

# OOP + Java II

**UNRN**

Universidad Nacional  
de **Río Negro**



# Cuestiones de estilo

que quedaron sueltas

**Al extender,  
sobreescibir solo  
para llamar a super  
no es correcto\***\*

*Excepto con el constructor*

# Minimizar el código duplicado

**Las clases,  
atributos y  
métodos llevan  
documentación**

---

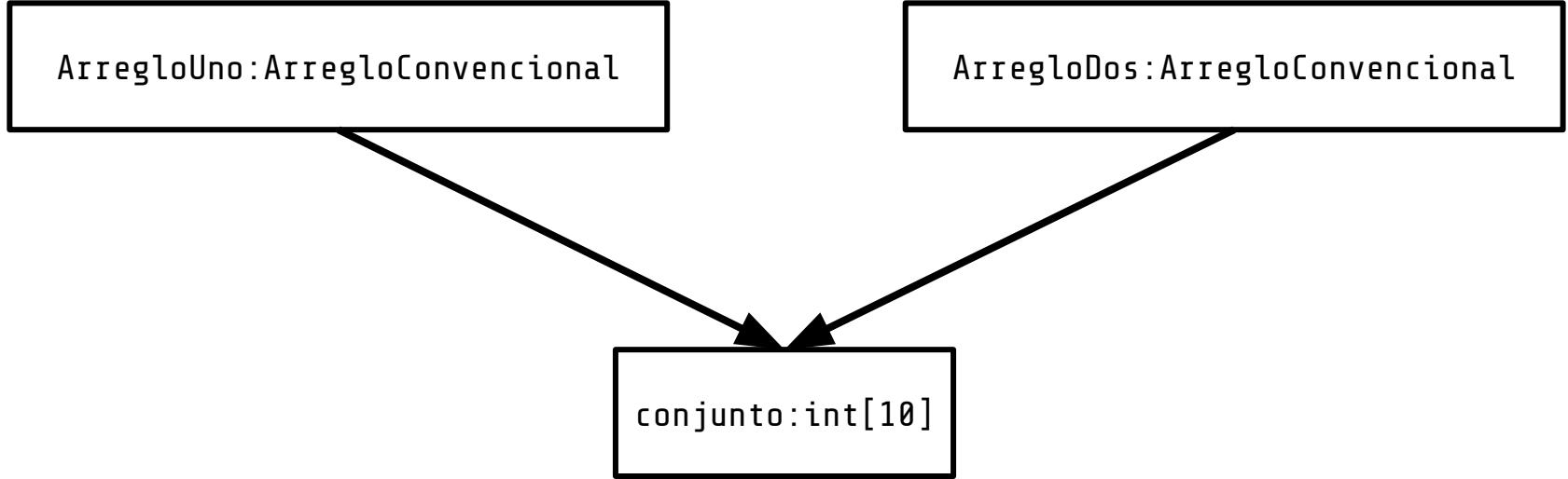
# Referencias II

# Dada esta implementación del constructor

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

# ¿Qué pasa en esta situación?

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



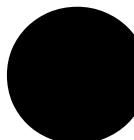
Ooops, ambas instancias apuntan al mismo arreglo

**UNRN**

Universidad Nacional  
de Río Negro

# Pero también, puede suceder algo parecido al revés.

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

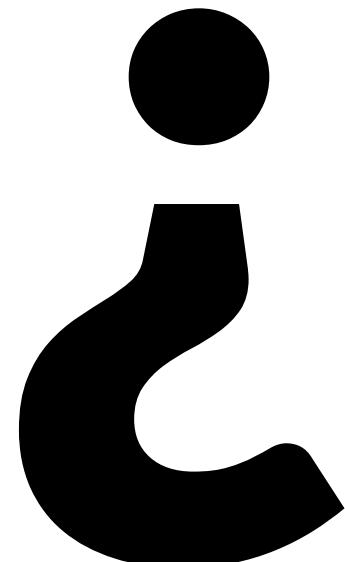
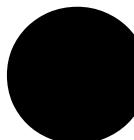


