Herencia

Emmanuel Cruz Hernández
emmanuel_cruzh@ciencias.unam.mx

10 de diciembre de 2020

Contenido

Introducción

Una de las características más valiosas en el código que hacemos es la posibilidad de **reutilizarlo**, es decir, tomar un código que ya trabaja para cierta aplicación parecida pero no idéntica.

Uno de los mayores beneficios de la Orientación a Objetos es la posibilidad de extender clases ya construidas (subclases) [1]. A esta acción se le conoce como **extensión de clases**.

Herencia

La *herencia* conlleva a la capacidad de reutilizar código de manera inteligente [1].
La clase original se conoce como *superclase* con respecto a la nueva; decimos que la subclase **hereda** los atributos y métodos de la superclase.

Super

Necesitamos una manera de especificar a qué constructor llamar, al haber extendido una clase.

La palabra clave **super** puede ser usada para explícitamente llamar al constructor de una superclase [2]:

super(<Parámetros>);

extends

Para especificar que una clase extiende a otra se hace con la palabra reservada **extends** en el encabezado de la clase.

public class <*Subclase*> **extends** <*Superclase*> {

    ...

}

Sobreescribir

Una subclase puede redefinir métodos de la superclase, para ello el método a redefinir debe tener la misma firma que en la clase padre.

Es conveniente usar la etiqueta *@Override* para evitar errores de compilación.

Método constructor

En el método constructor de una subclase se ocupa un constructor de la superclase.
Si en la definición del constructor del hijo no se usa explícitamente un constructor del padre, entonces implícitamente se usará el constructor sin parámetros del padre, para esto, debe existir una definición en la superclase de un constructor sin parámetros.

La primera instrucción debe ser la llamada al constructor del padre.

Ejemplos con herencia

Puedes consultar ejemplos sobre herencia en el siguiente enlace:

`https://codenowprogramming.000webhostapp.com/entradas/Java/12-herencia.php`

Polimorfismo

La palabra *polimorfismo* está formada con raíces griegas. Sus componentes léxicos son: *polys* que significa muchos, *morfo* que significa forma, más el sufijo *-ismo* que significa actividad o sistema.
Dicho lo anterior, polimorfismo quiere decir **cualidad de tener muchas formas**. [**?**]

Operador instanceof

El operador **instanceof** permite identificar de qué clase es cada objeto. Es un operador binario que tiene la siguiente forma:

<*Objeto*> **instanceof** <*Clase*>

que regresa el valor booleano verdadero si, en efecto, el objeto dado en la expresión d la izquierda es un ejemplar de la clase dada a la derecha y falso en otro caso.

Clases Abstractas

Una clase abstracta (**abstract class**) es un contenedor para declarar métodos y variables compartidos para usarse por subclases y para declarar una interfaz común de métodos y variables accesibles.

Una clase declarada como **abstract** puede incluir cualquier variable estándar y declaración de métodos, pero no puede ser usada en una expresión.
Puede también incluir declaraciones de métodos abstractos. La clase define un tipo, así que variables del tipo pueden ser declaradas y pueden mantener referencias a objetos de una subclase.

Declarar una clase abstracta

Para declarar una clase abstracta se hace de la siguiente manera:

public abstract class <*Nombre*> {

    ...

}

Clase concreta

Una subclase de una clase abstracta puede ser también abstracta. Sin embargo, en general es una clase concreta (es decir, diseñada para tener instancias u objetos).

Para declarar una clase concreta se recurre a la herencia y se usa la palabra reservada **extends**.

Interfaces

La declaración de interfaces permite la especificación de un tipo de referencia sin proveer una implementación en la forma en que una clase lo hace.
Esto provee de un mecanismo para declarar tipos que son distintos de las clases, lo que da una extensión importante a la forma en que los objetos y la herencia se usan en Java.

Conformación de tipos

Las interfaces explotan el concepto de “conformación de tipos”. Un tipo puede ser especificado por un nombre y un conjunto de métodos.
Un tipo puede “conformarse” con otro tipo si especifica el mismo conjunto de métodos, y cada método tiene el mismo nombre, tipos de parámetros y tipo de retorno.
Más aún, los dos tipos no tiene que estar relacionados por herencia, dando una mayor libertad a qué tipos pueden conformarse con otros tipos.

Declaración de interfaces

Una declaración de interfaz requiere el uso de la palabra clave **interface**, especificando un tipo de referencia, consistente en un nombre de tipo, un conjunto de declaraciones de métodos abstractos y un conjunto de variables finales estáticas (constantes).

Una interfaz se declara usando la siguiente sintaxis:

public interface <*Nombre*> {

    ...

}

Variables de interfaces

Las variables de la interfaz pueden ser declaradas usando la declaración estándar de variables estáticas, y deben ser inicializadas. Los únicos modificadores permitidos son public, static y private, pero son redundantes y no deben ser utilizados.
Una variable de interfaz puede ser inicializada por una expresión usando otra variable de interfaz, siempre y cuando esta última haya sido declarada textualmente antes de la inicialización.

Métodos de interfaces

Los métodos de interfaz son siempre abstractos, y son declarados de la misma forma que los métodos abstractos de las clases: sin cuerpo del método.
Los únicos modificadores permitidos son public y abstract, pero son también redundantes, y no deben ser usados.

implements

La palabra clave **implements** permite a una clase implementar (o mas bien, conformarse a) una o más interfaces.
El nombre de la clase en una declaración de clase se sigue de la palabra clave **implements** y una lista de uno o más nombres de interfaces separados por comas:

<Acceso> **class** *<NombreClase>* **implements** *<NombreInterfaces>* {

    ...

}

Sobre las interfaces I

El resto de la declaración de la clase se hace en forma normal. Si la clase no redefine todos los métodos declarados en la interfaz o interfaces a implementar, entonces la clase debe ser declarada como abstracta.
Una clase puede implementar cualquier número de interfaces, y también extender una clase al mismo tiempo.

Sobre las Interfaces II

Toda variable declarada en la interfaz se convierte en una variable estática de la clase.
Cualquier método declarado en la interfaz debe ya sea ser redefinido o la clase que implementa debe ser abstracta. Las subclases de la clase deben ser también abstractas o redefinir el resto de los métodos.

Ejemplo de clases abstractas e interfaces

Puedes consultar ejemplos sobre clases abstractas e interfaces en el siguiente enlace:

`https://codenowprogramming.000webhostapp.com/entradas/Java/13-abstractasInterfaces.php`

Bibliografía

Elisa Viso G. Canek Peláez V.
Introducción a las Ciencias de la Computación con Java.
Segunda edition, 2012.

Jorge L. Ortega Arjona.
Notas de Introducción al Lenguaje de Programación Java.
2004.