Anexo. Introducción a la herencia múltiple.

Vicente Benjumea García

Programación-II
Departamento de Lenguajes y Ciencias de la Computación.
E.T.S.I. Informática. Univ. de Málaga.

Anexo. Introducción a la herencia múltiple.

- Herencia múltiple y MRO.
- Herencia múltiple y MRO. Ejemplo.

Herencia múltiple y MRO

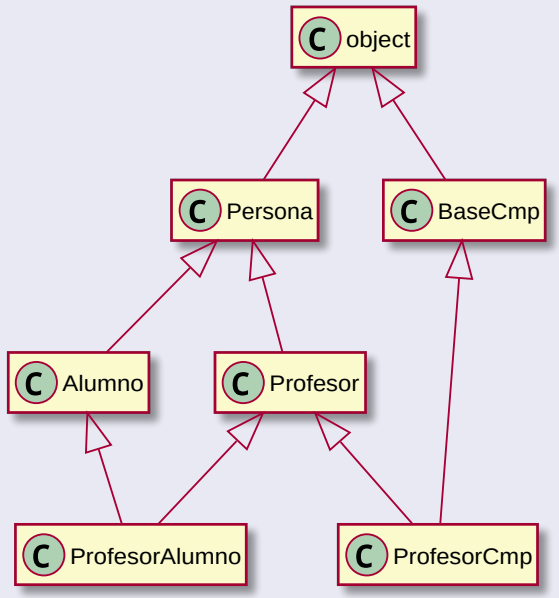
- La **herencia múltiple** se produce cuando una subclase hereda de **varias** superclases.
- La **herencia múltiple** puede dar lugar a relaciones jerárquicas **complejas**, por lo que no se debe abusar de esta característica.
- Usualmente existen dos situaciones donde la herencia múltiple es necesaria:
  1. La nueva clase que estamos definiendo presenta relaciones **es-un** con varias clases de **diferentes líneas jerárquicas**.
     - Un profesor que esté estudiando en la universidad, es un *Profesor*, y también es un *Alumno*.
  2. La nueva clase presenta una relación **es-un** con otra clase, pero además hereda de otras clases que proporcionan **funcionalidades adicionales** (*mixin*).
     - Un profesor-cmp es un *Profesor*, pero adicionalmente hereda de otra clase que proporciona las operaciones de comparación (*BaseCmp*).

- Definición de una clase derivada de múltiples clases base:

```
class SubClase(SuperClase1, SuperClase2, ...):
```

- Por ejemplo:

```
class Persona:  # hereda de object
    pass
class Profesor(Persona):
    pass
class Alumno(Persona):
    pass
class ProfesorAlumno(Profesor,Alumno):
    pass
class BaseCmp:  # hereda de object
    pass
class ProfesorCmp(Profesor, BaseCmp):
    pass
```



Herencia múltiple y MRO

- El **Orden de Resolución de Métodos** (**MRO**) (*Method Resolution Order*) determina la línea de invocación a los métodos de las superclases.
  - Para cada clase, el *MRO* se puede consultar:

```
print(Persona.__mro__)          # (Persona, object)
print(Alumno.__mro__)           # (Alumno, Persona, object)
print(Profesor.__mro__)         # (Profesor, Persona, object)
print(ProfesorAlumno.__mro__)   # (ProfesorAlumno, Profesor, Alumno, Persona, object)
print(BaseCmp.__mro__)          # (BaseCmp, object)
print(ProfesorCmp.__mro__)      # (ProfesorCmp, Profesor, Persona, BaseCmp, object)
print([c.__name__ for c in ProfesorCmp.__mro__])
```

- En caso de **herencia múltiple**, el **MRO no coincide** con la línea jerárquica de herencia, ya que esta última **no es lineal**.
  - Podemos apreciar como, en el MRO de la clase *ProfesorAlumno*, la clase siguiente a *Profesor* es **Alumno**, mientras que en los MRO de las clases *Profesor* y *ProfesorCmp*, la clase siguiente a *Profesor* es **Persona**.

Invocación a métodos de la superclase

- Cuando se invoca a un método de la superclase (`super().__init__()`, `super().metodo()`), mientras se programa el código, **no se sabe cual será la superclase**, ya que la clase podría estar en una jerarquía de herencia múltiple en un determinado momento, como el caso de la clase *Profesor* del ejemplo anterior.

