

9  
2

# Introducción al diseño 00

## Diseño del Software

Grado en Ingeniería Informática del Software

Curso 2024-2025

9  
2

# Introducción al diseño OO

1

## Elementos del modelo de objetos

¿Qué significa  
«orientado a  
objetos»?

No resulta fácil de definir.

*I have a cat named Trash. In the current political climate it would seem that if I were trying to sell him at least to a Computer Scientist, I would not stress that he is gentle to humans and is self-sufficient, living mostly on field mice. Rather, I would argue that he is object-oriented.*

*—Kind (1989)*

*Después de todo, un gato exhibe un comportamiento característico, responde a mensajes, posee una larga tradición de respuestas heredadas y se encarga de gestionar su propio estado interno de manera bastante independiente.*

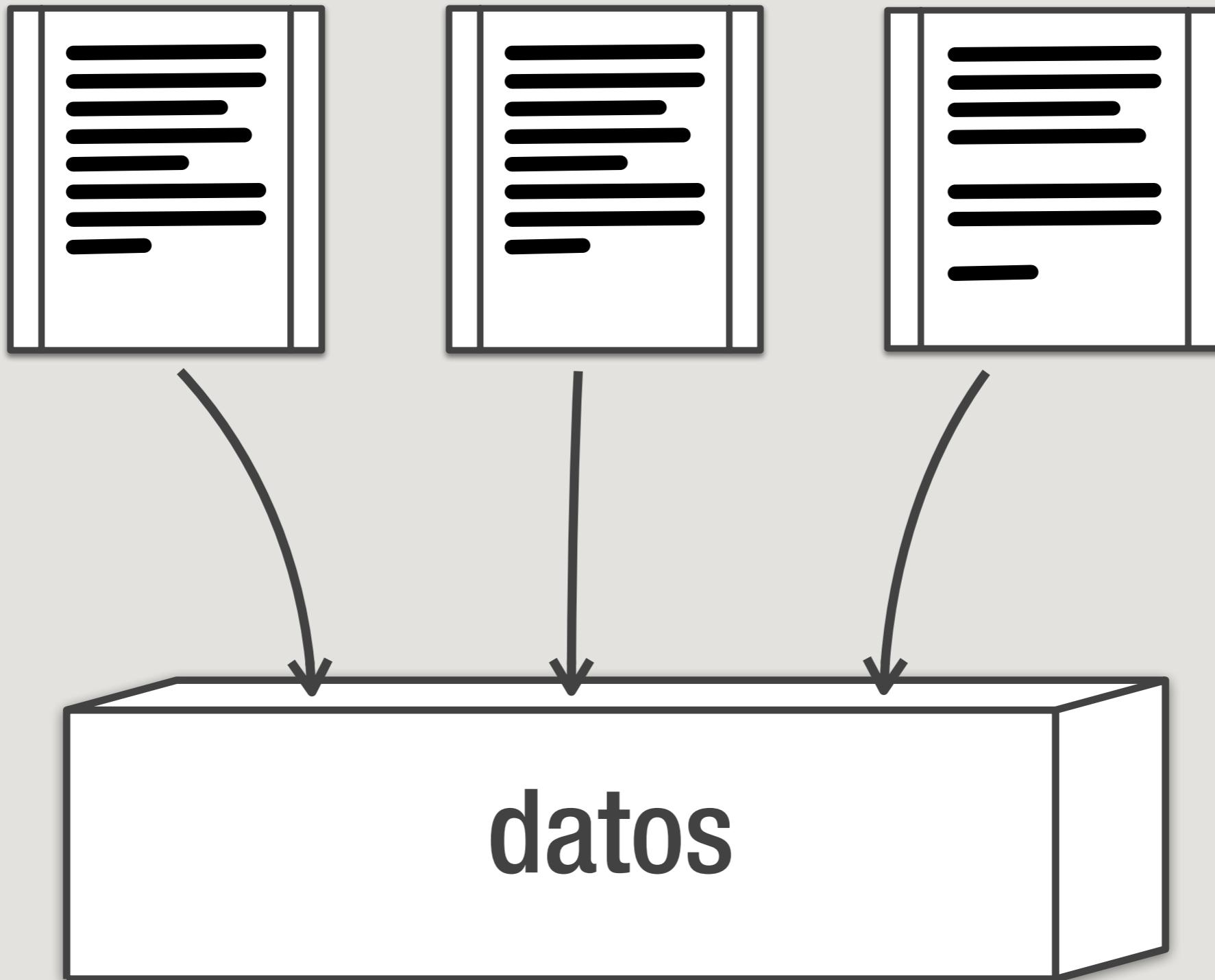
*—Kind (1989)*

Otra  
«definición»:

*X is Good*  
*Object-Oriented is Good*  
*Ergo, X is Object-Oriented*

**La programación estructurada  
separaba las estructuras de  
datos de los algoritmos que  
las manipulaban.**

# subrutinas



**En los lenguajes imperativos  
el modelo de invocación que  
se sigue es el de las funciones  
matemáticas.**

**f(x)**

$$f(x)$$

Se aplica una función a unos datos y se obtiene un resultado.

**f(x)**

Se decide qué hay que hacer y se distribuye en una jerarquía de funciones.

**En orientación a objetos no  
hay que seguir ese modelo.**

**tú.hazEsto()**

# tú.hazEsto()

Ahora se trata de decirle a  
alguien que haga algo.

# tú.hazEsto()

No solo hay que decidir el  
qué, sino también quién.

# tú.hazEsto()

Si se aprende a hacer las cosas sin un quién, luego no se ve la necesidad de introducirlo.



