

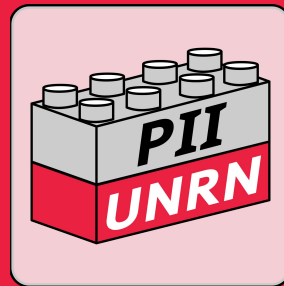
OOP + Java

UNRN

Universidad Nacional
de Río Negro

XI

2024





¿Dudas del TP5?



TP5

Revisemos algunos que me llamaron la atención.



**Abran
hilo**

Igual vamos a repetir el ejercicio



¿Preguntas?

Paquetes

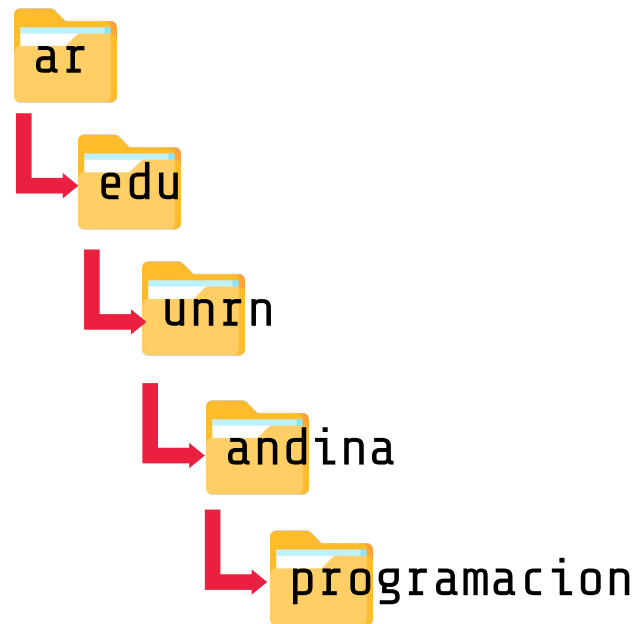
¿Qué son?

Una forma de organizar y agrupar funcionalidad en proyectos más grandes.

Formalmente, son la URL de la organización invertida

`ar.edu.unrn.andina.programacion.`

`arreglos`
`modulo`
`consola`



Pero suele ser largo

La primera de cada archivo java debe reflejar su ubicación

```
package ar.edu.unrn.andina.programacion.arreglos;
```

(Esto lo hace solo IntelliJ)

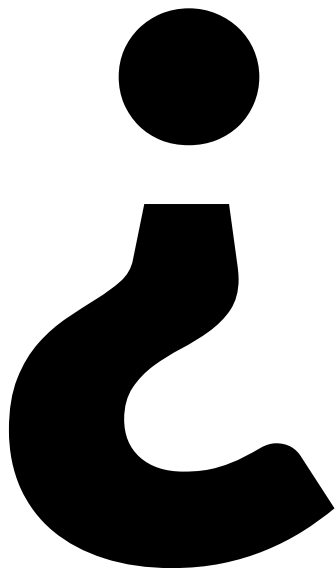


¿Preguntas?

Métodos

Detalles sobre el constructor



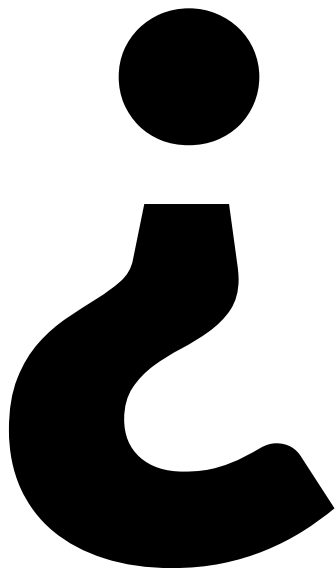


Qué hacia el constructor



Estructura base

```
public class MiClase{  
    private String unaCadena;  
  
    public MiClase(String argumento){  
        unaCadena = "Hola objetos " + argumento;  
        // El Constructor le da un valor a los atributos  
        // las instrucciones sobre la inicialización  
    }  
}
```



**Una clase, puede
tener más de uno**



opcionalmente
con argumentos

¡Múltiples constructores!

```
class MiClase{
    String unaCadena;

    MiClase(){
        unaCadena = "Hola Objetos";
        // Sin argumentos, y si no dice nada...
    }

    MiClase(String cadena){
        unaCadena = cadena; // Con argumentos
    }
}
```