Herencia múltiple y MRO

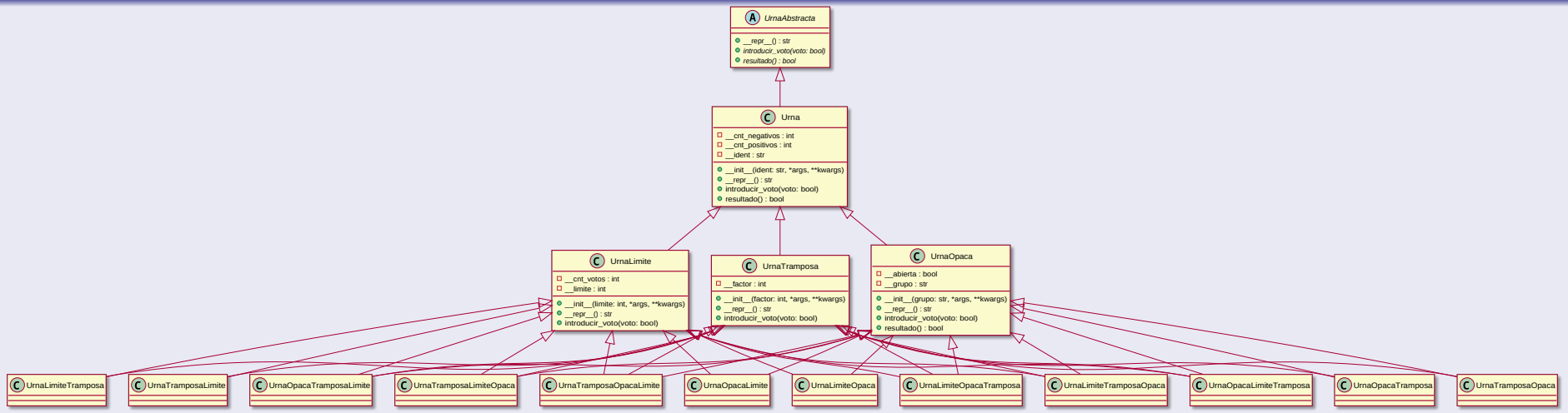
Herencia múltiple y creación de Objetos

- Para crear un objeto de una determinada clase, se invoca a su **constructor**, con el nombre de la clase, y entre paréntesis los **argumentos** necesarios para su construcción e inicialización. Dos alternativas para pasar los argumentos:
  1. Si los **nombres** de los parámetros son **distintos** en todos los constructores de la jerarquía de herencia de la clase, entonces se pueden proporcionar los **argumentos con nombre**, para todos los constructores, en cualquier orden.
  2. Conocemos el MRO de la clase del objeto que estamos creando, entonces se pueden proporcionar los **argumentos posicionales**, para todos los constructores, según el **orden del MRO** de la clase del objeto que estamos creando.

```
persona = Persona(nombre="María")                                    # Argumentos con nombre
alumno = Alumno(nombre="Pepe", titulacion="CIA")                     # Argumentos con nombre
profesor = Profesor(nombre="Juan", asignatura="Programación")        # Argumentos con nombre

profesoralumno = ProfesorAlumno(nombre="Ana", asignatura="Programación", titulacion="CIA") # Args nombre
basecmp = BaseCmp()                                                  # Argumentos con nombre
profesorcmp = ProfesorCmp(nombre="Eva", asignatura="Programación")   # Argumentos con nombre

persona = Persona("María")                                           # MRO(Per, Obj)
alumno = Alumno("CIA", "Pepe")                                       # MRO(Alum, Per, Obj)
profesor = Profesor("Programación", "Juan")                          # MRO(Prof, Per, Obj)

profesoralumno = ProfesorAlumno("Programación", "CIA", "Ana")        # MRO(ProfAlum, Prof, Alum, Per, Obj)
basecmp = BaseCmp()                                                  # MRO(BCmp, Obj)
profesorcmp = ProfesorCmp("Programación", "Eva")                     # MRO(ProfCmp, Prof, Per, BCmp, Obj)
```

Herencia múltiple y MRO. Ejemplo (I)

Diagrama de clases de la jerarquía



- `UrnaAbstracta`: es la base de la jerarquía de clases de urnas. Especifica el comportamiento básico, común a las urnas, introducir votos y mostrar el resultado de la votación.
- `Urna(UrnaAbstracta)`: define el comportamiento base de la urna, contabilizar los votos introducidos, y mostrar el resultado comparando la cuenta de votos.
- `UrnaLimite(Urna)`: sólo permite introducir un número de votos menor o igual a un límite pre-establecido. El resto de votos se ignoran.
- `UrnaOpaca(Urna)`: solo permite introducir votos mientras la votación está abierta. El resto de votos se ignoran. Cuando se consulta el resultado, se cierra la votación.
- `UrnaTramposa(Urna)`: intenta revertir el resultado de la votación. Si se introduce un voto distinto del resultado en ese momento, ese voto se multiplica por un determinado factor pre-establecido.
- Múltiples combinaciones de las clases anteriores . . .

