

Polimorfismo I

UNRN

Universidad Nacional
de Río Negro

XIII

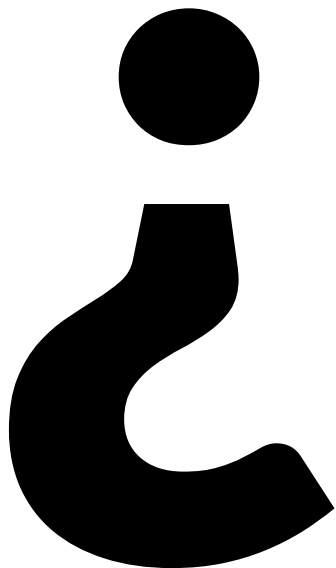
2024





Dudas del TP6





Dudas del TP7



**Entregar lo mismo 2
veces va a un ✖
(rehacer)**



¿Preguntas?

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(primerο.calcular());
```

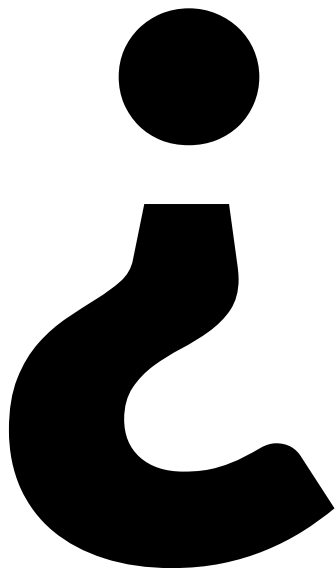
Suma

```
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(primerο, 40);  
System.out.println(segundo.calcular());
```



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



```
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();
    }
}
```

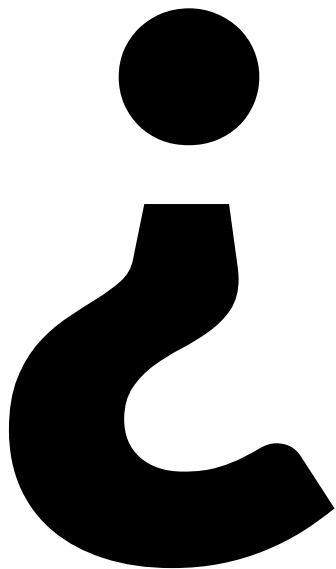
```
public class Suma{  
    private Suma izquierdo;  
    private Suma derecho;
```

```
    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?**



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

—

Pero si los dos tienen un método calcular...









Pero es necesario conectarlos

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


Pero es necesario conectarlos

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



abstract

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

Numero como 'operación'

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

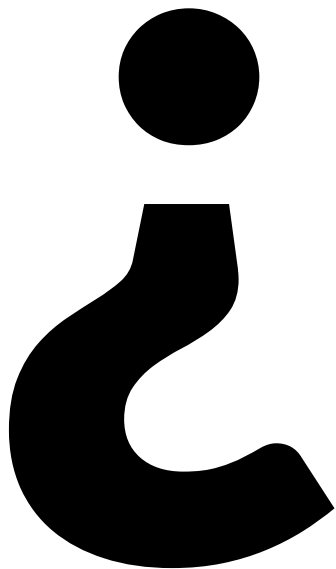
Suma como 'operación'

```
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
Numero o Suma?**





¿Preguntas?

**Esto es posible
por...**

herencia

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

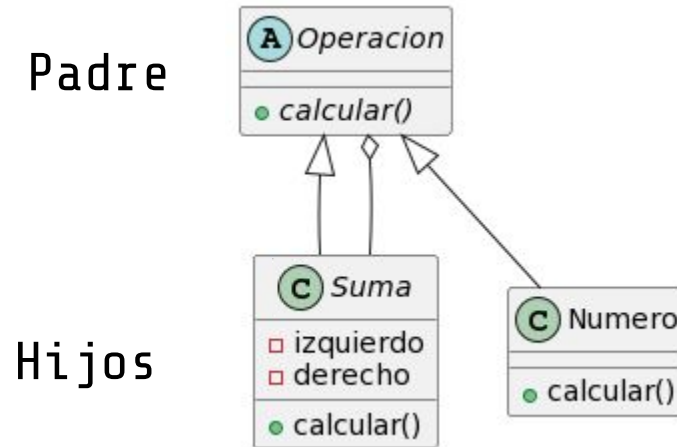
El “sub-tipado” *es un tipo* de polimorfismo

Todos los objetos tienen un solo Supertipo



Nothing is stronger than family.

La familia (hasta ahora)

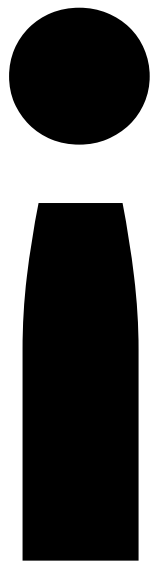


**Porque podemos
decir que
Suma es un
Operacion**

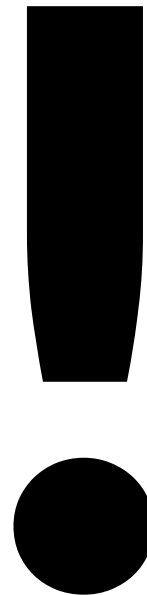
**Y que
Numero es un
Operacion**

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**



¿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?)

Operacion Binaria

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();  
    }  
}
```

