

Polimorfismo I

herencia

UNRN

Universidad Nacional
de Río Negro



Identidad

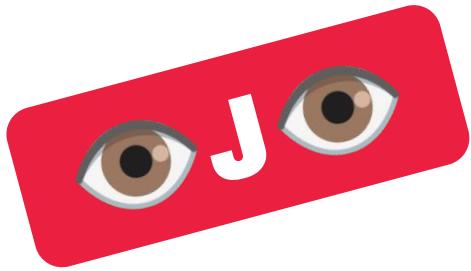


equals
es la igualdad
instancias de una
clase

*¿apuntan a
la misma
cosa?*

equals
es la igualdad de
dos objetos

*¿apuntan a
la misma
cosa?*



equals no es lo
mismo que ==

*¿apuntan a
la misma
instancia?*

==

**es la igualdad de
referencias**

Atributos

El contrato de equals

La relación de equivalencia

Reflexivo

Un objeto debe ser igual a sí mismo

Simétrico

`a.equals(b)` tiene que ser igual que `b.equals(a)`

Transitivo

si $a.equals(b)$ y $b.equals(c)$, entonces $a.equals(c)$

Consistente

El valor de `equals` solo cambia con los atributos,
no se admite aleatoriedad.

Las comparaciones base en equals

this == otro

otro == null

getClass() == otro.getClass()

¿es si mismo?

¿es nulo?

¿Mismo tipo?

La igualdad de dos usuarios

```
@Override  
public boolean equals(Object objeto) {  
    if (this == objeto){  
        return true;  
    }  
    if (objeto == null){  
        return false;  
    }  
    if (getClass() != objeto.getClass()){  
        return false;  
    }  
    Usuario user = (Usuario) objeto;  
  
    if (!fechaNacimiento.equals(user.fechaNacimiento)) return false;  
    if (!nombre.equals(user.nombre)) return false;  
    return apellido.equals(user.apellido);  
}
```



“pattern matching”



En lugar de hacer lo que vimos la clase pasada

```
if (getClass() == objeto.getClass()){
    Usuario user = (Usuario) objeto;
    // resto de la comparación.
}
```

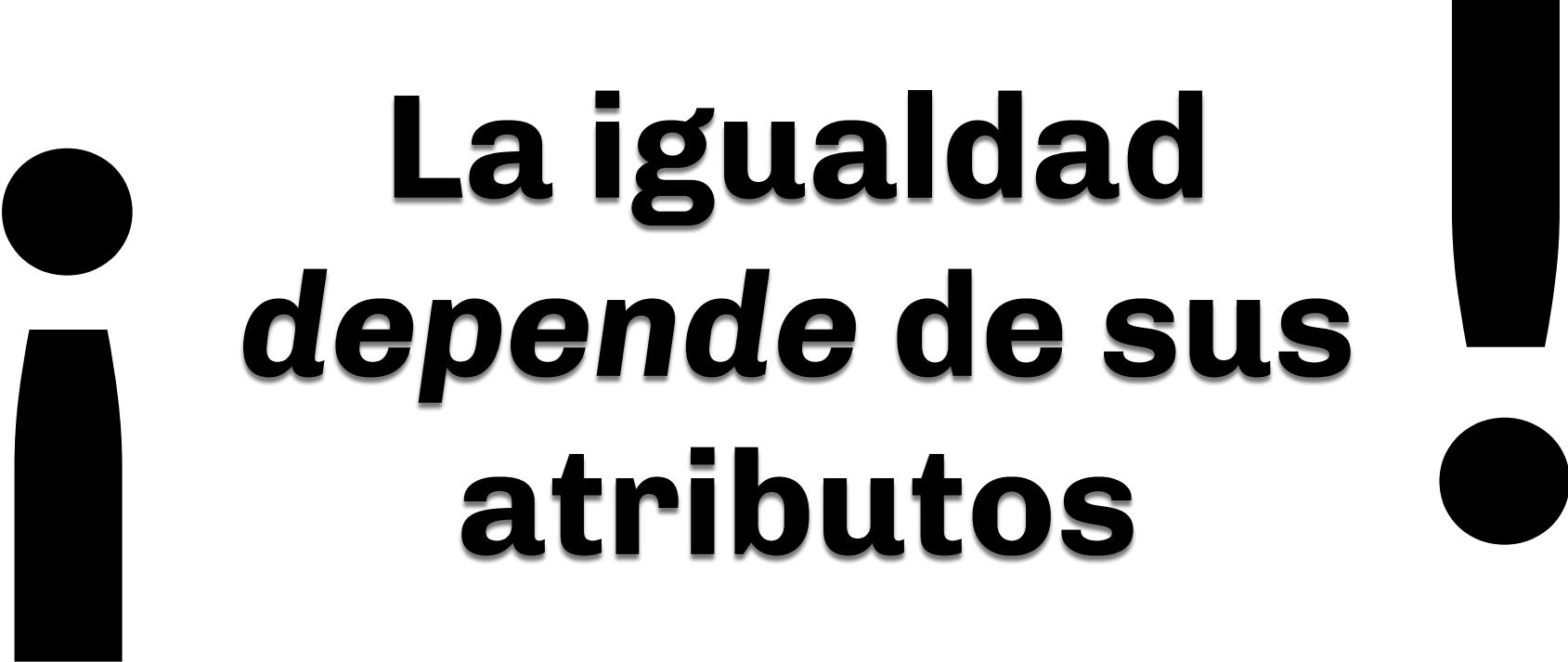
instanceof pattern matching

```
@Override  
public boolean equals(Object objeto) {  
    ...resto de equals  
    if (objeto instanceof Usuario user) {  
        boolean igualdad = this.nombre.equals(user.nombre);  
        igualdad = igualdad && this.apellido.equals(user.apellido);  
        return igualdad;  
        // Comparación de atributos  
    }  
}
```



¿Preguntas?





