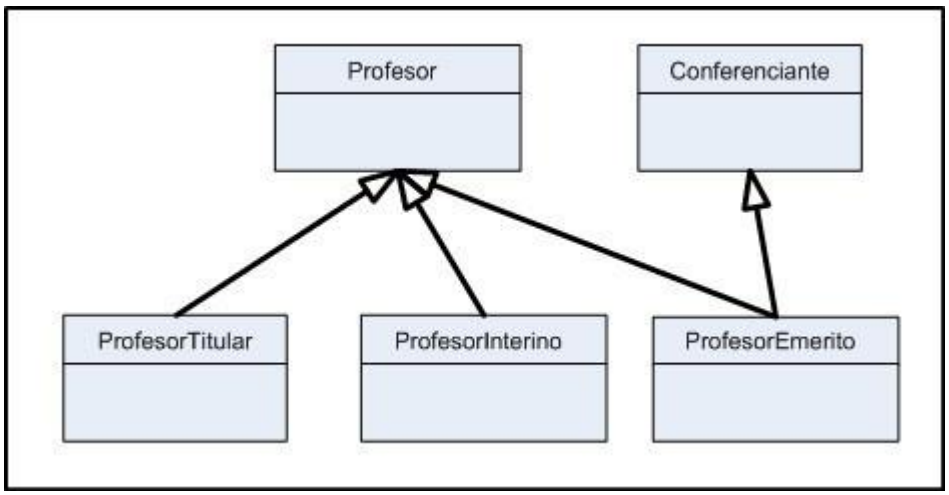
CONCEPTO DE INTERFACE Y HERENCIA MÚLTIPLE EN JAVA. IMPLEMENTS.

En apartados anteriores del tutorial hemos estudiado los conceptos de herencia y polimorfismo. Hasta ahora hemos considerado escenarios en que una clase hereda solo de otra clase. ¿Sería posible plantear un escenario donde una clase hereda de más de una clase (herencia múltiple)?



El esquema de la figura anterior representaría que hubiera clases como ProfesorEmerito que heredarían de dos clases: Profesor y Conferenciante. Esto sería un caso de herencia múltiple, y representaría que la subclase comparte las características de las dos superclases, y además tiene sus características específicas propias. La herencia múltiple, de cara a la consistencia de los programas y los lenguajes tiene una relativamente alta complejidad. De ahí que algunos lenguajes orientados a objetos la permitan y otros no. Java no permite la herencia múltiple, pero a cambio dispone de la construcción denominada “Interface” que permite una forma de simulación o implementación limitada de la herencia múltiple.

Ya hemos discutido el concepto de interfaz en alusión a la signatura de métodos o la información pública de las clases. También hemos hecho una primera aproximación al término interface en Java, y a modo de símil dijimos que podía considerarse como una norma de urbanismo en una ciudad. Vamos a profundizar en el concepto de interface dentro de Java. Un interface es una construcción similar a una clase abstracta en Java, pero con las siguientes diferencias:

a) En el encabezado se usa la **palabra clave interface** en lugar de class o abstract class. Por ejemplo public interface NombreDelInterface {…}

b) **Todo método es abstracto y público** sin necesidad de declararlo, es decir, no hace falta poner abstract public porque por defecto todos los métodos son abstract public. Por lo tanto un interface en Java no implementa ninguno de los métodos que declara: ninguno de sus métodos tiene cuerpo.

c) Las interfaces **no tienen ningún constructor.**

d) Un interfaz **solo admite campos de tipo “public static final”,** es decir, campos de clase, públicos y constantes. No hace falta incluir las palabras public static final porque todos los campos serán tratados como si llevaran estas

palabras. Recordemos que static equivalía a “de clase” y final a “constante”. Las interfaces pueden ser un lugar interesante para declarar constantes que van a ser usadas por diferentes clases en nuestros programas.

e) Una clase puede derivar de un interface de la misma manera en que puede derivar de otra clase. No obstante, se dice que **el interface se implementa (implements), no se extiende (extends)** por sus subclases. Por tanto para declarar la herencia de un interface se usa la palabra clave implements en lugar de extends.

Una clase puede implementar uno o varios interfaces en Java (se indica con implements NombreInterface1, NombreInterface2, …etc.), pero sólo puede extender una clase. Implementar varios interfaces en una sola clase es lo más parecido que tiene Java a la herencia múltiple.