# Encapsulamiento

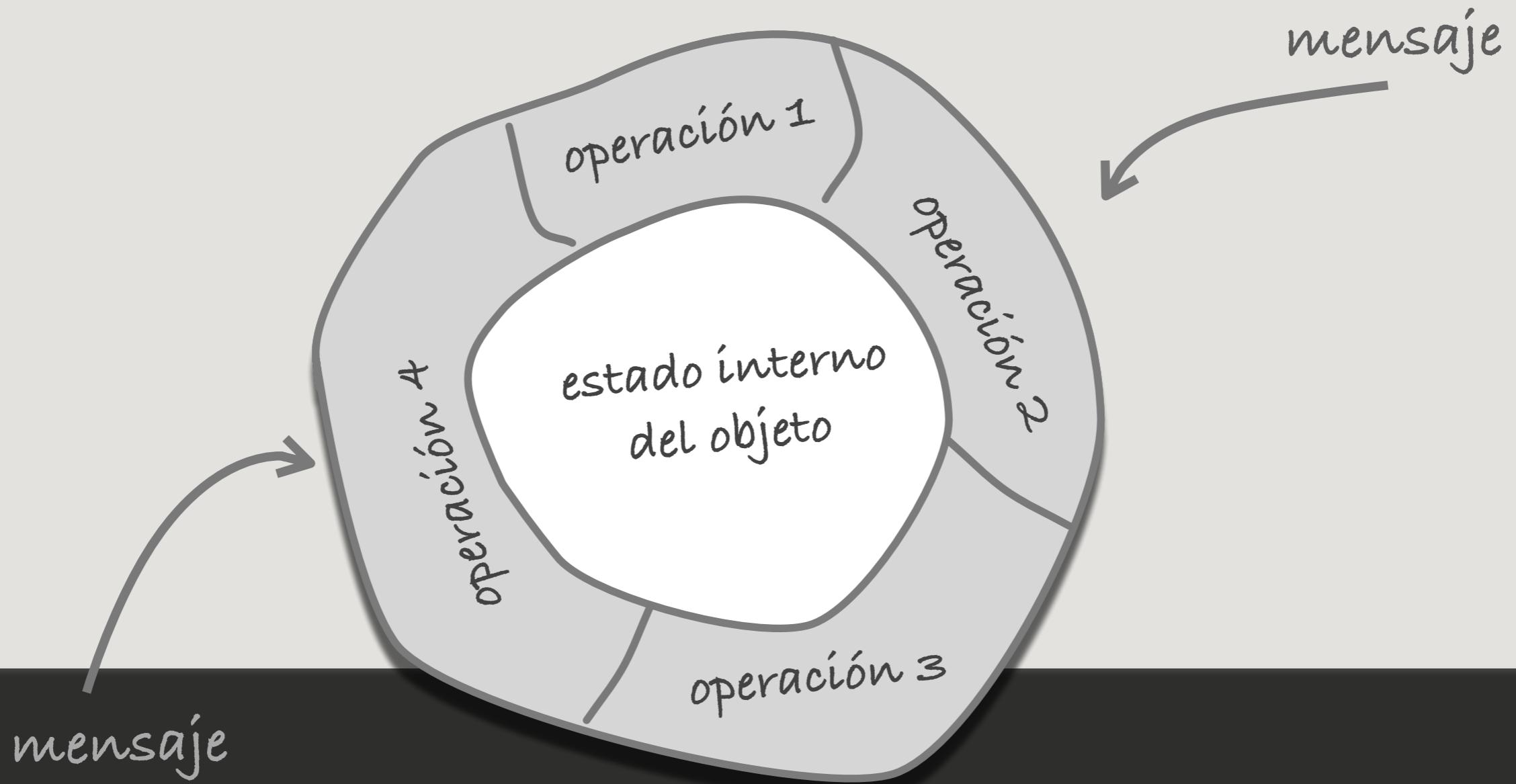


objeto

# Ocultación de la información

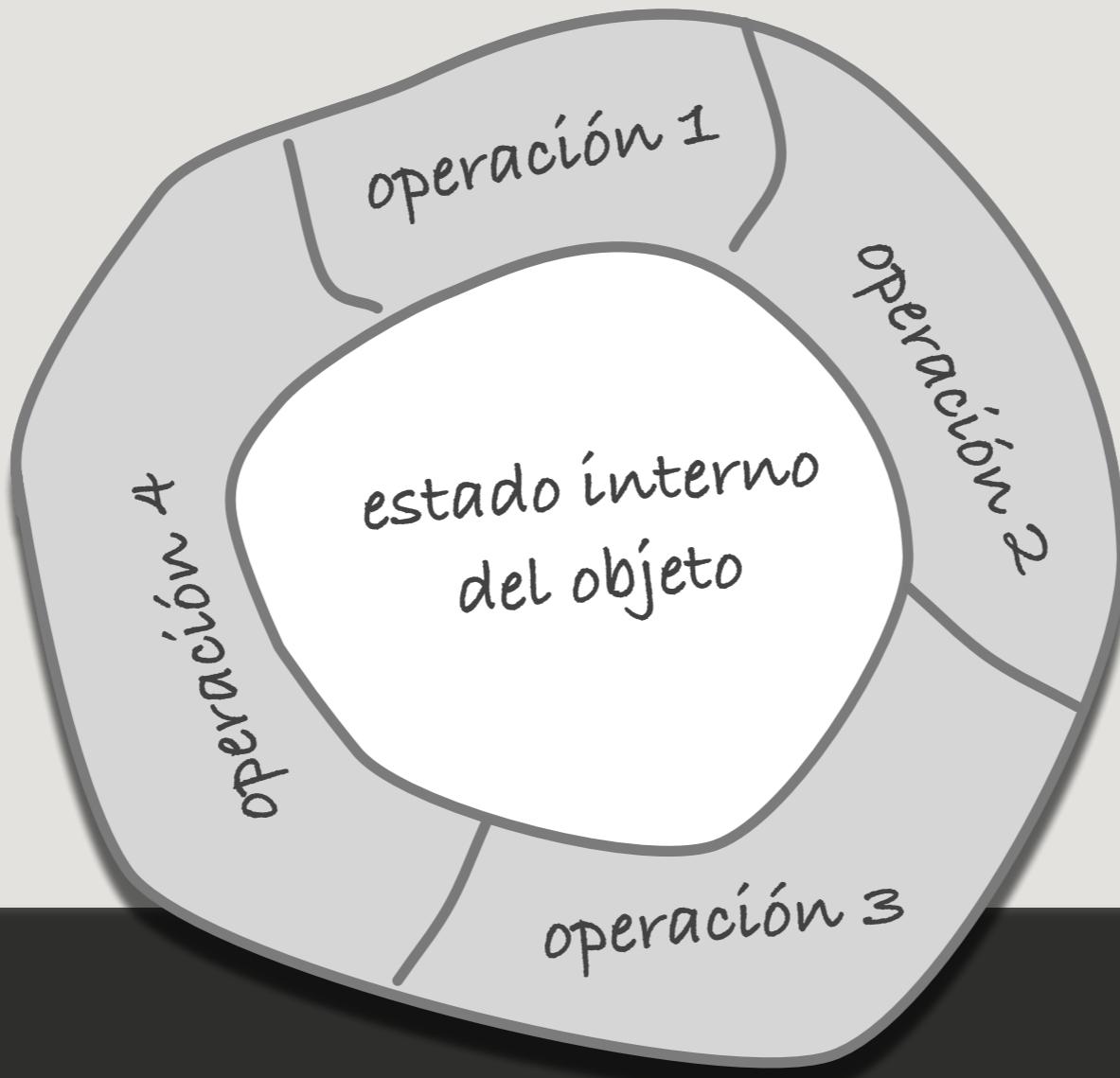
No se puede acceder al estado interno del objeto directamente.

Su representación es invisible desde el exterior del objeto.