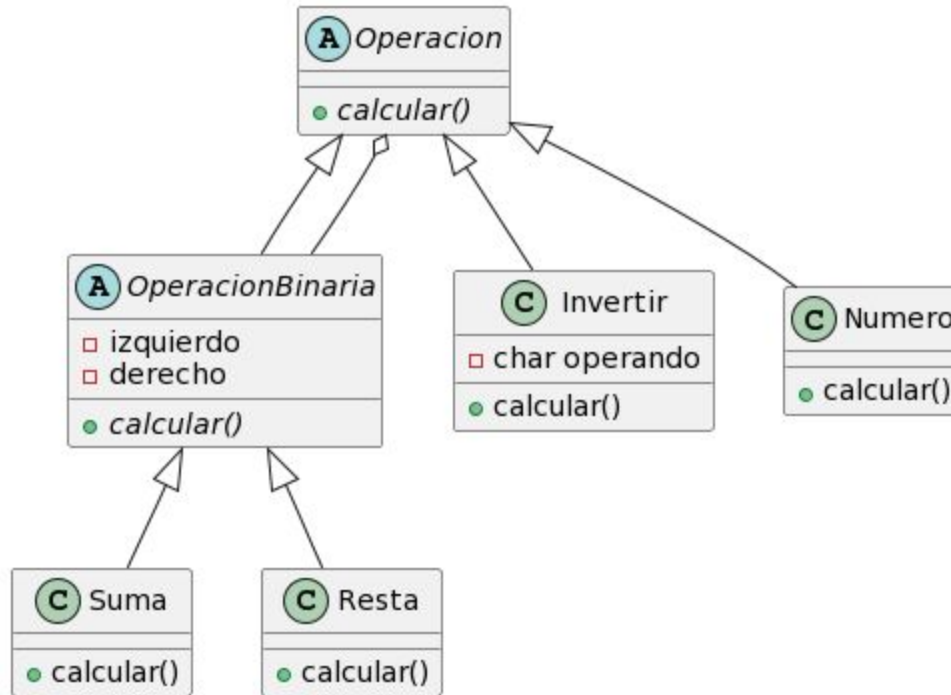
La 'Resta' versión 2

```
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(primeros, 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:
publico
privado

privado

No lo accede nadie
¡ni los hijos!

publico
privado
protegido

protegido

Es accesible por los hijos

**Como privado no
podríamos acceder a
los Operacion en
OperacionBinaria**



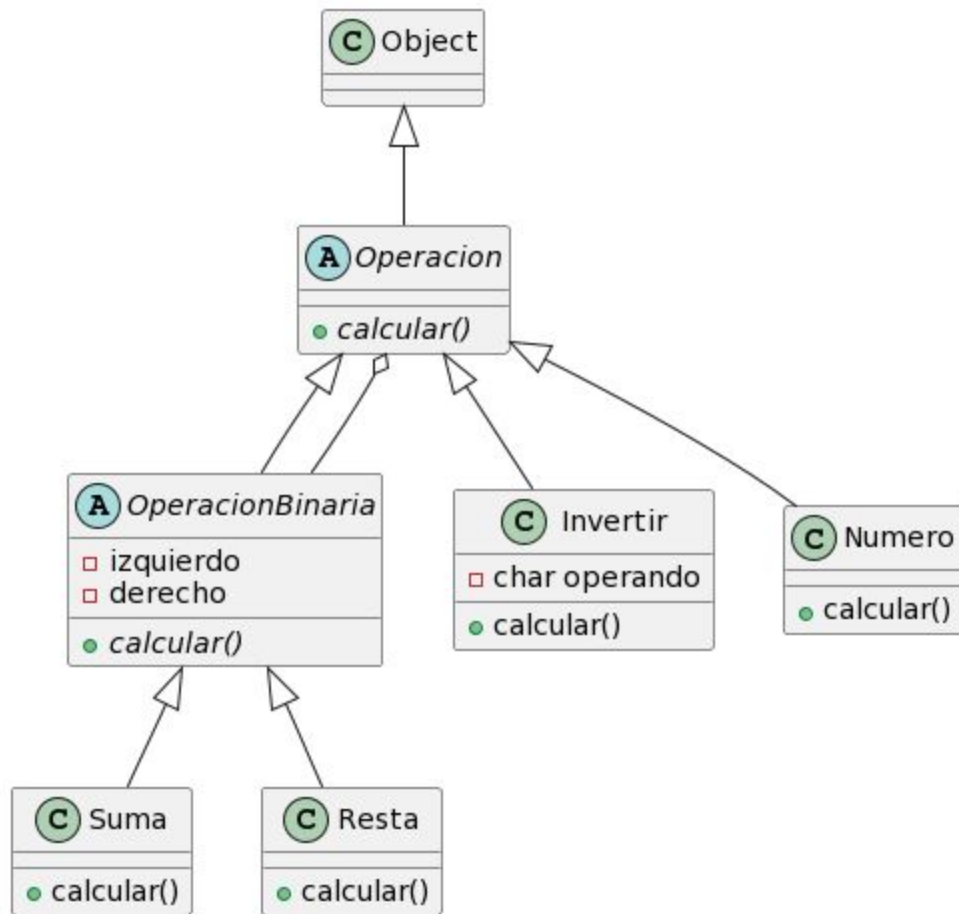
**La idea es compartir
atributos y
comportamiento**



¿Preguntas?

Y recuerden que en Java, todo hereda de Object

De manera directa o indirecta



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

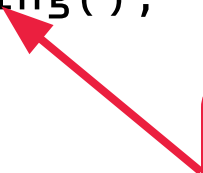


¿Preguntas?

algunas mejoras

Pero es necesario conectarlos

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



**Para que sea
considerado
Operacion**

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 'Inversion'

