ICC3100 – Paradigmas de Programación

POO

Relaciones a nivel de Clases

Matías Recabarren
mrecabarren@miuandes.cl

Facultad de Ingeniería/Universidad de los Andes/Chile

En esta clase

Herencia

Herencia

Teoría

Definición

De diccionario (RAE)

"Rasgo o rasgos morales, científicos, ideológicos, etc., que, habiendo caracterizado a alguien, continúan advirtiéndose en sus descendientes o continuadores."

Ciencia de la Computación

- Es una manera de formar una **nueva clase** utilizando otras clases ya existentes.
- La nueva clase posee todas las capacidades de las clases progenitoras.

Ejemplo

Ejemplo: Taxonomía Biológica

- Definición de Clases
  - Seres Vivos
    - Se alimentan, respiran, se reproducen, etc.
  - Animales
    - Se mueven, reproducción sexual, etc.
  - Mamíferos
    - Vertebrados, las hembras poseen glándulas mamarias, etc.
- Los **Animales** poseen todas las características de los **Seres Vivos**.
- Los **Mamíferos** poseen todas las características de los **Animales**.
  - Por ende, también poseen las características de los **Seres Vivos**.

Características de la Herencia

- Los lenguajes OOP permiten definir clases a partir de otras clases ya existentes.
  - A la clase de la cual se hereda se le llama **clase padre**, **superclase** o **clase base**.
  - La clase que hereda recibe el nombre de **clase hija**, **subclase** o **clase derivada**.
- Cada clase derivada hereda el estado (atributos) y el comportamiento (métodos) de la clase base.
- Cada clase derivada puede agregar variables de estado, y métodos específicos.
- Las clases derivadas no están limitadas a un solo nivel. Es posible declarar toda una **jerarquía** de clases de la profundidad que sea necesaria.

Ejemplo Herencia

Ejemplo: Herencia Bicicleta

- Clase Base: `Bicicleta`
- Clases Derivadas
  1. `MountainBike`
     - Agrega un atributo que mantiene el contenido de la botella de agua.
  2. `BMXBike`
     - Agrega un atributo con el ángulo de giro del manubrio.
     - Agrega métodos para realizar piruetas.
  3. `TandemBike`
     - Agrega un atributo con el número de asientos.
     - Agrega métodos para agregar y eliminar pasajeros.

Consecuencia de la Herencia

La clase **derivada** hereda todos los elementos de la clase **base**.

- Atributos y métodos.

La clase **derivada** solamente podrá acceder a los elementos de la clase **base** que *no sean privados*.

- Los elementos privados están dentro de la clase **derivada**, pero no pueden ser utilizados directamente.
- Si pueden utilizarse a través de un método no privado que los utilice.
- En Python la privacidad es **semántica**.

Consecuencia de la Herencia

La clase **derivada** puede **sobreescribir** métodos de la clase **base**.

- puede agregar un método con la misma *firma* que un método de la clase **base**.
  - La firma de un método está compuesta por su nombre y los parámetros que recibe.
- El método que sobreescribe puede invocar al método sobreescrito como parte de su ejecución.

Herencia - Llamada a métodos

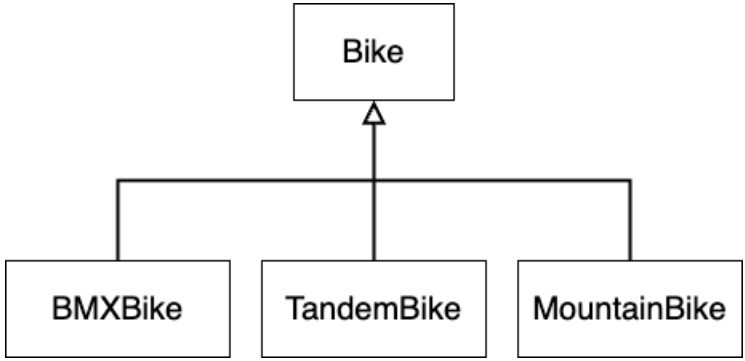
Volviendo al ejemplo de la Taxonomía Biológica, ¿Qué sucede si llamo al método `Respirar` de un objeto de la clase `Mamífero`?

- Primero se busca en la clase receptora: `Mamífero`.
- Si el método no se encuentra en esta clase se busca en la clase base: `Animal`.
- Si el método tampoco se encuentra en esta clase se busca en su clase base: `Ser Vivo`.
- Si no se encuentra el método en ninguna de las clases de la jerarquía se lanza un error.

