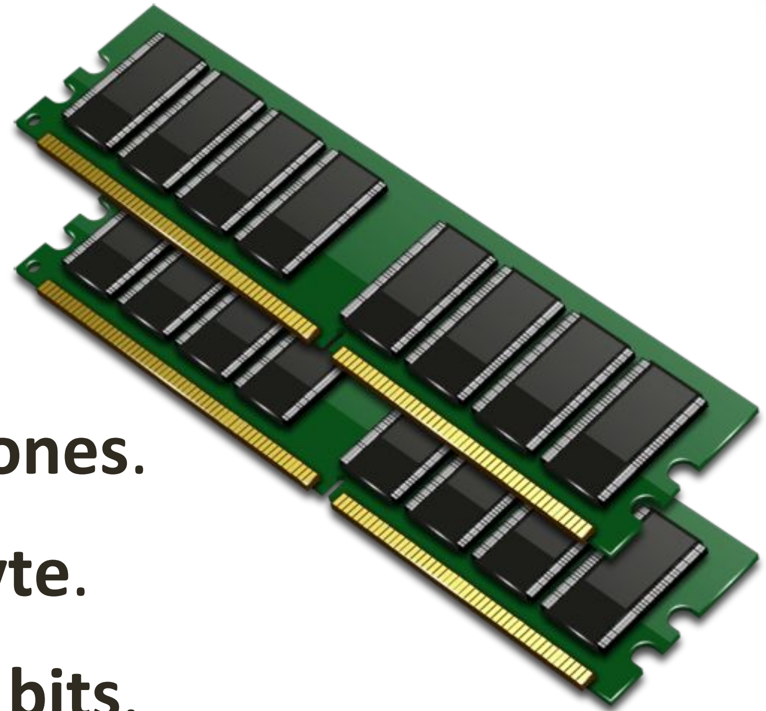
Hardware y Software

El **hardware** de una computadora es el conjunto de *componentes físicos*

El **software** de una computadora es el conjunto de *componentes lógicas*, datos, programas y los documentos que los respalda.

La memoria



La **memoria** es una de las componentes del hardware.

Es el dispositivo en el que se almacenan **datos** e **instrucciones**.

La unidad de medida es el **byte**.

Un byte es una secuencia de **bits**.

Cada bit toma un valor binario, esto es, 0 o 1.

Así, datos e instrucciones se representan en memoria como secuencias de 0 y 1.

La memoria

Podemos pensar la memoria como una secuencia de **celdas** cada una de las cuales tiene asociado una **dirección** y un **contenido**.

dirección	contenido
00000000	**00010001**
00000001	**11001000**
00000010	**11100110**
00000011	**01100001**
00000100	**01100101**
…	

Variables

Los lenguajes de programación permiten tener una visión **abstracta** de la **memoria**.

El programador no accede a memoria través de las direcciones de las celdas, sino a través de **variables**.

El programador no manipula el contenido de la celda en su representación binaria.

Cada variable mantiene un **valor** que depende del **tipo** establecido en una **declaración**.

Variables

La **declaración** de una variable establece su **nombre**, **alcance** y **tipo**.

Cuando el programa se ejecute cada variable va a quedar asociada a una **celda** de memoria y a un **valor**.

El tiempo de vida de una variable comienza cuando se reserva una celda y termina cuando se libera.

Variables

El valor de una **variable de tipo elemental** es un valor dentro del conjunto de valores que determina el tipo.

Ejemplo: *en el programa* *en la memoria*

```
int x = 237;
```


El valor de una **variable de tipo clase** es una **referencia**, que vincula a la variable con la celda de memoria que mantiene el **estado interno del objeto**.

Ejemplo: *en el programa* *en la memoria*

1. **Ciudad ciu;**
2. **ciu=new Ciudad(…);**

Variables

El valor de una **variable de tipo elemental** es un valor dentro del conjunto de valores que determina el tipo.

Ejemplo: *en el programa* *en la memoria*

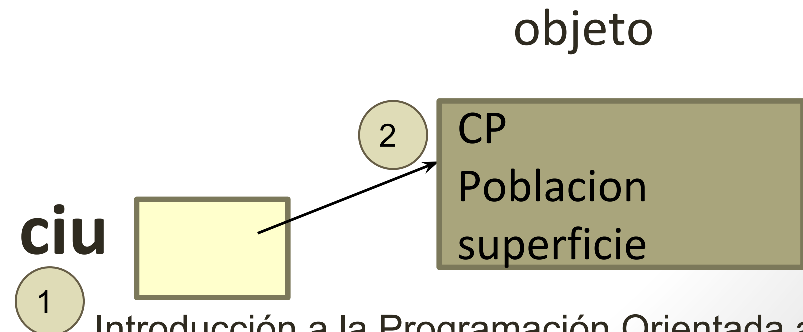
int x = 237; **X** | 237 |

El valor de una **variable de tipo clase** es una **referencia**, que vincula a la variable con la celda de memoria que mantiene el **estado interno del objeto**.

Ejemplo: *en el programa* *en la memoria*

(1) **Ciudad ciu;**

(2) **ciu=new Ciudad(…);**



En la Memoria

```
int x = 237;
Ciudad ciu;
ciu=new Ciudad(…);
```

dirección	contenido
00000000	**00010001**
x	**11101101 = $(237)_{10}$**
00000010	**11100110**
ciu	**01100001**
00000100	**01100101**
…	
01100001	CP Poblacion superficie
…	**01100101**

Variables

El alcance de una variable es el segmento de código en el cual es visible y puede ser usada.

El alcance de una **variable local** es el bloque dentro del cual se declara.

En Java el alcance de una **variable declarada como atributo de instancia privada** es global a la clase, es decir, la variable es visible en toda la clase.

Variables de tipo elemental

Una **variable de un tipo elemental** se **declara** mediante una instrucción como:

```
int i = 1;   int j;  char ch ;
```

Cuando se ejecuten estas instrucciones se reservan locaciones de memoria para **i**, **j** y **ch**.

La variable **i** queda inicializada en el misma instrucción de declaración.

Las variables **j** y **ch** no pueden ser usadas en una expresión hasta que no se les **asigne** un valor.

Variables de tipo elemental

El tipo determina el conjunto de valores que la variable puede recibir.

En ejecución **variable de tipo elemental** mantiene un valor atómico, indivisible, dentro del conjunto de valores establecidos por su tipo.

El valor se almacena en **memoria**, en una celda que se reserva en el momento que se ejecuta la declaración.

Variables de tipo elemental

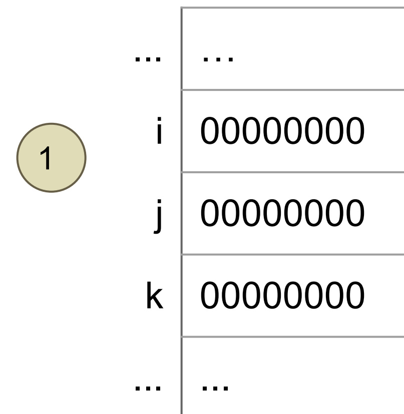
Cuando una **variable de un tipo elemental** aparece en una expresión, la expresión se computa reemplazando la variable por su valor.

Ejemplo

1 **int j, k, i;**

2 **j=20;   k=10;   i = 5;**

memoria



En una asignación el valor computado por la expresión se almacena como valor de la variable que aparece a la izquierda.

Variables de tipo elemental

Cuando una **variable de un tipo elemental** aparece en una expresión, la expresión se computa reemplazando la variable por su valor.

Ejemplo

memoria

(1) **int j, k, i;**

(2) **j=20; k=10; i = 5;**

	...	...
1	i	5
2	j	20
	k	10
	...	...

En una asignación el valor computado por la expresión se almacena como valor de la variable que aparece a la izquierda.

