

SEMINARIO

1

Herencia

Diseño del Software

Grado en Ingeniería Informática del Software

Curso 2024-2025





Vector



Stack

Pila

Una pila es una estructura de datos LIFO (el último que entra, el primero que sale).

Se caracteriza por, como mínimo, un par de operaciones para meter y sacar elementos.

push

pop

Vector

Un vector es, en esencia, una lista basada en arrays.

¿Qué creéis?

¿Por qué no?

¿Stack no debería ser una interfaz?

Si es una clase que heredase de Vector no definiría simplemente un tipo, sino que nos obligaría a una determinada implementación.

¿Stack no debería ser una interfaz?

Lo veremos en los próximos días en
clase de teoría.

¿Stack no debería ser una interfaz?

Por ahora centrémonos simplemente en si tiene sentido que sea un subtipo de Vector o no.



```
Stack<String> stack = new Stack<>();
stack.push("1");
stack.push("2");
stack.push("3");
stack.insertElementAt("¡Cuélame!", 1);
while (!stack.isEmpty()) {
    System.out.println(stack.pop());
}
```



```
Stack<String> stack = new Stack<>();
stack.push("1");
stack.push("2");
stack.push("3");
stack.insertElementAt("¡Cuélame!", 1);
while (!stack.isEmpty()) {
    System.out.println(stack.pop());
}
```



```
Stack<String> stack = new Stack<>();
stack.push("1");
stack.push("2");
stack.push("3");
stack.insertElementAt("¡Cuélame!", 1);
while (!stack.isEmpty()) {
    System.out.println(stack.pop());
}

// prints 3, ¡Cuélame!, 2, 1
```



```
Stack<String> stack = new Stack<>();
stack.push("1");
stack.push("2");
stack.push("3");
stack.insertElementAt("¡Cuélame!", 1);
while (!stack.isEmpty()) {
    System.out.println(stack.pop());
}

// prints 3, ¡Cuélame!, 2, 1
```

¿Qué ha ocurrido?

Que una pila, claramente,
no es un vector.

Es decir, estamos heredando operaciones
de Vector que claramente no pertenecen
al tipo abstracto Pila: violan su contrato.

¿Vemos cómo lo
hace Java?

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP Java SE 22 & JDK 22

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH Search X

Module java.base
Package java.util

Class Stack<E>

java.lang.Object
 java.util.AbstractCollection<E>
 java.util.AbstractList<E>
 java.util.Vector<E>
 java.util.Stack<E>

Type Parameters:
E - Type of component elements

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess, SequencedCollection<E>

public class Stack<E>
extends Vector<E>

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vector to be treated as a stack. The usual push and pop operations are provided, as well as a method to peek at the top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the Deque interface and its implementations, which should be used in preference to this class. For example:



La biblioteca de clases
de Java no siempre es
perfecta.

Inheritance is appropriate only in circumstances where the subclass really is a subtype of the superclass. In other words, a class B should only extend a class A only if an "is-a" relationship exists between the two classes. If you are tempted to have a class B extend a class A, ask yourself this question: Is every B really an A? If you cannot truthfully answer yes to this question, B should not extend A.

—Bloch (2008)

Inheritance is appropriate only in circumstances where the subclass really is a subtype of the superclass. In other words, a class B should only extend a class A only if an "is-a" relationship exists between the two classes. If you are tempted to have a class B extend a class A, ask yourself this question: Is every B really an A? If you cannot truthfully answer yes to this question, B should not extend A.

—Bloch (2008)

If the answer is no, it is often the case that B should contain a private instance of A and expose a smaller and simpler API; A is not an essential part of B, merely a detail of its implementation.

If the answer is no, it is often the case that B should contain a private instance of A and expose a smaller and simpler API; A is not an essential part of B, merely a detail of its implementation.

There are a number of obvious violations of this principle in the Java platform libraries. For example, a stack is not a vector, so Stack should not extend Vector. Similarly, a property list is not a hash table, so Properties should not extend Hashtable. In both cases, composition would have been preferable.

There are a number of obvious violations of this principle in the Java platform libraries. For example, a stack is not a vector, so Stack should not extend Vector. Similarly, a property list is not a hash table, so Properties should not extend Hashtable. In both cases, composition would have been preferable.

Pero...

¿Es suficiente el criterio «es un»?

iNO!