En resumen siempre se comienza buscando el método en el tipo del objeto, y de no encontrarse se empieza a **subir en la jerarquía**.

Notación de Herencia

En este curso utilizaremos la siguiente notación para representar herencia.



Herencia en python

Sintáxis

Para heredar de una clase se debe poner el nombre de esa clase entre paréntesis en la declaración.

```
class Person:
  def __init__(self, name: str) -> None:
    self.name: str = name

class Worker(Person):
  def __init__(self, name: str, company: str) -> None:
    self.name: str = name
    self.company: str = company
```

Función super()

La función super() se refiere a la superclase (la clase de la que la actual clase deriva) y te permite llamar a sus métodos.

```
class Person:
  def __init__(self, name: str) -> None:
    self.name: str = name

class Worker(Person):
  def __init__(self, name: str, company: str) -> None:
    super().__init__(name)
    self.company: str = company
```

Útil en la inicialización, pero también para acceder a otros métodos de la clase base.

Sobreescritura de métodos

Si defines un método en la clase derivada con el **mismo nombre** que un método en la clase base, el método de la clase base será sobrescrito.

```python
class Person:
  def __init__(self, name: str) -> None:
    self.name: str = name

  def greet(self) -> str:
    return f'Soy persona y me llamo {self.name}'

class Worker(Person):
  def __init__(self, name: str, company: str) -> None:
    super().__init__(name)
    self.company: str = company

  def greet(self) -> str:
    return super().greet() + f', trabajo en {self.company}'
```

Verificar relación de herencia

Puedes verificar si una clase es subclase de otra utilizando la función `issubclass()`, o si una instancia pertenece a una clase específica (o a una clase derivada de ella) usando `isinstance()`.

```
p1 = Person('Juan')
p2 = Worker('Ana', 'UANDES')

issubclass(Person, Worker)
isinstance(p2, Worker)
isinstance(p2, Person)
isinstance(p1, Worker)
```

¿Qué resultado entregan esas 4 llamadas?

Verificar relación de herencia

Puedes verificar si una clase es subclase de otra utilizando la función `issubclass()`, o si una instancia pertenece a una clase específica (o a una clase derivada de ella) usando `isinstance()`.

```
p1 = Person('Juan')
p2 = Worker('Ana', 'UANDES')

issubclass(Person, Worker) # verdadero
isinstance(p2, Worker) # verdadero
isinstance(p2, Person) # verdadero
isinstance(p1, Worker) # falso
```

¿Qué resultado entregan esas 4 llamadas?

Clase base-base

En python toda clase que no hereda explícitamente de otra, hereda de la clase **object**.

- Es decir, `object` es el punto superior de cualquier jerarquía.
- Toda clase hereda en algún punto de `object`.
- Algunos métodos relevantes de la clase `object`:
  - `__repr__(self)` → devuelve un string que representa inequivocamente al objeto.
  - `__str__(self)` → devuelve un string que representa al objeto de una manera legible. Se utiliza para una conversión informal a string, por ejemplo, por la función `print()`.

Sobrecarga de métodos

La sobrecarga de métodos se refiere a tener varios métodos con el mismo nombre, pero distintos tipos/cantidad de parámetros.

- Python **no soporta sobrecarga**
- En python la firma de un método es **solo su nombre**, no sus parámetros.
  - por ende si una clase derivada tiene un método con el mismo nombre que una clase base, pero con distintos parámetros, igualmente lo va a sobreescribir.
  - No se podrá acceder al método de la clase base desde un objeto de la clase derivada.

Privacidad

Python sigue la filosofía **somos todos adultos aquí**

- Entonces no hay restricciones de acceso a los elementos de una clase.
- Todos los métodos de la jerarquía son accesibles por las clases derivadas.
- Todos los métodos de la jerarquía son accesibles desde un objeto de la clase.

Si existe una convención:

- Todo identificador que comience con un guión bajo (_) se considera un elemento privado.
- Esto implica que **no debería usarse directamente** desde fuera de la clase.

Abstracción

Abstracción

Teoría

Definición

RAE - *Abstracto*

"Dicho del arte o de un artista: Que no pretende representar seres o cosas **concretas** y atiende solo a elementos de forma, color, estructura, proporción, etc."

Ciencia de la Computación - *Clase Abstracta*

Es una clase que no se encuentra definida en su totalidad o que su definición no es suficiente para crear objetos de ella.

