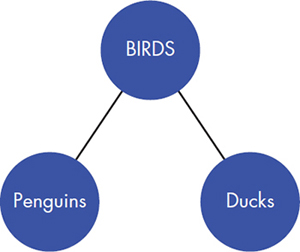
Aprende a programar con Minecraft

### **Entender la herencia**

La*herencia* se produce cuando las clases comparten los mismos métodos y atributos que otras clases. Por ejemplo, los patos son un tipo de ave. Comparten los mismos métodos que otras aves (volar, comer, etc.) y tienen los mismos atributos que otras aves (peso, envergadura, etc.). Así que se podría decir que *los patos* heredan sus atributos y métodos de la clase *aves*. [La figura 12-5](ch12.xhtml#ch12fig5) muestra esta relación en un diagrama.



*Figura 12-5: Tanto los pingüinos como los patos son clases de aves.*

La clase de la que heredan otras clases se llama *superclase*; la clase que hereda de la superclase se llama *subclase*.

La herencia es útil porque permite crear diferencias sutiles entre objetos similares. Por ejemplo, los pingüinos también son un tipo de ave, pero pueden nadar bajo el agua, a diferencia de la mayoría de las aves. Para representar a los pingüinos, tienes que crear una subclase que herede de la clase pájaro pero que tenga adaptaciones para que los pingüinos puedan nadar bajo el agua. Estas adaptaciones son la razón por la que creas subclases: puedes conservar las características principales de la superclase para evitar tener que escribir el código de nuevo, y sólo tienes que añadir los métodos y atributos que necesites en la subclase.

#### **Heredar una clase**

Cuando una subclase hereda de una superclase, la subclase puede utilizar todos los métodos y atributos de la superclase. La subclase también puede añadir clases y atributos adicionales sin alterar la superclase original.

Utilicemos el ejemplo de los pájaros para ilustrarlo. Primero escribiremos el código de la superclase Pájaro. Abre un nuevo archivo en IDLE, llámalo *clasepájaro.py* y añade el siguiente código para crear la clase:

*birdClass.py*

➊ class Bird(object):  
➋ def \_\_init\_\_(self, name, wingspan):  
self.name = name  
self.wingspan = wingspan  
  
➌ def birdcall(self):  
print("chirp")  
➍ def fly(self):  
print("flap")

Creamos una clase llamada Ave ➊, pero fíjate en que la clase Ave hereda de objeto. La clase objeto es una clase base sobre la que se construirán todas las demás clases. Todas las clases heredan de la clase objeto, y se utiliza cuando no hay otras superclases de las que heredar. Aunque haya varios niveles de herencia en los que muchas clases hereden unas de otras, la clase objeto siempre será la superclase utilizada en el nivel más alto de herencia.

El método \_\_init\_\_() de la clase Pájaro toma dos argumentos que establecen dos atributos: el nombre del pájaro y su envergadura ➋. Tiene dos métodos: birdcall() ➌ y fly() ➍. De momento, el método birdcall () sólo imprime "chirrido" y el método fly () sólo imprime "aleteo".

En el mismo archivo, crea un objeto llamado pájarojardín utilizando la clase Pájaro:

*birdClass.py*

gardenBird = Bird("Geoffrey", 12)  
gardenBird.birdcall()  
gardenBird.fly()

Este código imprimirá

chirp  
flap

Ahora que has creado una superclase, puedes crear una subclase que herede de la superclase pero que tenga su propio método. Lo harás en la siguiente sección.

#### **Añadir nuevos métodos a las subclases**

Añadamos una clase para pingüinos a *birdClass.py* y llamémosla Pingüino. Como los pingüinos pueden nadar bajo el agua, puedes añadir un método extra a la clase Pingüino llamado nadar():

*birdClass.py*

class Penguin(Bird):  
def swim(self):  
print("swimming")

Cuando defines una subclase y quieres que herede de otra superclase en lugar de un objeto, pones entre paréntesis el nombre de la superclase de la que hereda. Observa que no he creado un método \_\_init\_\_() para la clase Pingüino. La razón es que hereda de la clase Pájaro, por lo que utiliza el método \_\_init\_\_( ) de la clase Pájaro. Utilicemos ese método \_\_init\_\_( ) y probemos la función swim( ) creando un pingüino:

*birdClass.py*

sarahThePenguin = Penguin("Sarah", 10)  
sarahThePenguin.swim()

Este código mostrará lo siguiente:

swimming

La clase Pingüino también puede utilizar los métodos fly() y birdcall() porque los hereda de Bird.

*birdClass.py*

sarahThePenguin.fly()  
sarahThePenguin.birdcall()

En este caso, el resultado será el siguiente:

flap  
chirp

Pero aletear y piar no tienen sentido para un pingüino, porque los pingüinos no pueden volar y su canto de pájaro es más bien un graznido. Aprenderemos a anular métodos heredados y a solucionar esto en "[Anular métodos y atributos](ch12.xhtml#ch12lev1sec08)", en la [página 282](ch12.xhtml#page_282).