**La igualdad
depende de sus
atributos**

Pero hay algo para
respetar

El protocolo de equals y hashCode

¿hashcode?

Es su identificación

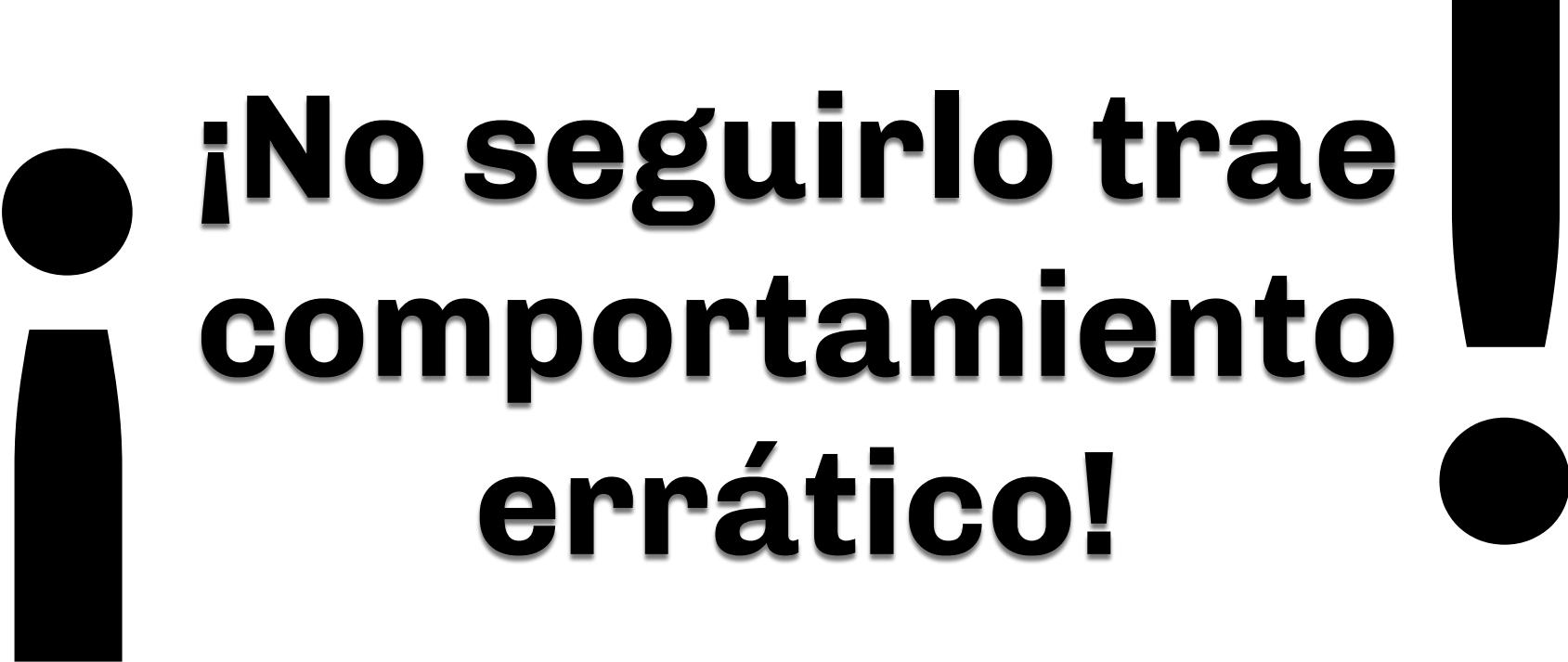
¡Y fuertemente usada para conjuntos y estructuras!

*Relacionado con
su posición en
memoria*

**Si dos objetos son
'equals'**

**Su hashCode
es igual**

```
@Override  
public int hashCode() {  
// Objects.hashCode(objeto);  
    return Objects.hash(todos, los, atributos, juntos);  
    return result;  
}
```



**¡No seguirlo trae
comportamiento
errático!**

**Por suerte es
fácilmente
testable**

*assertEquals de dos objetos
construidos con los mismos valores

iCollections!

Se usa con
Set - Conjuntos
Map - Diccionarios

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



herencia



1

Herencia

Es la relación entre clases de un mismo tipo.

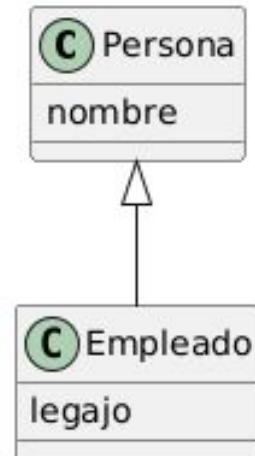
- La clase que hereda especializa a la madre.
- La clase madre es la generalización de la hija.