# Que pasa cuando

# Instanciamos y usamos su método.

```
Protegido uno = new Arreglo();
int[] elArreglo = uno.comoArreglo();

elArreglo[0] = -10;
System.out.println(uno.toString());
```

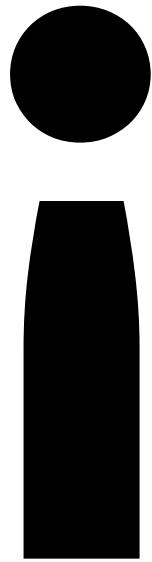


**Que veremos  
en la salida**

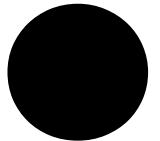
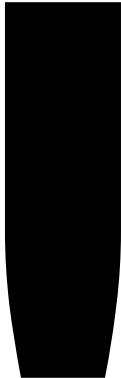
**A - Nada**

**B - -10**

**C - Una excepción**



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





**¿Preguntas?**



¿

y ahora

?

# ¿Yahora?

```
public class Cadena{  
    private String cadena;  
  
    public Cadena(String cadena){  
        this.cadena = cadena;  
    }  
}
```

# ¿Qué pasa en esta situación?

```
String mensaje = new String("hola mundo");  
Cadena primera = new Cadena(mensaje);  
Cadena segunda = new Cadena(mensaje);  
mensaje = mensaje + "adiós";
```

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

Pero igual siguen  
conectados, *aplica a*  
*toda referencia*

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?**



**Conectar así, es  
una herramienta  
poderosa**



---

Tio Ben dixit

**UNRN**

Universidad Nacional  
de Río Negro



**Abran  
hilo**

---

# Accesores

**Y es práctica habitual  
agregarlos como**

**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  
*comportamiento* 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.**

# Como

- Almacenamiento, en base de datos
- Transmisión por la red
- Interfaz gráfica

**Su uso **es** señal de  
un diseño  
que se quedó corto**

**Los métodos  
get / set no pueden  
ser usados para la  
lógica del  
problema**

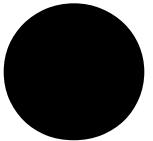
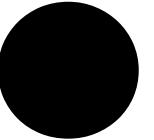
---

# Un ejemplo

---

# Para una persona

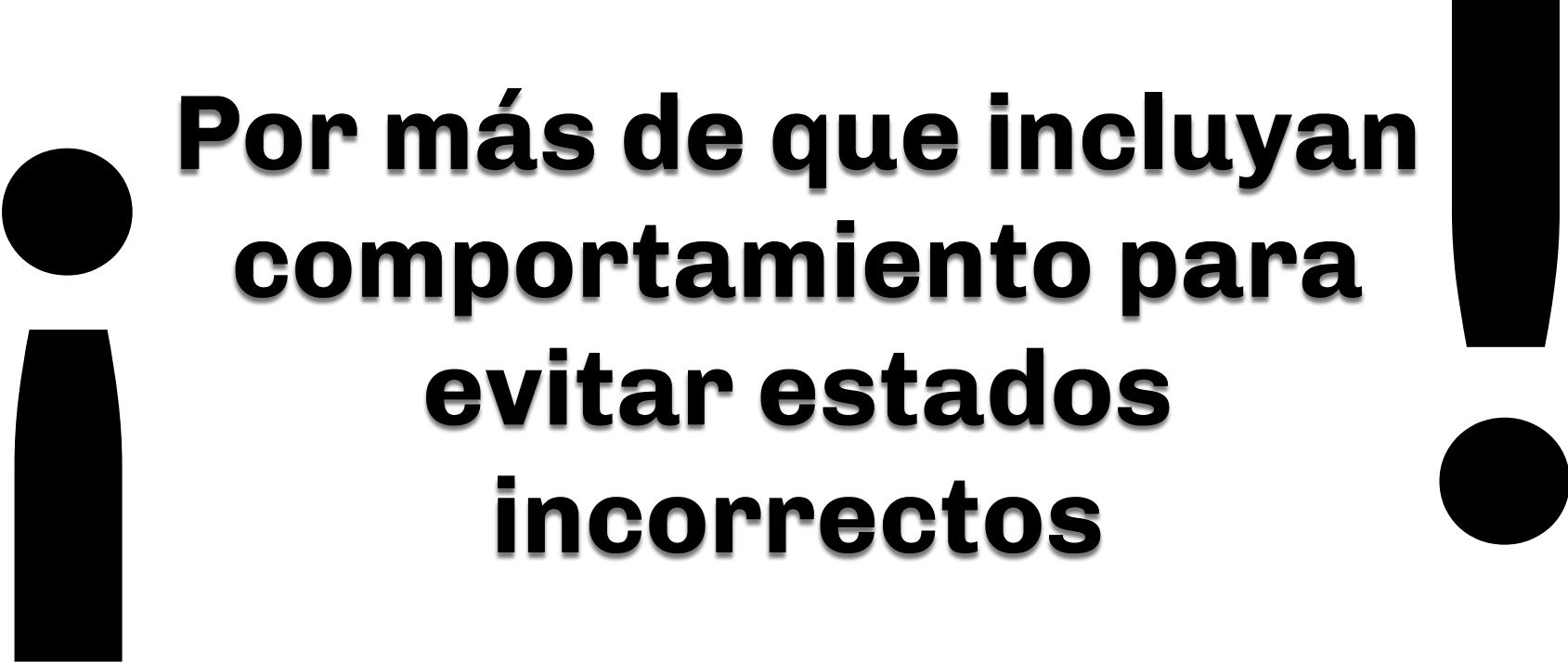
```
public class Persona {  
    private int edad;  
  
    public void setEdad(int edad) {  
        if (edad >= 0 && edad <= 150) {  
            this.edad = edad;  
        } else {  
            throw new IllegalArgumentException("Edad imposible.");  
        }  
    }  
}
```