Variables de tipo elemental

Cuando una **variable de un tipo elemental** aparece en una expresión, la expresión se computa reemplazando la variable por su valor.

Ejemplo

1 **int j, k, i;**

2 **j=20; k=10; i = 5;**

3 **i = j + 2 * k;**

memoria

	...	...
1	i	5
2	j	20
	k	10
	...	...

En una asignación el valor computado por la expresión se almacena como valor de la variable que aparece a la izquierda.

Variables de tipo elemental

Cuando una **variable de un tipo elemental** aparece en una expresión, la expresión se computa reemplazando la variable por su valor.

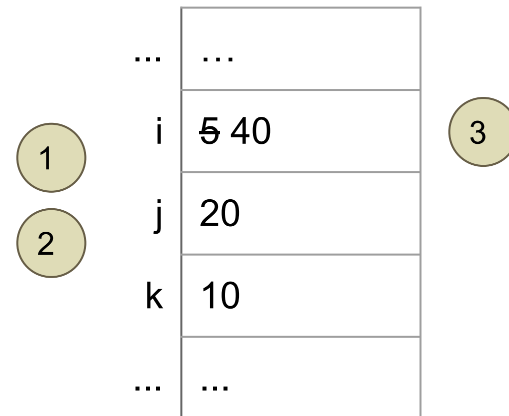
Ejemplo

memoria

1 **int j, k, i;**

2 **j=20; k=10; i = 5;**

3 **i = j + 2 * k;**



En una asignación el valor computado por la expresión se almacena como valor de la variable que aparece a la izquierda.

Variables de tipo clase

Una **variable de un tipo clase** se **declara** mediante una instrucción como:

```
Ciudad  s;
```

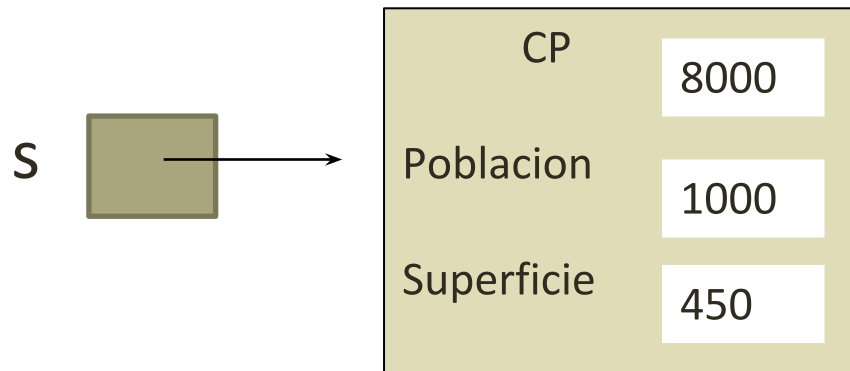
Cuando se ejecuta el código se reserva una **celda de memoria** que se inicializa como una **referencia nula**.

Gráficamente:

Variables de tipo clase

La instrucción: `s= new ciudad(8000,1000,450);`

1. <u>*Reserva*</u> memoria para mantener el estado interno del objeto de acuerdo a la estructura determinada por la clase **Ciudad**
2. <u>*Invoca*</u> al constructor de la clase que <u>*inicializa*</u> los atributos de instancia
3. <u>*Liga*</u> el objeto creado a la variable **s**



Variables de tipo clase

La instrucción

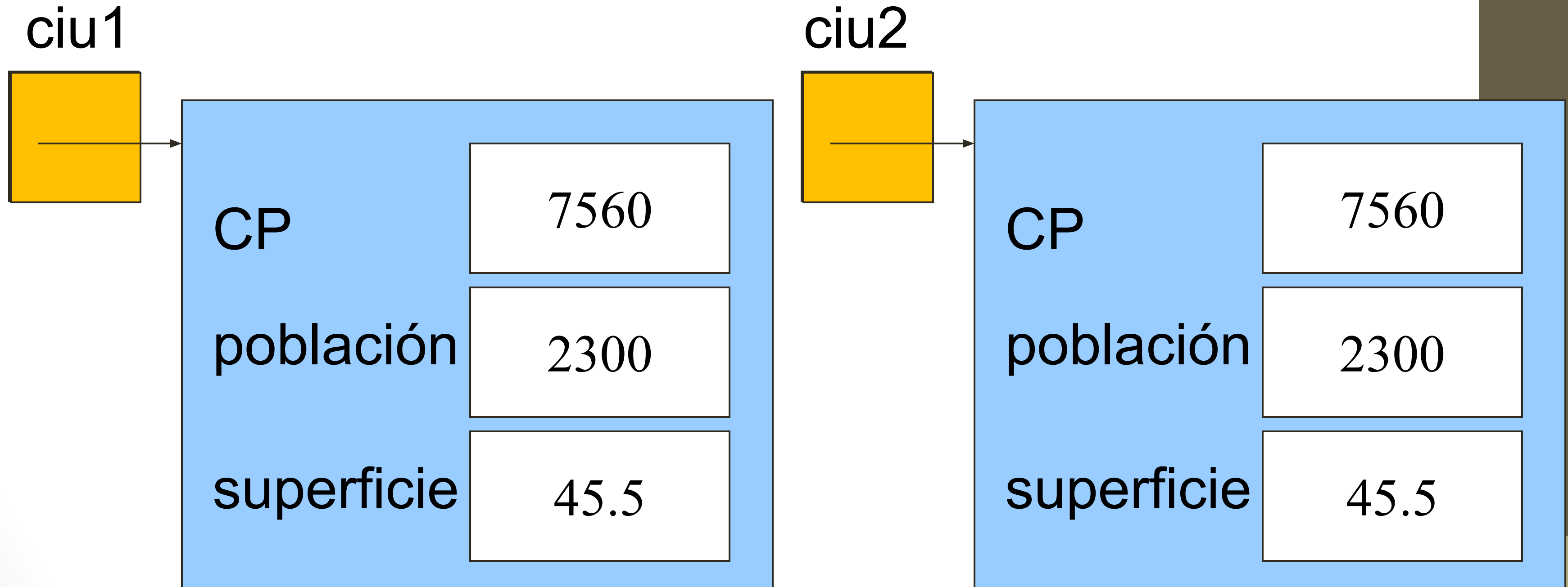
```
s.aumentarPoblacion(600);
```

Envía el mensaje **aumentarPoblacion(600)** al objeto ligado a la variable s.

En respuesta a este mensaje el objeto ejecuta el método que corresponde a su clase.