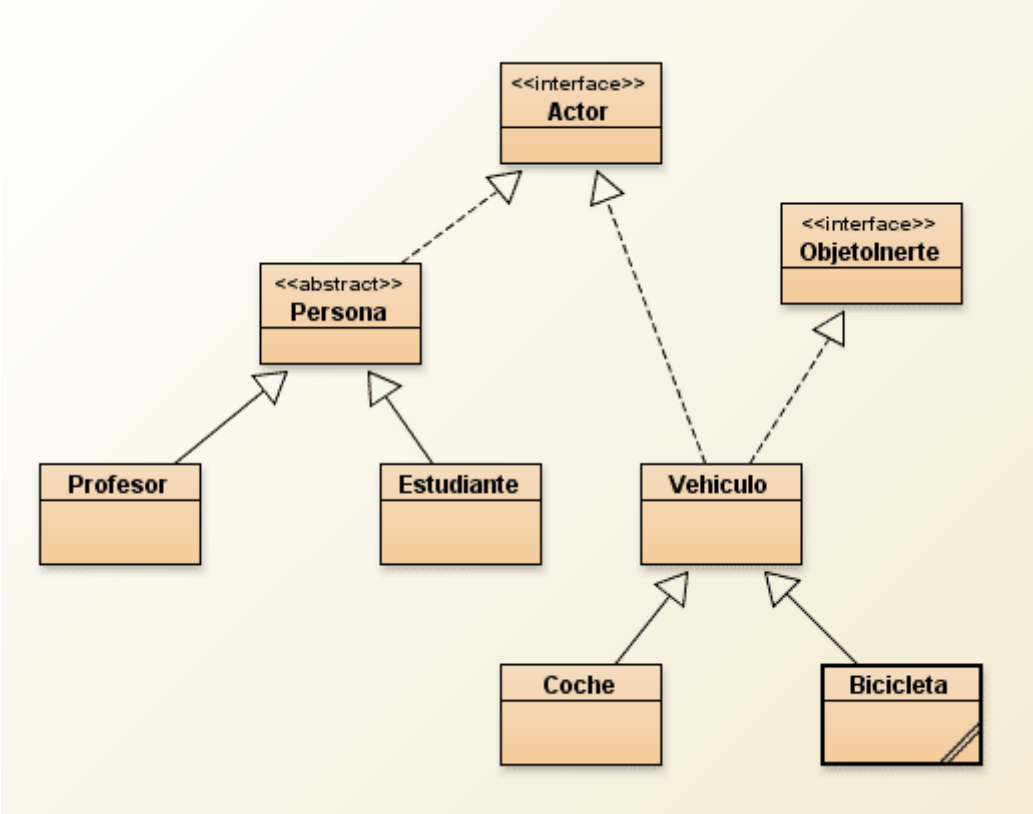
Podemos declarar variables del tipo interfaz, pero para inicializarlas utilizaremos una clase concreta que lo implemente. Por ejemplo List <String> miLista; declara una variable con el tipo de la interface List. La inicialización miLista = new List<String>(); no es posible porque no se puede crear un objeto del tipo definido por una interfaz. En cambio miLista = new LinkedList<String> (); sí es válido.

Diremos que **una interfaz en Java define un tipo cuyos métodos están todos sin implementar y que resulta equivalente a una herencia múltiple (de clases abstractas) en una clase.** Si una clase implementa una interface, puede suceder:

a) Que implemente los métodos de la interface sobreescribiéndolos (puede ser una clase concreta).

b) Que no implemente los métodos de la interface: obligatoriamente será una clase abstracta y obligatoriamente ha de llevar la palabra clave abstract en su encabezado para así indicarlo.

Consideremos un diagrama de clases como este, que podría emplearse para un programa de gestión en un centro educativo:



Vemos que las interfaces son identificadas por BlueJ con <<interface>> en la parte superior de su icono. Dentro de estas clases las relaciones quedan determinadas por este código:

public interface Actor {…}: define la interfaz actor ejemplo aprenderaprogramar.com.

public abstract class Persona implements Actor {…}: define la clase abstracta Persona como implementación de la interfaz Actor.

public class Profesor extends Persona{…}: define la clase Profesor como extensión de la clase Persona.

public class Estudiante extends Persona{…}: define la clase Estudiante como extensión de la clase Persona.

