` y `Universidad`.

Clase: Departamento.
Atributo: nombre.
Objetos: `dep1 = Departamento(Ïngeniería Estadística")`, `dep2 = Departamento(Ïnformática")`

Clase: Universidad.
Atributo: nombre.
Acción: `agregarDepartamento()`.
Objeto: `uni = Universidad(Üniversidad Nacional del Altiplano")`

```
class Departamento:
    def __init__(self, nombre):
        self.nombre = nombre

class Universidad:
    def __init__(self, nombre):
        self.nombre = nombre
        self.departamentos = []
```

```
9
10      def agregarDepartamento(self, departamento):
11          self.departamentos.append(departamento)
12
13  dep1 = Departamento("Ingenier a Estad stica")
14  dep2 = Departamento("Inform tica")
15
16  uni = Universidad("Universidad Nacional del Altiplano")
17  uni.agregarDepartamento(dep1)
18  uni.agregarDepartamento(dep2)
19
20  for d in uni.departamentos:
21      print(d.nombre)
```

Listing 2: Código Agregación

Ejecución:

```
Ingeniería Estadística
Informática
```

Relación (diagrama textual):

```
+------------------------+       +------------------------+
|      Universidad       |<>---->|      Departamento      |
+------------------------+       +------------------------+
| - nombre               |       | - nombre               |
| - departamentos[]      |       +------------------------+
+------------------------+

1                              *
(una universidad)      (varios departamentos)
```

Interpretación:

`Universidad` tiene una lista (`departamentos`) que almacena varios objetos de tipo `Departamento`.

La relación es uno a muchos ($1 \rightarrow *$).

El símbolo `<>` indica agregación: la universidad contiene, pero los departamentos pueden existir por separado.

Ejemplo 1

Crear las clases `Motor` y `Auto`.

Clase: Motor.
Atributo: tipo.
Acción: `encender()`.

Clase: Auto.
Atributo: marca.
Acción: `arrancar()`.
Objeto: `miAuto = Auto("Toyota")`

```python
class Motor:
    def __init__(self, tipo):
        self.tipo = tipo

    def encender(self):
        print(f"Motor: {self.tipo} encendido")

class Auto:
    def __init__(self, marca):
        self.marca = marca
        self.motor = Motor("Gasolinero")

    def arrancar(self):
        print(f"Auto: {self.marca} arrancando")
        self.motor.encender()

miAuto = Auto("Toyota")
miAuto.arrancar()
```

Listing 3: Código Composición Ejemplo 1

Ejecución:

```
Auto: Toyota arrancando
Motor: Gasolinero encendido
```

Relación (diagrama textual):

```
+------------------+      +--------------------+
|       Auto       |----->|       Motor        |
+------------------+      +--------------------+
| - marca          |      | - tipo             |
| - motor          |      +--------------------+
+------------------+      | + encender()       |
| + arrancar()     |      +--------------------+
+------------------+

1                         1
(cada Auto tiene)   (exactamente un Motor)
```

Interpretación:

`Auto` crea su propio `Motor` dentro del constructor (`self.motor = Motor("Gasolinero")`), por lo tanto, la existencia del motor depende del auto → **composición**.

Si el `Auto` deja de existir, su `Motor` también.

La relación es 1 a 1, y se representa con un rombo sólido () desde `Auto` hacia `Motor`.

Ejemplo 2

Crear las clases `Estudiante`, `Profesor`, `Curso` y `Universidad`.

Clase: Estudiante.
Atributos: nombre, dni, codigo_estudiante.
Acciones: `inscribirse()`, `mostrar_informacion()`.
Objetos: `est1`, `est2`

Clase: Profesor.
Atributos: nombre, dni, especialidad.
Acción: `mostrar_informacion()`.
Objetos: `prof1` a `prof6`

Clase: Curso.
Atributos: nombre_curso, profesor.
Acción: `agregar_estudiante()`, `mostrar_detalles()`.
Objetos: `curso1` a `curso6`