equals, copy, clone

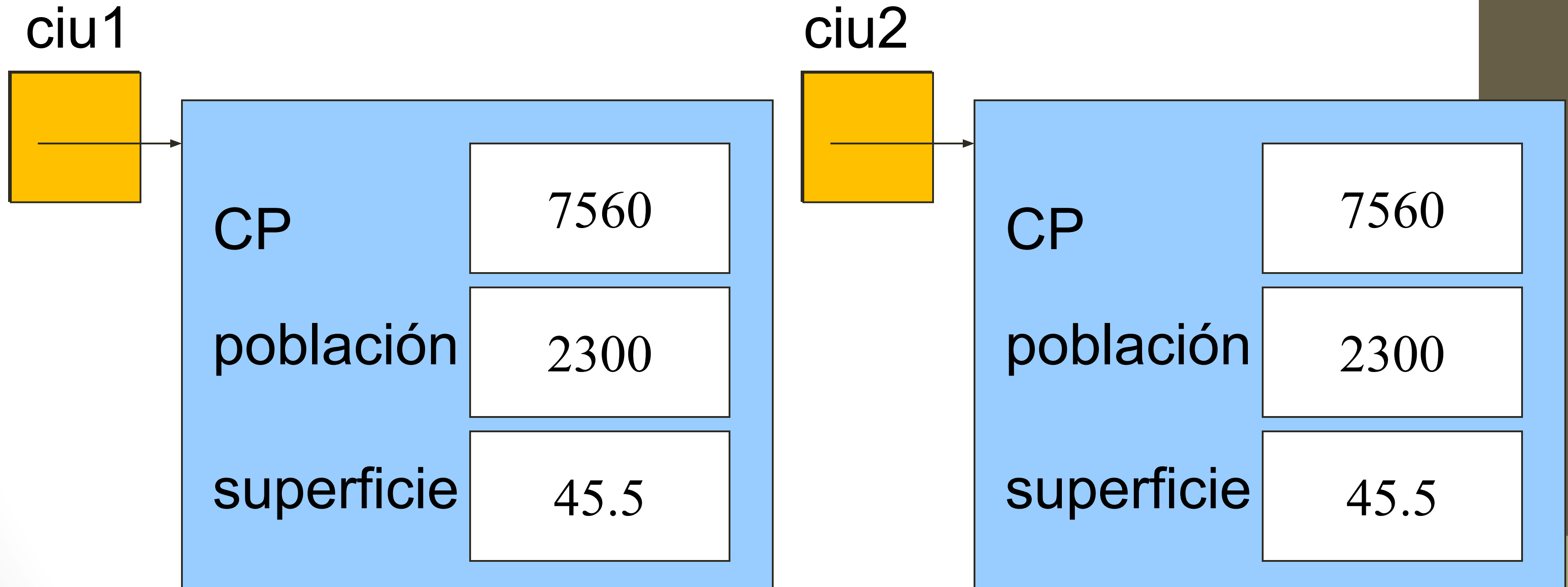
```
Ciudad ciu1,ciu2;
ciu1=new Ciudad(7560,2300,45.5);
ciu2=new Ciudad(7560,2300,45.5);
```




```
If (ciu1==ciu2)
```

equals, copy, clone

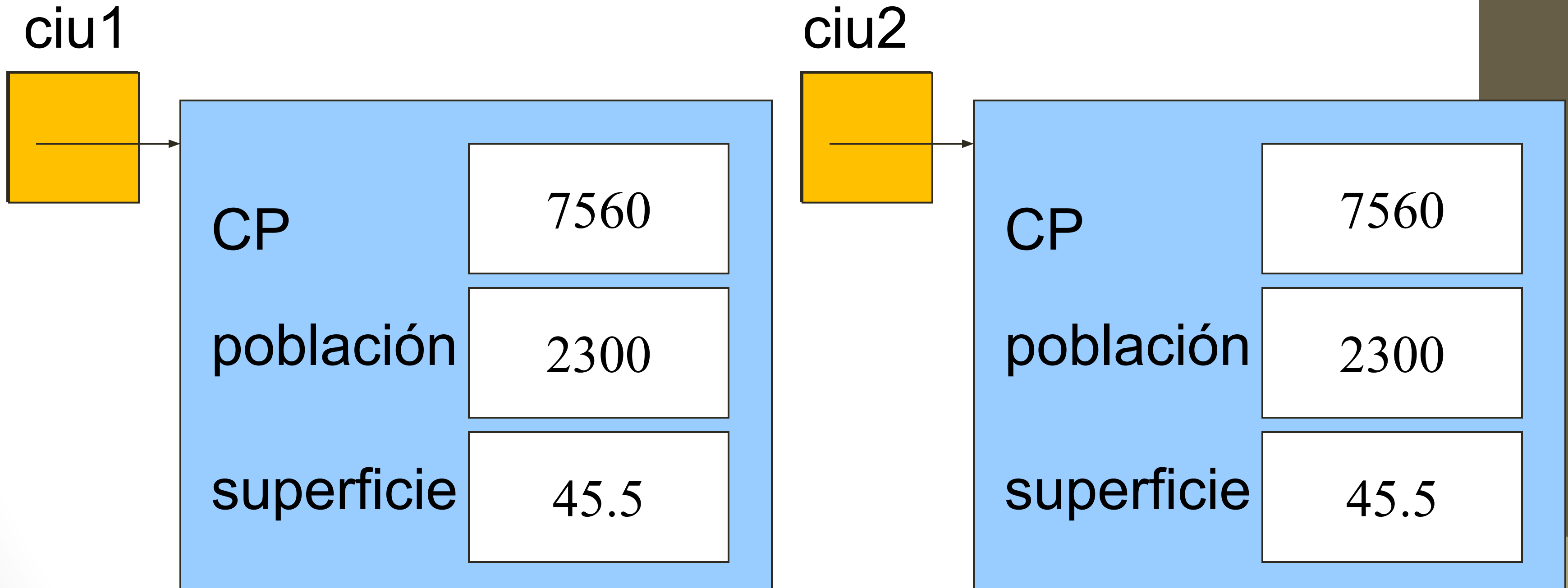
```
Ciudad ciu1,ciu2;
ciu1=new Ciudad(7560,2300,45.5);
ciu2=new Ciudad(7560,2300,45.5);
```




```
If (ciu1==ciu2) false
```

equals

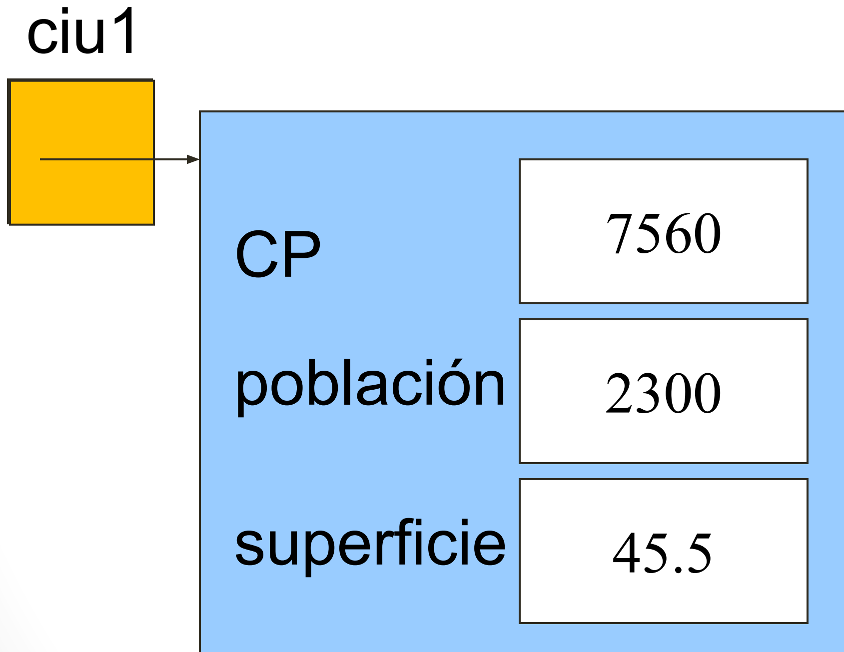
```
public  boolean  equals (Ciudad otra) {

  return  (CP == otra.obtenerCP() &&
     poblacion== otra.obtenerPoblacion()&&
    superficie== otra.obtenerSuperficie());}
```