¿Una ventana es un rectángulo?

¿Un pingüino es un pájaro raro?

¿Un dibujo es una figura?

¿Un círculo es un punto gordo?

**Las jerarquías
surgen como
una necesidad.**

¿De quién?

**De los
clientes.**

¿Un pingüino es un pájaro? Lo será solo si se quiere que el pingüino sea una de las formas de implementar dicha responsabilidad; es decir, si se quiere que pueda ser usado allá donde se requiera un pájaro.

¡Y el pingüino será a su vez una figura si se quiere que se pueda añadir a un dibujo para dibujarse!

Y por esa razón una ventana no deriva de rectángulo aunque, efectivamente, «ES UN» rectángulo (no pretende usarse como sustituto del mismo en ningún momento).

Que en español quede bien
una frase no es razón para
usar el polimorfismo (máxime
cuando esta puede expresarse
de formas distintas).

Criterio de pertenencia

Un objeto pertenecerá a una jerarquía si se quiere poder conectar a un cliente para este lo use como a uno más de la misma.

Para ofrecerle una implementación alternativa.

Criterio de pertenencia

Para ello deberá cumplir todas las responsabilidades de la interfaz.

Con que una no sea aplicable el objeto no pertenecerá a la jerarquía.

**¡Nunca debe
surgir una
jerarquía como
una mera
clasificación!**

¿Es una pila un vector al que se le añaden un par de operaciones push/pop? No importa, no es la forma de averiguarlo.

De lo que se trata es de: ¿será pasado un objeto Stack en los lugares donde se requiera un objeto Vector? ¿Es una alternativa de implementación? No.

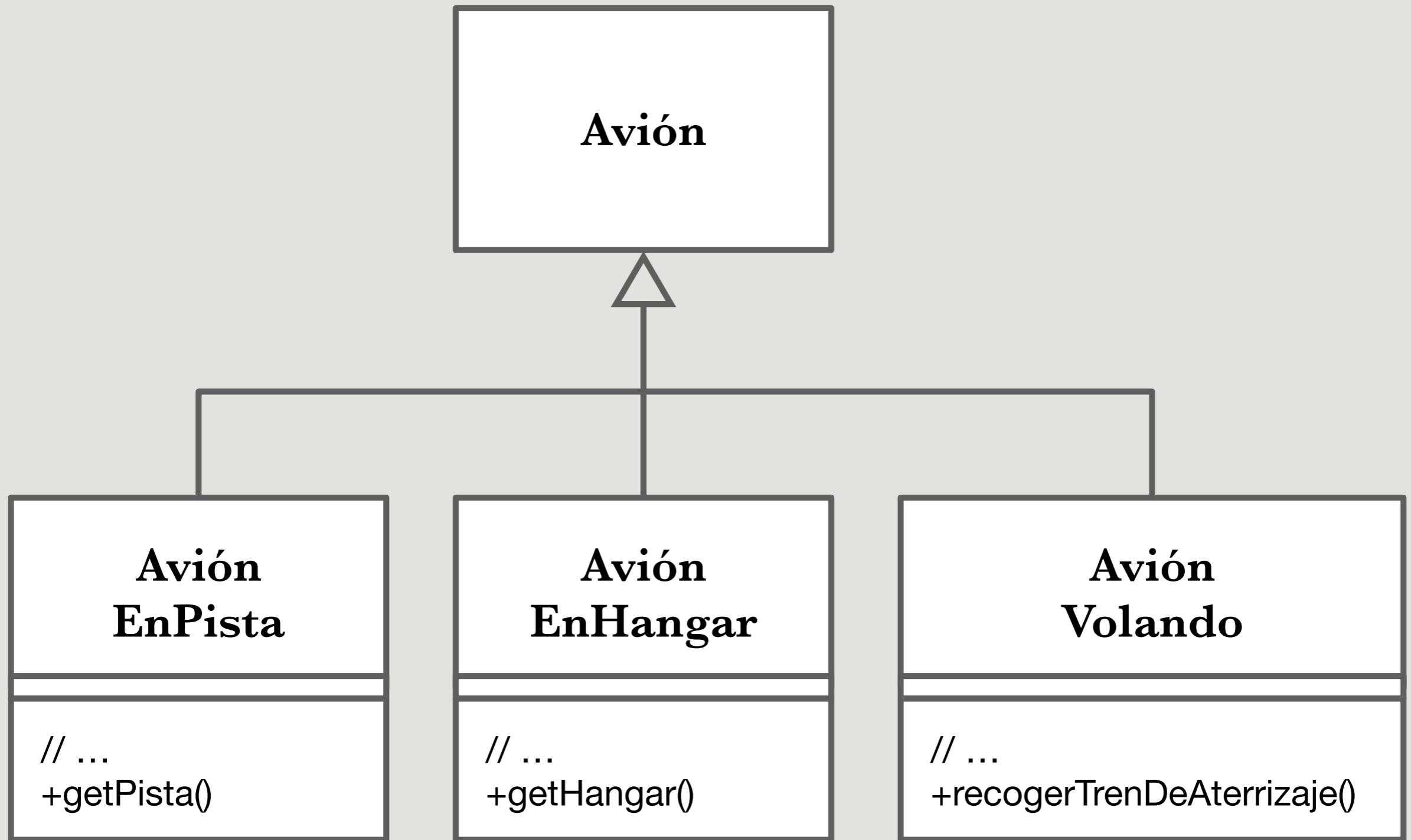
¿Cómo debería hacerse? La pila debería haber utilizado un vector en su implementación privada, protegiendo así los métodos de aquel que no debieran invocarse.