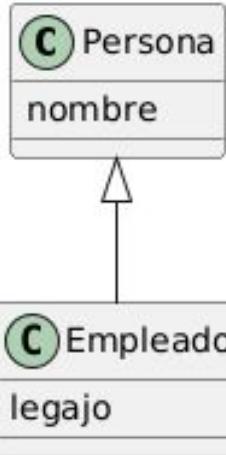
0

Sintaxis en Java

¡Solo uno!

```
class Persona {      class Empleado extends Persona {  
    String nombre;          String legajo;  
}  
}
```





Un **Empleado** es un **Persona**

El **Empleado** Pedro es una **Persona**
Pero la **Persona** José, quizás no sea un **Empleado**.

Acceso a atributos y métodos

La clase tiene acceso a todo lo publico y protegido de la clase extendida (padre)

Esto significa que...

```
class Animal {  
    String nombre;  
    public void comer() {  
        SOUT("Está comiendo.");  
    }  
}  
  
class Perro extends Animal {  
    public void ladRAR() {  
        SOUT("¡Guau!");  
    }  
}
```

```
void main(String[] args) {  
    Perro miPerro = new Perro();  
    miPerro.nombre = "Buddy";  
    miPerro.comer();  
    miPerro.ladRAR();  
}
```

Definición en partes de las clases

**A continuación
vamos a ver que
se puede hacer
con esto**

@override



**Abran
hilo**

4 Pillars of OOP

Encapsulation



Polymorphism



Inheritance



Abstraction



implementemos
una

calculadora

gradualmente más complicada

Operaciones matemáticas como clases

La Suma como clase

```
public class Suma{  
    private int izquierdo;  
    private int derecho;  
  
    public Suma(int izq, int dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
    public int calcular(){  
        return izquierdo + derecho;  
    }  
}
```

Ejemplo de uso

```
Suma primero = new Suma(10,20);
System.out.println(primeros.calcular());
```

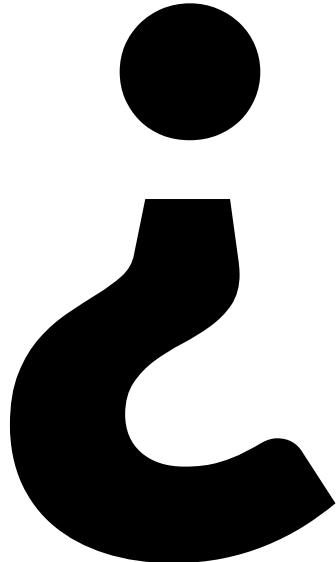
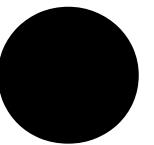
{ ¿Y si queremos
expresiones
más complejas? }

Suma

```
public class Suma{  
    private int izquierdo;  
    private int derecho;  
  
    public Suma(int izq, int dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
    public Suma(Suma izq, int dch){  
        izquierdo = izq.calcular();  
        derecho = dch;  
    }  
    public int calcular(){  
        return izquierdo + derecho;  
    }  
}
```

Ejemplo de uso

```
Suma primero = new Suma(10, 20);
Suma segundo = new Suma(primer, 40);
System.out.println(segundo.calcular());
```



**¿Y si queremos
cualquier
combinación de
sumas?**

Suma

```
public class Suma{  
    private Suma izquierdo;  
    private Suma derecho;  
  
    public Suma(Suma izq, Suma dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
  
    public int calcular(){  
        return izquierdo.calcular() + derecho.calcular();  
    }  
}
```

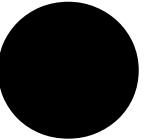


Is that legal

Suma

```
public class Suma{  
    private Suma izquierdo;  
    private Suma derecho;  
  
    public Suma(int izq, int dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
  
    public int calcular(){  
        return izquierdo.calcular() + derecho.calcular();  
    }  
}
```

Pero ahora
tenemos un
problema



**¿quién
guarda los
números?**

¡Puedes ser un número!

```
public class Numero{  
    private int valor;  
  
    public Numero(int valor){  
        this.valor = valor;  
    }  
  
    public int calcular(){  
        return valor;  
    }  
}
```

Que podemos expresar como algo parecido

Podemos expresar Suma como...

```
public class Suma{  
    private Suma izquierdo;  ¿Suma o Número?  
    private Suma derecho;    ¿Suma o Número?  
    private Número valorIzquierdo;  
    private Número valorDerecho;  
  
    public Suma(Número izq, Número dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
    public Suma(Suma izq, Suma dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
    public int calcular(){  
        return ... ; //cuál de los dos tenga valor y etc;  
    }  
}
```

Pero si los dos
tienen un método
calcular...



JAKE-CLARK.TUMBLR



Why don't we have both?