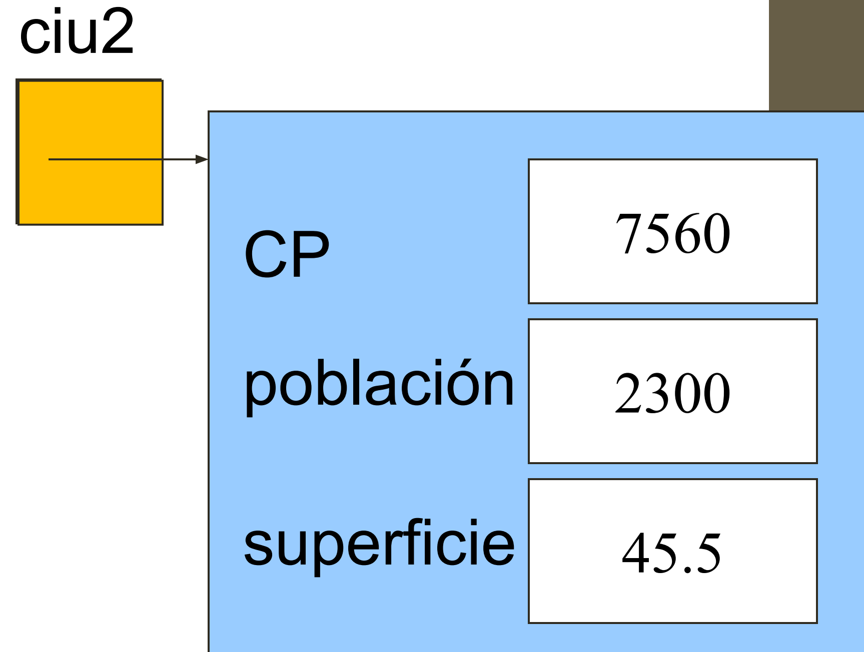



```
If (ciu1.equals (ciu2))
```

equals

```
public  boolean  equals (Ciudad otra) {

  return  (CP == otra.obtenerCP()  &&
     poblacion== otra.obtenerPoblacion()&&
    superficie== otra.obtenerSuperficie());}
```

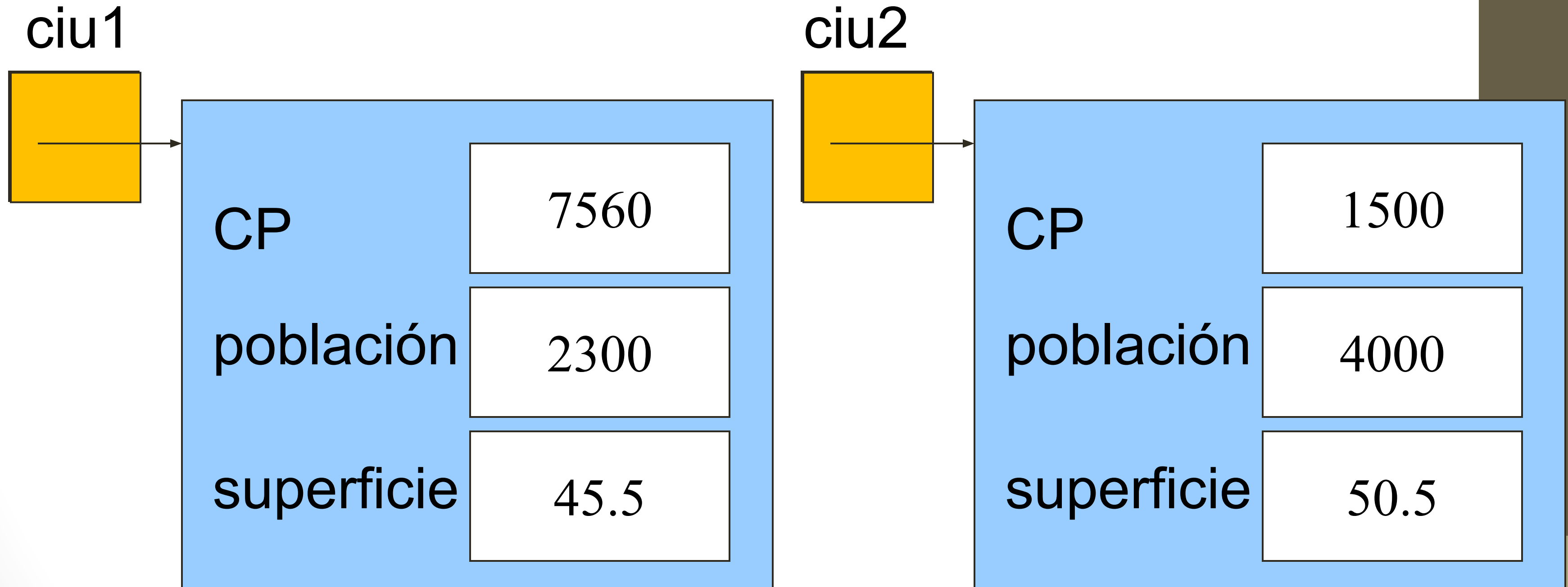





```
If (ciu1.equals (ciu2))    true
```

copy

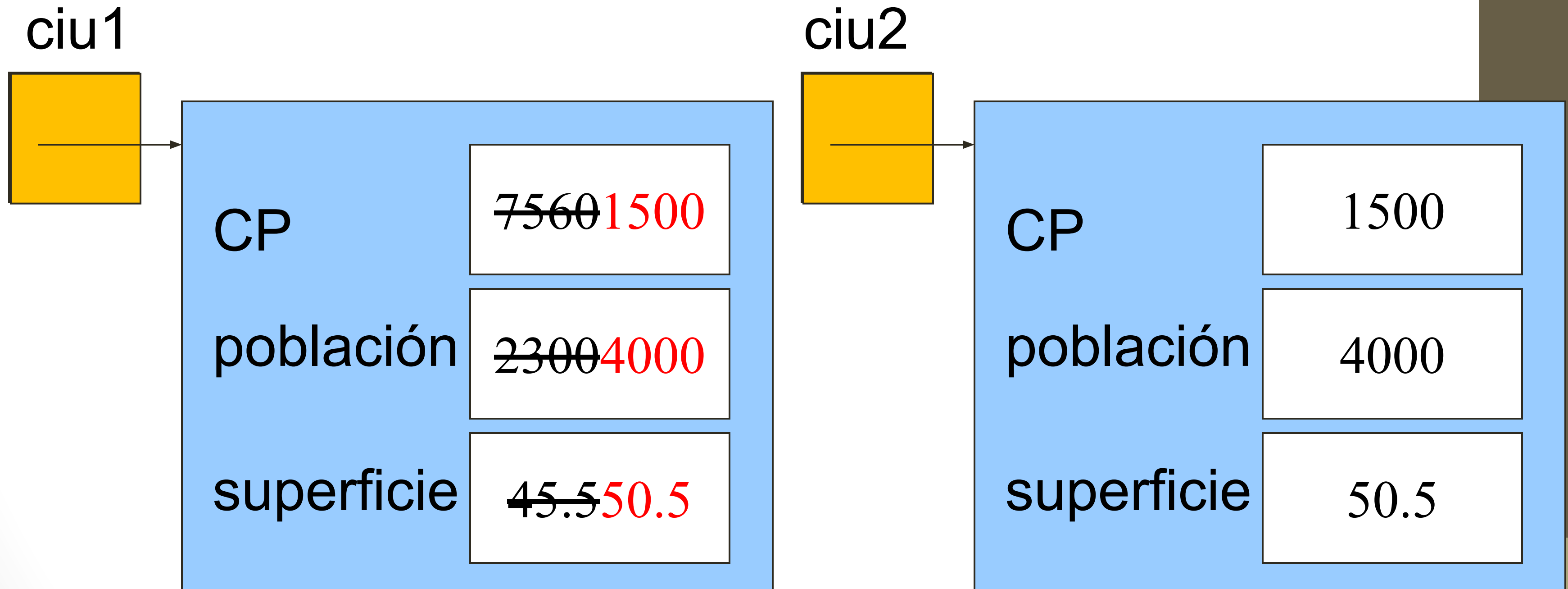
```
public  void  copy (Ciudad otra) {

     CP = otra.obtenerCP();
     poblacion=otra.obtenerPoblacion();
     superficie= otra.obtenerSuperficie(); }
```




```
ciu1.copy(ciu2);
```