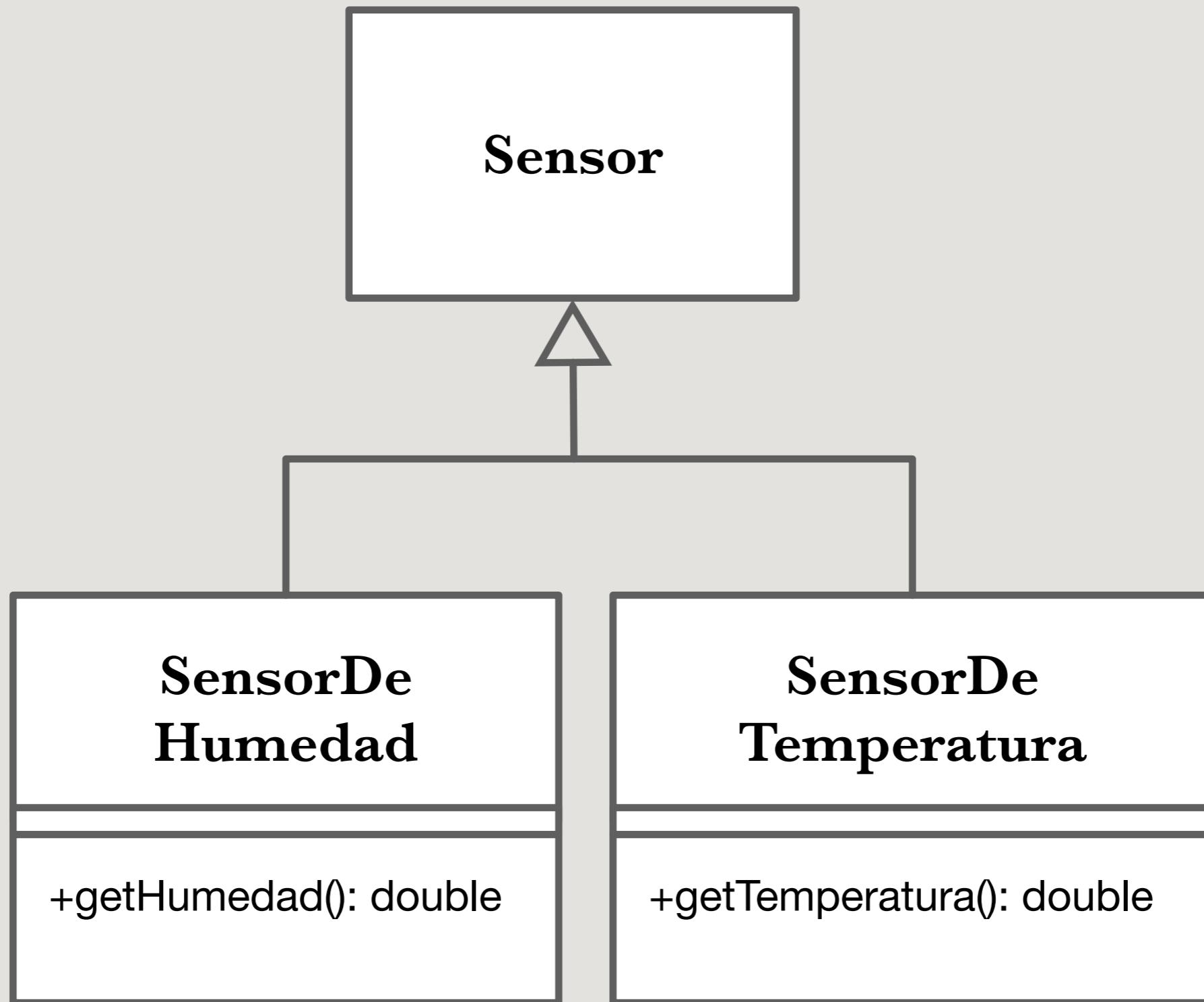
Para la torre de control un avión puede estar en el hangar, en la pista a punto de despegar (o tras el aterrizaje), en vuelo...