Dos instanciaciones válidas

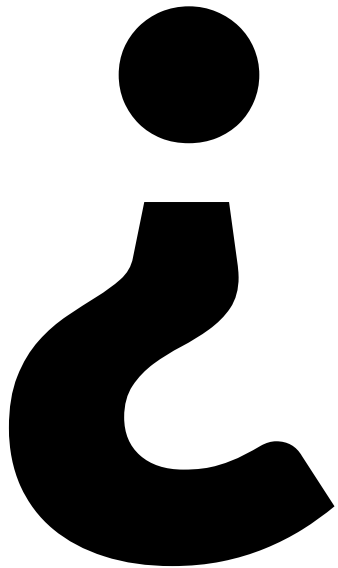
**¡Acá la clase se
vuelve objeto!**



```
MiClase uno = new MiClase();
```

```
MiClase dos = new MiClase("Roberto");
```

Clase, ¡hoy te convertís en héroe objeto!



**Como sabe a qué
constructor
llamar**



Introducción a

sobrecarga de métodos



Al llamar un método

Esto funciona con constructores

```
MiClase uno = new MiClase();
```

```
MiClase dos = new MiClase("Roberto");
```

**Se busca el método
cuyos tipos de
argumento coincidan**

OJO


¡Solo se ejecuta uno!

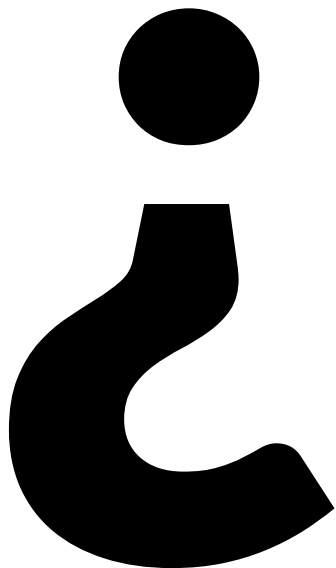


Encadenamiento de constructores

¡Pero se pueden encadenar!

```
class MiClase{  
    String unaCadena;  
  
    MiClase(){  
        this("el argumento por defecto");  
    }  
  
    MiClase(String argumento){  
        unaCadena = argumento; // Con argumentos  
    }  
}
```



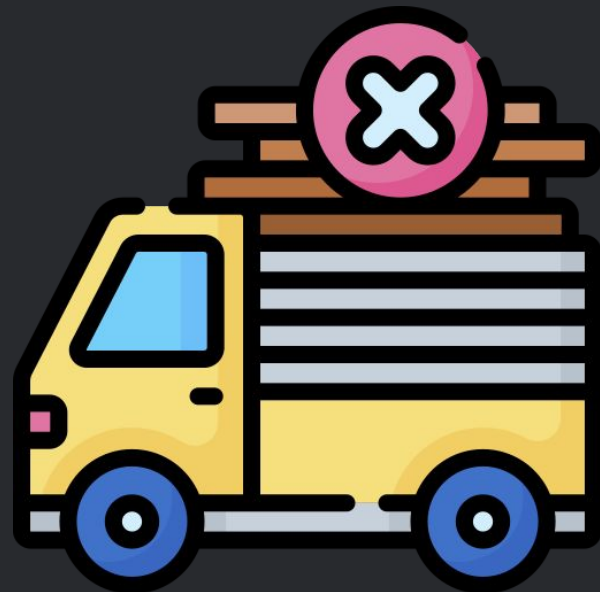


**Cuando
encadenar**



Para reducir la duplicación de código

Más sobre la Sobrecarga



Sobrecarga de métodos

```
class Sobrecargado{  
  
    int metodoSobrecargado(int a, int b);  
    int metodoSobrecargado(int a, float b);  
}
```

Sobrecarga de métodos

```
class Sobrecargado{
```

```
int metodoSobrecargado(int a, int b);    //1  
int metodoSobrecargado(int a, float b);  //2
```



```
int metodoSobrecargado(int uno, int dos); //3
```



```
}
```

solo 've' la combinación de tipos en los argumentos

El nombre que tenga el argumento no es tenido en cuenta

El orden de los tipos importa

```
class SobrecargadoDos{
```

```
void metodoSobrecargado(int a, float b);
```

```
void metodoSobrecargado(float a, int b);
```

```
}
```