copy

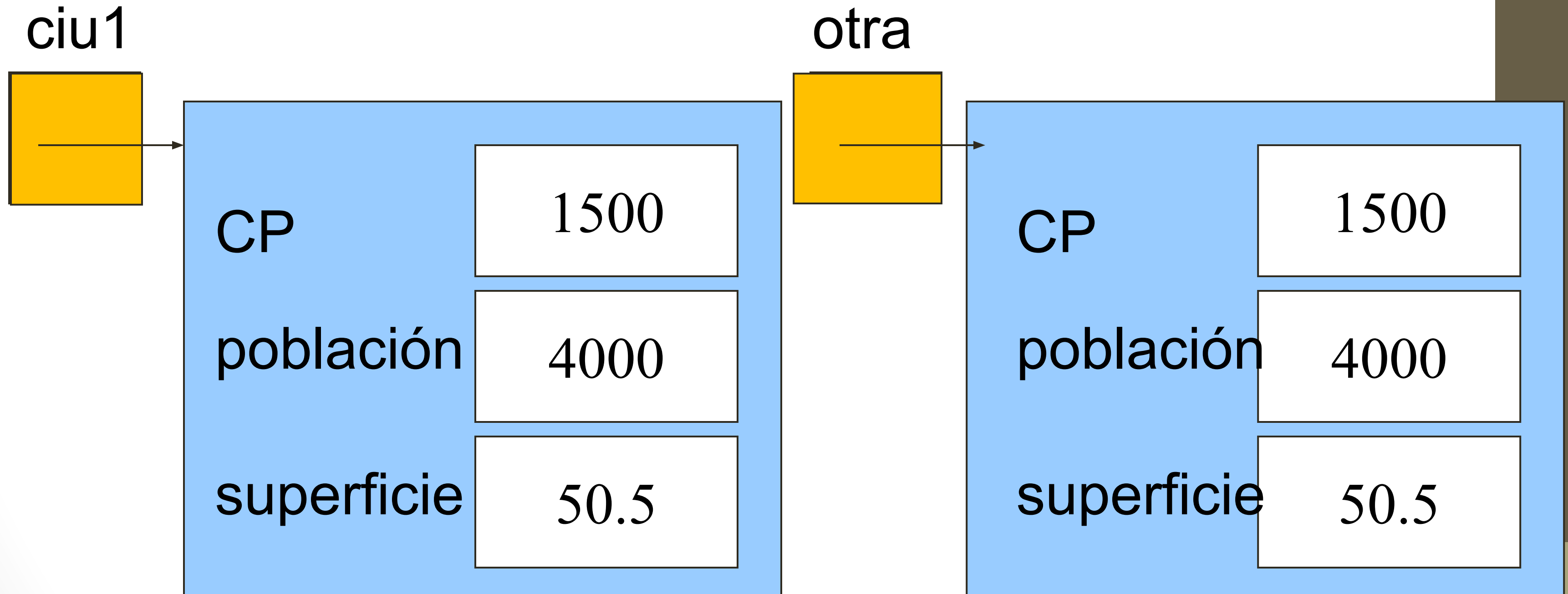
```
public  void  copy (Ciudad otra) {

     CP = otra.obtenerCP();
     poblacion=otra.obtenerPoblacion();
     superficie= otra.obtenerSuperficie(); }
```




```
ciu1.copy(ciu2);
```

clone

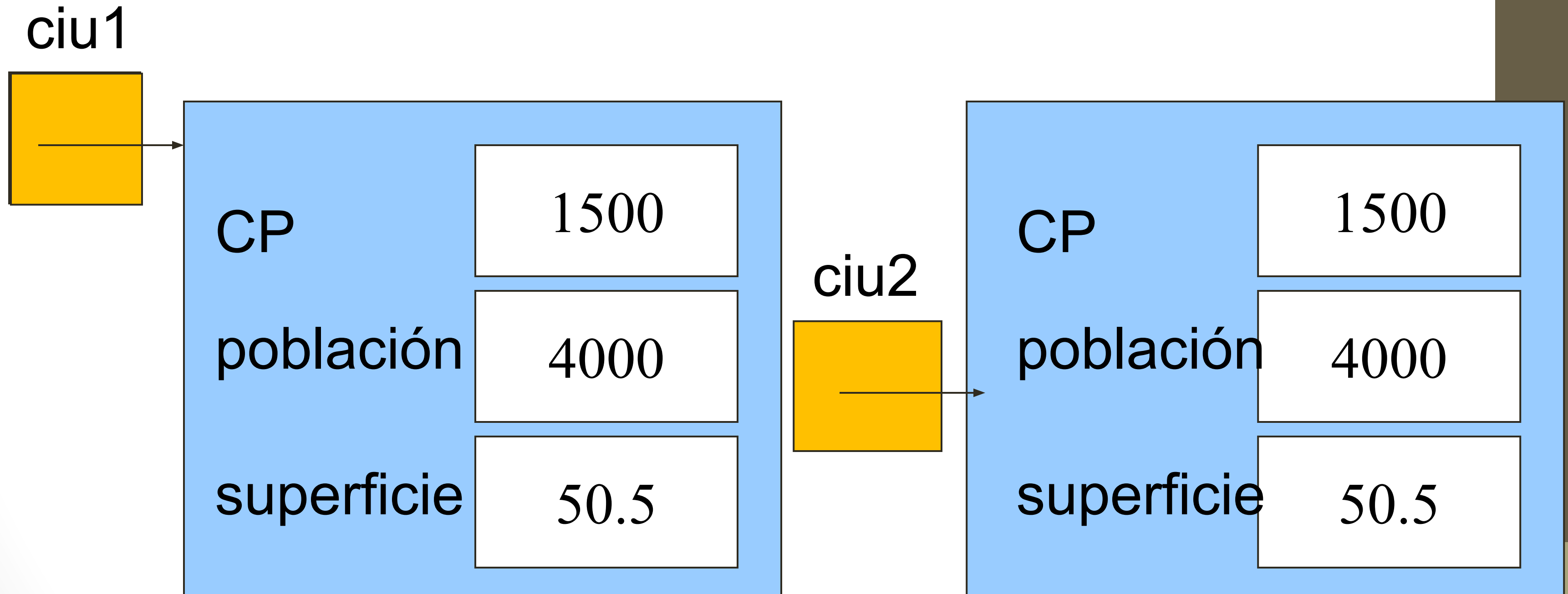
```
public  Ciudad  clone() {
 Ciudad otra;
 otra= new Ciudad(cp, poblacion, superficie);
 return (otra)}
```




```
ciu2 = ciu1.clone();
```

clone

```
public  Ciudad  clone() {
 Ciudad otra;
 otra= new Ciudad(cp, poblacion, superficie);
 return (otra)}
```




```
ciu2 = ciu1.clone();
```

Problemática

Qué sucede cuando alguno de los atributos de instancia de una clase **no es de tipo elemental**?

Cuando al menos uno de los atributos de instancia de una clase no es de tipo elemental, los métodos equals, clone se pueden implementar de dos maneras: **superficial** o en **profundidad**.

Por ejemplo, el equals se puede implementar:

- **Superficial**: en este caso, al momento de comparar los atributos de tipo no elemental (tipo clase), se compara su identidad (==).
- **Profundidad**: en este caso, al momento de comparar los atributos de tipo no elemental (tipo clase), se compara su equivalencia (equals).

Ejemplo

Ciudad

<<atributos de instancia>>
CP: entero
poblacion : entero
muni: Municipalidad

<<Constructores>>
Ciudad(cod :entero)
Ciudad(cod,p:entero,m:Municipalidad)
<<Comandos>>
//setters triviales
aumentarPoblacion(p:entero)
copy(c:Ciudad)
<<Consultas>>
//getters triviales
clone():Ciudad
equals(c:Ciudad):Boolean

Municipalidad

<<atributos de instancia>>
id: entero
cantEmpleados : entero

<<Constructores>>
Municipalidad(id :entero)
<<Comandos>>
//setters triviales
copy(m:Municipalidad)
<<Consultas>>
//getters triviales
clone():Municipalidad
equals(m:Municipalidad):Boolean

Ejemplo

En este caso la clase **Municipalidad** se implementa de la misma manera en la que veníamos trabajando (observar que todos sus atributos son de tipo elemental).

