**Introducción**

Los programas se crean a partir de objetos los cuales son instancias de clases.

Algunas clases están en la biblioteca de java y otras las escribe el programador.

Cuando empezamos a escribir un nuevo programa, buscamos clases útiles en la biblioteca y vemos las clases que hemos escrito en el pasado.

Este metodología OO para la programación significa que en vez de empezar los programas desde cero, nos basamos en el trabajo anterior.

Herencia

La herencia constituye una manera de resolver este problema.

Con la herencia podemos usar clases existente como la base para crear una clase modificar.

Considera la siguiente analogía.

Suponga que desea comprar un automóvil nuevo y v a una agencia en donde hay una variable de automóviles producidos en masa.

A usted le gusto uno en especial, pero no tiene esa característica especial que está buscando.

Al igual que la descripción de una clase, el automóvil se fabricó a partir de planos que describen a muchos automóviles idénticos.

Si estuviera disponible la herencia, usted podría especificar un automóvil que tuviera todas las características del automóvil producido en masa, pero con las características o cambios adicionales que usted requiere.

Como Usar la Herencia

Vamos a empezar con una clase similar a la que ya hemos usado varias veces en el libro.

En una clase para respetar a una esfera.

Una esfera tiene un radio y una posición en el espacio.

Al mostrar una esfera en la pantalla, debe aparecer como un circulo (el método para mostrar una esfera simplemente invoca al método de biblioteca drawOval).

El diámetro de la esfera esta fijo en 2 pixeles.

Solo hemos modelado las coordenadas x y y de una esfera (y no la coordenada *z*) debido a que vamos a mostrar una presentación bidimensional en la pantalla.

Ejemplo:



Protected

Esto se debe a que escribimos la clase de tal forma que se pueda utilizar para la herencia.

En el transcurso de este capítulo veremos lo que significa estos nuevos elementos.

Vamos a suponer que alguien escribió y probó esta clase, y la puso a disposición de otros para que la utilicen.

Ahora vamos a escribir un nuevo programa y necesitamos una clase muy parecida a esta, solo que para describir burbujas.

Esta nueva clase llamada Burbuja nos permite hacer cosas adicionales: modificar el tamaño de una burbuja y moverla en sentido vertical.

La limitación de la clase Esfera es que describe objetos que no se mueven y cuyo tamaño no se puede cambiar.

Primero necesitamos un método adicional que nos permita establecer un nuevo valor para el radio de la burbuja.

Podemos hacer esto sin alterar la clase existente; para ello debemos escribir una clase diferente que utilice el código que ya se encuentra en la clase Esfera.

Decimos que la nueva clase hereda las variables y los métodos de la clase anterior.

La nueva clase se convierte en una subclase de la anterior.

A la clase anterior se le llama la superclase de la nueva clase.

He aquí cómo podemos escribir la nueva clase:

Ejemplo;



**La Herencia en Acción**

Al igual que la clase Burbuja, es como que una clase tenga una superclase, la que a su vez tiene una superclase y así sucesivamente, hasta llegar a la parte superior del árbol de herencia.

No solo se heredan los elementos public y protected de la superclase inmediata, sino también todas las variables y los métodos public y protected de todas las superclase en el árbol de herencia.

Esto es similar al árbol genealógico: ustedes hereda de su madre, su abuela y así de manera sucesiva.

**Súper**

Algunas veces una clase necesita llamar a un método de su superclase inmediata, o de alguna de las clases del nivel superior en el árbol.

No hay problema con esto, ya que los métodos de todas las clases en los niveles superiores del árbol de herencia están disponibles, siempre y cuando estén etiquetados como public o protected.

El único problema que puede sugerir es cuando el método deseado de la superclase tiene el mismo nombre que un método de la clase actual (cuando se utiliza la redefinición o sobre escritura).

Para corregir este problema hay que anteponer al nombre del método la palabra clave súper.

Por ejemplo, para llamar el método mostrar de una superclase use lo siguiente:

Súper, mostrar (papel);

En general esto es más ordenado y corto que duplicar instrucciones, y nos puede ayudar a que un programa sea más conciso al utilizar al máximo los métodos existentes

 Ejemplo;



Si ahora escribimos una nueva clase llamada Globo Diferente que herede de la clase Globo, las opciones son:

1. No escribir un constructor en la subclase. Java supondrá que hay un constructor sin parámetros.
2. Escribir uno o más constructores sin llamadas a los constructores de la superclase.
3. Escribir uno o más constructores que llamen a un constructor apropiado de la superclase, utilizando súper.

 Ejemplo;



**Final**

Los procesos de heredar y redefinir, se enfocan en cambiar el comportamiento de las clases y los objetos.

La herencia es muy poderosa, pero algunas veces es reconfortante que algunas cosas estén fijas y no se puedan modificar.

Por ejemplo, es bueno saber exactamente qué es lo que hace sqrt, que es lo que hace drawLine, etcétera.

En la POO siempre existe el peligro de que alguien extienda las clases a las que estas pertenecen y en consecuencias cambie lo que hace.

Esto podría ser por error o en un intento inadvertido de ser útil.

Para evitar esto el programa puede describir un método como final.

Eso significa que no se puede redefinir.

La mayoría de los métodos de la biblioteca se describen como final.

Esto significa que cada vez que los use, puede estar completamente seguro de lo que hace.

Como hemos visto desde los primeros capítulos del libro, las variables también se pueden declarar como final.

Esto significa también que no podemos cambiar sus valores.

Son constantes.

Por ejemplo;

Final doublé cmPerInch = 2.54;

Declara una variable cuyo valor no se puede alterar.

Así, el prefijo final tiene el mismo significado sin importar que se utilice en una variable o en un método.

La palabra reservada **final** tiene tres usos:

1. Para creaciÃ³n de constantes con nombre.
2. Para evitar sobre escritura de métodos

Los métodos declarados como **final** no pueden ser sobre escritos.

1. Para evitar la herencia

Se usa **final** en la declaraciÃ³n de la clase para evitar que la clase sea heredada: O sea, todos sus métodos serán **final** implícitamente.

Ejemplo;



**Abstractas**

Considera un programa que mencione formas graficas de todos tipos y tamaños: círculos, rectángulos, cuadrados, triángulos, etc.

Estas distintas formas, similares a las clases que ya hemos visto en este capítulo, tiene información en común: su posición, coló y tamaño.

Vamos a declarar una superclase llamada Forma que describa los datos comunes.

Cada clase individua hereda esta información común.

He aquí la forma de describir esta superclase común:

Ejemplo;



Para mejor información consulte la siguiente pagina;

<http://profesores.elo.utfsm.cl/~agv/elo330/2s05/lectures/JAVA/Herencia_en_Java.html>