En cada una de esas situaciones tendrán sentido unas operaciones u otras (por ejemplo, saber el hangar o la pista en la que está, recoger el tren de aterrizaje, etcétera).





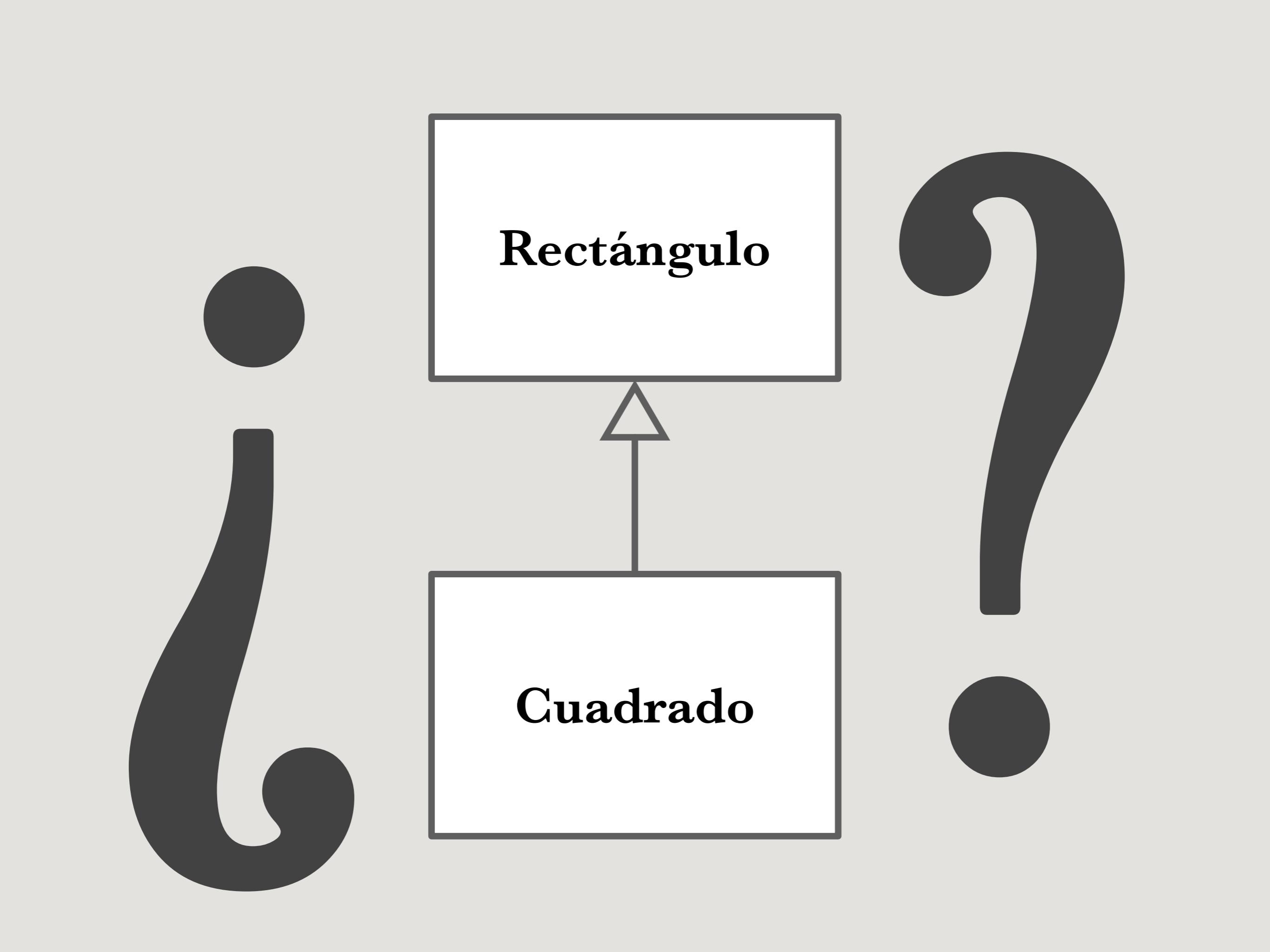


Nunca se pasará un sensor sin que importe cuál de los dos es. (No podemos tratarlos polimórficamente).

No es que tengan distintos métodos: aunque los dos se llamasen `getValor` y se subieran a la clase padre estaría mal.

Para los clientes serían objetos de tipos distintos, sin relación de herencia.

**Un cuadrado, ¿es
un rectángulo?**



Rectángulo

Cuadrado

¿Podemos sustituir un rectángulo por un cuadrado en todos los sitios en que se espere una referencia al primero?

Aquí no resulta tan obvio, ¿verdad?



Rectangle.java

```
class Rectangle {  
    private Point topLeft;  
    private double width;  
    private double height;  
  
    public void setWidth(double width) {  
        this.width = width;  
    }  
  
    public void setHeight(double height) {  
        this.height = height;  
    }  
  
    // ...  
}
```

Problema

Al heredar dichas implementaciones, un cuadrado violaría su definición.

Sus lados dejan de ser iguales.

Pero dicho problema tiene fácil solución, ¿no?



Square.java

```
class Square extends Rectangle {  
  
    // ...  
  
    public void setWidth(double width) {  
        super.setWidth(width);  
        super.setHeight(width);  
    }  
  
    public void setHeight(double height) {  
        super.setHeight(height);  
        super.setWidth(height);  
    }  
}
```



Square.java

```
class Square extends Rectangle {  
  
    // ...  
  
    public void setWidth(double width) {  
        super.setWidth(width);  
        super.setHeight(width);  
    }  
  
    public void setHeight(double height) {  
        super.setHeight(height);  