# Mensajes

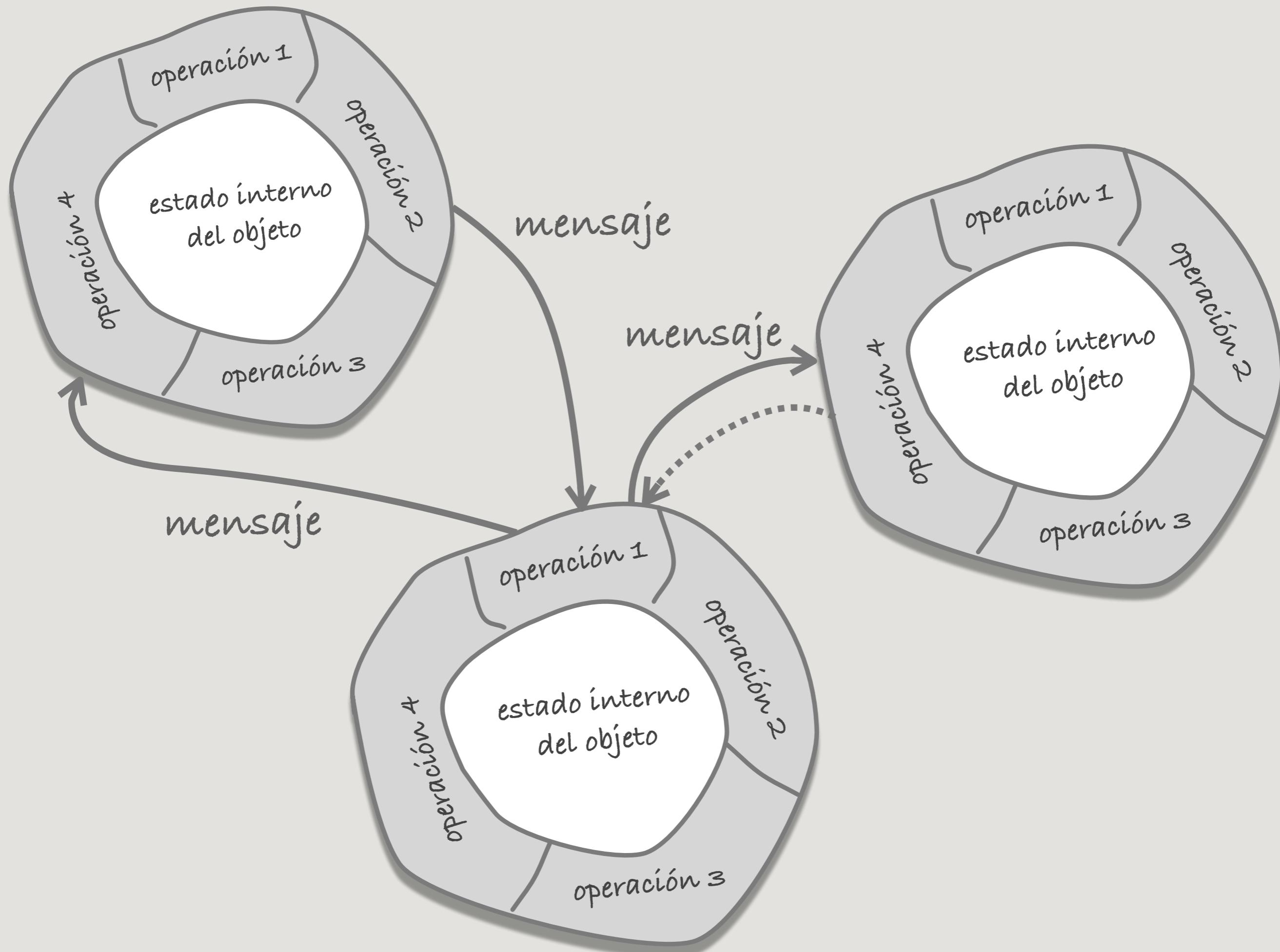


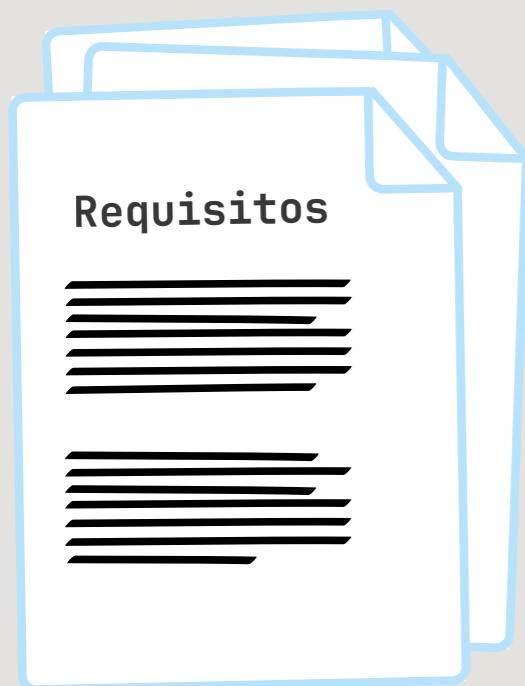


# Colaboraciones

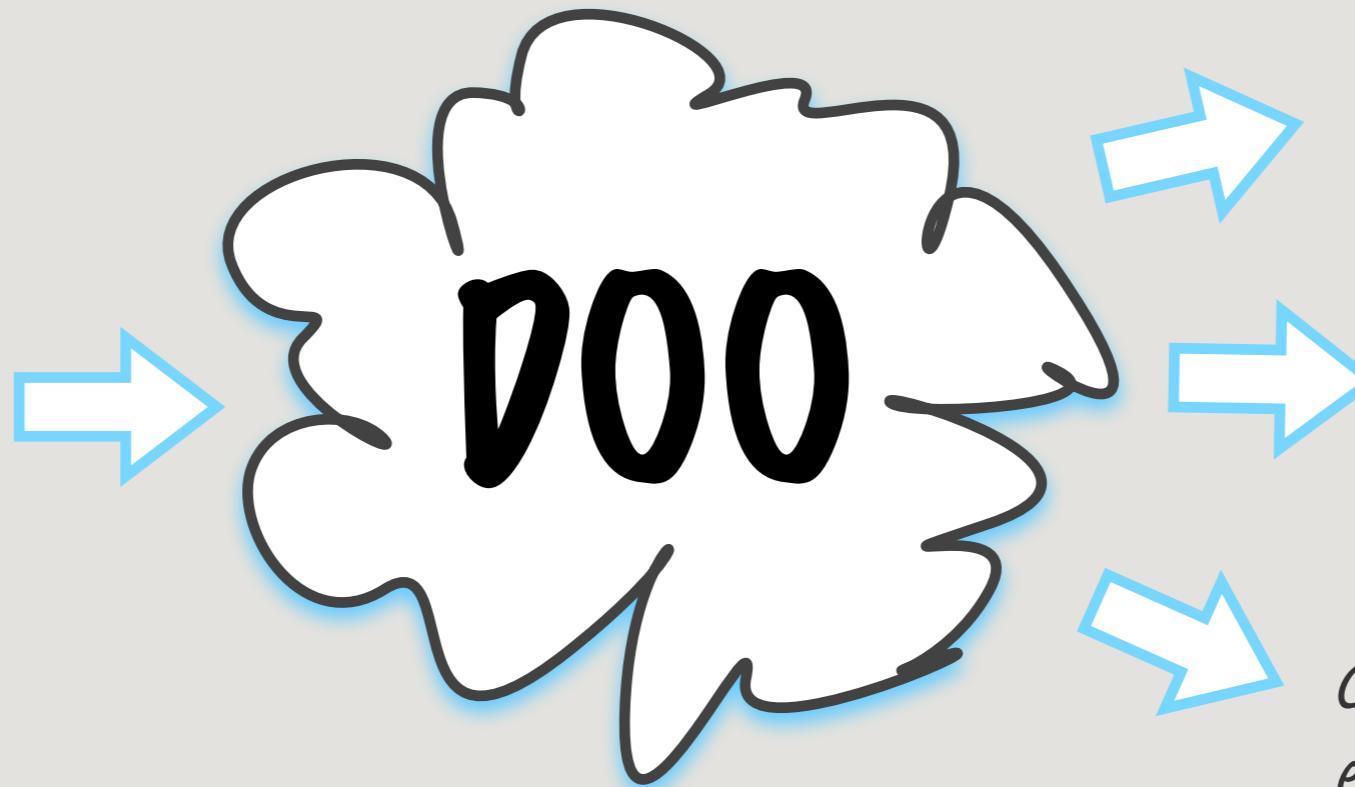
# Colaboraciones

El diseño OO construye las aplicaciones a base de una estructura de objetos que cooperan entre sí.