En el caso de la clase **Ciudad**, ahora tenemos un atributo que es de tipo Municipalidad (tipo clase), así que los métodos clone, copy e equals se pueden implementar de dos maneras: **superficial** o en **profundidad**.

Asumamos que la clase **Municipalidad** tiene sus métodos clone, copy e equals ya implementados.

Ejemplo - equals

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
    //<<consultas>>
    public boolean equals (Ciudad otra) {
        return CP == otra.obtenerCP() &&
                    poblacion==otra.obtenerPoblacion()&&
               muni.equals(otra.obtenerMunicipalidad());
    }
    ...
}
```

Implementación en profundidad.

Ejemplo - equals

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
    //<<consultas>>
    public boolean equals (Ciudad otra) {
        return CP == otra.obtenerCP() &&
                poblacion==otra.obtenerPoblacion()&&
            muni.equals(otra.obtenerMunicipalidad());
    }
    ...
}
```

id: 127
cantEmpleados: 750

id: 127
cantEmpleados: 750

ciu1

CP: 8000
poblacion : 10.000
muni:

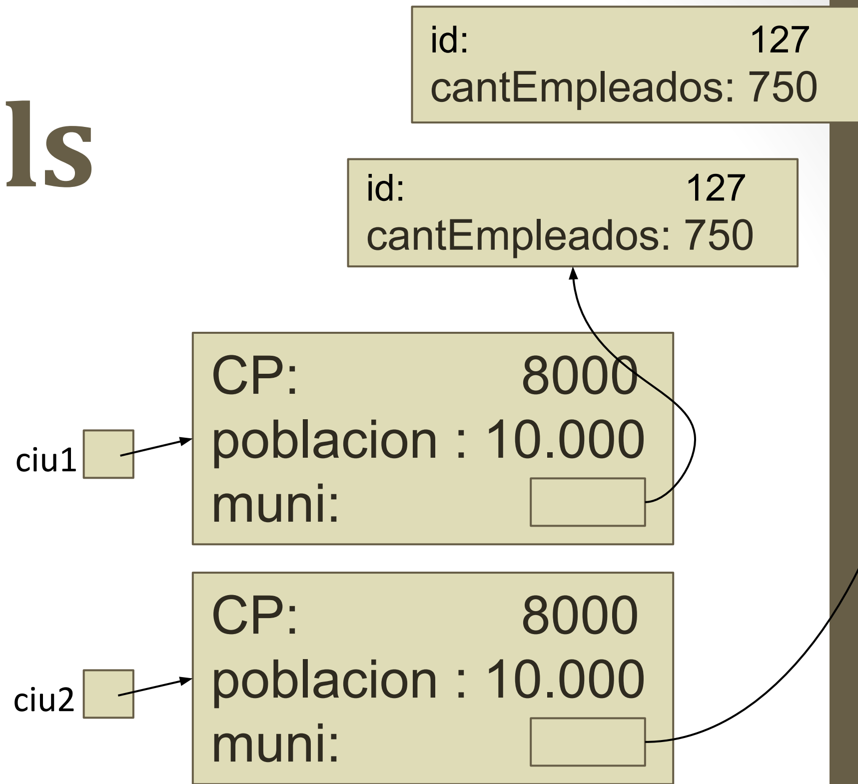
ciu2

CP: 8000
poblacion : 10.000
muni:

Implementación en profundidad.

```
ciu1.equals(ciu2)
```

Ejemplo - equals




```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
    //<<consultas>>
    public boolean equals (Ciudad otra) {
        return CP == otra.obtenerCP() &&
                poblacion==otra.obtenerPoblacion()&&
            muni.equals(otra.obtenerMunicipalidad());
    }
    ...
}
```

Implementación en profundidad.

```
ciu1.equals(ciu2)          true
```

Ejemplo - equals

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
    //<<consultas>>
    public boolean equals (Ciudad otra) {
        return CP == otra.obtenerCP() &&
                 poblacion==otra.obtenerPoblacion()&&
               muni == otra.obtenerMunicipalidad();
    }
    ...
}
```

id: 127
cantEmpleados: 750

id: 127
cantEmpleados: 750

ciu1

CP: 8000
poblacion : 10.000
muni:

ciu2

CP: 8000
poblacion : 10.000
muni:

Implementación superficial.

```
ciu1.equals(ciu2)
```

Ejemplo - equals

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
    //<<consultas>>
    public boolean equals (Ciudad otra) {
        return CP == otra.obtenerCP() &&
                poblacion==otra.obtenerPoblacion()&&
            muni == otra.obtenerMunicipalidad();
    }
    ...
}
```

id: 127
cantEmpleados: 750

id: 127
cantEmpleados: 750

ciu1

CP: 8000
poblacion : 10.000
muni:

ciu2

CP: 8000
poblacion : 10.000
muni:

Implementación superficial.

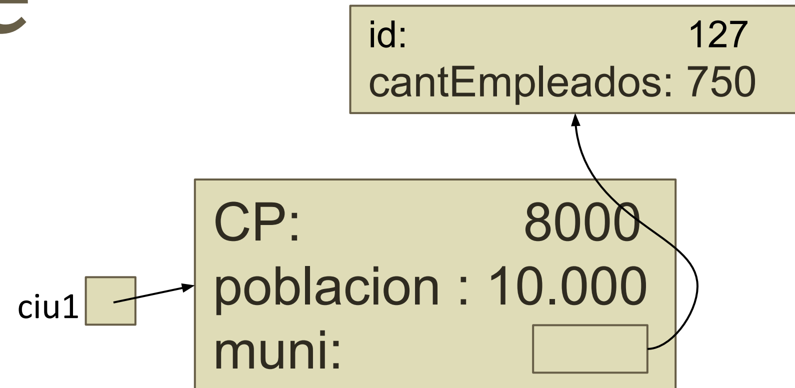
`ciu1.equals(ciu2)` false

Ejemplo - clone

```
Public class Ciudad{
     private int CP;
     private int poblacion;
     private Municipalidad muni;
     ...
    //<<consultas>>
    public Ciudad clone () {
        return new Ciudad(CP, poblacion,muni.clone());
    }
    ...
}
```

Implementación en profundidad.

Ejemplo - clone



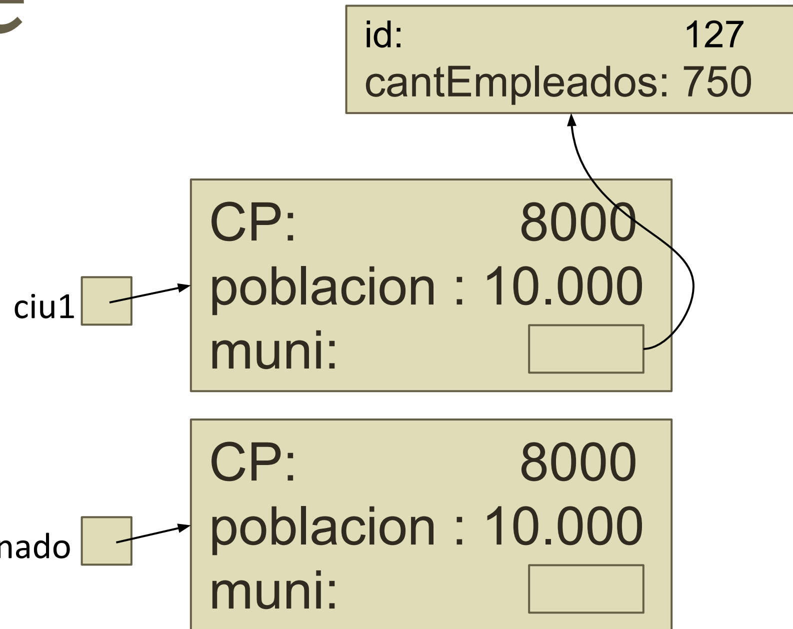

```
Public class Ciudad{
     private int CP;
     private int poblacion;
     private Municipalidad muni;
     ...
    //<<consultas>>
    public Ciudad clone () {
        return new Ciudad(CP, poblacion,muni.clone());
    }
    ...
}
```