Herencia múltiple y MRO. Ejemplo (II)

```python
from abc import ABC, abstractmethod

class UrnaAbstracta(ABC):

    @abstractmethod
    def introducir_voto(self, voto: bool) -> None:
        pass

    @abstractmethod
    def resultado(self) -> bool:
        pass

    # @override
    def __repr__(self) -> str:
        return f"UA()"
```

Herencia múltiple y MRO. Ejemplo (III)

```
class Urna(UrnaAbstracta):

    def __init__(self, ident: str, *args, **kwargs) -> None:
        self.__ident = ident
        self.__cnt_positivos = 0
        self.__cnt_negativos = 0
        super().__init__(*args, **kwargs)

    # @override
    def introducir_voto(self, voto: bool) -> None:
        if voto:
            self.__cnt_positivos += 1
        else:
            self.__cnt_negativos += 1

    # @override
    def resultado(self) -> bool:
        return (self.__cnt_positivos >= self.__cnt_negativos)

    # @override
    def __repr__(self) -> str:
        return (f"U({self.__ident}, {self.__cnt_positivos}, {self.__cnt_negativos},"
                f" {super().__repr__()})")
```

Herencia múltiple y MRO. Ejemplo (IV)

```python
class UrnaOpaca(Urna):

    def __init__(self, grupo: str, *args, **kwargs) -> None:
        self.__grupo = grupo
        self.__abierta = True
        super().__init__(*args, **kwargs)

    # @override
    def introducir_voto(self, voto: bool) -> None:
        if self.__abierta:
            super().introducir_voto(voto)
        # en otro caso, se ignora el voto

    # @override
    def resultado(self) -> bool:
        self.__abierta = False
        return super().resultado()

    # @override
    def __repr__(self) -> str:
        return f"UO({self.__grupo}, {self.__abierta}, {super().__repr__()})"
```

Herencia múltiple y MRO. Ejemplo (V)

```python
class UrnaLimite(Urna):

    def __init__(self, limite: int, *args, **kwargs) -> None:
        self.__limite = limite
        self.__cnt_votos = 0
        super().__init__(*args, **kwargs)

    # @override
    def introducir_voto(self, voto: bool) -> None:
        if self.__cnt_votos < self.__limite:
            super().introducir_voto(voto)
            self.__cnt_votos += 1  # después, por si introducir_voto lanza excepcion
        # en otro caso, se ignora el voto

    # @override
    def __repr__(self) -> str:
        return f"UL({self.__limite}, {self.__cnt_votos}, {super().__repr__()})"
```

Herencia múltiple y MRO. Ejemplo (VI)

```python
class UrnaTramposa(Urna):

    def __init__(self, factor: int, *args, **kwargs) -> None:
        self.__factor = factor
        super().__init__(*args, **kwargs)

    # @override
    def introducir_voto(self, voto: bool) -> None:
        if self.resultado() == voto:
            super().introducir_voto(voto)
        else:
            for idx in range(self.__factor):
                super().introducir_voto(voto)

    # @override
    def __repr__(self) -> str:
        return f"UT({self.__factor}, {super().__repr__()})"
```

Herencia múltiple y MRO. Ejemplo (VII)

```
class UrnaTramposaLimite(UrnaTramposa, UrnaLimite):
    pass

class UrnaLimiteTramposa(UrnaLimite, UrnaTramposa):
    pass

class UrnaTramposaOpaca(UrnaTramposa, UrnaOpaca):
    pass

class UrnaOpacaTramposa(UrnaOpaca, UrnaTramposa):
    pass

class UrnaLimiteOpaca(UrnaLimite, UrnaOpaca):
    pass

class UrnaOpacaLimite(UrnaOpaca, UrnaLimite):
    pass
```

Herencia múltiple y MRO. Ejemplo (VIII)

```
class UrnaOpacaLimiteTramposa(UrnaOpaca, UrnaLimite, UrnaTramposa):
    pass

class UrnaOpacaTramposaLimite(UrnaOpaca, UrnaTramposa, UrnaLimite):
    pass

class UrnaLimiteOpacaTramposa(UrnaLimite, UrnaOpaca, UrnaTramposa):
    pass

class UrnaLimiteTramposaOpaca(UrnaLimite, UrnaTramposa, UrnaOpaca):
    pass

class UrnaTramposaLimiteOpaca(UrnaTramposa, UrnaLimite, UrnaOpaca):
    pass

class UrnaTramposaOpacaLimite(UrnaTramposa, UrnaOpaca, UrnaLimite):
    pass
```