Análisis



Diseño orientado  
a objetos

*Un sistema de  
objetos que satisfacen  
los requisitos*

*una descripción del  
comportamiento  
público de esos objetos*

*Cómo se comunican  
esos objetos entre sí*

# Mensajes o peticiones, operaciones, métodos

Distintas caras de una misma  
moneda: las responsabilidades.

# Signatura de un método

Su nombre, los parámetros que recibe y el tipo que devuelve.

# **Un método representa dos cosas**

Declara un mensaje que es  
capaz de recibir el objeto.

**Un método representa  
dos cosas**

Y cómo reacciona ante la  
recepción del mismo.

# Interfaz de un objeto

El conjunto de todas las  
signaturas públicas definidas  
por él (qué podemos pedirle).

# Tipo

El nombre que denota una determinada interfaz.

# Tipo

Parte de la interfaz de un objeto  
puede estar caracterizada por  
un tipo, y el resto por otros.

¿Qué quiere decir?

**Síntomas de  
programación  
estructurada  
encubierta**

# Métodos largos

No se está delegando.

**Abundancia de  
llamadas a métodos  
del mismo objeto**

**«C con clases»**

# Preguntarse cómo se calcula el factorial

Lo importante es quién, no cómo.

# **La típica clase ‘Util’**

No se encuentra un quién: no se han asignado bien las responsabilidades.

Screenshot of the Java SE 22 & JDK 22 documentation for the `Math` class. The page shows the `PI` and `TAU` constants.

**CLASS** is selected in the navigation bar.

**SEARCH** bar contains "Search".

Constant	Description
<code>static final double PI</code>	The double value that is closer than any other to <i>pi</i> ( $\pi$ ), the ratio of the circumference of a circle to its diameter.
<code>static final double TAU</code>	The double value that is closer than any other to <i>tau</i> ( $\tau$ ), the ratio of the circumference of a circle to its radius.

## Method Summary

All Methods    Static Methods    Concrete Methods

Modifier and Type	Method	Description
static double	<code>abs(double a)</code>	Returns the absolute value of a double value.
static float	<code>abs(float a)</code>	Returns the absolute value of a float value.
static int	<code>abs(int a)</code>	Returns the absolute value of an int value.
static long	<code>abs(long a)</code>	Returns the absolute value of a long value.
static int	<code>absExact(int a)</code>	Returns the mathematical absolute value of an int value if it is exactly representable as an int, throwing <code>ArithmeticException</code> if the result overflows the positive int range.
static long	<code>absExact(long a)</code>	Returns the mathematical absolute value of an long value if it is exactly representable as an long, throwing <code>ArithmeticException</code> if the result overflows the positive long range.
static double	<code>acos(double a)</code>	Returns the arc cosine of a value; the returned angle is in the range 0.0 through <i>pi</i> .

# Enlace dinámico

Y su consecuencia: el  
polimorfismo

# Enlace dinámico

Es la asociación en tiempo de ejecución entre una petición a un objeto y un método concreto.



receptor.peticIÓN();

# Polimorfismo

El enlace dinámico permite sustituir en tiempo de ejecución un objeto por otro con su misma interfaz.

Y esto, queridos alumnos,  
es la verdadera esencia de  
la orientación a objetos.

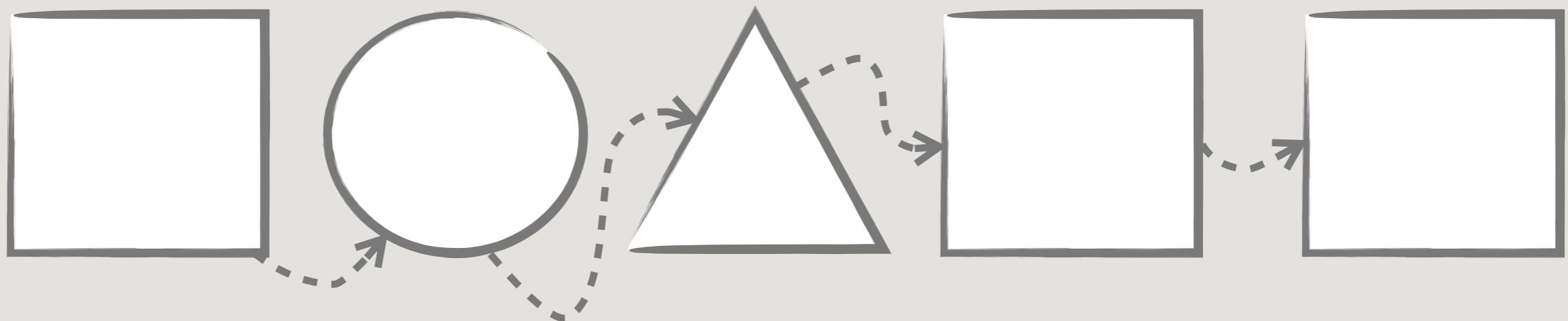
Un objeto no necesita  
«conocer» a quién está  
enviando un mensaje.

Escribid cómo sería  
el código para dibujar  
una lista de figuras



Dibujo.java

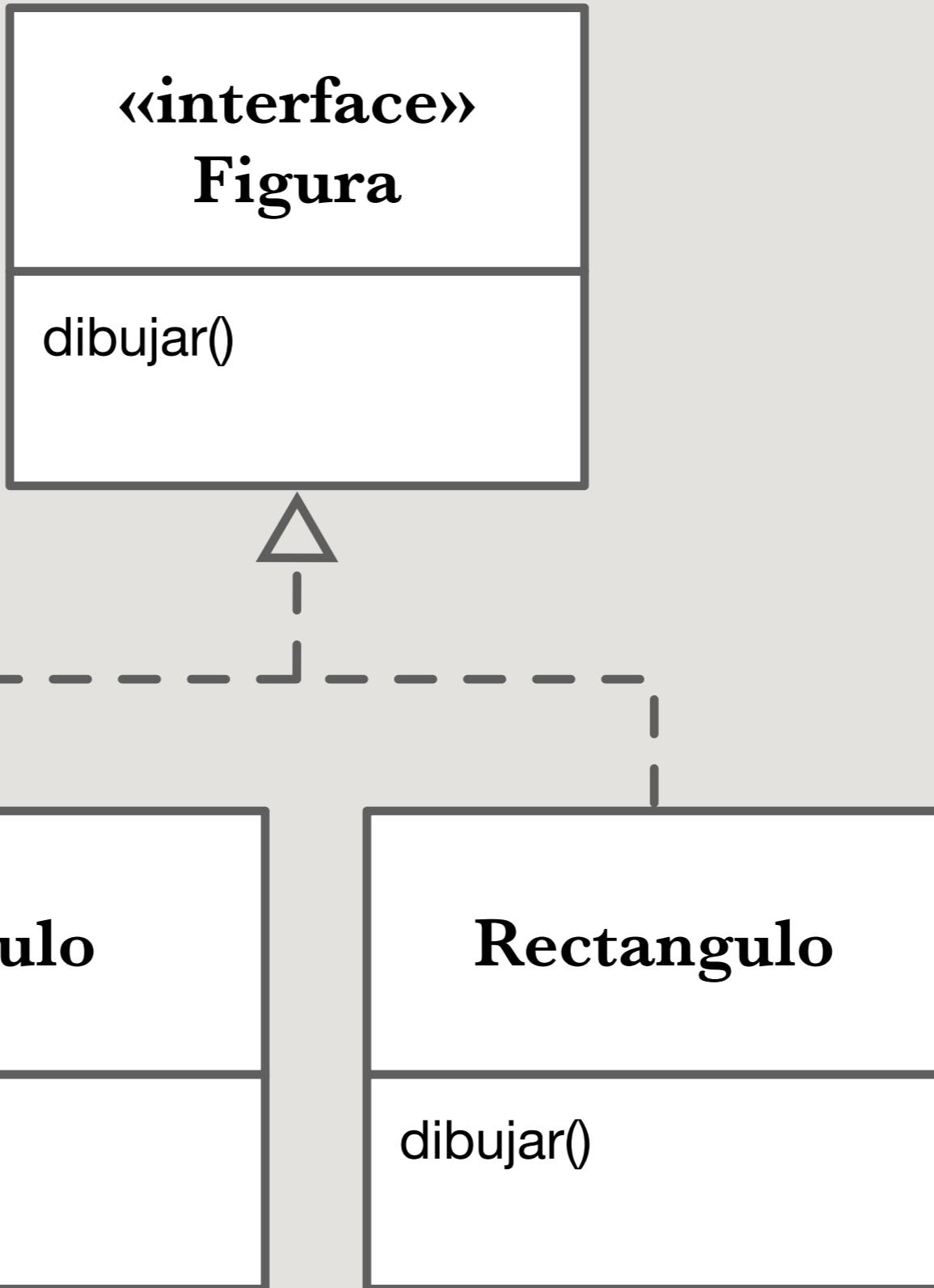
```
List<Figura> figuras = new ArrayList<>();
```