@Petirep

← JAKE-CLARK.TUMBLR



Pero es necesario conectarlos

```
public abstract class Operacion{  
    public abstract int calcular();  
}
```

Pero es necesario conectarlos

```
public abstract class Operacion{  
    public abstract int calcular();  
}
```

¿Sabemos qué operación es?

abstract

No es posible instanciar una clase abstracta
Le falta 'algo', en este caso, como calcular

Número como ‘Operación’

```
public class Numero extends Operacion{  
    private int valor;  
    public Numero(int valor){  
        this.valor = valor;  
    }  
    public int calcular(){  
        return valor;  
    }  
}
```

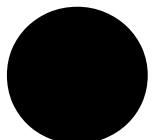
Suma como ‘operación’

```
public class Suma extends Operacion{
    private Operacion izquierdo;
    private Operacion derecho;

    public Suma(Operacion izq, Operacion dch){
        izquierdo = izq;
        derecho = dch;
    }
    public int calcular(){
        return izquierdo.calcular() + derecho.calcular();
    }
}
```

Y ahora para usarlo

```
Numero op1 = new Numero(10);
Numero op2 = new Numero(20);
Suma cuarto = new Suma(op1, op2); //10 + 20
System.out.println(cuarto.calcular());
```



**¿Por qué
Operacion y no
Número o Suma?**

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



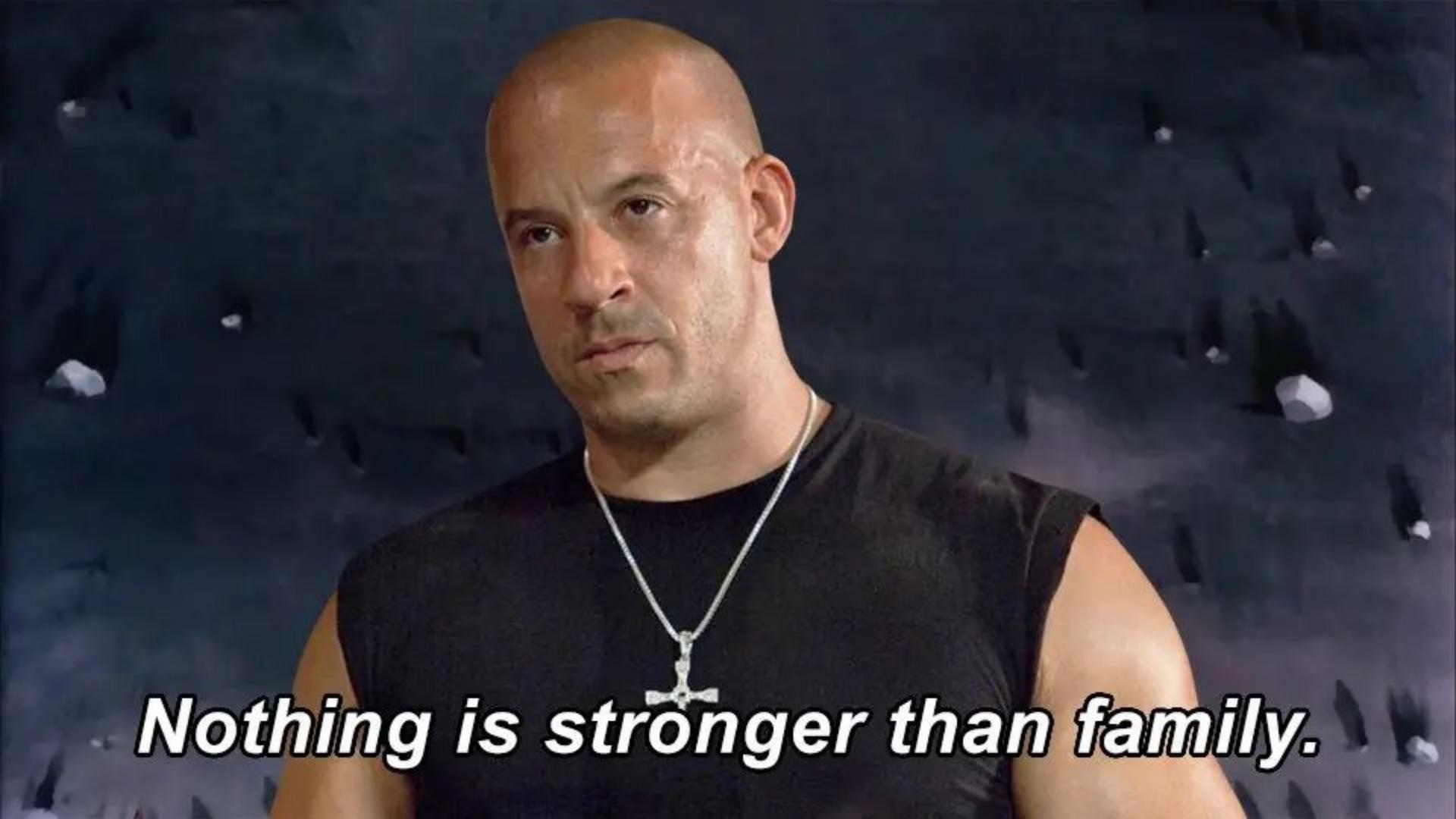
**Esto es posible
por...**

herencia

Es la especialización de una clase más general a algo más concreto

El “sub-tipado” es un tipo de polimorfismo

Todos los objetos tienen un solo Supertipo

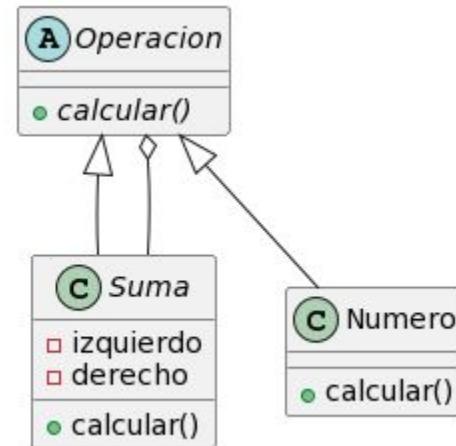
A portrait of Vin Diesel as Dominic Toretto from the Fast and Furious franchise. He is bald, wearing a black t-shirt and a silver chain necklace with a small cross pendant. He is set against a dark, star-filled background that looks like a meteor shower or a space scene.