Implementación en profundidad.

```
ciu_clonado = ciu1.clone()
```

Ejemplo - clone

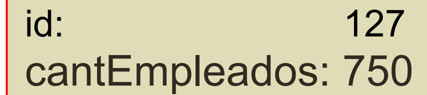
```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
   //<<consultas>>
   public Ciudad clone () {
      return new Ciudad(CP, poblacion,muni.clone());
   }
   ...
}
```

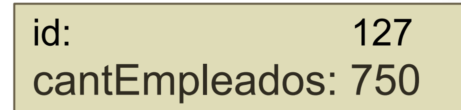


Implementación en profundidad.

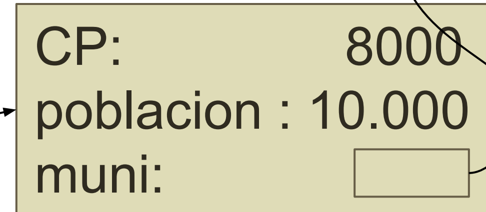
```
ciu_clonado = ciu1.clone()
```

Ejemplo - clone

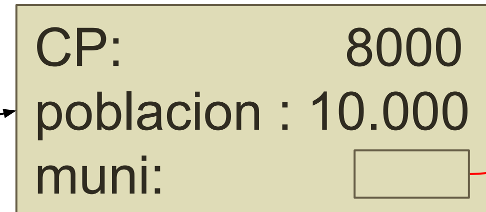




ciu1



ciu_clonado



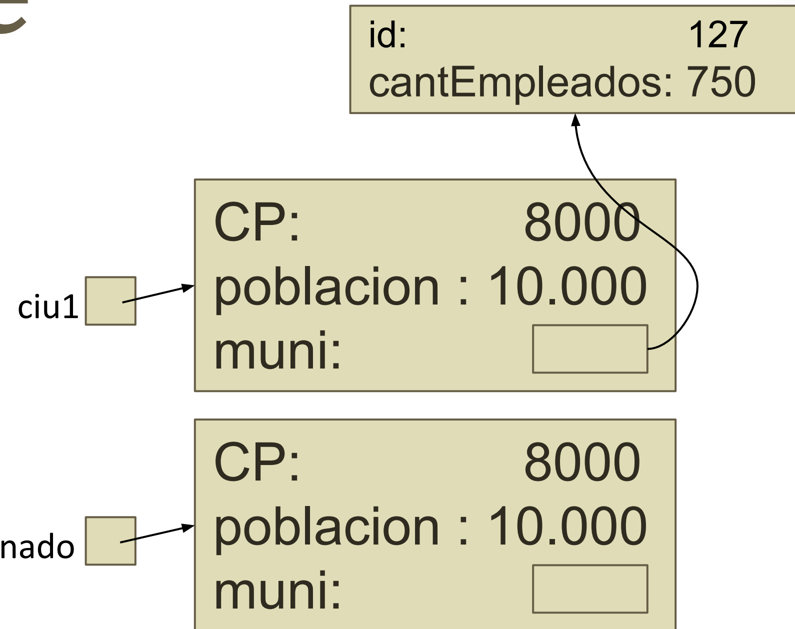

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
   //<<consultas>>
   public Ciudad clone () {
       return new Ciudad(CP, poblacion,muni.clone());
   }
   ...
}
```

Implementación en profundidad.

```
ciu_clonado = ciu1.clone()
```

Ejemplo - clone

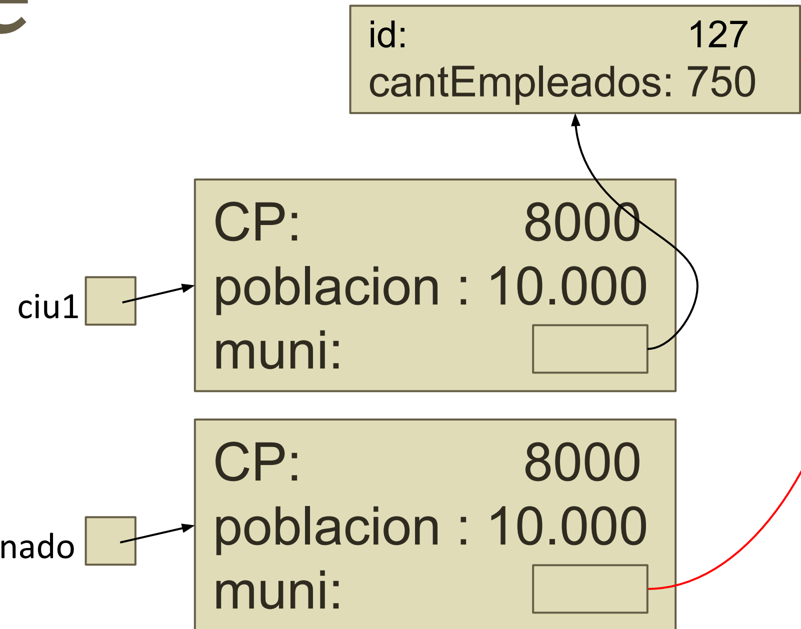
```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
   //<<consultas>>
   public Ciudad clone () {
      return new Ciudad(CP, poblacion,muni);
   }
   ...
}
```



Implementación superficial.

```
ciu_clonado = ciu1.clone()
```

Ejemplo - clone

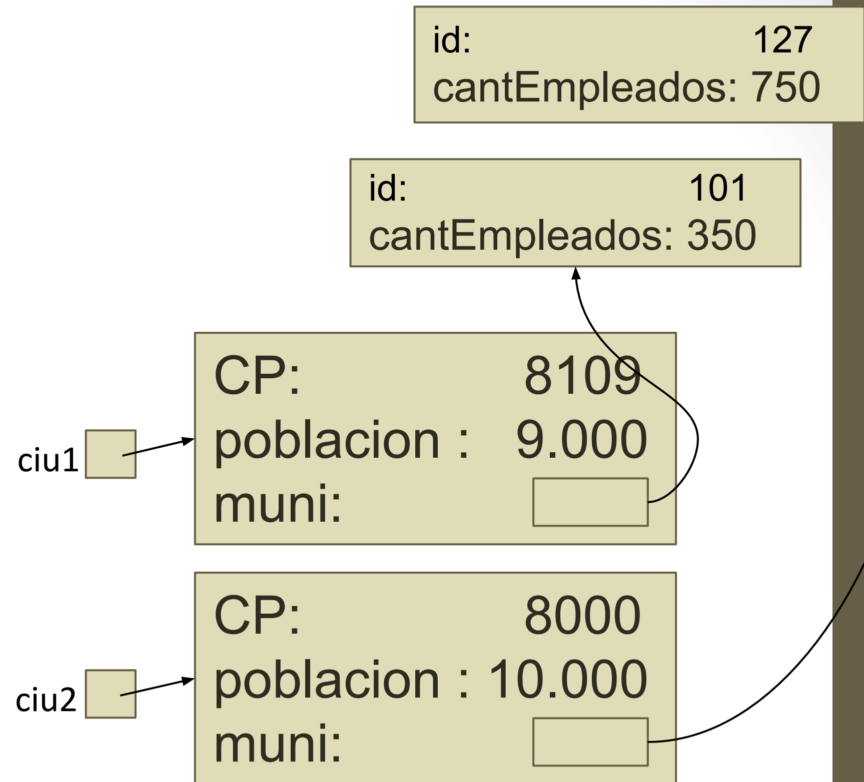



```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    …
   //<<consultas>>
   public Ciudad clone () {
      return new Ciudad(CP, poblacion,muni);
   }
   …
}
```

Implementación superficial.