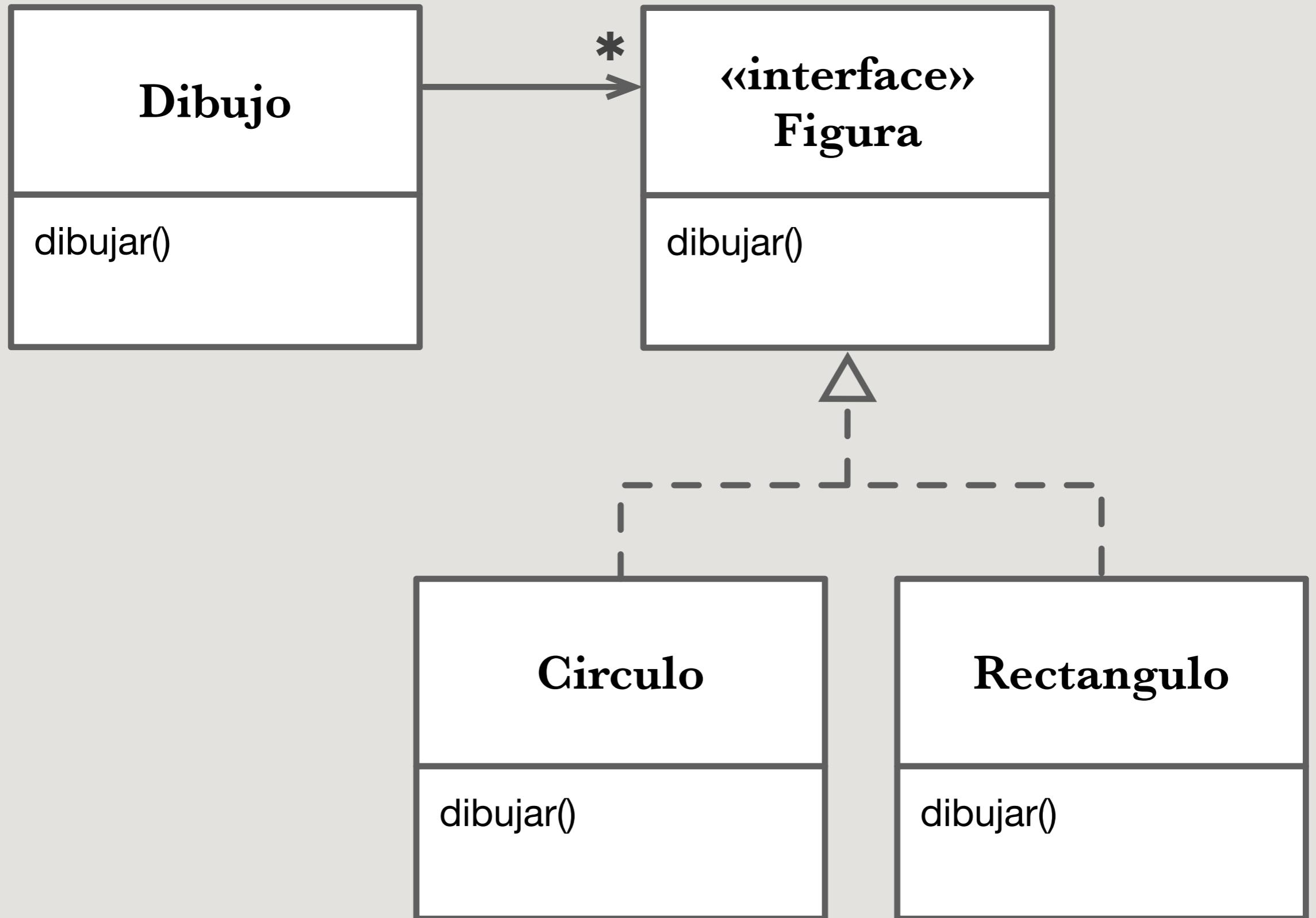




Dibujo.java

```
for (Figura figura : figuras) {  
    figura.dibujar();  
}
```





# Clases

O cómo especificar la interfaz  
de los objetos

# Paso 1

Elegir qué nombre se quiere dar a ese conjunto de operaciones

# Paso 2

Elección de responsabilidades  
y definición de métodos

# Paso 3

## Implementación

Nótese la distinción hecha entre los pasos dos y tres

Ejemplo



# Paso 1



A screenshot of a Java code editor window titled "Cronómetro.java". The window has three standard OS X-style buttons (red, yellow, green) at the top left. The code editor area contains the following Java code:

```
class Cronómetro {  
}
```

Le damos un nombre apropiado

# Paso 2

```
class Cronómetro {  
    public void ponerEnMarcha() {...}  
    public void parar() {...}  
    public long tiempoTranscurrido() {...}  
}
```

Elección de responsabilidades y  
definición de métodos

# Creación de objetos



Creación de objetos

Punto p;



Creación de objetos

Punto p; // ¿Qué es p?



## Creación de objetos

Punto p; // ¿Qué es p?

// ...

p = new Punto(30, 46);

# Referencias

Una referencia siempre apunta a un objeto en memoria (o a «null»).

# Referencias

El tipo de la referencia  
determina qué mensajes  
puede recibir el objeto.

