Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Docente:

Ing. Coyla Idme Leonel

Alumno:

Ticona Miramira Roberto Angel

Relaciones entre clases

En la POO en Python, las relaciones entre clases permiten modelar como los objetos interactuan entre si en un sistema. Existen 3 relaciones fundamentales que definen como 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.
- Acción:
- Objeto: prof = Profesor(”Dr. Morillos”)
- Clase: Curso.
- Atributos: nombre, profesor.
- Acción:
- Objeto: curso = Curso(”Muestreo”, prof)

Código

```python
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)
```

Ejecución

```
Dr. Morillos
```

Relación

```
+----------------+           +----------------+
|    Profesor    |           |     Curso      |
+----------------+           +----------------+
| - nombre       |           | - nombre       |
|                |           | - profesor     |
+----------------+           +----------------+
        ^                            |
        | 1                          | 1
        +----------- tiene ---------->+
```

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).

» Agregación

Es un tipo especial de asociación. Se refiere a una relación de ”todo-parte” donde una clase (el todo) contiene a otras (las partes), pero las partes pueden existir independientemente del todo.

Características

- Relación ”tiene un” (has a).
- El objeto contenido no es destruido si el objeto contenedor desaparece.
- Los objetos se pueden reutilizar en otras partes.

Ejemplo

Crear las clases departamento y universidad con sus respectivos atributos y establecer el tipo de relación.

- Clase: Departamento.
- Atributo: nombre.
- Acción:
- Objetos: dep1 = Departamento(”Ingeniería Estadística”), dep2 = Departamento(”Informática”)
- Clase: Universidad.
- Atributo: nombre.
- Acción: agregarDepartamento()
- Objetos: uni = Universidad(”Universidad Nacional del Altiplano”)

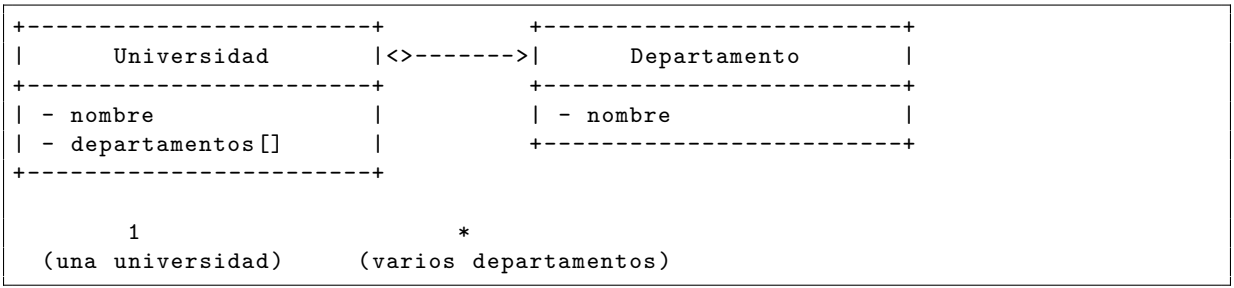
Código

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

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

    def agregarDepartamento(self, departamento):
        self.departamentos.append(departamento)

dep1 = Departamento("Ingeniería Estadística")
dep2 = Departamento("Informática")

uni = Universidad("Universidad Nacional del Altiplano")

uni.agregarDepartamento(dep1)
uni.agregarDepartamento(dep2)

for d in uni.departamentos:
    print(d.nombre)
```

Ejecución

```
Ingeniería Estadística
Informática
```

Relación



Interpretación

- Universidad tiene una lista (departamentos) que almacena varios objetos de tipo Departamento.
- La relación es uno a muchos (1 -> *).
- El símbolo <> indica agregación (la universidad contiene, pero los departamentos pueden existir por separado).

» Composición

Es un caso más fuerte que la agregación, también es una relación ”todo-parte”, pero con propiedad total y dependencia de ciclo de vida. Si el objeto contenedor se destruye, sus partes también.

Características

- Relación fuerte de pertenencia.
- Se crean y destruyen junto con el objeto contenedor.

Ejemplo 1

Crear las clases motor y auto con sus respectivos atributos y establecer su relación.

- Clase: Motor.
- Atributo: tipo.
- Acción: encender()
- Objeto:
- Clase: Auto.
- Atributo: marca.
- Acción: arrancar()
- Objeto: miAuto = Auto(”Toyota”)

Código

```
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()
```

Ejecución

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

Relación

```
+------------------+       --------------------+
|       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, universidad cada uno con sus respectivos atributos y establecer el tipo de relación

- Clase: Estudiante.
- Atributos: nombre, dni, codigo_estudiante.
- Acciones: inscribirse(), mostrar_informacion()
- Objeto: est1 = Estudiante(), est2 = Estudiante()
- Clase: Profesor.
- Atributos: nombre, dni, especialidad.
- Acción: mostrar_informacion()
- Objeto: prof1 = Profesor(), prof2 = Profesor(), ... , prof6 = Profesor()
- Clase: Curso.
- Atributos: nombre_curso, profesor.
- Acción: agregar_estudiante(), mostrar_detalles()
- Objeto: curso1 = Curso(), curso2 = Curso(), ... , curso6 = Curso()
- Clase: Universidad.
- Atributos: nombre.
- Acción: agregar_cursos, mostrar_cursos()
- Objeto: univ = Universidad("Universidad Nacional del Altiplano")