```
public class Invertir extends Operacion{
    private Operacion Valor;

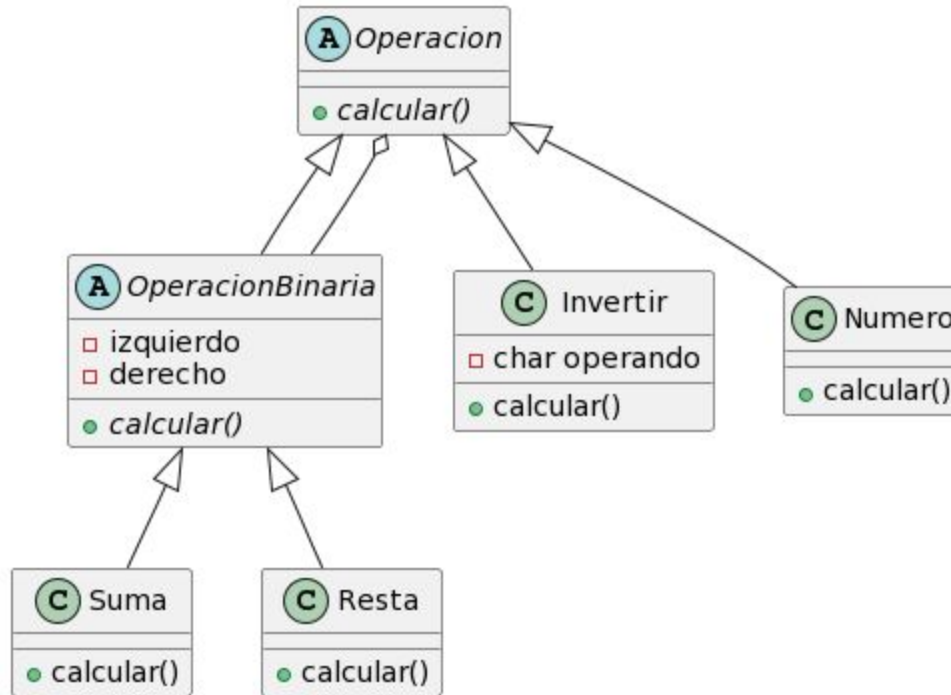
    public Invertir(Operacion valor){
        this.valor = valor;
    }