# Atributos

La memoria del objeto  
(que no su cerebro)

Terminad de implementar  
nuestro cronómetro



CronómetroTest.java

```
class CronómetroTest {  
  
    public static void main(String[] args) {  
  
        Cronómetro cronómetro = new Cronometro();  
        cronómetro.ponerEnMarcha();  
  
        // operación a cronometrar  
        // ...  
  
        cronómetro.parar();  
  
        // ¿tiempo transcurrido?  
        System.out.println(cronómetro.tiempoTranscurrido());  
    }  
}
```



Cronómetro.java

```
class Cronómetro {  
  
    public void ponerEnMarcha() {  
        // Recordar a qué hora se recibió este mensaje  
    }  
  
    public void parar() {  
        // Calcular (y guardar) el tiempo transcurrido  
    }  
  
    public long tiempoTranscurrido() {  
        // ...  
    }  
}
```



Cronómetro.java

```
class Cronómetro {  
  
    public void ponerEnMarcha() {  
        horaDeComienzo = System.currentTimeMillis();  
    }  
  
    public void parar() {  
        parado = true;  
        tiempo = System.currentTimeMillis() - horaDeComienzo;  
    }  
  
    public long tiempoTranscurrido() {  
        return tiempo;  
    }  
  
    // ...  
}
```



Cronómetro.java

```
class Cronómetro {  
  
    // ...  
  
    private boolean parado;  
    private long horaDeComienzo;  
    private long tiempo;  
}
```

*Y esa es la única función de los atributos*

Guardar el estado del  
objeto entre distintas  
peticiones

*O lo que es lo mismo...*

Dar soporte a la  
implementación

Si la clase se hubiera implementado de otra forma habrían surgido otros atributos.

Y cada vez que esta cambie  
los atributos probablemente  
también lo harán.

¿A los clientes de la clase  
les interesan estos detalles  
de implementación?

inno!

*Así que...*

Si empezamos a  
diseñar una clase por  
sus atributos...

WAI

# Criterio

¿Este dato afecta al comportamiento de otra operación como para que deba ser recordado?

*La siguiente definición aparece en un libro*

Un objeto es un conjunto de datos que opcionalmente puede incluir operaciones para manipular dichos datos.



# Métodos

Criterios de diseño

# **No hacer asunciones**

Un objeto no puede presuponer una secuencia de mensajes determinada.

# Cohesión

Un objeto debe estar centrado en una única funcionalidad (lo veremos).

# Parámetros

Cómo diseñar la signatura de  
los métodos

# Parámetros

Además del estado interno del propio objeto, algunas operaciones pueden necesitar información adicional.

# Parámetros

Los parámetros son la forma de proporcionársela al enviar el mensaje al objeto.



Calculadora.java

```
class Calculadora {  
  
    public int sumar(int a, int b) {  
        return a + b;  
    }  
}
```

# Ejemplo



Printer.java

```
class Printer {  
  
    public void print(String document,  
                      PaperSize PaperSize,  
                      boolean color,  
                      Orientation orientation) {  
        // ...  
    }  
}
```

*una pista...*

# El cambio

¿Qué ocurre si se añade un nuevo modo de ahorro de tinta?



Printer.java

```
class Printer {  
  
    public void print(String document,  
                      PaperSize PaperSize,  
                      boolean color,  
                      Orientation orientation) {  
        // ...  
    }  
}
```



Printer.java

```
class Printer {  
  
    public void print(String document,  
                      PaperSize PaperSize,  
                      boolean color,  
                      Orientation orientation,  
                      boolean ecoMode) {  
  
        // ...  
    }  
}
```

Nos acabamos de  
cargar a todos  
los **clientes** de la  
impresora.

# Operandos y opciones

# Operandos

La información necesaria para que la operación pueda ser ejecutada.

# Operandos

Tienden a permanecer estables.

# Opciones

Todo lo que no es imprescindible.

# Opciones

Suelen indicar distintas formas de llevar a cabo la operación.

*una pista*

¿Es posible encontrar  
un valor predeterminado  
para ese parámetro?



Printer.java

```
class Printer {  
  
    public void print(String document,  
                      PaperSize PaperSize,  
                      boolean color,  
                      Orientation orientation) {  
        // ...  
    }  
}
```



Printer.java

```
class Printer {  
  
    public void print(String document) {  
        // Imprime en función de cómo esté configurada  
        // la impresora  
    }  
  
    public void setPaperSize(PaperSize paperSize) { ... }  
    public void setColor(boolean color) { ... }  
    public void setOrientation(...) { ... }  
}
```



PrinterTest.java

```
class PrinterTest {  
  
    public static void main(String[] args) {  
        Printer printer = new Printer();  
        printer.setColor(true);  
        printer.print("myDocument.doc");  
    }  
}
```

¿Qué ocurre si se introduce  
ahora una nueva opción?

**Los clientes no se  
ven afectados**

**¡Incluso aunque la utilicen!  
(sin ser conscientes de ello).**



PrinterTest.java

```
class PrinterTest {  
  
    public static void main(String[] args) {  
        Empleado pepe = new Empleado();  
        Printer impresora = new Printer();  
  
        printer.setEcoMode(true);  
        pepe.trabaja(impresora)  
    }  
}
```

# **Mezclar operadores y opciones es otro síntoma de programación estructurada encubierta**

Significa no entender todavía la  
noción de estado de un objeto.

# Estado de un objeto

Algo más que el valor de sus  
atributos

*una definición típica*

*Es la combinación de los  
valores de los atributos de  
un objeto.*

*Pero...*

¿Tiene sentido definir algo tan importante (el estado del objeto) en función de algo tan irrelevante como hemos dicho que son sus atributos?

# **Estado de un objeto**

Determina la respuesta para cada uno de sus mensajes.

Información que es necesario guardar para que, al recuperarla, se obtengan exactamente las mismas respuestas ante la misma secuencia infinita de mensajes.

Normalmente coincidirá con el valor de sus atributos (de ahí la definición anterior), pero no siempre.

Estado abstracto  
frente a  
estado concreto

Si en una clase Temperatura al pedirle los datos en Celsius están en Fahrenheit... ¿se cambia el estado del objeto?

No, porque no cambian  
sus **respuestas**.

Es decir, no cambia su  
comportamiento.

Incluso aunque, dependiendo de cómo  
esté implementada, pudieran variar los  
valores de sus atributos.

# **Estado concreto**

Se define en función de los atributos.

# Estado concreto

Este, dependiendo de la implementación de la clase, sí podría variar.

# **Estado abstracto**

Se define en función de las respuestas a los mensajes.

# **Estado abstracto**

Es el que nos interesa y al que  
nos referiremos siempre que  
hablemos de estado.

*En definitiva...*

*Decimos que dos objetos tienen el mismo estado si para la misma secuencia infinita de mensajes devuelven las mismas respuestas.*

*O, lo que es lo mismo,*

*Decimos que dos objetos tienen  
el mismo estado si se comportan  
igual.*



# Métodos de acceso

La verdadera razón de su uso



Persona.java

```
class Persona {  
  
    public int edad;  
  
    // ...  
}
```



Persona.java

```
class Persona {  
  
    private int edad;  
  
    public int getEdad() {  
        return edad;  
    }  
  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
  
    // ...  
}
```

¡¿No estamos en las mismas  
solo que con más código?!

# Controlar la modificación

Los errores ya no se propagan sin control: se sabe quién está corrompiendo los datos.



Persona.java

```
class Persona {  
  
    private int edad;  
  
    public void setEdad(int edad) {  
        if (edad < 0 || edad > 120)  
            throw new IllegalArgumentException("Edad inválida");  
        this.edad = edad;  
    }  
  
    // ...  
}
```

Pero esa no es su  
mayor ventaja

Ni el motivo por el que debemos  
hacer los atributos privados.

