Tipos de métodos e interacciones entre objetos

Colaboración y composición

{desafío} latam_

Codificar un programa con clases, atributos y métodos utilizando colaboración y composición para resolver un problema de baja complejidad acorde al lenguaje Python.

- Unidad 1:
  Introducción a la programación orientada a objetos con Python
- Unidad 2:
  Tipos de métodos e interacciones entre objetos
- Unidad 3:
  Herencia y polimorfismo
- Unidad 4:
  Manejo de errores y archivos

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Describe los conceptos de colaboración y composición distinguiendo sus diferencias de acuerdo al paradigma de orientación a objetos.*

{desafío} latam_

¿A qué se refiere la colaboración en programación orientada a objetos?

{desafío} latam_

/* Colaboración */

Colaboración en Python

Que una clase colabore con otra, quiere decir que una clase debe ser **instanciada** dentro de la otra, en este caso, se creará la clase Superficie, y luego la clase Pelota. La clase Pelota debe instanciar dentro de ella a la clase Superficie, para poder generar un rebote (se ha simplificado para enfocarnos en la colaboración).

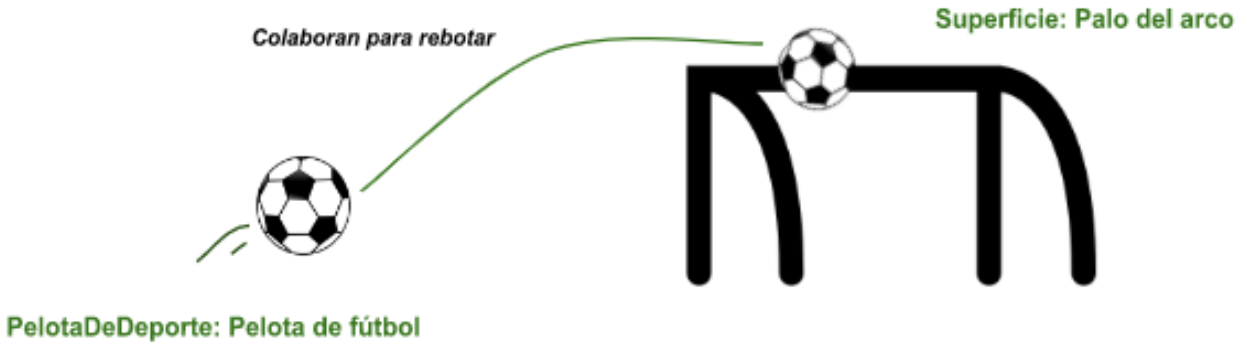
```
class Superficie():
    def __init__(self):
        self.__resistencia = 2

    @property
    def resistencia(self):
        return self.__resistencia

class Pelota():
    def rebotar(self, altura: float):
        # Se instancia la clase que colabora con Pelota
        s = Superficie()
        rebotes = []
        while altura > 0:
           rebotes.append(altura)
           rebotes.append(0)
           # La instancia de Superficie colabora con Pelota para rebotar
           altura //= s.resistencia
        return rebotes

p = Pelota()
# Salida: [10, 0, 5, 0, 2, 0, 1, 0]
print(p.rebotar(10))
```

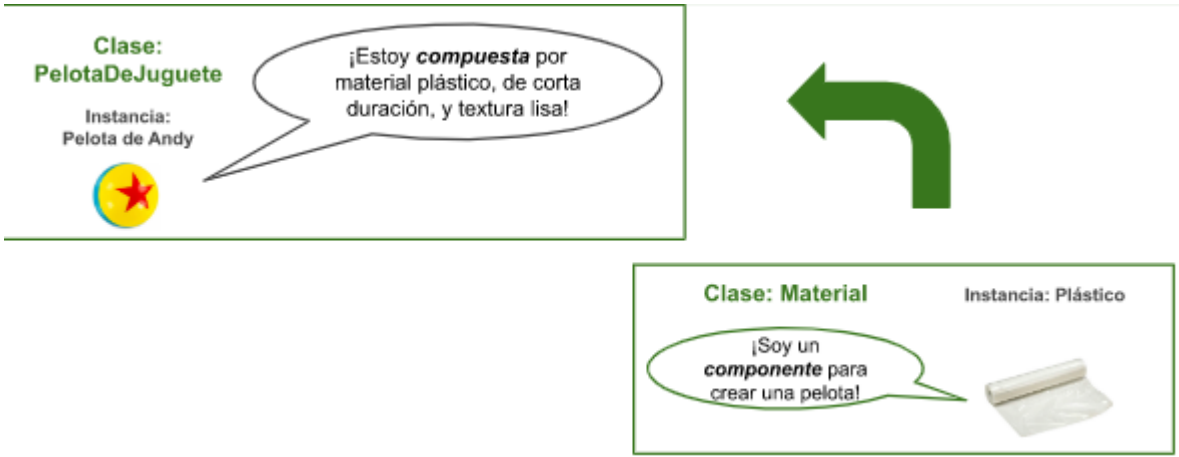
¿Qué es la colaboración entre objetos?



