Polimorfismo e Interfaces

Ejemplo Práctico (Case Study): sistema de nómina utilizando polimorfismo

Conversión descendente (downcasting) y operador **instanceof**;

Observación de ingeniería de software (SIO): conversión descendente permitida pero evitar; **ClassCastException**

Métodos **final** NO pueden ser sobrescritos (not **overriden**)

Métodos **private** y **static** son implícitamente **final**

Clases **final** NO pueden ser superclases

Todos los métodos en una clase **final** son implícitamente **final**

E.g. clase **String**

Al hacerse una clase **final** se evita que los programadores creen subclases que podrían ignorar las restricciones de seguridad.

Si algo puede ser **final**, debe ser **final**

Observación de ingeniería de software (SIO): en la API de Java, la vasta mayoría de clases NO se declara como **final**

No llame desde los constructores a los métodos que puedan sobrescribirse (overridable methods), e.g. método de validación **static**; es aceptable llamar a un método **static** desde un constructor, siempre y cuando el método no llame -directa o indirectamente- un método de instancia que pueda sobrescribirse.

Estandarización de las interacciones; las interfaces son una herramienta que requiera que las clases no relacionadas implementen un conjunto de métodos comunes; los objetos de software también se comunican a través de interfaces

La declaración de una interfaz empieza con la palabra clave **interface** y sólo puede contener constantes y métodos **abstract.** Todos los métodos que se declaran en una interfaz son de manera implícita **public** y **abstract**, y todos los campos son implícitamente **public**, **static** y **final**. Gpp: declarar SIN palabras clave

Para especificar que una clase implementa a una interfaz, agregamos la palabra clave **implements** y el nombre de la interfaz al final de la primera línea de la declaración de nuestra clase.

Implementar una interfaz es como firmar un contrato con el compilador que diga, “Declararé todos los métodos especificados por la interfaz, o declararé mi clase como **abstract**.”

Al igual que una clase **public**, una interfaz **public** se debe declarar en un archivo con el mismo nombre que la interfaz, y con la extensión de archivo .java.

A menudo, una interfaz se utiliza en vez de una clase **abstract** cuando no hay una implementación predeterminada que heredar.

Observación de ingeniería de software (SIO): muchos desarrolladores sienten que las interfaces son una tecnología de modelado aún más importante que las clases, en especial con las nuevas mejoras a las interfaces en Java SE 8

En el capítulo 15, Archivos, flujos y serialización de objetos, veremos la noción de etiquetado de interfaces (también conocidas como interfaces marcadoras), que son interfaces vacías que no tienen métodos ni valores constantes. Se utilizan para agregar relaciones del tipo es-un a las clases, e.g. serialización de objetos: **Serializable** al final de la primera línea de la declaración de su clase

UML expresa la relación entre una clase y una interfaz a través de una relación conocida como realización. Se dice que una clase realiza, o implementa, los métodos de una interfaz.

Cuando una clase implementa a una interfaz, se aplica la misma relación es-un que proporciona la herencia.
Observación de ingeniería de software (SIO): cuando el parámetro de un método se declara con un tipo de superclase o de interfaz, el método procesa en forma polimórfica al objeto que recibe como argumento.
Observación de ingeniería de software (SIO): al utilizar una referencia a la interfaz, podemos invocar de manera polimórfica a cualquier método declarado en la interfaz, en sus superinterfaces (una interfaz puede extender a otra) y en la clase **Object**.

Buena práctica de programación (GPP): Al declarar un método en una interfaz, seleccione un nombre para el método que describa su propósito en forma general, ya que podría implementarse por muchas clases no relacionadas.

Java no permite que las subclases hereden más de una superclase, pero sí que una clase herede de una superclase e implemente tantas interfaces como necesite.

La API de Java contiene numerosas interfaces, y muchos de los métodos de la API de Java reciben argumentos de interfaz y devuelven valores de interfaz

Mejoras a las interfaces de Java SE 8

Métodos **default** de una interfaz:

Antes de Java SE 8, los métodos de las interfaces solamente podían ser métodos **public** y **abstract**, pero en Java SE 8 las interfaces también pueden contener métodos **public default** con implementaciones predeterminadas concretas que especifiquen cómo deben realizarse las operaciones cuando una clase implementadora no sobrescriba los métodos

Cuando una clase implementa una interfaz de Java SE 8, la clase “firma un contrato” con el compilador que dice: “Declararé todos los métodos **abstract** especificados por la interfaz o declararé mi clase como **abstract**.” La clase implementadora no tiene que sobrescribir los métodos **default** de la interfaz, pero puede hacerlo si es necesario. Sio: los métodos **default** de Java SE 8 le permiten evolucionar las interfaces existentes al agregar nuevos métodos a esas interfaces sin descomponer el código que las usa.

Métodos **static** de una interfaz:

Antes de Java SE 8, era común asociar con una interfaz a una clase que tuviera métodos ayudantes **static** para trabajar con objetos que implementaran la interfaz; sin embargo, con los métodos **static** de una interfaz de Java SE 8, dichos métodos ayudantes pueden ahora declararse directamente en las interfaces, e.g. el método **sort** de **Collections** puede ordenar objetos de cualquier clase que implemente a la interfaz **List**

A partir de Java SE 8, cualquier interfaz que contenga sólo un método **abstract** se conoce como interfaz funcional. Las interfaces funcionales se usan mucho con las nuevas capacidades lambda de Java SE 8 que presentaremos en el capítulo 17 (Lambdas y flujos de Java SE 8)