# Ocultamiento de la información

Cuanto menos sepa una clase de otra,  
mejor (criterio de independencia).

# Criterio de independencia

Una clase no debe saber de otra ni su implementación, ni sus atributos, ni sus modos de operación.

# Criterio de independencia

Únicamente los mensajes que puede  
recibir.



Persona.java

```
class Persona {  
    private int edad;  
  
    public int getEdad() {  
        return edad;  
    }  
  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
  
    // ...  
}
```

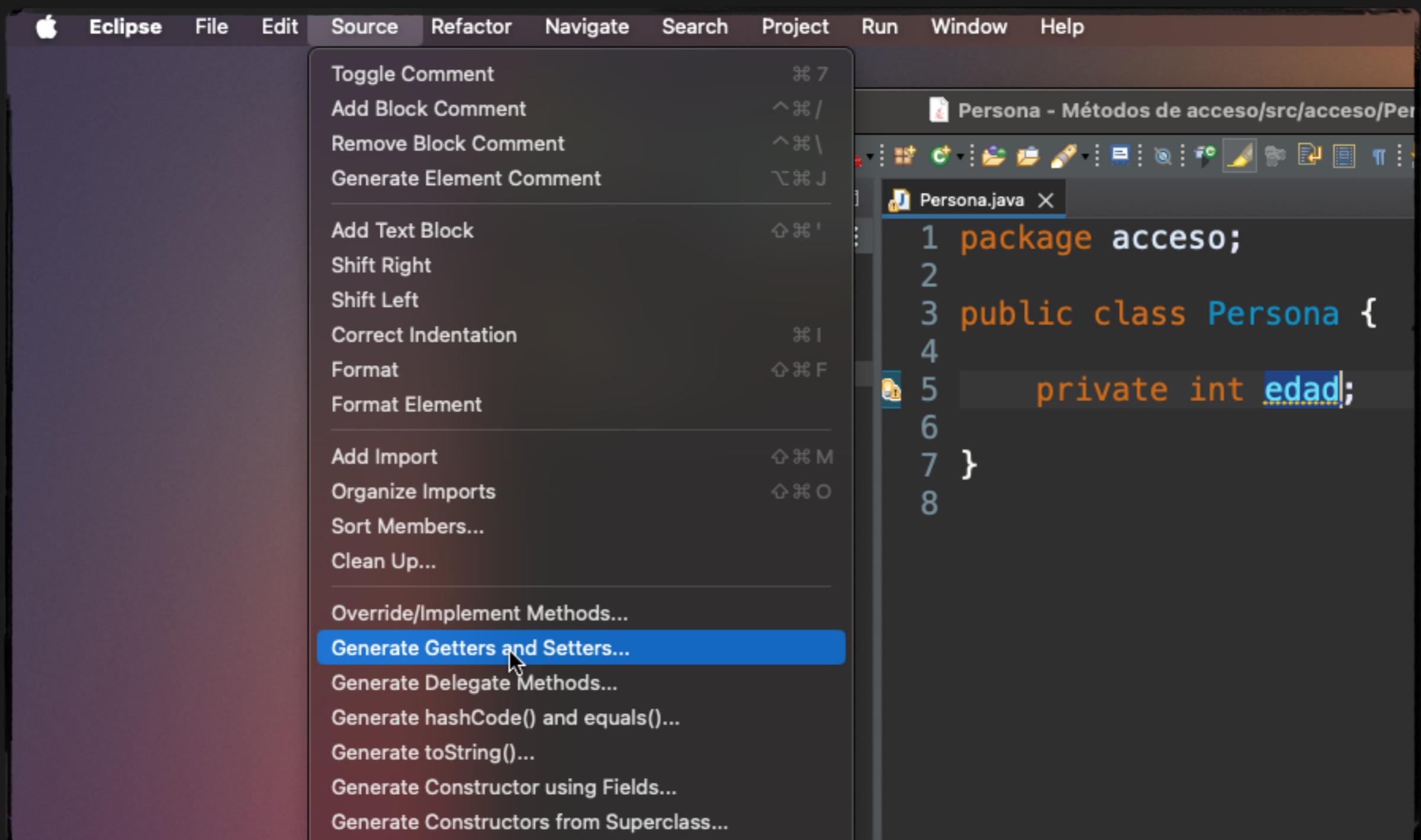


Persona.java

```
class Persona {  
    private Date fechaDeNacimiento;  
  
    public void getEdad() {  
        // Cambia la implementación, no el método  
        // (que forma parte del contrato de la  
        // clase) => los clientes no se enteran  
    }  
  
    // ...  
}
```

*según lo anterior...*

¿Qué os parece esto?





Andry Strong

# Diseño orientado a objetos

Pequeña guía de diseño  
basado en responsabilidades

**Lo difícil es extraer  
responsabilidades y saber  
a quién asignárselas**

Los métodos tradicionales no sirven.

**¡Cómo vamos a extraer  
clases sin saber primero  
qué tienen que hacer!**

Hay que centrarse en el qué.

*Pero...*

**¡¿No era eso en lo  
que consistía la  
programación  
estructurada?!**



**Falta un matiz**

No separarlo jamás del quién.

# Paso 1

# Elegir una tarea a realizar (el qué)

Es decir, extraer las responsabilidades.

# Paso 2

# Decidir quién debe hacerla

¿Alguna de las clases o interfaces que ya tenemos?

Si es así, asignársela.

# Decidir quién debe hacerla

En caso contrario ya tenemos una nueva clase o interfaz.

¡Y con una responsabilidad concreta!

# Paso 3

# Decidir las operaciones

Atómicas, para que puedan combinarse.

# **Decidir las operaciones**

Parámetros con la información  
imprescindible.

# **Decidir las operaciones**

Sacar partido de objetos que recuerdan su estado.

# Paso 4

# Implementar las operaciones

Cualquiera sin errores.

# Sin errores

Constructores para iniciar  
siempre el objeto en un estado  
válido.

# Sin errores

Dejar el objeto siempre en un estado válido al salir de un método.

# Sin errores

Comprobar los parámetros.

# Implementar las operaciones

No preocuparse por la eficiencia hasta el final.

**No preocuparse por  
la eficiencia**

Generalmente es  
suficientemente rápida.

# No preocuparse por la eficiencia

No tiene sentido optimizar  
(en estas fases tempranas)  
algo que puede desaparecer  
al factorizar.

Los patrones de diseño nos ayudarán en los pasos dos y tres para algunos problemas de diseño habituales.