Los objetos que forman parte de una colaboración no dependen uno del otro para existir.

Composición de objetos

La composición es otra forma de **interacción entre objetos de distinta clase**, donde una clase tiene un atributo que es instancia de otra clase. Esta, posee el atributo se denomina "**clase compuesta**", mientras que la clase a la cual pertenece el atributo de la clase compuesta se denomina "**clase componente**".



Composición

La composición es también llamada **agregación fuerte**, siendo la agregación la interacción entre objetos donde una clase padre tiene una clase hija, como atributo de ella. En la composición, la clase padre corresponde a la clase compuesta, y la clase hija corresponde a la clase componente.

La diferencia entre la **agregación "normal"** y la **agregación "fuerte"** (composición), es que en la primera la instancia de la clase hija puede existir en forma independiente de su clase padre, en cambio, en la composición la instancia específica del componente requiere necesariamente de la existencia de la clase compuesta para existir.

Agregación en Python

Para condicionar que la instancia hija pueda existir en forma independiente de la instancia padre, se debe:

- dar al constructor de la clase padre una instancia de la clase hija como argumento.
- asignarlo a un atributo de instancia.

De esta forma, se debe primero crear una instancia de la clase hija (es decir, puede existir por sí misma), que se usa como argumento para crear la clase padre.

Recordar que Python no es estrictamente tipado, por lo que si no se indica el tipo de dato del atributo en el constructor, el uso de la agregación es difícil de deducir.

Agregación en Python

Ejemplo

```python
class Material():
    def __init__(self, nombre: str, duracion: str, textura: str):
        self.nombre = nombre
        self.duracion = duracion
        self.textura = textura

class Pelota():
    def __init__(self, tamanio: int, color: str, material: Material):
        self.tamanio = tamanio
        self.color = color
        # La pelota tiene un material
        self.material = material

# El material existe en forma independiente de la pelota
m = Material("Plástico", "Corta", "Lisa")
p = Pelota(16, "Amarillo", m)

# Salida: <class '__main__.Material'>
print(type(p.material))
# Salida: Plástico
print(p.material.nombre)
```

/* Composición */

Composición en Python

Para condicionar que una instancia hija no pueda existir por sí sola independiente de una clase padre, se debe:

- crear la instancia de la clase hija (componente) dentro del constructor de la clase padre (compuesto).
- asignar esta instancia a un atributo, es decir, el componente no se debe usar como argumento al momento de crear la instancia.

Por lo tanto, la clase compuesta debe contener la información necesaria para crear la instancia de la clase componente dentro de su constructor.

Composición en Python

Ejemplo

```python
from abc import ABC, abstractmethod
class Material(ABC):
   @abstractmethod
   def romper(self):
       pass
class MaterialPlastico(Material):
   nombre = "Plástico"
   duracion = "Corta"
   def __init__(self, textura: str):
       self.textura = textura
   def romper(self):
       pass
class Pelota():
   def __init__(self, tamanio: int, color: str, textura: str):
       self.tamanio = tamanio
       self.color = color
       self.textura = textura
       # La pelota está compuesta por un componente material
       self.material = MaterialPlastico(self.textura)

p = Pelota(16, "Amarillo", "Lisa")
# Salida: Plástico
print(p.material.nombre)
```

Usar conjuntamente objetos de distintas clases

Mediante colaboración y composición

Se puede usar conjuntamente colaboración y composición, complementando los usos de cada una, para resolver una problemática utilizando el paradigma orientado a objetos.

A su vez, con el fin de crear una mejor estructura del programa, que sea fácil de escalar en el tiempo, y que ayude a evitar la generación de errores, se puede también hacer uso de clases abstractas que sirvan de plantilla para distintos tipos similares de objetos. Y por su lado, cada clase puede encapsular su estado, haciendo uso de propiedades para controlar el acceso y modificación de sus atributos.

