Prácticas de Lenguajes, Tecnologías y Paradigmas de programación. Curso 2020-21

Parte I: Java

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

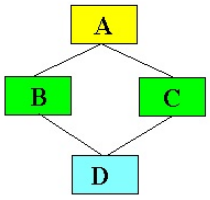
Práctica 2 - Material de lectura previa 1

Interfaces en Java

Índice

1. ¿Qué es una interfaz?

En clases de teoría se ha estudiado la herencia como un tipo de *Polimorfismo Universal de Inclusión*. También se han estudiado las Clases Abstractas, su utilidad y uso en *Java*. En la primera práctica se realizaron ejercicios con herencia incluyendo las clases abstractas. Una clase abstracta es una clase de la que no se pueden crear objetos y puede contener métodos abstractos. Este tipo de métodos pueden verse como una serie de exigencias para las clases derivadas. El tipo de herencia usado con estas clases es simple: una clase sólo hereda de otra clase, pero puede tener muchas clases derivadas. Las interfaces introducen cierta flexibilidad en la herencia de *Java*, ya que una clase de cualquier tipo puede heredar de varias interfaces (incluidas las propias interfaces), y con ello incrementar la capacidad del polimorfismo en el lenguaje.



En algunos contextos se dice que las interfaces introducen cierto grado de *herencia múltiple*. El concepto general de este tipo de herencia consiste en que una clase puede heredar de varias clases. Esto presenta algunos problemas que se han de resolver estableciendo alguna política en el tratamiento de la herencia. Un ejemplo de esta problemática es el *problema del diamante*, el cual ocurre cuando dos clases `B` y `C` heredan un método `m` de una clase `A` y adaptan el método heredado a la clase derivada usando sobrescritura. Si otra clase `D` hereda las dos versiones del método `m` de las clases `B` y `C` ¿Cuál de las dos versiones debe usarse ante una instancia de `D`? Existen muchos lenguajes con herencia múltiple (*C++*, *Eiffel*, *Python*, *Ruby* ...), y cada uno implementa su propia política para resolver este problema.

Las interfaces son lo más parecido a la herencia múltiple que implementa *Java* aunque no resuelve los problemas inherentes a este tipo de herencia. Estas clases se asemejan a una clase abstracta pura en la que todos los métodos son abstractos y públicos (sin necesidad de declararlos explícitamente como tal), y sus atributos son estáticos y finales (constantes). Esto implica que no tiene constructores y no se pueden crear objetos de una interfaz. Cuando una clase `C` hereda de una interfaz `I`, se dice que `C` ***implementa*** `I`.

2. Utilidad de las interfaces de Java

Un *Tipo Abstracto de Datos* (TAD) es un conjunto de valores y operaciones que cumple con los principios de abstracción, ocultación de la información y se puede manejar sin conocer su representación interna, es decir, son independientes de la implementación. Dicho de otra forma, un TAD permite separar la especificación de una clase (qué hace) de la implementación (cómo

lo hace). El uso de TAD's da lugar a programas más robustos y menos propensos a errores. Las interfaces son clases que se usan para especificar TAD's. Al implementar una interfaz, las clases extienden su funcionalidad estando obligadas ellas y/o sus derivadas a implementar los métodos abstractos heredados de la interfaz. Las implementaciones de los métodos abstractos deben usar la estructura de datos de la clase en la que se implementan. De esta forma se garantiza que a los objetos de la clase que implementa una interfaz se les pueden aplicar los métodos heredados.

Al definir interfaces también se permite la existencia de *variables polimórficas* definiéndolas con el tipo de una interfaz, y con ello también se permite la invocación polimórfica de métodos sobre estas variables. Esto restringe el uso de los objetos de una clase que implementa una interfaz a la funcionalidad especificada en la interfaz.

Otra característica de las interfaces es que nos permiten declarar *constantes* que van a estar disponibles para todas las clases que las implementen. Esto ahorra código evitando tener que escribir las mismas declaraciones de constantes en diferentes clases, a la vez que se concentran en un único lugar con la mejora que supone en el mantenimiento del código.

Una característica adicional es la *documentación* incluida en la interfaz. La propia sintaxis define qué métodos han de incluirse en una clase para cumplir con la interfaz. Se necesita documentación adicional respecto a las características comunes de las clases que la implementen y para qué servirán esos métodos. En realidad, si se implementa una interfaz, lo que se hace es ajustarse a una *norma*. Por ejemplo si se desea que los objetos de una clase se puedan comparar para ponerlos en un orden, se implementa la interfaz `Comparable` definida en el API de *Java*. Esta exige la implementación del método `compareTo()`, el perfil del método establece la norma de que el método debe devolver un valor de tipo `int` y la documentación de la interfaz establece cómo ha de funcionar: devolver 0 si son iguales y un entero negativo o positivo dependiendo de qué objeto de los dos comparados es mayor. Muchas clases predefinidas implementan esta interfaz.

Existen algunas normas aconsejables en el uso de las interfaces, entre ellas destacamos dos:

- Respetar el *principio de segregación*. Las clases derivadas de una clase que implementa una interfaz no deberían depender de interfaces que no utiliza.

- Evitar la *contaminación de la interfaz*. Esto ocurre cuando se añade un método a una clase base simplemente porque algunas de las clases derivadas lo usan.

3. Diferencias entre interfaz y clase abstracta

Sintácticamente, una interfaz es una clase completamente abstracta, es decir, es simplemente una lista de métodos no implementados que puede incluir la declaración de constantes.[1] Una clase abstracta, además puede incluir métodos implementados y variables.

Una clase abstracta la usamos cuando deseamos definir una abstracción que englobe objetos de las distintas clases que heredan de ella. Con ello, se puede hacer uso del polimorfismo en la misma jerarquía "vertical"de clases. Una interfaz permite elegir qué clases dentro de una o diferentes jerarquías de clases incorporan una determinada funcionalidad extra. Es decir, no fuerza una relación jerárquica, simplemente permite que clases no necesariamente relacionadas puedan tener algunas características similares en su comportamiento.

4. Sintaxis

Definición de una interfaz. Una interfaz puede extender de varias interfaces pero de ninguna clase. Aunque los métodos que especifica son abstractos **no** se explicita la palabra `abstract` ya que por defecto todos los métodos son abstractos. Tampoco pueden ser privados ni `protected`.[2]

```
[modifVisibilidad] interface nomInterfaz [extends listaInterfaces]
  {
   [CONSTANTES]
   [ [modificador] tipoDevuelto nombreMetodo1([parámetros]);
                  ...
     [modificador] tipoDevuelto nombreMetodoN([parámetros]); ]
  }
```

Ejercicio 1 *Según la definición anterior, ¿es posible definir interfaces vacías? ¿conoces alguna?*

Implementación de una interfaz. Una clase solamente puede derivar de una clase base (`extends`), pero puede implementar varias interfaces escribiendo sus nombres separados por una coma después de la palabra reservada `implements`.

```
[listaModificadores] class NomClase [extends NomClase]
                          [implements listaInterfaces]
  {
       ...
  }
```

[1] Java 8 ya permite incluir métodos por defecto y métodos estáticos en las interfaces, pero en esta práctica suponemos el uso de una versión anterior de Java.

[2] Los corchetes indican opcionalidad y `listaInterfaces` es una lista de nombres de interfaces separados por comas.