Nothing is stronger than family.

La familia (hasta ahora)

Padre

Hijos



**Porque podemos
decir que
Suma es un
Operacion**

**Y que
Número es un
Operación**

polimorfismo

Una interfaz común entre clases de diferente tipo
Donde se espera al padre, puede ir cualquiera de sus hijos



Tratandolos de la misma manera

A través de lo que su padre/supertipo

```
public abstract Operacion{  
    public abstract int calcular();  
}
```

Esto es lo que “sabe hacer” (y nada más)

¿Cómo funcionan las substituciones?

Si se espera una Suma, solo sus ‘hijos’ pueden substituirlo.

Lo mismo con Operación, solo sus ‘hijos’ pueden substituirlo

Donde se espera al padre, puede ir cualquiera de sus hijos

**Establece un
denominador común
de comportamiento**

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



¿Y si agregamos más operaciones?

Resta como ‘operación’

```
public class Resta{  
    private Operacion izquierdo;  
    private Operacion derecho;  
  
    public Resta(Operacion izq, Operacion dch){  
        izquierdo = izq;  
        derecho = dch;  
    }  
    public int calcular(){  
        return izquierdo.calcular() - derecho.calcular();  
    }  
}
```

**Mucho
duplicado...
(¿no les parece?)**

OperacionBinaria

atributos

Operando izquierdo

Operando derecho

métodos

int calcular()

¿Pero para qué?

```
public abstract class OperacionBinaria extends Operacion{  
    protected Operacion izquierda;  
    protected Operacion derecha;  
  
    public OperacionBinaria(Operacion izq, Operacion dch){  
        this.izquierdo = izq;  
        this.derecho = dch;  
    }  
    public abstract int calcular();  
}
```

La 'Suma' versión 2

```
public class Suma extends OperacionBinaria{  
    public int calcular(){  
        return izquierdo.calcular() + derecho.calcular();  
    }  
}
```

La 'Resta' versión 2

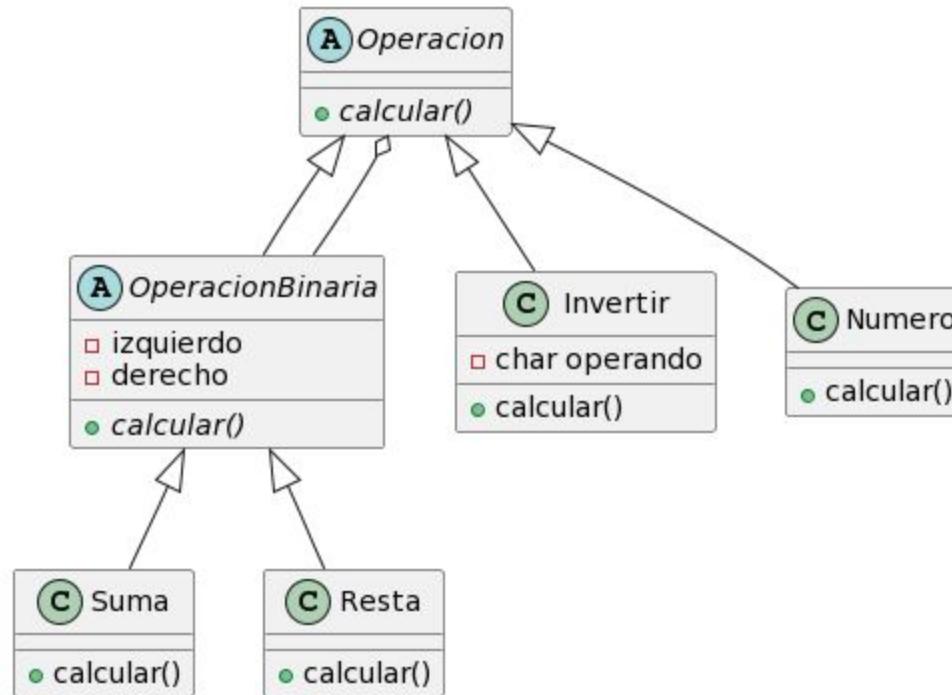
```
public class Resta extends OperacionBinaria{  
    public int calcular(){  
        return izquierdo.calcular() - derecho.calcular();  
    }  
}
```

Un ejemplo completo

```
Numero n = new Numero(10);
Numero m = new Numero(20);
Suma primero = new Suma(n, m);
Resta segundo = new Resta(primer, m);

System.out.print(segundo.toString() + "=");
System.out.println(segundo.calcular());
```

El diagrama de clases



Hay un detalle
que ***quizás*** pasó
desapercibido

¿Lo qué?

```
public abstract class OperacionBinaria{  
    protected Operacion izquierda;  
    protected Operacion derecha;  
}
```

...