Clase: Universidad.
Atributo: nombre.
Acción: `agregar_cursos()`, `mostrar_cursos()`.
Objeto: `univ = Universidad(Üniversidad Nacional del Altiplano")`

```
class Estudiante:
    def __init__(self, nombre, dni, codigo_estudiante):
        self.nombre = nombre
        self.dni = dni
        self.codigo_estudiante = codigo_estudiante
        self.cursos = []

    def inscribirse(self, curso):
        self.cursos.append(curso)
        curso.agregar_estudiante(self)

    def mostrar_informacion(self):
        print(f"Estudiante: {self.nombre} DNI: {self.dni} C digo: {
           self.codigo_estudiante}")
        print("Cursos inscritos:")
        for curso in self.cursos:
            print(f"  {curso.nombre_curso}")

class Profesor:
    def __init__(self, nombre, dni, especialidad):
        self.nombre = nombre
        self.dni = dni
        self.especialidad = especialidad

    def mostrar_informacion(self):
```

```
25          print(f"Profesor: {self.nombre} DNI: {self.dni},
                Especialidad: {self.especialidad}")
26
27  class Curso:
28      def __init__(self, nombre_curso, profesor):
29          self.nombre_curso = nombre_curso
30          self.profesor = profesor
31          self.estudiantes = []
32
33      def agregar_estudiante(self, estudiante):
34          if estudiante not in self.estudiantes:
35              self.estudiantes.append(estudiante)
36
37      def mostrar_detalles(self):
38          print(f"\nCurso: {self.nombre_curso}")
39          self.profesor.mostrar_informacion()
40          print("Estudiantes inscritos:")
41          for est in self.estudiantes:
42              print(f"  {est.nombre} ({est.codigo_estudiante})")
43
44  class Universidad:
45      def __init__(self, nombre):
46          self.nombre = nombre
47          self.cursos = []
48
49      def agregar_cursos(self, curso):
50          self.cursos.append(curso)
51
52      def mostrar_cursos(self):
53          for curso in self.cursos:
54              curso.mostrar_detalles()
55
56  # Creaci n de objetos
57  prof1 = Profesor("Ing. Coyla Leonel", "01323043", "Programaci n")
58  prof2 = Profesor("Ing. Tito Jose", "02839212", "Programaci n")
59  prof3 = Profesor("Dr. Valvede Confesor", "02839312", "Estad stica")
60  prof4 = Profesor("Ing. Torres Fred", "03987412", "Programaci n")
61  prof5 = Profesor("Ing. Roque Elvis", "94847311", "Estad stica")
62  prof6 = Profesor("Ing. Rossel Luis", "83474711", "Programaci n")
63
64  curso1 = Curso("Lenguajes de Programaci n II", prof1)
65  curso2 = Curso("Sistema de gesti n de base de datos", prof2)
66  curso3 = Curso("Modelos discretos", prof3)
67  curso4 = Curso("Programaci n num rica", prof4)
68  curso5 = Curso("Inferencia estad stica", prof5)
69  curso6 = Curso("An lisis y dise os de sistemas de informaci n",
        prof6)
70
```

```python
est1 = Estudiante("Milena Kely", "01345621", "2025007")
est2 = Estudiante("Henry Quispe", "23345182", "2025089")

univ = Universidad("Universidad Nacional del Altiplano")
univ.agregar_cursos(curso1)
univ.agregar_cursos(curso2)

est1.inscribirse(curso1)
est1.inscribirse(curso2)
est1.inscribirse(curso3)
est1.inscribirse(curso4)
est1.inscribirse(curso5)
est1.inscribirse(curso6)
est2.inscribirse(curso2)

print(univ.nombre)
univ.mostrar_cursos()
est1.mostrar_informacion()
est2.mostrar_informacion()
```

