Interfaces

Agustín J. González

ELO-326: Seminario I

2do. Sem. 2001

Introducción

- En C++ se permite la herencia múltiple; es decir, es posible derivar una clase de dos o más clases bases simultáneamente.
- La herencia múltiple es manejable en casos simples, pero puede llegar a ser muy compleja si datos son heredados de múltiples cases bases.
- Java NO soporta múltiple herencia.
- Java Sí soporta “herencia” múltiple de ***interfaces***

Definición

- Una interface puede ser considerada como una clase abstracta (i.e. No podemos crear instancias de ella) con las siguientes características:
  - Todos los miembros son públicos (no hay necesidad de declararlos públicos)
  - Todos los métodos son abstractos (no se requiere declararlos como abstract)
  - Todos los campos son static y final. Se usa para definir valores constantes.

Definición e Implementación de Interfaces

- Definición de la Interfaz : Sorteable.java
- Cualquier clase puede que implemente esta interfaz pasa a ser un subtipo (la clase puede tomar el lugar de la interfaz).
- De esta forma es posible crear algoritmos “polimorfos” que pueden aceptar como objetos instancias de clases que implementan la interfaz. Estos algoritmos sólo se relacionan con los objetos a través de los métodos que la interfaz define, los cuales son implementados por la clase del objeto.
- Implementación de la Interfaz : IntArray , esta clase implementa la interfaz Sorteable
- La clase IntArrar implementa todos los métodos de la interfaz y pasa a ser un proveedor concreto de los servicios prometidos por Sorteable.
- Ahora objetos de tipo IntArray pueden ser pasados como parámetros Sorteable.
- Una clase puede implementar múltiples interfaces.
  public class ClassX extends ClassY implements InterfaceA, InterfaceB,..
  { // cuerpo de la clase }

Uso de Interfaces

- Java ya define y usa muchas interfaces; entre ellas: Runnable para hilos, Cloneable para copia, ImageObserver/ImageConsumer para desacoplar la carga de imágenes, Transferable para copy-and-paste, y listeners para el manejo de eventos.
- Una vez que la interfaz ha sido definida, se transforma en un tipo para Java sobre el cual nuevos servicios pueden ser definidos.
- Por ejemplo, consideremos una interfaz para indicar servicios de ordenamiento.

```
////// SortAlgorithm.java //////
public interface SortAlgorithm
{   // sorts a in index range
    public void sort(Sortable a, int begin, int end);
    // sorts a in its entire range
    public void sort(Sortable a);
}
```

Uso de Interfaces (cont)

- Ahora podemos crear una clase que implemente esta interfaz usando algún algoritmo de ordenamiento, por ejemplo Quicksort.java.
- En esta implementación los métodos sort son polimorfos en el sentido que aceptan cualquier objeto Sorteable (en realidad objetos que son instancia de una clase que implemente Sortable).
- Una vez desarrolladas estas interfaces podemos desarrollar programas que las usen. Por ejemplo veamos esta prueba TestSort.java
- El cual al ser ejecutado genera la siguiente salida:

Input array:

2, 9, -12, 8, 17, -99, 54, 3

Increasing array:

-99, -12, 2, 3, 8, 9, 17, 54

Decreasing array:

54, 17, 9, 8, 3, 2, -12, -99

- Notar que podemos intercambiar algoritmos muy fácilmente por ejemplo creando otras clases como bubblesort que implemente SortAlgorithm.

Extensión de Interfaces

- Una interfaz puede definirse basada en varias interfaces existentes.
- Por ejemplo
  interface Accessible extends Readable, Writeable, Executable
  {
  ....
  }
- Ahora una clase o interfaz puede ser un subtipo (puede tomar el lugar de) de más de una interfaz o clase.
- Pueden ocurrir conflictos cuando encontramos métodos o campos con el mismo nombre.
- Conflicto con métodos: Si dos métodos son exactamente iguales (igual nombre, parámetros y valor retornado), no hay problema. Si sólo difieren en el valor retornado, la “herencia” falla.
- Conflicto con nombre de campos: En realidad no hay conflicto ya que estos nombres son estáticos y siempre pueden ser accedidos con el calificador de su tipo, como en Sorteable.DECREASING.