**¿Cómo  
describen ese  
método?**

# Pero lo más importante

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



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

—

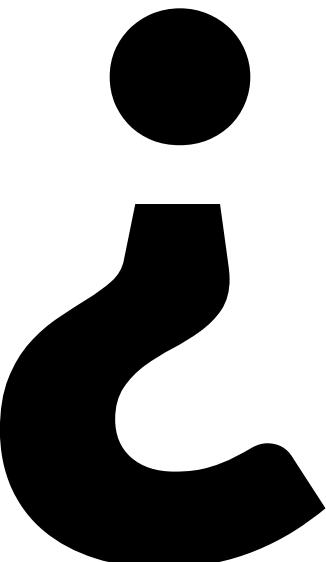
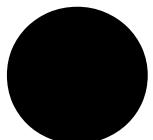
Es señal de un  
diseño incompleto

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



**¿Preguntas?**





# **Como podemos preservar el encapsulamiento**

Implementando

# Inmutabilidad

# Volviendo a Número como 'operación'

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

¿Qué es realmente?

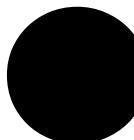
# Agregando este método... ¿En qué se transforma?

```
public class ¿¿?? extends Operacion{
    private int valor;

    public Numero(int valor){
        this.valor = valor;
    }
    public int calcular(){
        return valor;
    }
    public void modificar(int nuevoValor){
        this.valor = nuevoValor;
    }
}
```

# ¿Qué es realmente?

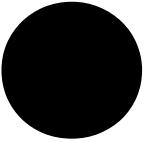
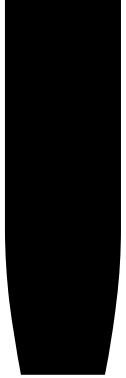
```
public class ¿¿?? extends Operacion{  
    private int valor;  
  
    public Numero(int valor){  
        this.valor = valor;  
    }  
    public int calcular(){  
        return valor;  
    }  
}
```



# ¿Podrían coexistir?



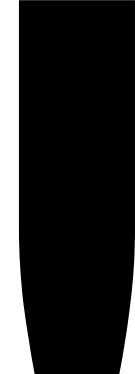
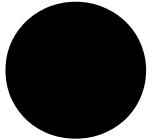
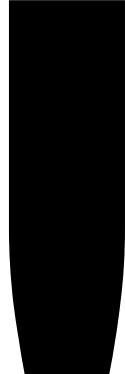
Sí



