Java no permite herencia múltiple auténtica

La herencia múltiple puede provocar colisión en la herencia, por ejemplo una la clase base B hereda un atributo *nombre* de una superclase A1 y otro atributo también *nombre* de otra superclase A2. Los lenguajes que la permiten tienen mecanismos específicos para solventar este problema. Java simplemente no permite la herencia múltiple porque consideró importantes los problemas de mantenimiento que puede generar permitir esta herencia múltiple. El siguiente código no compila, no es posible un extends múltiple.

```
class A1{
   int nombre;
}
class A2{
   int nombre;
}

class B extends A1,A2{
   int b;
}
```

Se puede "simular", es decir, conseguir un efecto equivalente al de la herencia múltiple, utilizando por separado o conjuntamente dos mecanismos: interfaces y delegación.

una forma de conseguir herencia múltiple: extends + implements

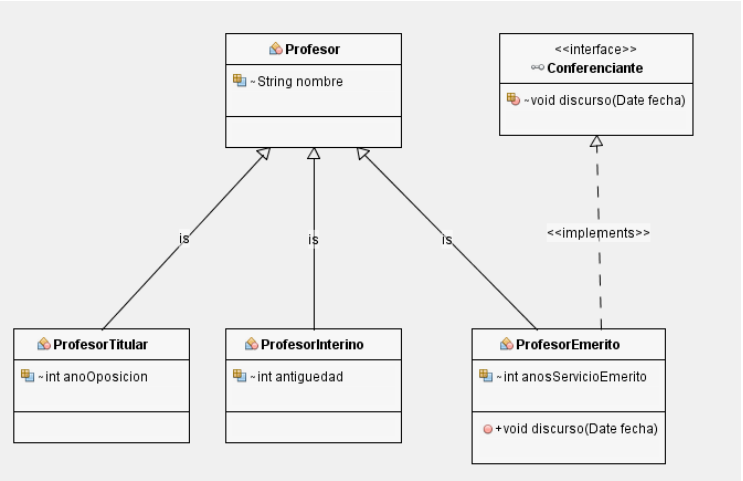
Una clase tiene una superclase principal que es con la que se usa extends. El resto de las herencias hay que diseñarlas de forma que consistan en implementar interfaces en lugar de extender clases.

class X extends A, B,C,D en Java no es posible, se hace:

class X extends A implements B,C,D

Esto da soporte a la herencia múltiple si parte de lo que se quiere heredar "encaja" con un interface, es decir, se quiere heredar "un comportamiento" a través del interface

Ejercicio U5_B7A_E1: Traduce a Java el siguiente ejemplo en el que un profesor emérito extiende a profesor pero también necesitamos que implemente a un Conferenciante. Como conferenciante es un "comportamiento" (todo métodos, sin estado, o sea sin atributos instancia), puede definirse como un interface y heredarse vía implements.



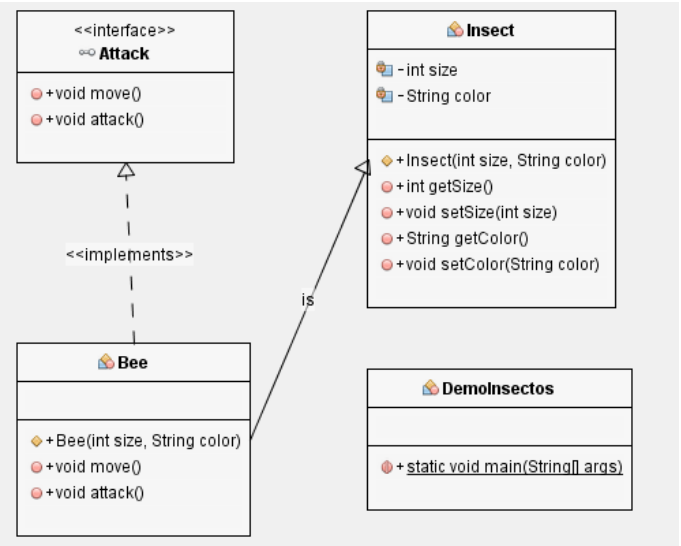
Para que se pueda ejecutar el siguiente main

```
public class App {
   public static void main(String[] args) {
      ProfesorEmerito pe= new ProfesorEmerito();
      pe.discurso( new Date(45788));
   }
}
```

nota: en java se puede usar la ñ como cualquier otro carácter unicode, no obstante en el ejemplo usamos n en lugar de ñ (en año) ya que la herramienta easy uml empleada para generar el diagrama genera error al usar el caracter ñ.

Ejercicio U5_B7_E2:

Tenemos un comportamiento "ataque". Atacar consiste realmente en dos acciones, un movimiento previo como volar, correr, saltar, etc. y el ataque en sí como morder, picar, golpear,etc. Una abeja es un insecto y por otro lado "ataca" su ataque consiste en primero "fly" (volar) para luego "sting" (aguijonear o picar). Escribe la siguiente estructura



de forma que

```
public class DemoInsectos {
   public static void main(String[] args) {
      Bee a = new Bee(1, "black");
      a.attack();
   }
}
```

produzca la salida:
fly
sting

conclusiones:

1. La herencia múltiple “pura” genera problemas de mantenimiento y se decidió no permitirla directamente en Java
2. Pero como al mismo tiempo es una necesidad del modelado de objetos se intenta simular
3. La simulación más simple es con la combinación extends + implements. Esto soluciona algunas situaciones que ocurren en contextos de herencia múltiple, pero no todas. Nos hará falta composición para mejorar la simulación.