```
ciu_clonado = ciu1.clone()
```

Ejemplo - copy

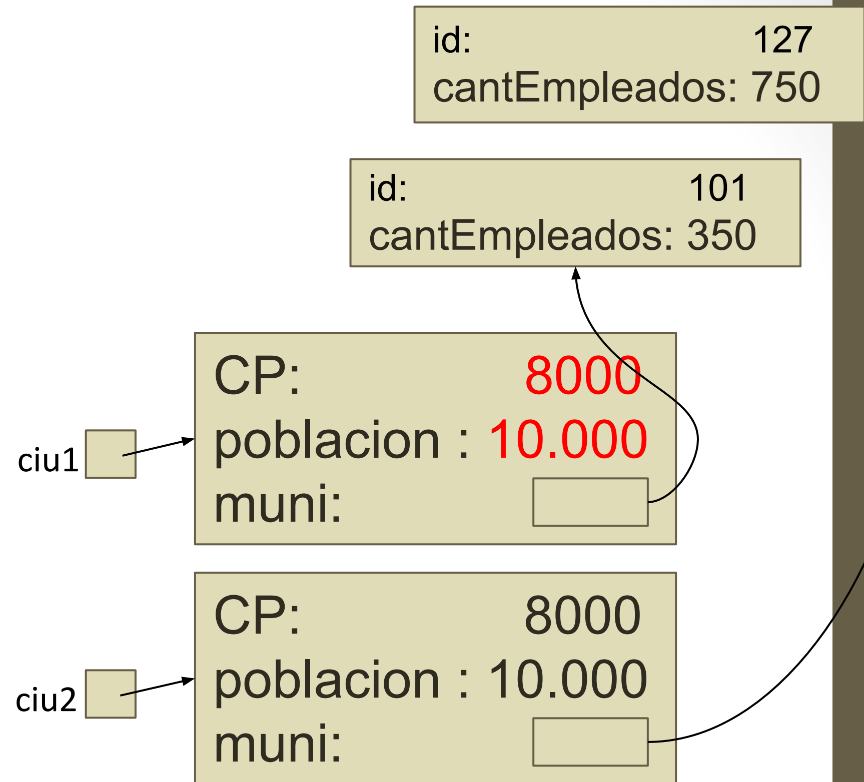



```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    …
    //<<comandos>>
    public void copy (Ciudad otra) {
            CP= otra.obtenerCP();
            poblacion=otra.obtenerPoblacion();
            muni.copy(otra.obtenerMunicipalidad());
    }
    …
}
```

Implementación en profundidad.

```
ciu1.copy(ciu2)
```

Ejemplo - copy



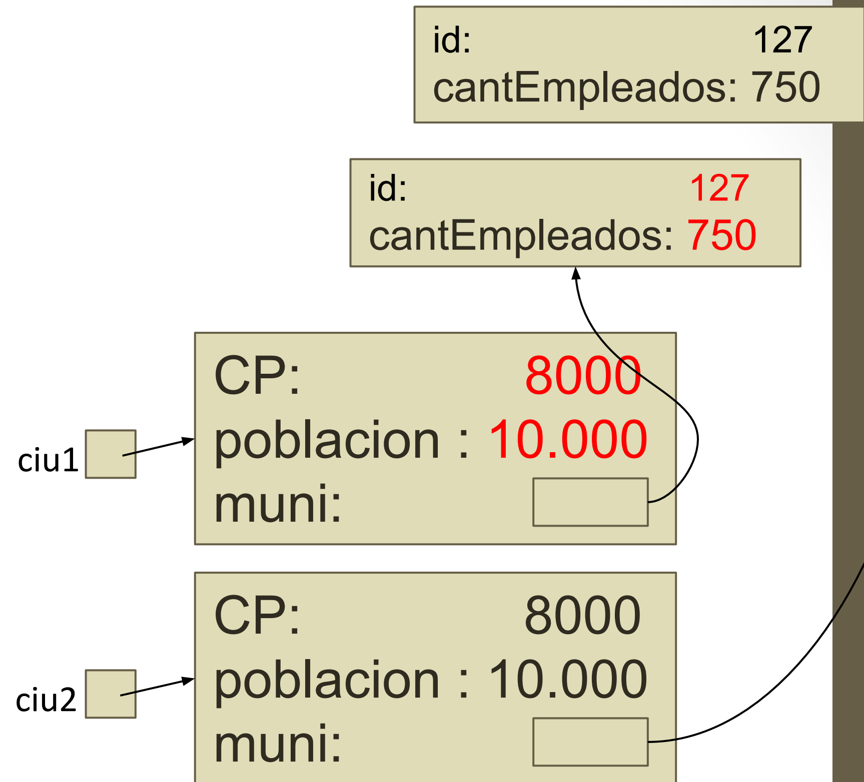

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    …
    //<<comandos>>
    public void copy (Ciudad otra) {
            CP= otra.obtenerCP();
            poblacion=otra.obtenerPoblacion();
            muni.copy(otra.obtenerMunicipalidad());
    }
    …
}
```

Implementación en profundidad.

`ciu1.copy(ciu2)`

Ejemplo - copy

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    ...
    //<<comandos>>
    public void copy (Ciudad otra) {
            CP= otra.obtenerCP();
            poblacion=otra.obtenerPoblacion();
            muni.copy(otra.obtenerMunicipalidad());
    }
    ...
}
```

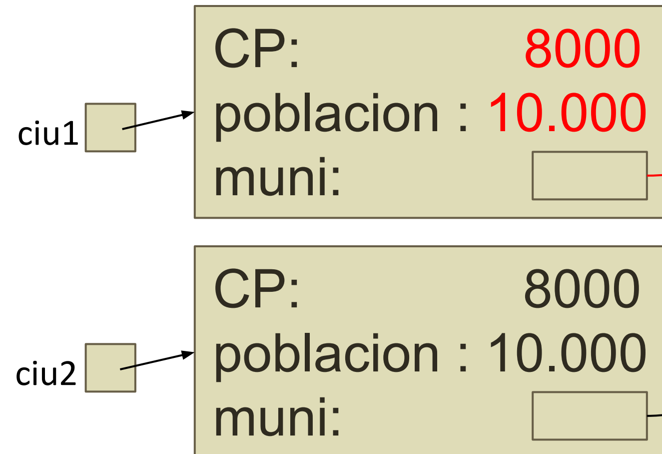


Implementación en profundidad.

```
ciu1.copy(ciu2)
```

Ejemplo - copy

```
Public class Ciudad{
    private int CP;
    private int poblacion;
    private Municipalidad muni;
    …
    //<<comandos>>
    public void copy (Ciudad otra) {
            CP= otra.obtenerCP();
            poblacion=otra.obtenerPoblacion();
            muni = otra.obtenerMunicipalidad();
    }
    …
}
```



Implementación superficial.

`ciu1.copy(ciu2)`

Resumen

- Al momento de comparar hay que tener en cuenta:

  - El operador **==** se utiliza para comparar igualdad de tipos elementales (char, boolean, int,float);

  - Si se utiliza el operador **==** entre dos variables de tipo clase el resultado sólo será **verdadero si ambas variables referencian al mismo objeto.**

  - Si se desea comparar la **igualdad entre dos objetos** se debe hacer mediante el método **equals** que **compara campo a campo el valor de sus atributos de instancia.**