Ejercicio guiado

"Módulo de venta de productos"

{desafío} latam_

Módulo de venta de productos

Desde la tienda “Mis Mascotas”, la cual se dedica a la venta de productos para mascotas (perros, gatos y exóticos), te han solicitado desarrollar un prototipo para el backend del módulo de venta de su sitio web de e-commerce. El prototipo debe ser una aplicación de consola en Python, donde los ingresos de cada detalle de venta (producto y cantidad) deben realizarse mediante input.

Por ahora, solo se debe considerar el detalle de la venta (tanto su ingreso como mostrarlo por pantalla). El detalle de la venta está constituido a su vez por una lista de ítems, donde cada ítem posee un nombre del producto y una cantidad.

Al ejecutarse el script principal del programa, el cual debe importar la o las clases necesarias para su funcionamiento, se debe solicitar el ingreso del nombre del producto del ítem del detalle de venta y su cantidad asociada. Además, se debe seguir solicitando ítems hasta que el usuario indique lo contrario. Una vez finalizado el ingreso de ítems, se debe mostrar en pantalla el detalle de la venta.

Módulo de venta de productos

Solución

Paso 1

Antes de escribir el código, se debe identificar las clases presentes en el problema, y las relaciones entre ellas. En este caso, las clases identificadas son 3:

- Venta
- DetalleVenta
- DetalleVentaItem

Módulo de venta de productos

Solución

Paso 2

Identificar las relaciones entre las distintas clases, que permitirán resolver el algoritmo del problema:

Venta y DetalleVenta:

- Venta tiene un DetalleVenta.
- Dado el contexto explicado, un DetalleVenta no puede existir en forma independiente, sino que siempre se asocia a una Venta específica.
- Por lo tanto, la relación entre estas clases es de composición, siendo la Venta el compuesto y el DetalleVenta el componente.

Venta y DetalleVentaItem:

- Venta usa un DetalleVentaItem para agregar un ítem a su detalle.
- Para que exista una Venta, no es necesario que posea un DetalleVentaItem; solo lo requiere para realizar la acción de agregar ítems al detalle.
- Por lo tanto, la relación de estas clases es de colaboración, donde DetalleVentaItem colabora con Venta para agregar un ítem a su detalle.

Módulo de venta de productos

Solución

Paso 2

Identificar las relaciones entre las distintas clases, que permitirán resolver el algoritmo del problema:

DetalleVenta y DetalleVentaItem:

- Venta tiene una lista de DetalleVentaItem.
- El DetalleVentaItem se agrega una vez ya creada la instancia DetalleVenta. Es decir, DetalleVentaItem existe de forma independiente de DetalleVenta (se crea dentro de Venta).
- Por lo tanto, la relación de estas clases es de agregación, siendo DetalleVenta la clase padre, y DetalleVentaItem la clase hija.

Módulo de venta de productos

Solución

Paso 3

En un archivo **venta.py**, se crea la clase de menor jerarquía, que en este caso es **DetalleVentaItem**. Siguiendo las buenas prácticas, se define sus atributos de instancia (producto y cantidad) como atributos privados con valores asignables por parámetro en el constructor.

```
# archivo venta.py
class DetalleVentaItem():
   def __init__(self, producto: str, cantidad: int):
       self.__producto = producto
       self.__cantidad = cantidad
```

Módulo de venta de productos

Solución

Paso 4

A continuación, en la misma clase, se define las propiedades de producto y cantidad para poder acceder a los valores de los atributos privados. Los valores solo se asignan al momento de crear la instancia, por lo que no es necesario crear setters.

```
@property
    def producto(self):
        return self.__producto

    @property
    def cantidad(self):
        return self.__cantidad
```

{desafío} latam_

Módulo de venta de productos

Solución

Paso 5

Luego, se crea la segunda clase de menor jerarquía, que sería **DetalleVenta**. En su constructor, sólo se define la lista de ítems como atributo privado, como una lista vacía (no se tiene ítems al momento de crear el detalle de venta).

```
class DetalleVenta():
   def __init__(self):
       self.__items = []
```