Herencia múltiple y MRO. Ejemplo (IX)

```
VOTACION = [ "s s n n n n n", "s s s s s s n n" ]

def prueba_votacion(ident: str, urna: UrnaAbstracta) -> None:
    for idx in range(len(VOTACION)):
        print()
        print(ident, f"Iteración-{idx+1}")
        print(urna)
        for voto in VOTACION[idx].split():
            urna.introducir_voto(voto.lower() == "s")
        print(urna)
        print(ident, f"Iteración-{idx+1}", f"Resultado intermedio: {urna.resultado()}")
    print()
    print(urna)
    print(ident, f"Resultado final: {urna.resultado()}")
```

Herencia múltiple y MRO. Ejemplo (X)

```
def print_mros() -> None:
    clases = [ UrnaAbstracta, Urna, UrnaOpaca, UrnaLimite, UrnaTramposa,
               UrnaTramposaLimite, UrnaLimiteTramposa,
               UrnaTramposaOpaca, UrnaOpacaTramposa,
               UrnaLimiteOpaca, UrnaOpacaLimite,
               UrnaOpacaLimiteTramposa, UrnaOpacaTramposaLimite,
               UrnaLimiteOpacaTramposa, UrnaLimiteTramposaOpaca,
               UrnaTramposaLimiteOpaca, UrnaTramposaOpacaLimite ]
    print()
    print("MROs de las clases")
    for clase in clases:
        print([c.__name__ for c in clase.__mro__])
```

Herencia múltiple y MRO. Ejemplo (XI)

```
def prueba_args_nombre() -> None:
    lista_urnas = [
        # UrnaAbstracta(),   # Can't instantiate abstract class UrnaAbstracta
        Urna(ident="Ident-Votación"),
        UrnaOpaca(grupo="Grupo-Votación", ident="Ident-Votación"),
        UrnaLimite(limite=10, ident="Ident-Votación"),
        UrnaTramposa(factor=2, ident="Ident-Votación"),
        UrnaTramposaLimite(factor=2, limite=10, ident="Ident-Votación"),
        UrnaLimiteTramposa(limite=10, factor=2, ident="Ident-Votación"),
        UrnaTramposaOpaca(factor=2, grupo="Grupo-Votación", ident="Ident-Votación"),
        UrnaOpacaTramposa(grupo="Grupo-Votación", factor=2, ident="Ident-Votación"),
        UrnaLimiteOpaca(limite=10, grupo="Grupo-Votación", ident="Ident-Votación"),
        UrnaOpacaLimite(grupo="Grupo-Votación", limite=10, ident="Ident-Votación"),
        UrnaOpacaLimiteTramposa(grupo="Grupo-Votación", limite=10, factor=2, ident="Ident-Votación"),
        UrnaOpacaTramposaLimite(grupo="Grupo-Votación", factor=2, limite=10, ident="Ident-Votación"),
        UrnaLimiteOpacaTramposa(limite=10, grupo="Grupo-Votación", factor=2, ident="Ident-Votación"),
        UrnaLimiteTramposaOpaca(limite=10, factor=2, grupo="Grupo-Votación", ident="Ident-Votación"),
        UrnaTramposaLimiteOpaca(factor=2, limite=10, grupo="Grupo-Votación", ident="Ident-Votación"),
        UrnaTramposaOpacaLimite(factor=2, grupo="Grupo-Votación", limite=10, ident="Ident-Votación"),
        ]
    print()
    print("Prueba Argumentos con Nombre")
    for urna in lista_urnas:
        prueba_votacion(type(urna).__name__, urna)
```

