Aprende a programar con Minecraft

### **Sobrescribir métodos y atributos**

Una subclase puede redefinir métodos y atributos de su superclase. Esto es útil cuando quieres utilizar el mismo nombre para un método pero quieres que se comporte de forma diferente en la subclase.

En "[Comprender la herencia](ch12.xhtml#ch12lev1sec07)", en [la página 277](ch12.xhtml#page_277), creamos una clase Pájaro y una clase Pingüino. La clase Pingüino heredó de Ave, por lo que compartió todos sus métodos. Pero los pingüinos no pueden volar, y su canto de pájaro es más un graznido que un gorjeo. Así que debemos cambiar los métodos fly() y birdcall() para reflejar esto. Abre *birdClass.py* y añade este código:

*birdClass.py*

class Penguin(Bird):  
def swim(self):  
print("swimming")  
  
➊ def birdcall(self):  
print("sort of a quack")  
  
➋ def fly(self):  
print("Penguins cannot fly :(")

He hecho dos cambios en la clase Pingüino. He añadido un método birdcall( ) ➊ y un método fly () ➋. Como ambos métodos se escriben igual que en la superclase Pájaro, anularán los métodos de la superclase.

Llama a los métodos añadiendo este código a *clasepájaro.py*:

*birdClass.py*

sarahThePenguin.fly()  
sarahThePenguin.birdcall()

Ahora, cuando ejecutes el programa, obtendrás este resultado:

Penguins cannot fly :(  
sort of a quack

Anular un método de una superclase cambiará lo que hace el método para la subclase, pero no para la superclase. Así, los pingüinos no podrán volar, pero otros pájaros que hereden de Pájaro podrán volar.

También puedes sobrescribir el método \_\_init\_\_() en una subclase. Esto significa que cuando se crea el objeto de la subclase, puede tener atributos o comportamientos diferentes a los de la superclase.

Por ejemplo, creemos una subclase Loro de Pájaro en el mismo archivo. Los loros pueden ser de distintos colores, así que incluyamos un argumento extra en el método \_\_init\_\_() para un atributo de color:

*birdClass.py*

class Parrot(Bird):  
➊ def \_\_init\_\_(self, name, wingspan, color):  
self.name = name  
self.wingspan = wingspan  
self.color = color

He incluido un nuevo método \_\_init\_\_ () para la clase Loro que tiene un argumento extra, color ➊, en comparación con la clase original Pájaro.

Ahora, cuando creamos un nuevo objeto Loro, podemos acceder al atributo color. También podemos acceder a los métodos birdcall() y fly( ) porque se heredaron de la superclase Bird:

*birdClass.py*

freddieTheParrot = Parrot("Freddie", 12, "blue")  
print(freddieTheParrot.color)  
freddieTheParrot.fly()  
freddieTheParrot.birdcall()

Este código producirá lo siguiente:

blue  
flap  
chirp

Recuerda que puedes sobrescribir cualquier método que una subclase herede de una superclase; incluso puedes sobrescribir el método \_\_init\_\_(). Esto te da mucho control sobre los objetos y sus numerosos atributos y métodos.

Después de todos los cambios que hemos hecho en *clasepájaro.py*, el programa final tiene el siguiente aspecto. También está disponible en los recursos del libro en [*https://www.nostarch.com/programwithminecraft/.*](https://www.nostarch.com/programwithminecraft/)

*birdClass.py*

class Bird(object):  
def \_\_init\_\_(self, name, wingspan):  
self.name = name  
self.wingspan = wingspan  
  
def birdcall(self):  
print("chirp")  
  
def fly(self):  
print("flap")  
  
  
class Penguin(Bird):  
def swim(self):  
print("swimming")  
  
def birdcall(self):  
print("sort of a quack")  
  
def fly(self):  
print("Penguins cannot fly :(")  
  
  
class Parrot(Bird):  
def \_\_init\_\_(self, name, wingspan, color):  
self.name = name  
self.wingspan = wingspan  
self.color = color  
  
gardenBird = Bird("Geoffrey", 12)  
gardenBird.birdcall()  
gardenBird.fly()  
  
sarahThePenguin = Penguin("Sarah", 10)  
sarahThePenguin.swim()  
sarahThePenguin.fly()  
sarahThePenguin.birdcall()  
  
freddieTheParrot = Parrot("Freddie", 12, "blue")  
print(freddieTheParrot.color)  
freddieTheParrot.fly()  
freddieTheParrot.birdcall()

Probarás a anular métodos y atributos en la próxima misión.

#### **Misión nº 73: Árbol fantasma**

Has creado varias formas de edificios fantasma. Llevémoslo al siguiente nivel y creemos un árbol fantasma. Es una idea increíble, pero ¿cómo podemos hacerlo? La clase Edificio es para edificios, que tienen paredes y techos; los árboles no tienen paredes ni techos. ¡No te preocupes! Puedes evitarlo modificando tu clase Edificio fantasma.

Al igual que los edificios fantasma, el árbol fantasma aparecerá y desaparecerá utilizando los métodos build() y clear(). Pero los métodos tienen que funcionar de forma diferente porque los árboles tienen un aspecto distinto al de las casas. Así que tienes que crear una clase que herede de la clase Edificio y anular los métodos build( ) y clear( ).

Para empezar, he cogido la función que crea un árbol del archivo *forest.py*[(página 153](ch08.xhtml#page_153)) y la he puesto en [el Listado 12-6](ch12.xhtml#ch12ex6). Cópiala en un nuevo archivo llamado *ghostTree.py* en la carpeta de *clases*.

*ghostTree.py*

from mcpi.minecraft import Minecraft  
mc = Minecraft.create()  
  
# Paste the ghostHouse.py program here  
# Create a Tree class here  
  
➊ def growTree(x, y, z):  
""" Creates a tree at the coordinates given """  
wood = 17  
leaves = 18  
  
# Trunk  
mc.setBlocks(x, y, z, x, y + 5, z, wood)  
  
# Leaves  
mc.setBlocks(x - 2, y + 6, z - 2, x + 2, y + 6, z + 2, leaves)  
mc.setBlocks(x - 1, y + 7, z - 1, x + 1, y + 7, z + 1, leaves)  
  
# Create build() and clear() methods for the Tree class here

*Listado 12-6: Una función para crear un árbol*

Para terminar el programa, copia y pega el código de la clase Edificio de *ghostHouse.* py en el nuevo archivo. A continuación, crea una nueva clase llamada Árbol que herede de la clase Edificio. Dentro de la clase Árbol, añade un método build() y un método clear() para anular los métodos de la clase Edificio y construir un árbol en lugar de una casa. Asegúrate de incluir el argumento self delante de los atributos en el método final growTree() ➊.

Cuando hayas creado el programa, crea un objeto Árbol llamado árbol fantasma. Llama al método build( ) para que aparezca el árbol, espera un poco y luego haz que desaparezca utilizando clear().

[La Figura 12-7](ch12.xhtml#ch12fig7) muestra el resultado de mi programa.



*Figura 12-7: ¡Es un árbol espeluznante!*

**OBJETIVO EXTRA: BOSQUE FANTASMA**

Modifica el código de *ghostTree.py* para construir un bosque fantasma. ¿Qué tipo de tesoros crees que podrías encontrar en un bosque fantasma?

[anterior](ch12_8.html)[Subtema 9 de 10: (Ver todo)](ch12.html)[siguiente](ch12_10.html)