7.E. Interfaces.

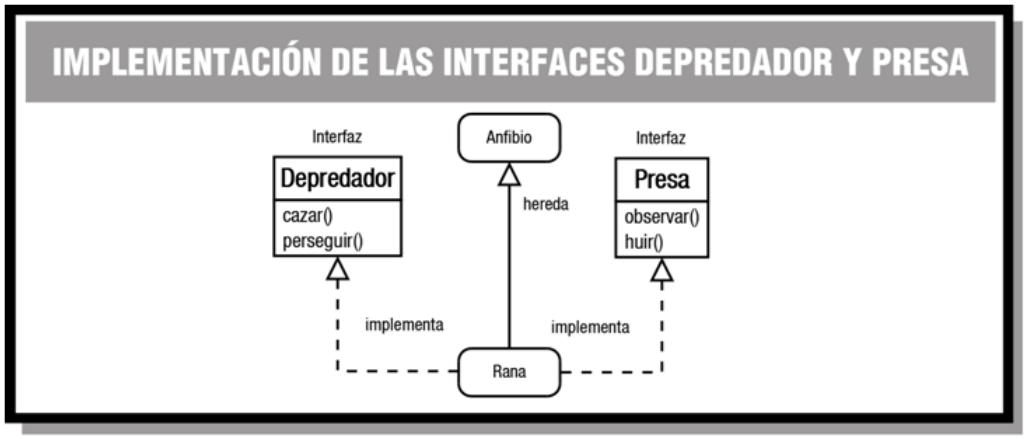
1. Interfaces.

1.5. Simulación de la herencia múltiple mediante el uso de interfaces.

Una **interfaz** no tiene **espacio de almacenamiento** asociado (no se van a declarar objetos de un tipo de interfaz), es decir, no tiene **implementación**.

En algunas ocasiones es posible que interese representar la situación de que "una clase **X** es de tipo **A**, de tipo **B**, y de tipo **C**", siendo **A**, **B**, **C** **clases disjuntas** (no heredan unas de otras). Hemos visto que sería un caso de **herencia múltiple** que Java no permite.

Para poder simular algo así, podrías definir tres **interfaces A**, **B**, **C** que indiquen los comportamientos (métodos) que se deberían tener según se pertenezca a una supuesta clase **A**, **B**, o **C**, pero sin implementar ningún método concreto ni atributos de objeto (sólo interfaz).



De esta manera la clase **X** podría a la vez:

1. Implementar las interfaces **A**, **B**, **C**, que la dotarían de los comportamientos que deseaba heredar de las clases **A**, **B**, **C**.
2. Heredar de otra clase **Y**, que le proporcionaría determinadas características dentro de su taxonomía o jerarquía de objeto (atributos, métodos implementados y métodos abstractos).

En el ejemplo que hemos visto de las interfaces **Depredador** y **Presa**, tendrías un ejemplo de esto: la clase **Rana**, que es subclase de **Anfibio**, implementa una serie de **comportamientos** propios de un **Depredador** y, a la vez, otros más propios de una **Presa**. Esos **comportamientos** (**métodos**) no forman parte de la **superclase Anfibio**, sino de las **interfaces**. Si se decide que la clase **Rana** debe de llevar a cabo algunos otros **comportamientos adicionales**, podrían añadirse a una **nueva interfaz** y la clase **Rana implementaría** una tercera **interfaz**.

De este modo, con el mecanismo "**una herencia pero varias interfaces**", podrían conseguirse resultados similares a los obtenidos con la **herencia múltiple**.

Ahora bien, del mismo modo que sucedía con la **herencia múltiple**, puede darse el problema de la **colisión de nombres** al implementar dos **interfaces** que tengan un **método con el mismo identificador**. En tal caso puede suceder lo siguiente:

- Si los dos métodos tienen **diferentes parámetros** no habrá problema aunque tengan el mismo nombre pues se realiza una **sobrecarga** de métodos.
- Si los dos métodos tienen **un valor de retorno de un tipo diferente**, se producirá un **error de compilación** (al igual que sucede en la sobrecarga cuando la única diferencia entre dos métodos es ésa).

Si los dos métodos son **exactamente iguales en identificador, parámetros y tipo devuelto**, entonces solamente se podrá **implementar uno de los dos métodos**. En realidad se trata de un solo método pues ambos tienen la misma interfaz (mismo identificador, mismos parámetros y mismo tipo devuelto).

Recomendación

La utilización de nombres idénticos en diferentes **interfaces** que pueden ser implementadas a la vez por una misma clase puede causar, además del problema de la **colisión de nombres**, dificultades de **legibilidad** en el código, pudiendo dar lugar a confusiones. Si es posible intenta evitar que se produzcan este tipo de situaciones.

Autoevaluación

Dada una clase Java que implementa dos interfaces diferentes que contienen un método con el mismo nombre, indicar cuál de las siguientes afirmaciones es correcta.

- Si los dos métodos tienen un valor de retorno de un tipo diferente, se producirá un error de compilación.
- Si los dos métodos tienen un valor de retorno de un tipo diferente, se implementarán dos métodos.
- Si los dos métodos son exactamente iguales en identificador, parámetros y tipo devuelto, se producirá un error de compilación.
- Si los dos métodos tienen diferentes parámetros se producirá un error de compilación.

Ejercicio resuelto

Localiza en la API de Java algún ejemplo de clase que implemente varias interfaces diferentes (puedes consultar la documentación de referencia de la API de Java).

Solución:

Existen una gran cantidad de clases en la API de Java que implementan **múltiples interfaces**. Aquí tienes un par de ejemplos:

- La clase javax.swing.JFrame, que implementa las **interfaces** WindowConstants, Accessible y RootPaneContainer:
  - public class JFrame extends Frame
  - implements WindowConstants, Accessible, RootPaneContainer

- La clase java.awt.component, que implementa las **interfaces** ImageObserver, MenuContainer y Serializable:
  - public abstract class Component extends Object
  - implements ImageObserver, MenuContainer, Serializable