Código

```
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} Código:
            ↪ {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
```

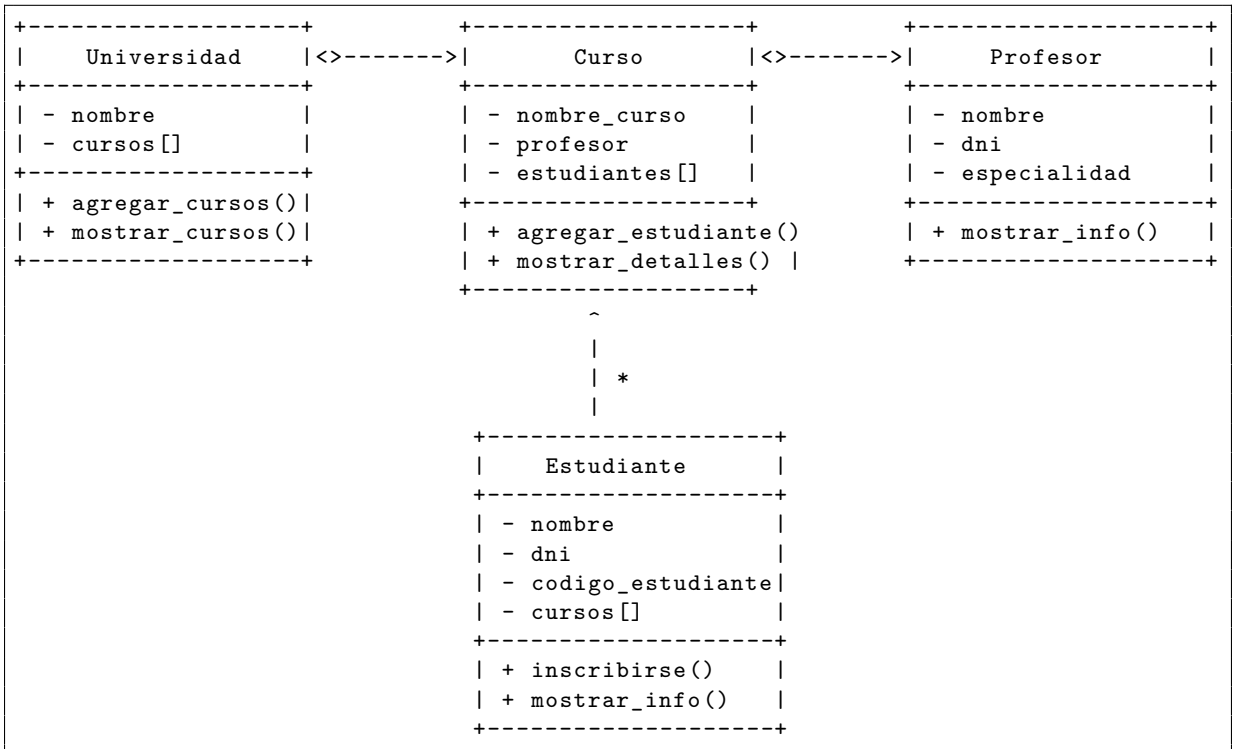
```
23
24      def mostrar_informacion(self):
25          print(f"Profesor: {self.nombre} DNI: {self.dni}, Especialidad:
                ↪ {self.especialidad}")
26
27  class Curso:
28      def __init__(self, nombre_curso, profesor):
29          self.nombre_curso = nombre_curso
30          self.profesor = 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  prof1 = Profesor("Ing. Coyla Leonel", "01323043", "Programación")
57  prof2 = Profesor("Ing. Tito Jose", "02839212", "Programación")
58  prof3 = Profesor("Dr. Valvede Confesor", "02839312", "Estadística")
59  prof4 = Profesor("Ing. Torres Fred", "03987412", "Programación")
60  prof5 = Profesor("Ing. Roque Elvis", "94847311", "Estadística")
61  prof6 = Profesor("Ing. Rossel Luis", "83474711", "Programación")
62
63  curso1 = Curso("Lenguajes de Programación II", prof1)
64  curso2 = Curso("Sistema de gestión de base de datos", prof2)
65  curso3 = Curso("Modelos discretos", prof3)
66  curso4 = Curso("Programación numérica", prof4)
67  curso5 = Curso("Inferencia estadística", prof5)
68  curso6 = Curso("Análisis y diseños de sistemas de información", prof6)
69
70  est1 = Estudiante("Milena Kely", "01345621", "2025007")
71  est2 = Estudiante("Henry Quispe", "23345182", "2025089")
72
73  univ = Universidad("Universidad Nacional del Altiplano")
74  univ.agregar_cursos(curso1)
75  univ.agregar_cursos(curso2)
76
77  est1.inscribirse(curso1)
78  est1.inscribirse(curso2)
79  est1.inscribirse(curso3)
80  est1.inscribirse(curso4)
81  est1.inscribirse(curso5)
82  est1.inscribirse(curso6)
83  est2.inscribirse(curso2)
84
```

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

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




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

```
30 - 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.