# Fichas CRC

Clase-Responsabilidades-  
Colaboraciones

View

*Render the Model.*

*Transform coordinates.*

*Controller*

*Model*

Controller

*Interpret user input.*

*Distribute control.*

*View*

*Model*

Model

*Maintain problem related info.*

*Broadcast change notification.*



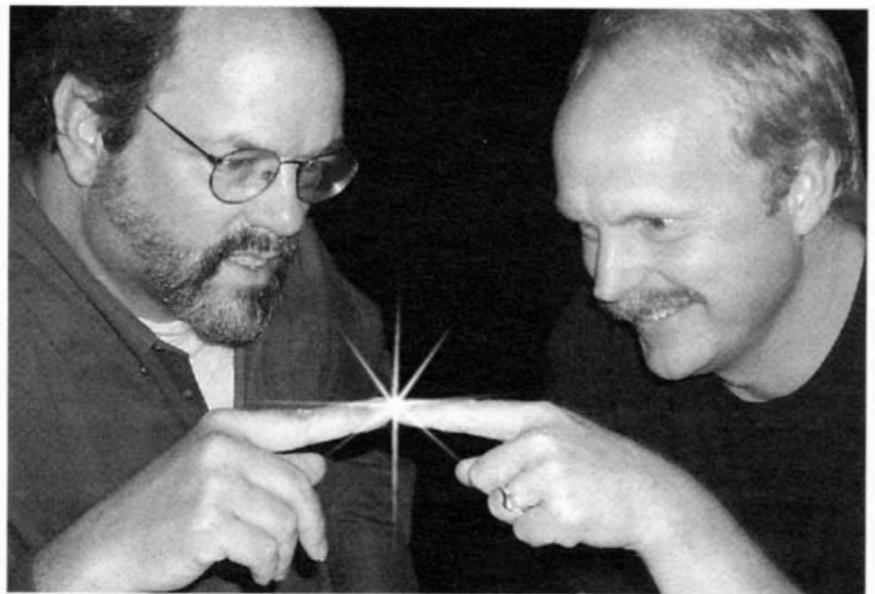
## Ward And Kent

[WardCunningham](#) and [KentBeck](#) wrote a lot of Smalltalk code ([HotDraw](#), [DiagrammingDebugger](#), [LiterateProgramBrowser](#), etc.) and a fair amount of tech reports together in [TekLabs](#). They wrote the original [CrcCard](#) paper from Ward's idea (<http://c2.com/doc/oopsla89/paper.html>).

[WardAndKent](#) remind me sometimes of Fred Astaire and Gene Kelly. They both dance superbly, but not at all the same. Ward has anti-gravity, Kent has atomic energy. -- [RonJeffries](#)

[InfoWorld](#) published this picture of the extreme programmer's secret handshake in their October 9, 2000 issue. The photograph is by [LowellLindstrom](#) who caught [WardAndKent](#) "pairing" at [XpImmersionFour](#).

So which one is ET?



**Kent Beck and Ward Cunningham**

**Company** Consultants, FirstClass Software and Cunningham & Cunningham

**Claim to fame** Extreme programming

These two innovators bear watching as they promote a new methodology for software development called extreme programming, or XP. Based on four core values — communication, simplicity, feedback, and courage — this controversial approach to software development promises to better integrate the business and technical teams across your company, increase collaboration, and deliver results of a higher quality. But XP's location-centric approach may not work as well for distributed development projects, and coding in pairs as defined by XP may not work at some companies. Additional data is needed to determine success vs. failure ratios of XP vs. other approaches. And it remains to be seen how XP will mesh with the advent of the virtual organization.

— Maggie Biggs

I was surprised and pleased by the sparkle. Their art department said we had far and away the best picture. I agree. -- [KentBeck](#)

*Text contained in the above image, for those who can't see it:*



# A Laboratory For Teaching Object-Oriented Thinking

*Kent Beck, Apple Computer, Inc.*

*Ward Cunningham, Wyatt Software Services, Inc.*

*From the OOPSLA'89 Conference Proceedings*

*October 1-6, 1989, New Orleans, Louisiana*

*And the special issue of SIGPLAN Notices*

*Volume 24, Number 10, October 1989*

## Contents

1. [Problem](#)
2. [Perspective](#)
3. [CRC Cards](#)
4. [Experience](#)
5. [Conclusion](#)
6. [References](#)
7. [Appendix](#)

## Introduction

It is difficult to introduce both novice and experienced procedural programmers to the anthropomorphic perspective necessary for object-oriented design. We introduce CRC cards, which characterize objects by class name, responsibilities, and collaborators, as a way of giving learners a direct experience of objects. We have found this approach successful in teaching novice programmers the concepts of objects, and in introducing experienced programmers to complicated existing designs.

### 1. Problem

The most difficult problem in teaching object- oriented programming is getting the learner to give up the global knowledge of control that is possible with procedural programs, and rely on the local knowledge of objects to accomplish their tasks. Novice designs are littered with regressions to global thinking: gratuitous global variables, unnecessary pointers, and inappropriate reliance on the implementation of other objects.

Because learning about objects requires such a shift in overall approach, teaching objects reduces to teaching the design of objects. We focus on design whether we are teaching basic concepts to novices or the subtleties of a complicated design to experienced object programmers.

Rather than try to make object design as much like procedural design as possible, we have found that the most effective way of teaching the idiomatic way of thinking with objects is to immerse the learner in the "object-ness" of the material. To do this we must remove as much familiar material as possible, expecting that



SPECIAL OFFERS Keep up with new releases and promotions. [Sign up to hear from us.](#)



[Join](#) | [Sign In](#) [View Your Cart](#)

Search



Store ▾ Formats ▾ Deals & Promotions Video Training Imprints ▾ Explore ▾ Community ▾

[Home](#) > [Articles](#) > [Software Development & Management](#) > [Agile](#)

## CRC Cards: An Agile Thinking Tool

Aug 25, 2009

[Contents](#) [Print](#)

Page 1 of 6 [Next >](#)

Rebecca Wirfs-Brock shows you how to use CRC cards as an Agile thinking tool.

### Like this article? We recommend



[Object Design: Roles, Responsibilities, and Collaborations](#)

[Learn More](#)

[Buy](#)

### InformIT Promotional Mailings & Special Offers

I would like to receive exclusive offers and hear about products from InformIT and its family of brands. I can unsubscribe at any time.

[Privacy Notice](#)

Email Address

[Submit](#)

This year marks the 20th anniversary of the World Wide Web and the CRC card. I know you know about the web. But you should also know about CRC cards. Kent Beck and Ward Cunningham introduced CRC cards— Class–Responsibility–Collaborator cards—in 1989 as a tool for teaching object-oriented thinking. Since then, CRC cards have moved far beyond the classroom. They can be useful in short design sessions or for interactively telling design stories. Written into a design document or summarized on a wiki, they can also present a high-level, conceptual design overview.

I prefer using 4 by 6 inch lined index cards. They give me enough room to write without feeling cramped. By convention, the name of the class or design element is written at the top, responsibilities on the left 2/3 of the card, and collaborators listed the right (a collaborator is another name for a helper). But don't be misled by the name—while the first C stands for class, but you can also use CRC cards to describe responsibilities of components, subsystems, services, or interfaces.

It's interesting to compare the original CRC cards written by Kent Beck and Ward Cunningham that describe the Model–View–Controller pattern with the CRC cards in *Patterns of Software Architecture* (Buschmann et al, 1996):

[Beck and Cunningham MVC Cards:](#)

[POSA MVC Cards:](#)



# Rebecca Wirfs-Brock

Pionera del diseño orientado a objetos

Rebecca Wirfs-Brock es una pionera del diseño de software basado en el paradigma de objetos y es la inventora de Responsibility Driven Design o RDD, según sus siglas en inglés, una técnica que ha influido en otras técnicas de diseño más recientes, que quizás conozcas mejor, como TDD (Test Driven Development), BDD (Behaviour Driven Development) o DDD (Domain Driven Development).

Ilustraciones por [Emilio Rodríguez Vila](#).

Textos por [Rafael Luque](#).

Colabora