Repasando los calificadores de acceso...

Tenemos:
público
privado

Calificadores de acceso

privado

No lo accede nadie
¡ni los hijos!

**publico
privado
protegido**

Calificadores de acceso

protegido

Es accesible por los hijos

Como privado no
podríamos
acceder a los
Operacion en
OperacionBinaria

**La idea es
compartir
atributos y
comportamiento**

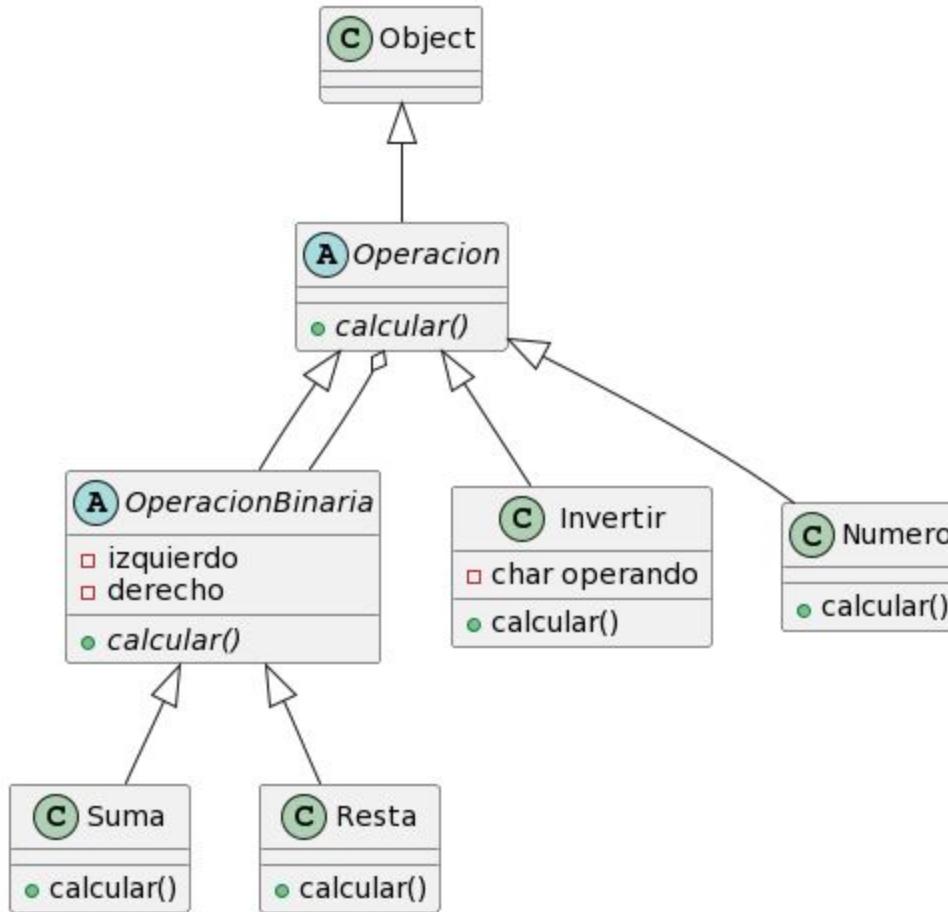
A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



Y recuerden que en Java, todo hereda de Object

De manera directa o indirecta



**Por lo que podemos
redefinir
equals / hashCode
toString**

A yellow cube with two large white question marks on its faces, resembling a power-up item from the Super Mario video game series.

¿Preguntas?



algunas mejoras

Podemos ‘forzar’ la implementación de `toString`

```
public abstract Operacion{  
    public abstract int calcular();  
    public abstract String toString();  
}
```

Una pequeña mejora

```
public abstract class OperacionBinaria{  
    protected Char simbolo;  
    protected Operacion izquierda;  
    protected Operacion derecha;  
  
    public OperacionBinaria(Operacion izq, Operacion dch){  
        this.izquierdo = izq;  
        this.derecho = dch;  
    }  
    protected OperacionBinaria(char op, Operacion izq, Operacion dch){  
        this(izq, dch);  
        this.operando = op;  
    }  
    public abstract int calcular();  
  
    public String toString(){  
        return "(" + izquierdo.toString() + simbolo + derecho.toString() + ")";  
    }  
}
```

La 'Suma' versión 3

```
public class Suma extends OperadorBinario{  
  
    public Suma(Operacion izq, Operacion dch){  
        super("+", izq, dch);  
    }  
  
    public int calcular(){  
        return izquierdo.calcular() + derecho.calcular();  
    }  
}
```

Como Operacion ahora agrega otro método abstracto...

```
public class Numero extends Operacion{
    private int valor;
    public Numero(int valor){
        this.valor = valor;
    }
    public int calcular(){
        return valor;
    }
    public String toString(){
        return String(valor);
    }
    public void cambiarValor(int nuevoValor){
        this.valor = nuevoValor;
    }
}
```