- Se utiliza como base para la definición de otras clases derivadas.
- Puede dejar algunos *métodos* sin definir.
  - Estos métodos son los métodos abstractos.
  - Solamente se define la firma del método.
  - **Deben** ser definidos por las clases derivadas.

Consecuencias

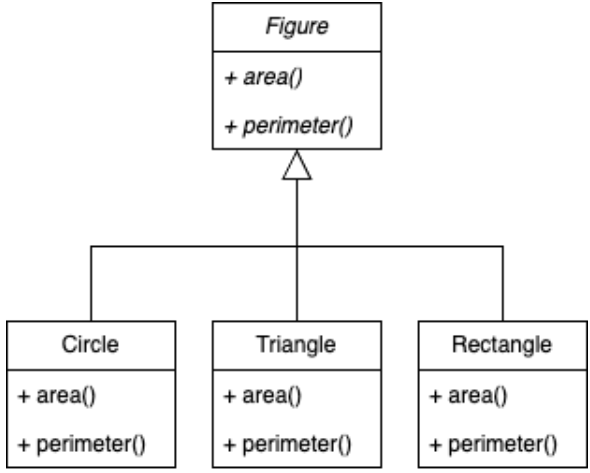
- Una clase abstracta **no puede ser instanciada**.
- Toda clase que tenga un método abstracto **debe** ser una clase abstracta.
- Una variable del tipo de una clase abstracta puede referenciar a cualquier objeto de las clases derivadas de esta que no sean abstractos.
  - La misma regla que la herencia.

Ejemplo: Figuras

Supongamos que estamos interesados en modelar las figuras geométricas, queriendo obtener su área y perímetro.

- Creamos una clase abstracta `Figure`.
  - ¿Por qué abstracta?
  - Porque una figura no es un elemento concreto.
  - ¿Cómo calculo el área de una *figura* cualquiera?
- Agregamos dos métodos abstractos.
  - `area`
  - `perimeter`
- Creamos clases derivadas, que implementen ambos métodos según la figura que sean.
  - Las clases `Circle`, `Rectangle` y `Triangle` implementarán los métodos según sus fórmulas para el cálculo.

Ejemplo: Figuras



Abstracción

Abstracción en Python

Clase abstracta

Python no soporta directamente la abstracción

- Pero si define un módulo con elementos que permite incorporarla
- **Abstract Base Classes** → `import abc`
- Una clase abstracta se genera heredando directamente de la clase `ABC`

```
from abc import ABC

class Figure(ABC):
  ...
```

Métodos abstractos

De todas formas python no tomará la clase como abstracta hasta que tenga métodos abstractos.

- Para definir un método como abstracto se le debe agregar el decorador `@abstractmethod`
- Solo clases que hereden de `ABC` pueden usar este decorador.
- Las clases derivadas estarán obligadas a definir todos los métodos abstractos.

```python
from abc import ABC, abstractmethod

class Figure(ABC):
  @abstractmethod
  def area(self) -> float:
    ...
```

Métodos abstractos

```
f = Figure()
```

Lo de arriba lanzará error `TypeError`

```
class Circle(Figure):
  def __init__(self, radius: float) -> float:
    self.radius: float = radius
```

Si no se definen los métodos abstractos, entonces la clase derivada también será abstracta.

Extender ejemplo

¿Cómo harían que las figuras pudieran **sumarse** con otra figura?

Tarea 1

Fechas relevantes

Laboratorio

Hay laboratorios esta semana, ejemplos con herencia muy útiles para la tarea.

Fechas de las evaluaciones

- ~~Laboratorio 1 → Martes 25 de marzo.~~
- Tarea 1 → Martes 22 de abril 10am (EF - online)
  - ~~Enunciado → Martes 25 de marzo~~
  - **EP1 → Martes 8 de abril (presencial)**
- Laboratorio 2 (Tarea 1) → Martes 22 de abril
- Laboratorio 3 → Martes 13 de mayo
- Tarea 2 → Martes 3 de junio 10am (EF - online)
  - EP1 → Martes 20 de mayo (presencial)
- Laboratorio 4 (Tarea 2) → Martes 3 de junio
- Laboratorio 5 → Martes 17 de junio
- Tarea 3 → Martes 8 de julio (EF - presencial)
  - EP1 → Martes 24 de junio (presencial)
  - Evaluación → Martes 8 de julio (presencial)