INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

Material para el curso de Programación Orientada a Objetos

Contenido

1.3 Introducción a buenas prácticas de programación

abstract

Esta palabra reservada nos permite forzar a que una clase o método sea implementado en una sub clase, y puede aplicarse de 2 maneras:

- A toda una clase. Una clase abstracta debe derivarse pues no puede instanciarse de manera directa. por ejemplo, en Java tenemos la clase Number de la cual heredan Integer, Long, Float, etc. (Estas clases se denominan envolventes o wrapping clases porque proveen de métodos utilitarios “alrededor” de un atributo de tipo int, long, float, etc.). Las clases abstractas nos permiten organizar varias subclases específicas bajo otra clase que no tiene uso en concreto.

- A un método en particular. Un método abstracto debe ser sobre escrito en una sub clase pues no puede ser llamado de manera directa, y sirve para contener funcionalidad que aplica a varias subclases específicas pero que no tiene significado concreto en la súper clase. Cuando usamos el modificador *abstract* no debemos poner cuerpo al método, es decir, llaves, sino solamente un punto y coma después del encabezado o firma.

Con un solo método que sea abstracto, toda la clase se vuelve abstracta por definición, ya que no puede instanciarse si no está completa; pero una clase abstracta puede tener todos sus métodos completos y de todos modos No puede instanciarse.

final

Este modificador es de alguna manera lo contrario, porque se usa para impedir que se altere un elemento mediante la herencia, y puede aplicarse de 3 maneras:

- A un método en particular. Nos permite proteger la funcionalidad de dicho método en todas las sub clases, por ejemplo, para evitar que se altere algún calculo por cuestiones de confidencialidad. Es lo opuesto a usar *virtual* en C++.
- A una variable en particular. En otras palabras, la volvemos una constante, pues su valor no puede cambiar después de ser inicializada:
  *public static final int SEGUNDOS_POR_DIA = 60 * 60 * 24;*
  Debemos en este caso usar juntos los 3 modificadores mostrados.
- A toda una clase. Una clase final simplemente no puede heredarse.

Las palabras *final* y *abstract* son mutuamente excluyentes.

Interfaces

Vimos que, a diferencia de C++, Java no permite la herencia múltiple por que puede provocar ambigüedad, sin embargo, tenemos las interfaces para simularla. Una interface es una especie de clase abstracta donde todos sus métodos contienen definiciones pero nada de código, es decir, son métodos abstractos. Dicho código debe ser proporcionado por la clase que *implemente* la interface.

Tienen mucho en común con las clases: Ambas existen en paquetes, ambas son tipos de datos validos para declarar variables, y ambas pueden contener métodos.

Pero existen diferencias significativas:

- los métodos en una interface contienen solamente definiciones pero no código.
- Una clase implementa una interface, mientras que hereda de otra clase.
- Una clase puede implementar más de una interface.

por ejemplo:

```
public abstract interface Actor {
        public abstract void hablar ();
        public abstract void actuar ();
}
public abstract interface Criminal {
        public abstract void hablar ();
        public abstract void robar ();
}

public class Politico extends Persona implements Actor, Criminal {
        // código para implementar los métodos hablar (), actuar (), robar ()
}
```

Las interfaces y todos sus métodos son siempre abstractos puesto que deben ser implementados en alguna otra parte, el uso de la palabra *abstract* es opcional. Este ejemplo muestra los beneficios de la herencia múltiple:

- Se garantiza que *Político* tenga todos los métodos definidos en *Actor y Criminal.* A menos que también fuera abstracto, debe implementar todos los métodos definidos en todas sus interfaces, por eso se dice que una interface es también una obligación contraída de codificar dichos métodos.
- *Actor y Criminal* son tipos de datos.
- Un *Político* puede ser visto como una *Persona, Actor o Criminal,* para efectos del polimorfismo.

Pero evitando los problemas potenciales como en el caso de C++ : Ya que ni *Actor* ni *Criminal* contienen ningún código, la implementación de hablar () en *Político* no puede ser ambigua.

Referencias

"Mastering Java 2.0"
Sun Microsystems