Otras operaciones

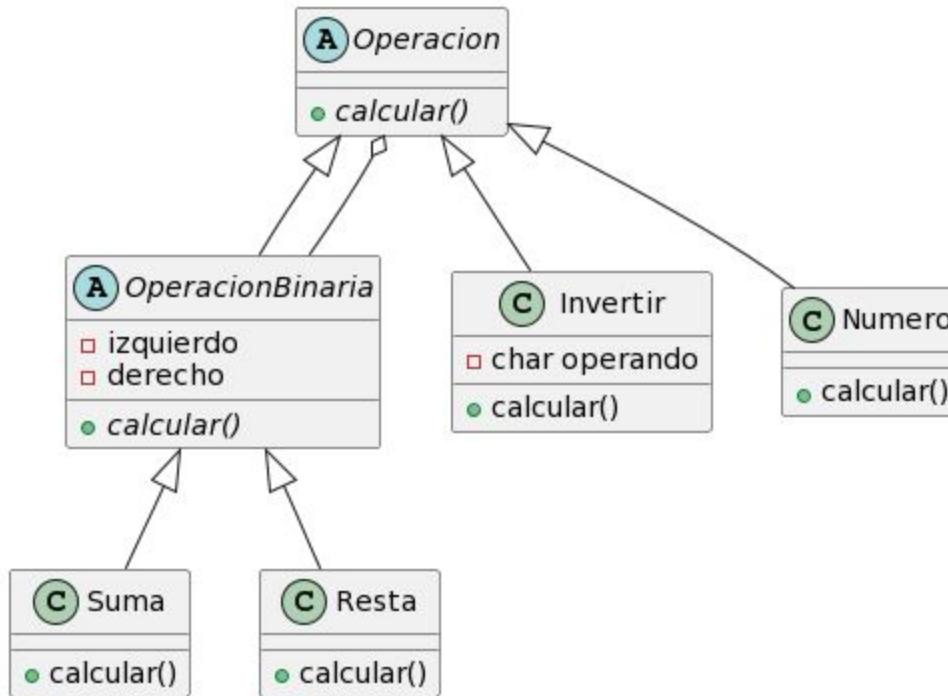
La 'Inversión'

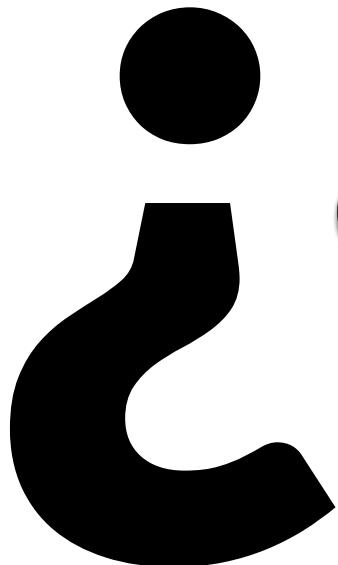
```
public class Invertir extends Operacion{  
    private Operacion Valor;  
  
    public Invertir(Operacion valor){  
        this.valor = valor;  
    }  
  
    public int calcular(){  
        return -valor.calcular();  
    }  
    public comoCadena(){  
        return "-" + valor.toString();  
    }  
}
```

práctica

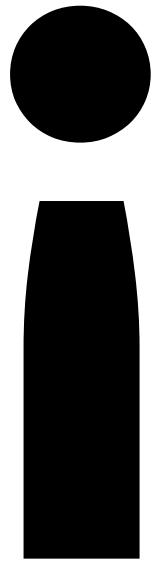
Completen la calculadora y agreguen un `main` que le dé uso.

El diagrama de clases (OperacionBinaria no agrega nada)

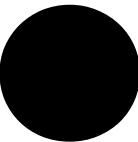
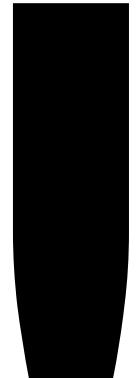




**¿Qué
operaciones se
les ocurren?**



**La clave está en
minimizar lo
duplicado**



**Si hay mucho
copypasteo lo
marcaré como**



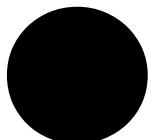
PLUS
ULTRA

Operaciones múltiples

```
public abstract class OperacionMultiple extends Operacion{  
    private List<Operacion> operandos;  
  
    public OperacionMultiple(){  
        operandos = new ArrayList<Operacion>();  
    }  
  
    public void agregar(Operacion op){  
        operandos.add(op);  
    }  
}
```

MultiSuma

```
public class MultiSuma extends OperacionMultiple{  
    public int calcular(){  
        int calculo;  
        for (op : operandos){  
            calculo = calculo + op.calcular();  
        }  
        return calculo;  
    }  
}
```



ArrayList<Operacion>



ArrayList<*Tipo*>

TP7

Calculadora

TP5

Devolución

Conclusiones y cuestiones interesantes

Si una clase no tiene comportamiento

¿Cuál es su propósito?

**Suena a pensar en
struct, con los
datos primero**

Los nombres compuestos

Quizas sean lo mismo, algo como
RequisitoAntiguedad
RequisitoEdad
En una licencia de conducir

El comportamiento tiene que referirse a sus atributos

cambiarFormacion en el entrenador sin una conexión al equipo.

¿Existe como tal?

Una “Herramienta de Taller”

¿Qué “tipos de” hay?