        super.setWidth(height);  
    }  
}
```

Ahora, cuando alguien cambia el ancho del cuadrado, su altura cambia también (y viceversa).

De ese modo, las invariantes^{*} del cuadrado se mantienen intactas.

* Las propiedades de la clase que deben ser ciertas siempre, independientemente del estado.

Pero...



```
void g(Rectangle r) {  
    r.setWidth(5);  
    r.setHeight(4);  
    assert r.area() == 20;  
}
```



```
void g(Rectangle r) {  
    r.setWidth(5);  
    r.setHeight(4);  
    assert r.area() == 20;  
}
```

¿Qué ha pasado?

Un método de algún cliente cambió el ancho y el alto de lo que creía ser un rectángulo.

El área, naturalmente, no es correcto.

¡Cómo podía pensar el autor del método g que cambiar el ancho de un rectángulo iba a cambiar también su alto!

Para ese método, al menos, los rectángulos y cuadrados no son intercambiables.

Conclusiones

Sobre la validez de un diseño

Un diseño, considerado aisladamente, no se puede decir si es o no válido. La validez solo puede ser expresada en función de sus clientes.

O, dicho de otro modo, de las suposiciones razonables que harán los usuarios de dicho diseño.

Sobre el criterio «ES-UN»

Pero, ¿qué ocurre entonces? Después de todo,
¿acaso un cuadrado no es un también un rectángulo?
¿No cumple la famosa relación «es un»?

**¡No para el autor
del método g!**

Para él, un cuadrado,
definitivamente, no es un
rectángulo.

¿Por qué?

¡Porque no se comportan igual!

Así pues...

El criterio para decidir si un objeto es un subtipo de otro depende de su comportamiento, desde el punto de vista de las suposiciones razonables que puedan hacer los usuarios de dicho diseño (es decir, sus clientes).

Por cierto...

¿Sería así en todos los casos?

¿El cuadrado nunca podría ser una subclase del rectángulo?

Pensadlo.