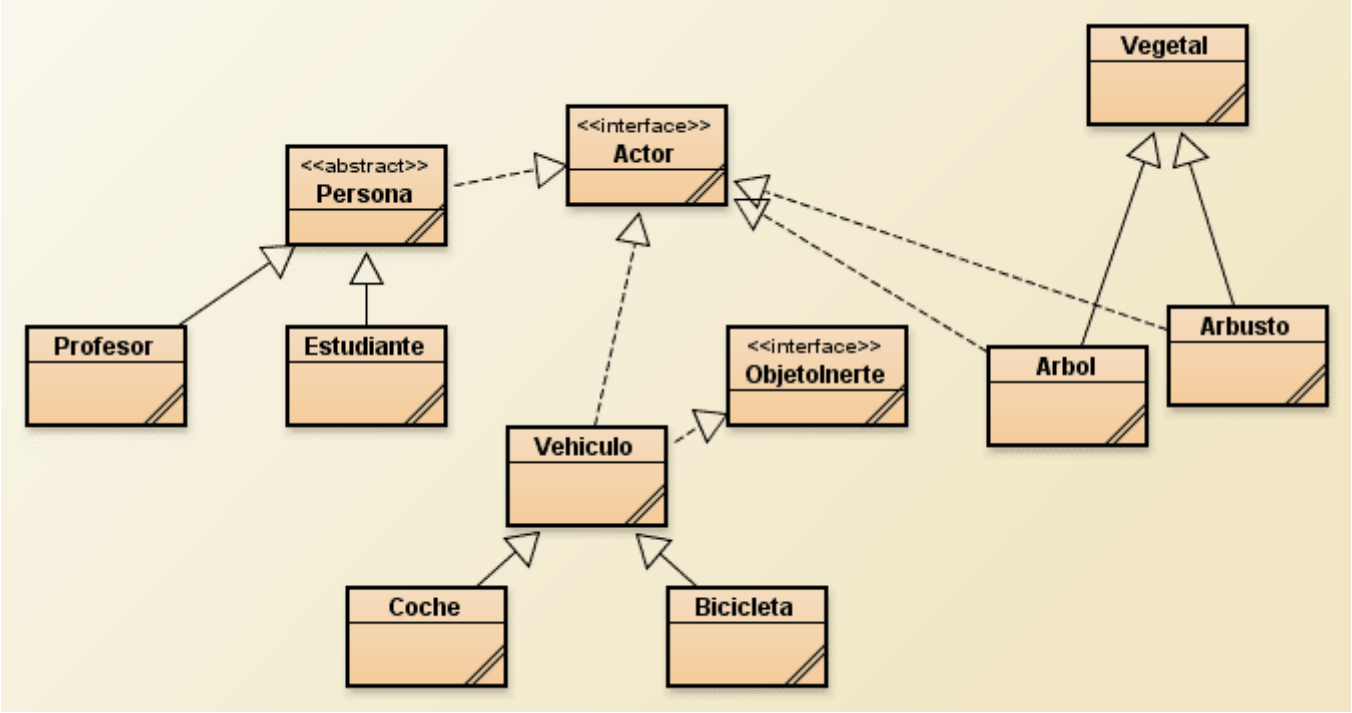
public interface ObjetoInerte {…}: define la interfaz ObjetoInerte.

public class Vehiculo implements Actor, ObjetoInerte {…}: define que la clase Vehiculo implementa a dos interfaces, la interfaz Actor y la interfaz ObjetoInerte, es decir, que un vehículo es a la vez Actor y ObjetoInerte.

public class Coche extends Vehiculo {…}: define la clase Coche como extensión de la clase Vehiculo.

public class Bicicleta extends Vehiculo {…}: define la clase Bicicleta como extensión de la clase Vehiculo.

Una clase podría heredar de otra e implementar una o varias interfaces. En este caso en primer lugar se pone la relación de herencia respecto a la superclase y a continuación las interfaces que implementa. Por ejemplo en este esquema:



La definición de la clase Arbol sería así: *public class Arbol extends Vegetal implements Actor {…}.* La clase hereda de Vegetal e implementa a Actor.

¿Cómo saber si una clase es candidata a ser definida como una interfaz?

a) Si necesitamos algún método con cuerpo ya sabemos que no va a ser una interfaz porque todos los métodos de una interfaz han de ser abstractos.

b) Si necesitamos que una clase "herede" de más de una superclase, esas superclases son candidatas a ser interfaces.

c) En algunos casos es igual de viable definir una clase como interfaz que como clase abstracta, pero puestos en esta situación preferiremos optar por una interfaz porque es más flexible y extensible: nos va a permitir que una clase implemente varias interfaces (aprovechamos la herencia múltiple de las interfaces). En cambio, una clase no puede heredar de varias clases.

Recordar que los campos declarados **son campos estáticos aunque no se indique específicamente:**

```
public interface Actor {

int activo = 1;
```

```
int inactivo = 0;

// … resto del código de la interface ejemplo aprenderaprogramar.com

      }
```

En este caso activo e inactivo se comportan como *public static final* (constantes) para todas las clases que implementen esta interfaz.

¿Cuál es uno de los intereses principales de usar interfaces? Poder hacer uso del polimorfismo: por ejemplo poder reunir en una colección objetos del tipo interface pero que están implementados en distintas clases. O poder tratar en un bucle objetos de distintos tipos pero que pertenecen al mismo supertipo porque implementan una interfaz.

EJERCICIO

Responde a las siguientes preguntas:

a) ¿Una clase puede heredar **de dos clases** en Java?

b) ¿Una interface Java puede tener métodos que incluyan una sentencia while? ¿Una interface Java puede tener métodos que incluyan una sentencia *System.out.println*?

c) ¿Un objeto Java puede ser **del tipo** definido por una interface? ¿Un objeto Java puede ser al mismo tiempo del tipo definido por una interface y del tipo definido por una clase que no implementa la interface? ¿Un objeto Java puede ser al mismo tiempo del tipo definido por una interface y del tipo definido por una clase que implementa la interface?