}

También OK

Pero con el retorno no alcanza

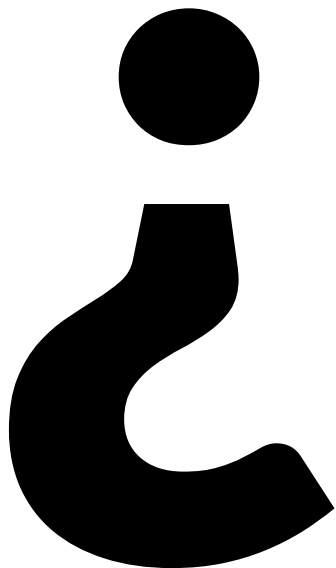
```
class SobrecargadoDos{
```

```
int metodoSobrecargado();  
float metodoSobrecargado();
```

```
}
```

}



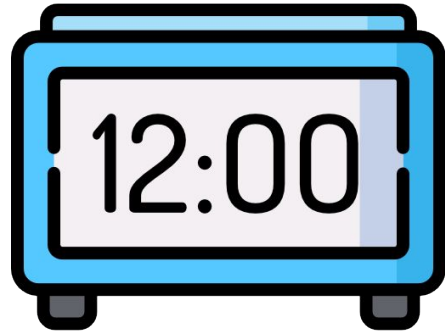


**Para que se
usa**



**Da contexto
específico bajo el
mismo nombre**

Mismo comportamiento, diferente información de entrada



```
clase Tiempo  
suma(int segundos)  
suma(Tiempo otro)
```



¿Preguntas?

Veamos más de cerca Object

1

**Todas las clases
extienden a `Object` o
a quien indiquemos
con `extends`**

**El resultado de esto,
es que todo es un
Object**

2

**Es el mínimo común
denominador en
comportamiento**

Object define (entre otras cosas que no vamos a ver)

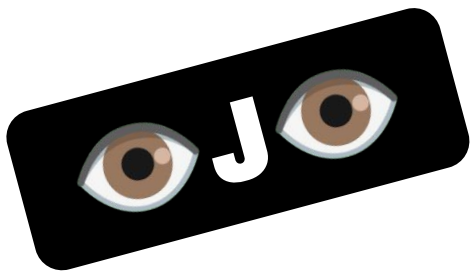
- equals - si este es igual a otro
- hashCode - un número que representa
- toString - representación textual del contenido

Clase usuario

```
public class Usuario {  
  
    private LocalDate fechaNacimiento;  
    private String nombre;  
    private String apellido;  
  
    public User(LocalDate fechaNacimiento, String nombre, String apellido) {  
        this.fechaNacimiento = fechaNacimiento;  
        this.nombre = nombre;  
        this.apellido = apellido;  
    }  
}
```

**iguales
es la igualdad de
dos objetos**

**¿apuntan a
la misma
cosa?**



**equa^ls no es lo
mismo que ==**

==

**es la igualdad de
referencias**

**¿apuntan al
mismo
lugar?**

Atributos

El contrato de equals

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
```

¿es si mismo?

```
otro == null
```

¿es nulo?

```
getClass() == otro.getClass()
```

¿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);
}
```

¿es si mismo?

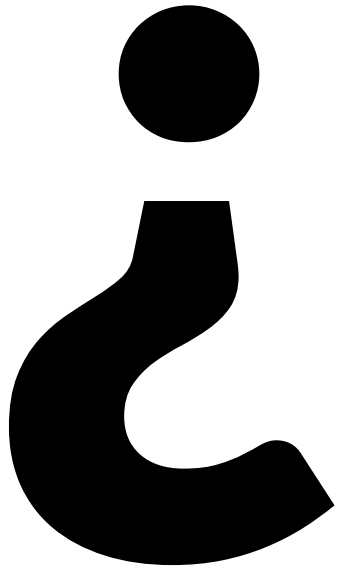
¿es nulo?

¿Mismo tipo?