Herencia múltiple y MRO. Ejemplo (XII)

```
def prueba_args_mro() -> None:
    lista_urnas = [
        # UrnaAbstracta(),   # Can't instantiate abstract class UrnaAbstracta
        Urna("Ident-Votación"),
        UrnaOpaca("Grupo-Votación", "Ident-Votación"),
        UrnaLimite(10, "Ident-Votación"),
        UrnaTramposa(2, "Ident-Votación"),
        UrnaTramposaLimite(2, 10, "Ident-Votación"),
        UrnaLimiteTramposa(10, 2, "Ident-Votación"),
        UrnaTramposaOpaca(2, "Grupo-Votación", "Ident-Votación"),
        UrnaOpacaTramposa("Grupo-Votación", 2, "Ident-Votación"),
        UrnaLimiteOpaca(10, "Grupo-Votación", "Ident-Votación"),
        UrnaOpacaLimite("Grupo-Votación", 10, "Ident-Votación"),
        UrnaOpacaLimiteTramposa("Grupo-Votación", 10, 2, "Ident-Votación"),
        UrnaOpacaTramposaLimite("Grupo-Votación", 2, 10, "Ident-Votación"),
        UrnaLimiteOpacaTramposa(10, "Grupo-Votación", 2, "Ident-Votación"),
        UrnaLimiteTramposaOpaca(10, 2, "Grupo-Votación", "Ident-Votación"),
        UrnaTramposaLimiteOpaca(2, 10, "Grupo-Votación", "Ident-Votación"),
        UrnaTramposaOpacaLimite(2, "Grupo-Votación", 10, "Ident-Votación"),
        ]
    print()
    print("Prueba Argumentos MRO")
    for urna in lista_urnas:
        prueba_votacion(type(urna).__name__, urna)
```

Herencia múltiple y MRO. Ejemplo (XIII)

```
def main() -> None:
    print_mros()
    prueba_args_nombre()
    prueba_args_mro()

if __name__ == "__main__":
    main()
```

Herencia múltiple y MRO. Ejemplo (XIV)

```
# UrnaTramposaLimite Iteración-1
# UT(2, UL(10, 0, U(Ident-Votación, 0, 0, UA())))
# UT(2, UL(10, 9, U(Ident-Votación, 2, 7, UA())))
# UrnaTramposaLimite Iteración-1 Resultado intermedio: False
#
# UrnaTramposaLimite Iteración-2
# UT(2, UL(10, 9, U(Ident-Votación, 2, 7, UA())))
# UT(2, UL(10, 10, U(Ident-Votación, 3, 7, UA())))
# UrnaTramposaLimite Iteración-2 Resultado intermedio: False
#
# UT(2, UL(10, 10, U(Ident-Votación, 3, 7, UA())))
# UrnaTramposaLimite Resultado final: False
```

```
# UrnaLimiteTramposa Iteración-1
# UL(10, 0, UT(2, U(Ident-Votación, 0, 0, UA())))
# UL(10, 7, UT(2, U(Ident-Votación, 2, 7, UA())))
# UrnaLimiteTramposa Iteración-1 Resultado intermedio: False
#
# UrnaLimiteTramposa Iteración-2
# UL(10, 7, UT(2, U(Ident-Votación, 2, 7, UA())))
# UL(10, 10, UT(2, U(Ident-Votación, 8, 7, UA())))
# UrnaLimiteTramposa Iteración-2 Resultado intermedio: True
#
# UL(10, 10, UT(2, U(Ident-Votación, 8, 7, UA())))
# UrnaLimiteTramposa Resultado final: True
```

Herencia múltiple y MRO. Ejemplo (XV)

```
# UrnaTramposaOpaca Iteración-1
# UT(2, UO(Grupo-Votación, True, U(Ident-Votación, 0, 0, UA())))
# UT(2, UO(Grupo-Votación, False, U(Ident-Votación, 0, 0, UA())))
# UrnaTramposaOpaca Iteración-1 Resultado intermedio: True
#
# UrnaTramposaOpaca Iteración-2
# UT(2, UO(Grupo-Votación, False, U(Ident-Votación, 0, 0, UA())))
# UT(2, UO(Grupo-Votación, False, U(Ident-Votación, 0, 0, UA())))
# UrnaTramposaOpaca Iteración-2 Resultado intermedio: True
#
# UT(2, UO(Grupo-Votación, False, U(Ident-Votación, 0, 0, UA())))
# UrnaTramposaOpaca Resultado final: True
```

```
# UrnaOpacaTramposa Iteración-1
# UO(Grupo-Votación, True, UT(2, U(Ident-Votación, 0, 0, UA())))
# UO(Grupo-Votación, False, UT(2, U(Ident-Votación, 1, 0, UA())))
# UrnaOpacaTramposa Iteración-1 Resultado intermedio: True
#
# UrnaOpacaTramposa Iteración-2
# UO(Grupo-Votación, False, UT(2, U(Ident-Votación, 1, 0, UA())))
# UO(Grupo-Votación, False, UT(2, U(Ident-Votación, 1, 0, UA())))
# UrnaOpacaTramposa Iteración-2 Resultado intermedio: True
#
# UO(Grupo-Votación, False, UT(2, U(Ident-Votación, 1, 0, UA())))
# UrnaOpacaTramposa Resultado final: True
```