# Como Constante y Variable

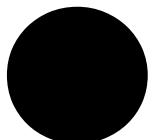
**Los tipos inmutables  
devuelven copias en  
las operaciones que  
modifican el estado**

**Y los mutables,  
directamente  
modifican el  
estado del objeto**



**La  
documentación  
es importante!**

**No está de más  
usar final para  
reforzar el hecho  
de que no cambian**



# Como diseñar e identificar inmutabilidad

# Cuál funciona como *mutable* y cuál *inmutable*

```
public Tiempo sumar(segundos)
```

```
public void sumar(segundos)
```

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?**



**La otra  
opción**

# Copiando los objetos que entran

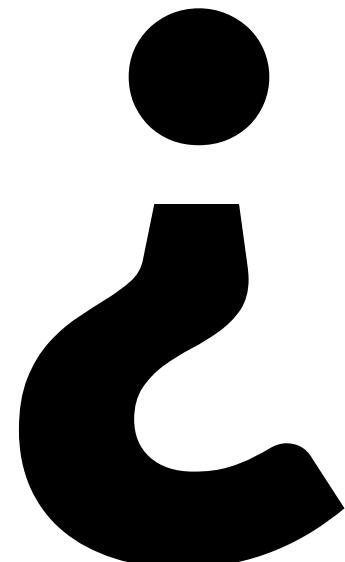
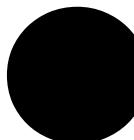
# Copiando los objetos que salen

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

---

# Para arreglos vimos

## Arrays.copyOf



**Y nuestros  
objetos**

# Con un constructor de copia

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

---

# Mucho más fácil

```
return new Punto(this);
```

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?**



-

Para sumar a los  
**Tipos de  
constructores**

# 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;  
    }  
}
```

# Parametrizado

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

# Privado

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

**Para impedir la instanciación de una clase**

# Inicialización de atributos

```
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**

# Bloque de inicialización de instancia

```
public class ConAtributos{  
    private int posicion;  
  
    {  
        posicion = POR_DEFECTO;  
        // inicializador atributos  
    }  
  
}
```

**No puede lanzar excepciones**

---

# **Orden de instancia**

- 1. asignación directa en la declaración**
- 2. inicializador de instancia**
- 3. constructor**

# La inicialización de los atributos va en el 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;  
    }  
}
```

**La utilización de  
atributos estáticos  
debe de estar  
justificada**

# Inicializadores

# Bloque static

```
public class ConAtributosDeClase{  
    private static int contador;  
    static {  
        contador = 0;  
        // resto del inicializador estático  
    }  
}
```

**La ‘construcción’ de lo estático**

# Bloque de inicialización de instancia

```
public class ConAtributosDeClase{  
    private static int contador;  
    private int posicion;  
    static {  
        contador = 0;  
        // inicializador estático  
    }  
    {  
        // inicializador atributos  
    }  
}
```

---

# **Orden de instancia**

- 1. declaración estática**
- 2. inicializador estático**
- 3. declaración de atributos**
- 4. inicializador de instancia**
- 5. constructor**



**¿Preguntas?**



---

# Paquetes II



# package · paquete

Una agrupación de clases e interfaces  
relacionadas bajo un nombre

# Declaración de un paquete

```
package ar.unrn;
```

*La estructura de directorios  
tiene que ser la misma*

# ¿Como importar algo de otro paquete?

*Debajo de package en la clase*

```
import nombre.del.paquete.Clase;
```

```
import nombre.del.paquete.*;
```

**no recomendado**

**1**

# Ayuda a la organización

**2**

# **Evitan los conflictos de nombre**

**El paquete  
java.lang  
Ya está importado**

**No hacer  
import paquete.\*  
Solo traer lo que  
necesitamos**

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?**



# Excepciones en los constructores

**¡Perfectamente normal!**  
**(no olviden documentarlos)**

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?**



# Hasta la próxima



**unrn.edu.ar**

**UNRN**

Universidad Nacional  
de Río Negro



| unrnionegro