Capítulo 1

Herencia

Las colecciones son un concepto importante y poderoso al diseñar con objetos. En este capítulo veremos cómo se modela el conocimiento de un conjunto de referencias de un objeto y su utilización para resolver un problema concreto.

Índice general

1.1. Introducción

En capítulos precedentes introdujimos el concepto de *clase*: un molde que define la estructura y comportamiento de los objetos creados a partir de ella. Las clases nos ofrecen tanto un marco conceptual como ventajas prácticas: por un lado nos permiten agrupar objetos similares bajo una misma abstracción, dándoles un nombre común (el nombre de la clase), y por otro lado nos permiten definir una única vez el comportamiento de un objeto y reutilizarlo en varios objetos. Por ejemplo, podemos podemos definir una clase `Heroe` que permita desplazar nuestros héroes entre distintos casilleros de la siguiente manera:

```
Object subclass: #Heroe
   instanceVariableNames: ''.

Heroe >> desplazarseA: unCasillero
    unCasillero recibirHeroe: self.
```

Consideremos ahora que queremos introducir distintas *especies* de héroes: arqueros y guerreros. Arqueros y guerreros son héroes que van a participar en distintas batallas épicas y por lo tanto deben poder hacer daño. Además, tanto arqueros como guerreros poseen, obviamente, distintas características de combate que podemos representar en código utilizando distintas clases `Arquero` y `Guerrero`:

```
Object subclass: #Arquero
   instanceVariableNames: 'flecha'.

Arquero >> danoCausado
    ^ flecha poder + self modificadorDestreza.

Object subclass: #Guerrero
   instanceVariableNames: 'arma'.

Guerrero >> danoCausado
    ^ arma poder + self modificadorFuerza.
```

Usar distintas clases para guerreros y arqueros nos permite definir distintos comportamientos para cada uno de ellos. Sin embargo, nuestros nuevos héroes ya no comparten el comportamiento que definimos anteriormente en nuestra clase `Heroe`. Para solucionar este problema, este capítulo introduce el concepto de **herencia**. La herencia es una relación entre clases que permite definir super-clasificaciones y sub-clasificaciones de objetos, generar distintos niveles de abstracción y compartir comportamiento entre distintas clases. Estos distintos niveles de abstracción llevan a la aparición de clases concretas y clases abstractas. Finalmente este capítulo discute distintos criterios de aplicación de herencia.

1.2. Clasificando Objetos con Herencia

Guille ▶*diagrama de venn*◀

1.3. Herencia

1.4. Method Lookup

1.5. `self` vs `super`

method lookup revisited 2 self vs super

1.6. Clases concretas vs clases abstractas

Al generar superclases sigo perdiendo información, gano en generalidad. Un Ave quizás no tenga sentido instanciarlo. Si en mi aplicación no voy a instanciar aves (porque representan un concepto demasiado general) entonces la clase es abstracta. Diferencia entre Smalltalk y Java:

1) En Smalltalk no tiene sentido crear un ave, aunque podría. La clase es abstracta cuando no tengo intención de crear instancias de esa clase (porque no tiene sentido).

2) En Java no puedo crear un ave aunque quisiera. La clase es abstracta y el compilador me impide generar instancias de esa clase. Son dos filosofías distintas. public abstract class Ave forma parte de la definición misma de clase

ejemplo de herencia con superclase abstracta ejemplo de herencia con superclase concreta

1.7. Criterios para utilizar herencia

1.7.1. Usar Objetos vs Clases

1.7.2. Especializar vs Generalizar

Otra forma de ver la herencia: subclasificamos un concepto conocido, lo refinamos. Si mi hija no conoce lo que es una tonina, yo le puedo explicar: "Y... es como un delfín pero negro" (una especie de ... pero que ...; mostrando tanto en lo que se parece como en lo que se diferencia).

Tenemos conceptos conocidos: golondrina, colibrí, torcaza, paloma, gorrión. Muchos de estos pájaros tendrán cosas en común pero se diferencian en algo... Una vez que reconocí varios objetos, puedo abstraer una clase. Pierdo información, porque lo que obtengo es más general (una golondrina genérica, en lugar de esta o aquella golondrina, que era de color azul y yo llamaba pepita). Ahora vamos a trabajar con otro tipo de abstracción: tengo varias clases (conceptos) y llego a una jerarquía de clases de la más general a las más particulares:

Otra forma de ver la herencia: subclasificamos un concepto conocido, lo refinamos. Si mi hija no conoce lo que es una tonina, yo le puedo explicar: "Y... es como un delfín pero negro" (una especie de ... pero que ...; mostrando tanto en lo que se parece como en lo que se diferencia).

1.7.3. subclase vs subtipo

1.7.4. Herencia vs composición