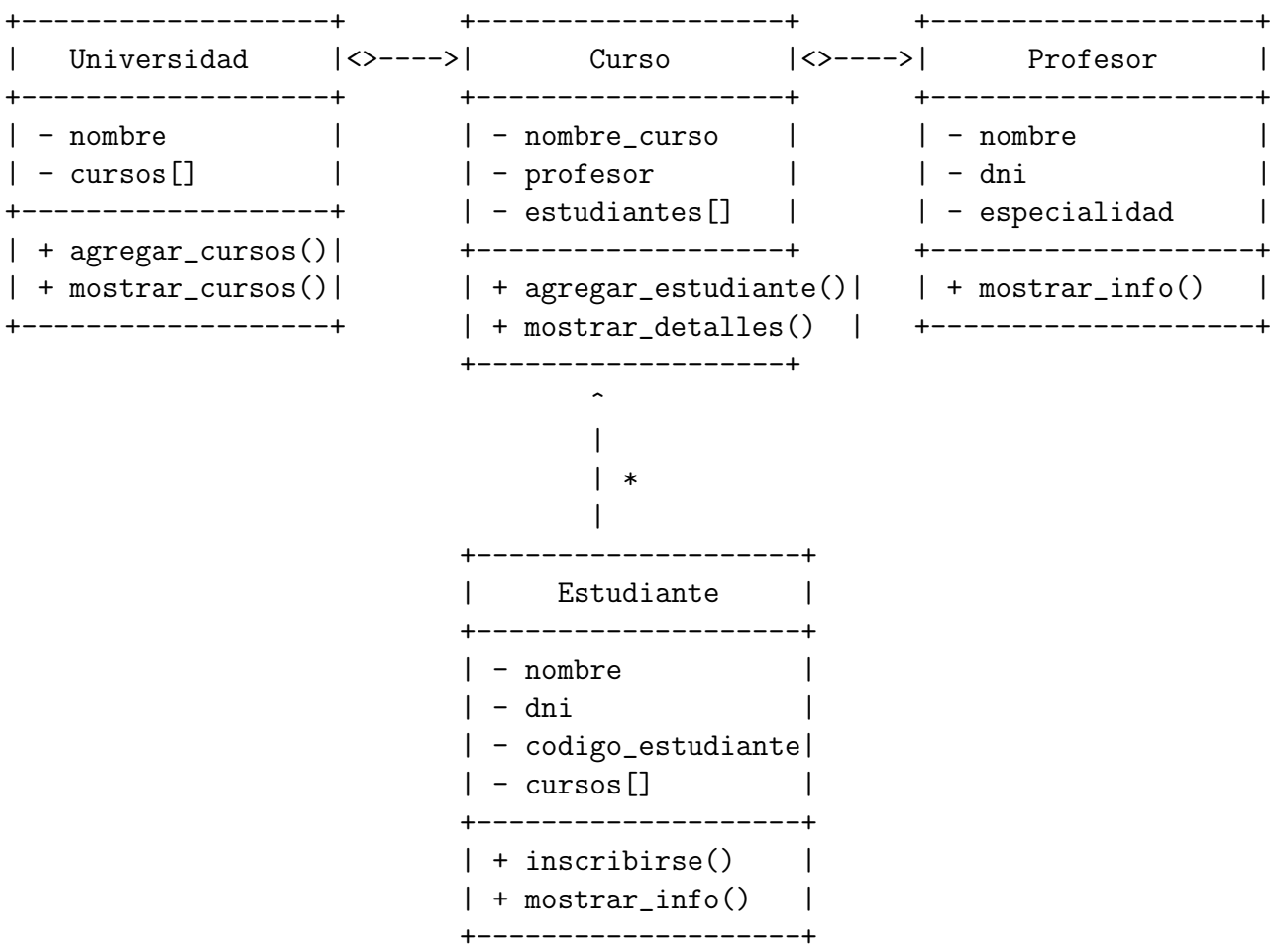
Listing 4: Código Composición Ejemplo 2

Ejecución:

```
Universidad Nacional del Altiplano

Curso: Lenguajes de Programación II
Profesor: Ing. Coyla Leonel DNI: 01323043, Especialidad: Programación
Estudiantes inscritos:
  Milena Kely (2025007)

Curso: Sistema de gestión de base de datos
Profesor: Ing. Tito Jose DNI: 02839212, Especialidad: Programación
Estudiantes inscritos:
  Milena Kely (2025007)
  Henry Quispe (2025089)

Estudiante: Milena Kely DNI: 01345621 Código: 2025007
Cursos inscritos:
  Lenguajes de Programación II
  Sistema de gestión de base de datos
  Modelos discretos
  Programación numérica
  Inferencia estadística
  Análisis y diseños de sistemas de información

Estudiante: Henry Quispe DNI: 23345182 Código: 2025089
```

```
Cursos inscritos:
  Sistema de gestión de base de datos
```

Relación (diagrama textual):




```
Relaciones:
- Universidad <>--* Curso → Agregación (una universidad contiene muchos cursos)
- Curso <>--1 Profesor → Asociación (cada curso tiene un profesor)
- Estudiante *--* Curso → Asociación bidireccional (muchos a muchos)
```

Interpretación:

Universidad – Curso → Agregación: La universidad contiene varios cursos. Los cursos pueden existir independientemente.

Curso – Profesor → Asociación (1 a 1): Cada curso tiene exactamente un profesor.

Estudiante – Curso → Asociación bidireccional (N a N): Un estudiante puede inscribirse en varios cursos y cada curso puede tener varios estudiantes.