Comparamos su estado

Al entrar Object en juego, las conversiones pueden fallar

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



**No podemos usar
la sobrecarga
con equals**

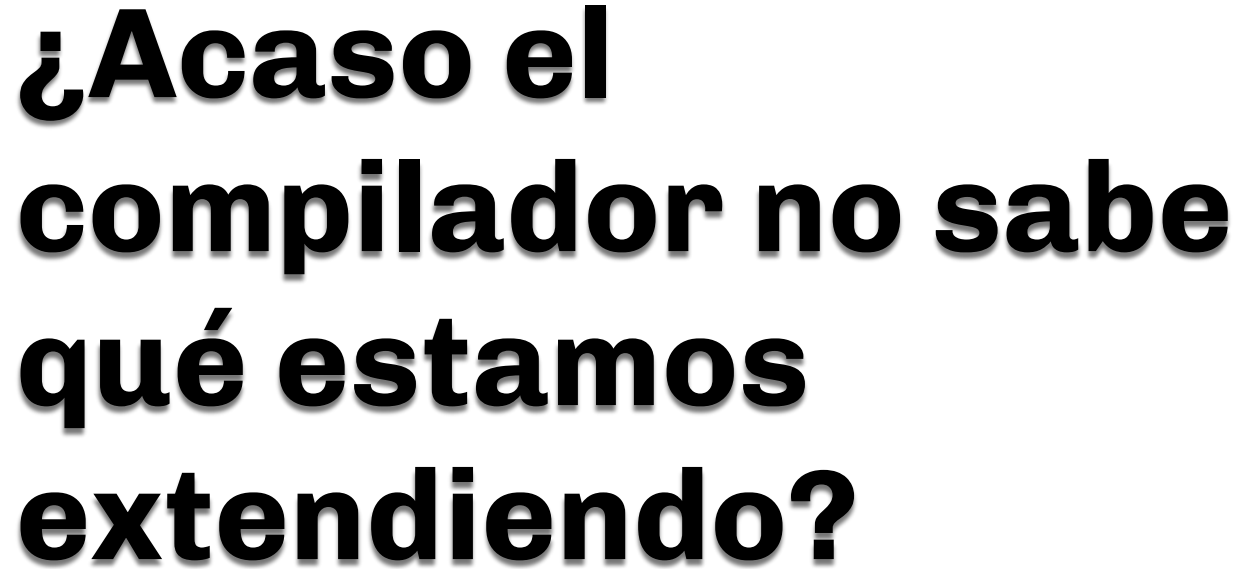


**Teóricamente sí,
pero...**

**No es posible
garantizar que los
atributos del
contrato se cumplan**

Por otro lado:

¿@Override?



**¿Acaso el
compilador no sabe
qué estamos
extendiendo?**

**Evita que la
sobrecarga nos
sorprenda**

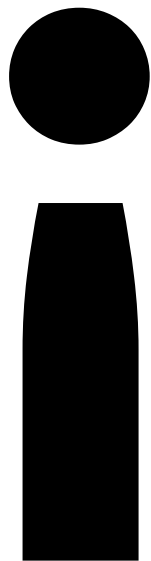
Si solo implementamos la sobrecarga, esto no puede suplantar la versión de `Object` (no coincide el tipo)

```
public boolean equals(Object objeto);
```

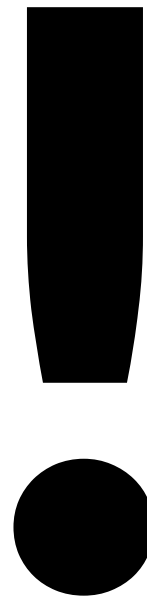
```
public boolean equals(Usuario otro);
```



¿Preguntas?



**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!

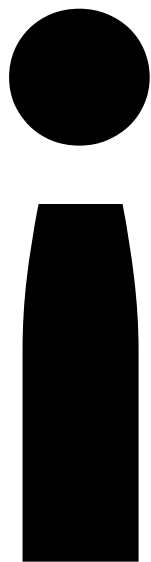


***Relacionado con su
posición en
memoria***

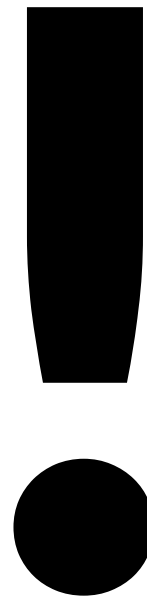
**Si dos objetos son
'equals'**

**Su hashcode
es igual**


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



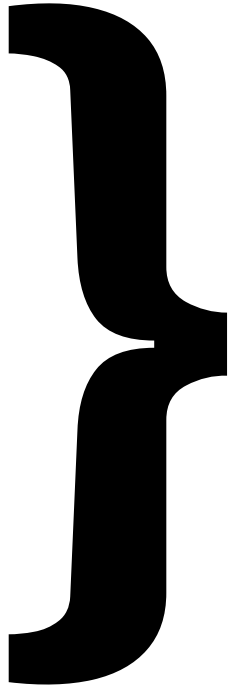
**¡No seguirlo trae
comportamiento
errático!**





**Por suerte es
fácilmente
testeable**

***assertEquals de dos objetos
construidos con los mismos
valores**



iCollections!

Se usa con
Set - Conjuntos
Map - Diccionarios

instanceof



¿Preguntas?

TP6

Clases y Objetos 1

unrn.edu.ar