Módulo de venta de productos

Solución

Paso 6

En la misma clase, definir el método **agregar_item**, el cual recibe por parámetro la instancia de la clase y una instancia de **DetalleVentaItem**. Dentro del método, se agrega la instancia de ítem recibida a la lista de ítems de la instancia de la clase.

```
def agregar_item(self, item: DetalleVentaItem):
        self.__items.append(item)
```

Módulo de venta de productos

Solución

Paso 7

En la misma clase, sobrecargar el método **`__str__`** de forma que al usar una instancia de **DetalleVenta** como argumento de print, se muestre en pantalla la información de todos los ítems ingresados.

```
def __str__(self):
        retorno = ("::::::::: DETALLE DE ESTA VENTA :::::::::\n"
        "PRODUCTO\tCANTIDAD\n")

        items = [
            f"{i.producto}\t\t{i.cantidad}\n"
            for i in self.__items
        ]
        return f"{retorno}{''.join(items)}"
```

Módulo de venta de productos

Solución

Paso 8

Finalmente, se crea la clase de mayor jerarquía, que en este caso es Venta. En su constructor, se asigna una instancia de **DetalleVenta** (creada dentro del constructor) al atributo privado de detalle.

```
class Venta():
   def __init__(self):
       self.__detalle = DetalleVenta()
```

Módulo de venta de productos

Solución

Paso 9

En la misma clase, crear un método de instancia **modificar_detalle**, que reciba por parámetro la instancia, el nombre del producto dado por el usuario para agregar al detalle, y su cantidad asociada.

```
def modificar_detalle(self, producto: str, cantidad: int):
```

Módulo de venta de productos

Solución

Paso 10

Dentro del método anterior, generar una instancia de **DetalleVentaItem** con el producto y cantidad ingresados. Luego usar esa instancia como argumento para el método **agregar_item** del detalle de la instancia.

```
def modificar_detalle(self, producto: str, cantidad: int):
        detalle_venta_item = DetalleVentaItem(producto, cantidad)
        self.__detalle.agregar_item(detalle_venta_item)
```

Módulo de venta de productos

Solución

Paso 11

Para terminar la clase **Venta**, se debe agregar la propiedad detalle, la cual retorna la instancia de detalle dentro de la clase.

```
    @property
    def detalle(self):
        return self.__detalle
```

Paso 12

En un archivo llamado **programa.py**, importe la clase Venta del módulo venta y cree una instancia de ella.

```
#archivo programa.py
from venta import Venta
venta = Venta()
```

Módulo de venta de productos

Solución

Paso 13

Consultar al usuario si desea agregar un ítem a la venta y almacenar opción.

```
opcion = int(input(
    "¿Desea ingresar un ítem a la venta?\n"
    "1. Sí\n2. No\n"))
```

Paso 14

Mientras la opción sea 1, consultar y almacenar en variables el producto y cantidad asociada para el ítem de la venta.

```
while opcion == 1:
   producto = input("\nIngrese nombre del producto vendido:\n")
   cantidad = int(input("\nIngrese cantidad vendida del producto:\n"))
```

Módulo de venta de productos

Solución

Paso 15

A continuación, dentro del ciclo, llamar al método modificar_detalle de la instancia de venta creada, usando como argumento el producto y la cantidad ingresados por el usuario.

```
venta.modificar_detalle(producto, cantidad)
```

Paso 16

Al final del ciclo, volver a consultar si se desea ingresar un ítem a la venta.

```
    opcion = int(input(
        "\n¿Desea ingresar un ítem a la venta?\n"
        "1. Sí\n2. No\n"))
```

Módulo de venta de productos

Solución

Paso 17

Finalmente, fuera del ciclo, imprimir el detalle de la venta ingresada.

```
print(venta.detalle)
```

Módulo de venta de productos

Solución

Ejemplo de salida

```
¿Desea ingresar un ítem a la venta?
1. Sí
2. No
1

Ingrese nombre del producto vendido:
spikes

Ingrese cantidad vendida del producto:
1

¿Desea ingresar un ítem a la venta?
1. Sí
2. No
1

Ingrese nombre del producto vendido:
k/d

Ingrese cantidad vendida del producto:
2

¿Desea ingresar un ítem a la venta?
1. Sí
2. No
2
:::::::: DETALLE DE ESTA VENTA :::::::::
PRODUCTO        CANTIDAD
spikes          1
k/d             2
```

¿Cómo se genera colaboración a nivel de código?

¿A qué se refiere la composición en programación orientada a objetos?

Próxima sesión...

- *Desafío evaluado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales