

OOP + Java II

UNRN

Universidad Nacional
de Río Negro

XII

2024





Sobre dictado presencial

imposible

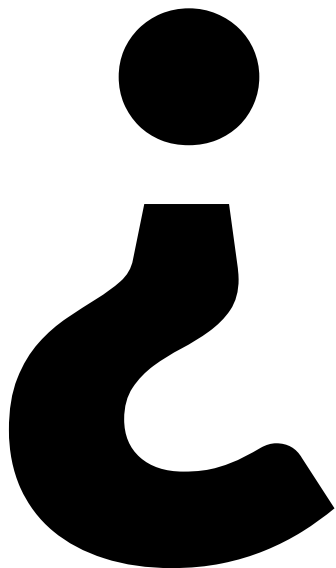
Los horarios con aula *ampliable* me son inviables.

— \ _ (ツ) _ / —

por otro lado

un recordatorio

Las reentregas por correcciones son *solo* con el formulario



Dudas del TP6



toString

Dado el

```
arreglo = {1,2,3,4,5};
```

La cadena resultante debiera ser:

```
[1, 2, 3, 4, 5]
```

Formato esperado

Modificar

```
public void modificar(int valor, int posicion){
```

Este falta, para cambiar el valor en una posición.



**Abran
hilo**

TP5

**Conclusiones
y entregas como para revisar**

¿es clase u objeto?

Goku

¿De qué tipo sería?

¿Existe como tal?

Una “Herramienta de Taller”

¿Que “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 cardinalidad.

Al mencionar que tiene un

Circuito, un Perro, en lugar de describirlo específicamente, 'deleguen' es responsabilidad a otra clase.

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

Aprovechen la composición de objetos

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**

TP4

(Cierra completamente el viernes)



¿Preguntas?



**Abran
hilo**

Tipos de constructores

Parametrizado

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

Por 'defecto'

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona() {  
        this("desconocido", 0);  
    }  
  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

De copia

```
public class Punto {  
    private int x;  
    private int y;  
  
    public Punto(Punto otroPunto) {  
        this.x = otroPunto.x;  
        this.y = otroPunto.y;  
    }  
}
```

Vamos a ver *para* que se crean estos constructores

Privado

```
public class Utilidades {  
    private Utilidades() { ; }  
  
    public static void metodoUtil() {  
        Método estático de la clase utilitaria  
    }  
}
```

Para impedir la instanciación de una clase

No es un constructor, pero es parecido al 'por defecto'

```
public class Punto {  
    private int x = 0;  
    private int y = 0;
```



Aunque se puede asignar directamente.

```
    public Punto(Punto otroPunto) {  
        this.x = otroPunto.x;  
        this.y = otroPunto.y;  
    }  
}
```

Es mejor usar un constructor



¿Preguntas?

Miembros de clase

Una operación adicional para Arreglo

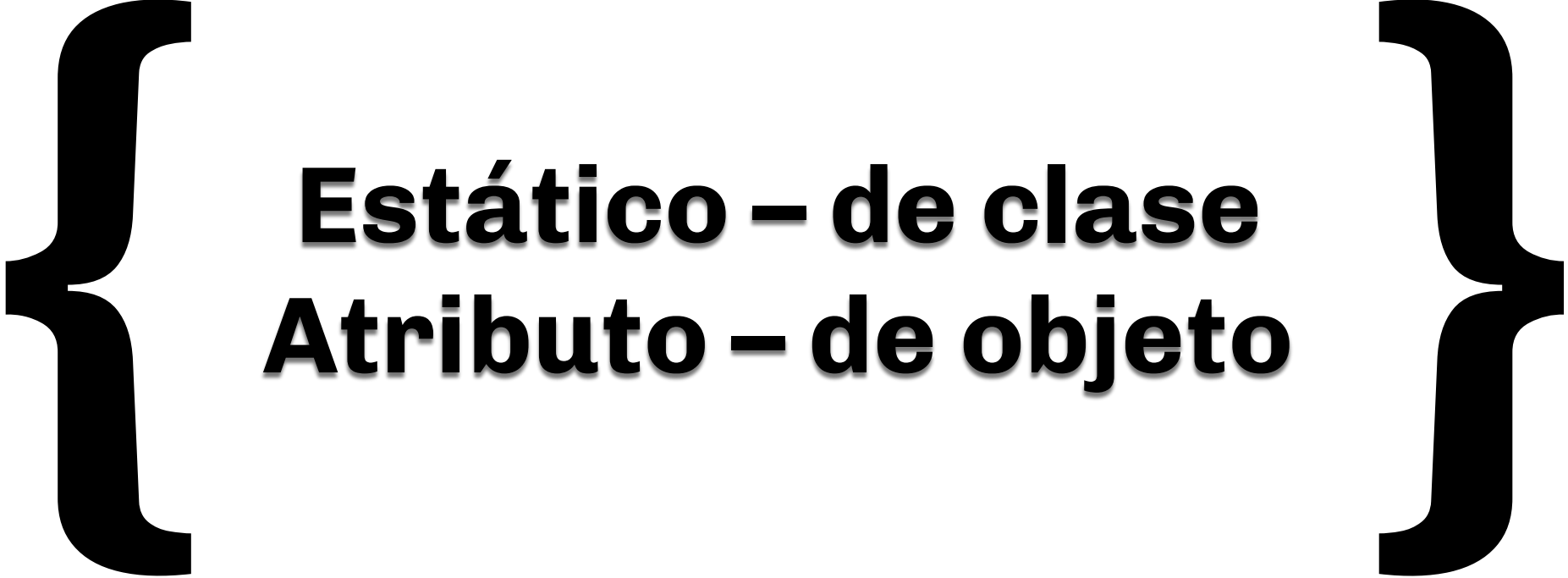
```
class Arreglo{
    int[] arreglo;

    ArregloDinamico(int largo){
        this.arreglo = new int[largo];
    }

    static void copiar(int[] destino, int[] origen){ ... }
}
```

static

Es un método de clase, no tiene asociada una instancia de la misma.
(no hay `this`)



Estático – de clase
Atributo – de objeto

¿Que diferencias hay?

```
void copiar(int[] origen){ ... }
```

```
static void copiar(int[] destino, int[] origen){ ... }
```

**Aplican a métodos
pero a atributos
también**

Un atributo static es compartido entre todas las instancias

```
public class Arreglo{
    private int[] arreglo;
    private static int contador = 0;

    public Arreglo() {
        ...
        contador++;
        ...
    }
    public int cuantosHay() {
        return contador;
    }
}
```



¿Preguntas?

Referencias II

Algo que he observado (con un ejemplo 'sintético')

```
public class Cosa{  
    private int[] arreglo;  
  
    public Cosa(int[] arreglo){  
        this.arreglo = arreglo;  
    }  
}
```

¿Que pasa cuando primera cambia arreglo?

```
int[] conjunto = new int[10];  
Arreglo primera = new Arreglo(conjunto);  
Arreglo segunda = new Arreglo(conjunto);
```

Pero también, puede suceder algo parecido al revés.

```
public class Arreglo{  
    private int[] arreglo;  
  
    public Cosa(int[] arreglo){  
        this.arreglo = arreglo;  
    }  
  
    public int[] comoArray(){  
        return this.arreglo;  
    }  
}
```

**Es un problema
cuando hay
referencias a
objetos *mutables***

Prueben con String o con un Integer

Pero aplica a toda referencia

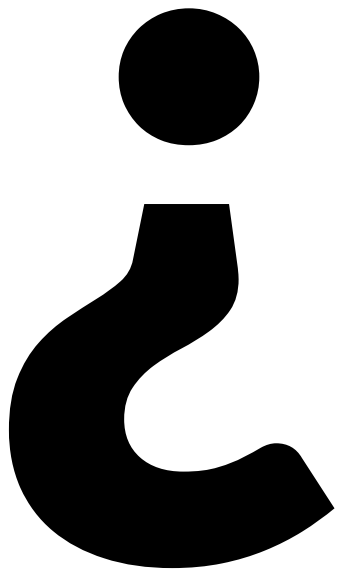


¿Preguntas?

**Esto también puede
provocar...**

Dada una clase muy simple

```
public class Protegido{  
    private int[] arreglo;  
  
    public Protegido(int largo){  
        this.arreglo = new int[largo];  
    }  
    public int[] comoArray(){  
        return this.arreglo;  
    }  
    public String toString(){  
        return Arrays.toString(arreglo);  
    }  
}
```

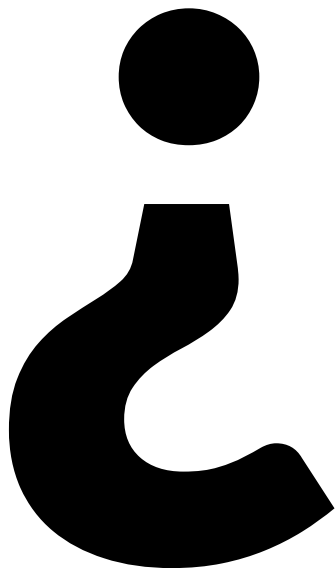


Que pasaria sí...



Instanciamos y usamos su método.

```
Protegido uno = new Protegido();  
int[] unArreglo = uno.obtenerArreglo();  
  
unArreglo[0] = -10;  
System.out.println(uno.toString());
```



**Que veremos en
la salida**





A - Nada

B - -10

C - Una excepcion

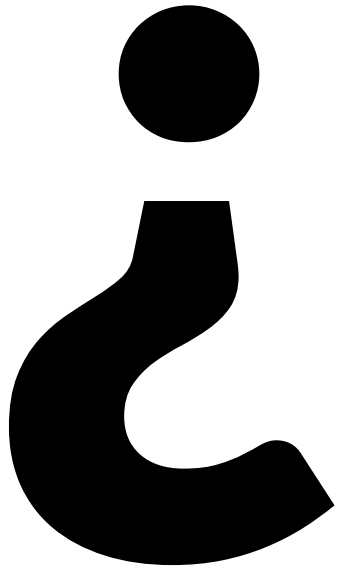




**Las referencias
pueden romper con
el encapsulamiento.**



¿Preguntas?



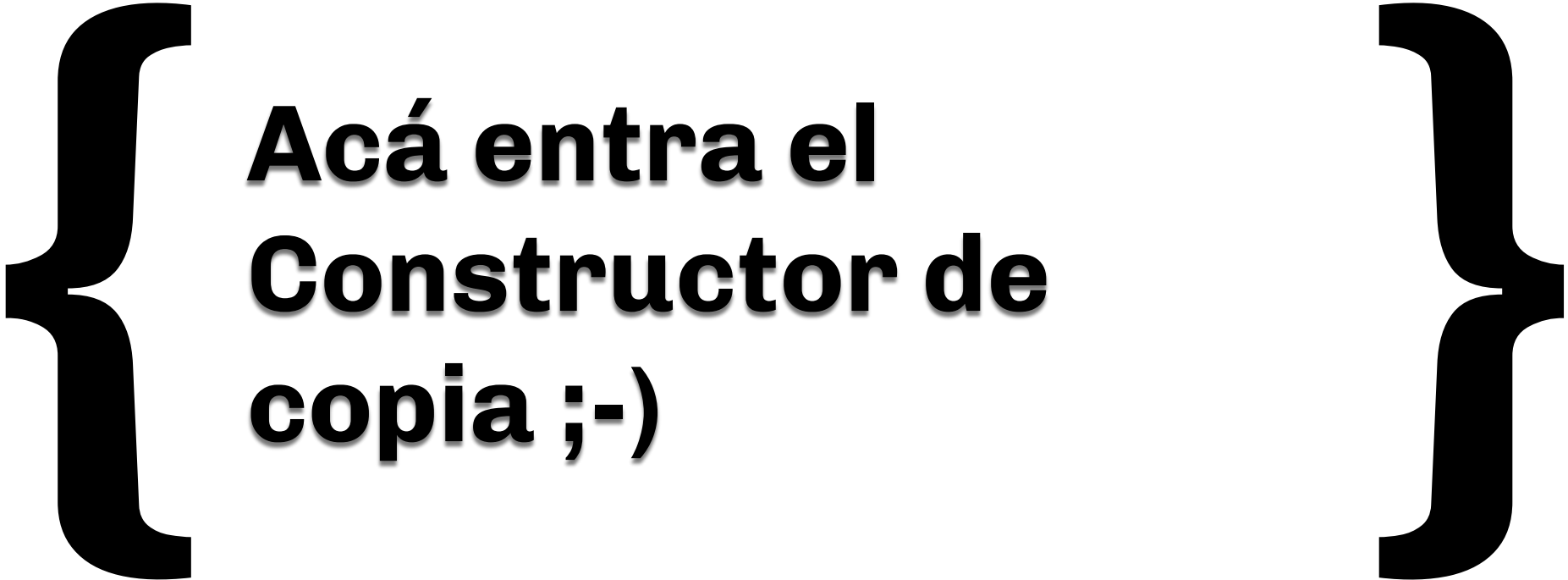
**Como se
resuelve**



Copiando los objetos que entran

Creando una copia de los objetos que salen

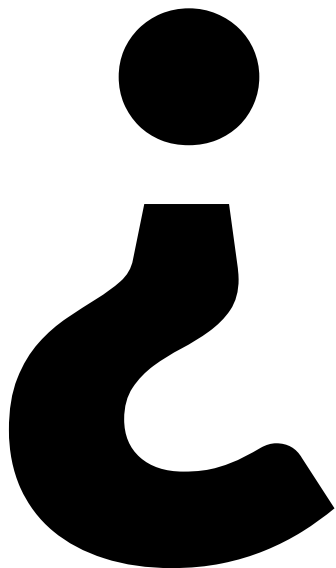
**Esencialmente,
cortando el vínculo
entre lo interno y
externo**



**Acá entra el
Constructor de
copia ;-)**

Con un constructor por copia es más fácil.

```
return new EstaClase(this);
```



**Que pasa si
nuestro objeto
tiene otros
objetos**



Tipos de copia

Copia superficial

Solo llamamos al constructor de copia en la referencia que retornamos

```
return new EstaClase(this);
```


Copia profunda

Llamamos al constructor de copia en todos los objetos que estén vinculados al objeto.

Para retornar un verdadero duplicado.



**Y todo esto,
¿por que?**

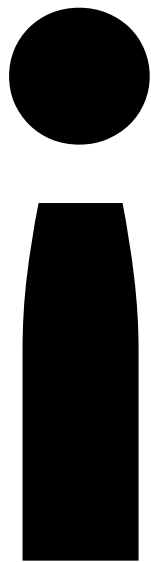


**Es práctica habitual
agregar métodos
'accesores'
get/set atributo**

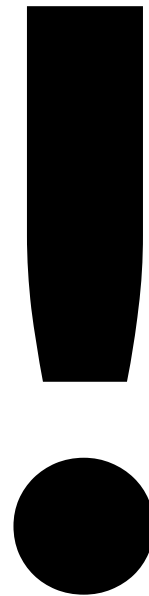
Lo van a ver por todos lados



**También son
conocidos como
getters
y
setters**



**Vuelan por el aire el
encapsulamiento**



**Depender de los
accesores para el
funcionamiento base de
las clases**

es un error

**Pero esto no explica por
qué están por todos
lados**

**Se usan para cuestiones
que no tienen que ver con
el comportamiento.**

Usos de accesorios

Almacenamiento en base de datos

Transmisión por la red

Interfaz gráfica

Su uso **debe de
hacer ruido, puede
ser señal de un
diseño insuficiente**

Un ejemplo

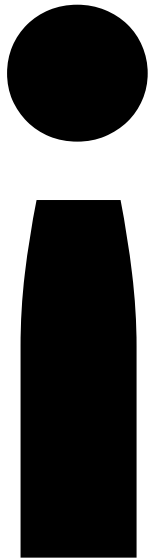
Para una persona

```
public class Persona {  
    private int edad;  
  
    public void setEdad(int edad) {  
        if (edad >= 0 && edad <= 150) {  
            this.edad = edad;  
        } else {  
            throw new IllegalArgumentException("Edad  
imposible.");  
        }  
    }  
}
```

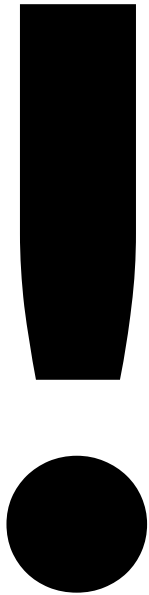
¿Como describen ese método?

**Pero lo mas
importante**

**En términos del
comportamiento de
una persona.**



**Por más de que
incluyan
comportamiento para
evitar estados
incorrectos**



**Es señal de un
diseño incompleto**



**Igual es mejor que
dejar los atributos
públicos**



¿Preguntas?



Taller de Diseño II

'pattern matching'

Esta operación se puede resumir con la sintaxis nueva

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

instanceof pattern matching

```
if (objeto instanceof Usuario user) {  
    // resto de la comparación.  
}
```

unrn.edu.ar