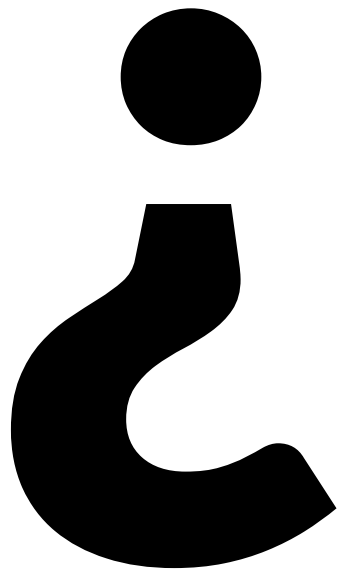
    public int calcular(){
        return -valor.calcular();
    }
    public comoCadena(){
        return "-" + valor.toString();
    }
}
```

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?**

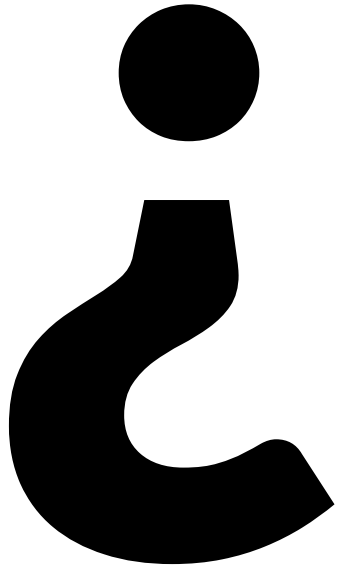


***PLUS
ULTRA***

Operaciones múltiples

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

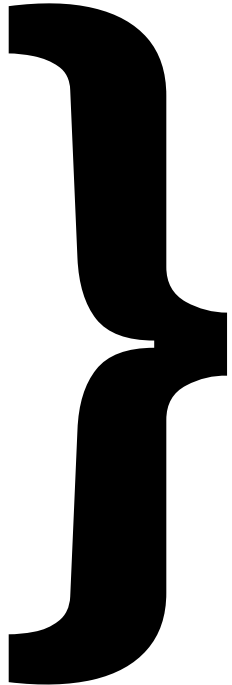
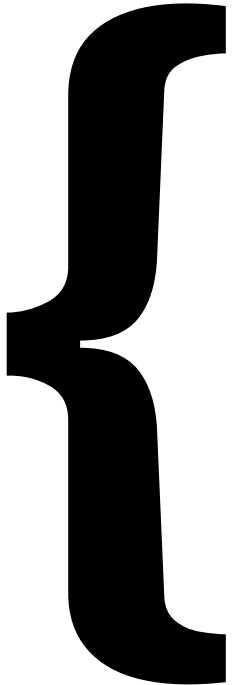
```
public class MultiSuma extends OperacionMultiple{  
    public int calcular(){  
        int calculo;  
        for (op : operandos){  
            calculo = calculo + op.calcular();  
        }  
        return calculo;  
    }  
}
```



ArrayList<Operacion>



ArrayList<*Tipo*>



**La próxima clase
vamos a ver que
son los <>**

TP8

PolyCalculadora

unrn.edu.ar





¿Preguntas?