Pero antes, volvamos a Minecraft y creemos unos nuevos edificios fantasma utilizando la herencia.

#### **Misión nº 72: Hotel Fantasma**

Tanto las casas como los hoteles son tipos de edificios: tienen puertas, ventanas, habitaciones, escaleras y paredes. Los hoteles no son más que casas elegantes con extras como balcones, muchas habitaciones y una bonita entrada.

¿Cómo puedes programar unos hoteles fantasma utilizando el código que ya has creado para las casas fantasma? La estructura básica de los edificios es la misma. Digamos que la única diferencia es que los hoteles fantasma tienen métodos adicionales para crear alfombras dentro de las habitaciones y añadir flores alrededor del borde del edificio. Eso significa que la clase hotel fantasma puede heredar todos los métodos de la clase casa fantasma. Entonces, lo único que necesita la clase hotel fantasma son dos métodos extra para las alfombras y las flores.

En IDLE, crea un nuevo archivo y guárdalo como hotelfantasma *.py* en la carpeta *clases*. Copia y pega en el archivo el código de la clase Edificio del programa *ghostHotel*.py.

Crea una nueva clase llamada FancyBuilding que herede de la clase Building. La clase FancyBuilding debe tener un nuevo método llamado upgrade() que añada alfombra dentro del edificio y flores alrededor de las paredes. [El listado 12-5](ch12.xhtml#ch12ex5) muestra mi código para el método upgrade( ), pero siéntete libre de personalizar tus hoteles.

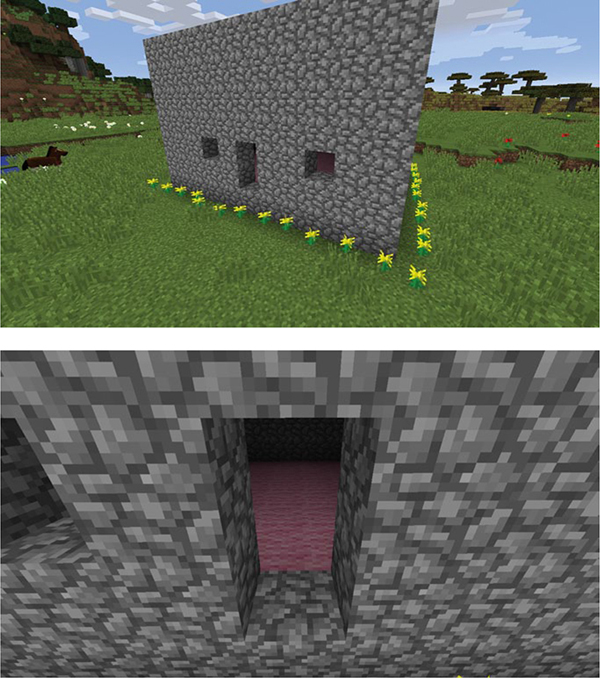
*ghostHotel.py*

# Create a FancyBuilding class here  
  
def upgrade(self):  
# Carpet  
mc.setBlocks(self.x + 1, self.y, self.z + 1,  
self.x + self.width - 1, self.y, self.z + self.depth - 1,  
35, 6)  
  
# Flowers  
mc.setBlocks(self.x - 1, self.y, self.z -1,  
self.x - 1, self.y, self.z + self.depth + 1,  
37)  
mc.setBlocks(self.x - 1, self.y, self.z - 1,  
self.x + self.width + 1, self.y, self.z – 1,  
37)  
mc.setBlocks(self.x + self.width + 1, self.y, self.z - 1,  
self.x + self.width + 1, self.y, self.z + self.depth + 1,  
37)  
mc.setBlocks(self.x - 1, self.y, self.z + self.depth + 1,  
self.x + self.width + 1, self.y, self.z + self.depth = 1,  
37)  
  
# Create an instance of the FancyBuilding class  
# Call the build() and upgrade() methods

*Listado 12-5: Un método de la* *clase* FancyBuilding que añade alfombra *y* flores al edificio

Cuando hayas creado la clase y añadido el nuevo método, crea una instancia de la clase FancyBuilding y llámala hotel fantasma. Construye el hotel fantasma con el método build( ) y, a continuación, añade los elementos adicionales con el método upgrade().

[La Figura 12-6](ch12.xhtml#ch12fig6) muestra mi elegante hotel fantasma.



*Figura 12-6: ¡Mira qué flores y qué alfombra!*

**OBJETIVO EXTRA: ALDEA FANTASMA**

En la Misión nº 71, creaste un pueblo fantasma en el que todos los edificios parecían más o menos iguales. Es raro ver edificios idénticos en los pueblos reales. Cambia el programa del pueblo fantasma creando varias clases que hereden de la clase Edificio. Podrías crear una clase Tienda, una clase Hospital y una clase Restaurante, por ejemplo. Luego, cuando crees los objetos, podrás elegir qué tipo de edificio crear utilizando las distintas clases.

[anterior](ch12_7.html)[Subtema 8 de 10: (Ver todo)](ch12.html)[siguiente](ch12_9.html)