¿Es una sola cosa?

Una película con la información que la
describe

y

la película con acciones de reproducción

¿Es comportamiento o un tipo dé?

Comportamiento como:

- Pastoreo
- Cuida
- Caza
- Compañía

Que es lo que sería y cuál debiera de ser

¿Es un atributo?

La cantidad del contenido, como en un inventario de un personaje,
¿no sale del contenedor de lo aloja?

Esta información sale del conjunto y su respectiva

Al mencionar que tiene un

Círculo, un Perro, en lugar de describirlo específicamente, ‘deleguen’ esa responsabilidad a otra clase.

(¡cada una con su propio comportamiento y estado!)

Cuidado con redundar estado

Un triángulo puede definirse por sus Puntos, de los cuales salen los ángulos, segmentos, y ángulos.

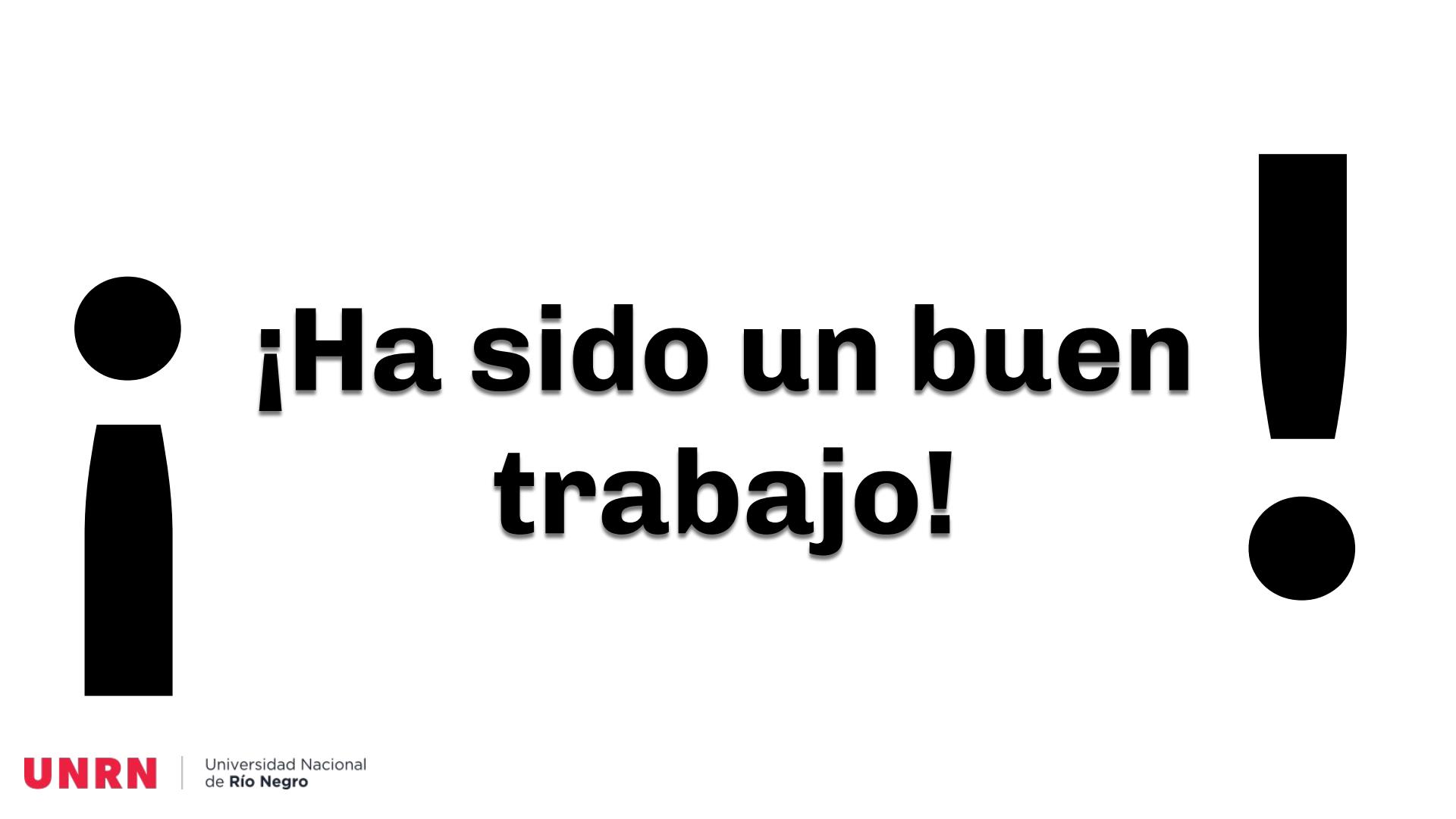
Es tentador guardar la información completa en lugar de calcularla.

Son más cuentas, pero es menos estado que mantener.

Si una clase tiene cosas ‘vacías’

Si todos los vagones tienen una “capacidad pasajeros” que no usamos según el uso del vagón, entonces hay que generalizarlo.

Entonces es potencialmente tres objetos.



**¡Ha sido un buen
trabajo!**



¿Preguntas?





Abran
hilo

unrn.edu.ar

UNRN

Universidad Nacional
de Río Negro



| unrnionegro