

Punteros y referencias en Java

PUNTERO

Un puntero es una referencia a una posición de la memoria, donde hay un valor concreto.

Object o = new Object();

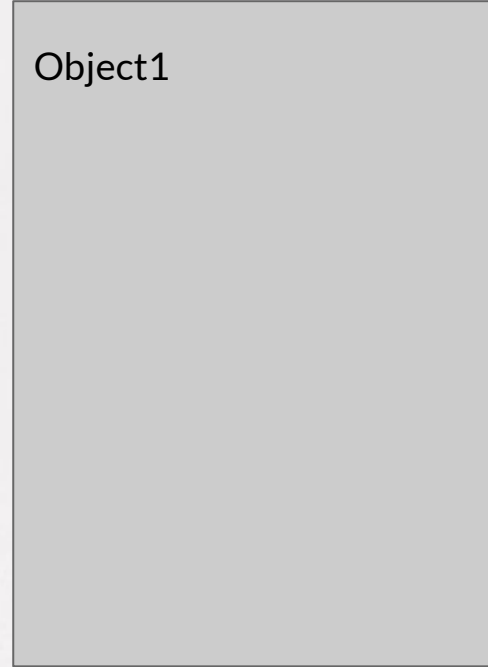
VARIABLES

O



MEMORIA

Object1



```
Object o = new Object()  
Object b = o
```

VARIABLES

MEMORIA

o

b



Object1

¿Qué pasaría si haces esto con tipos de datos primitivos, es decir sin usar el new?

```
Object o = new Object()  
Object b = new Object()  
O.next = b;
```


VARIABLES

o →
o.next →
b →

MEMORIA

Object1

Object 2

¿Y ESTO QUÉ NOS PERMITE HACER?

Crear estructuras de datos relacionadas entre sí.

```
public class LetraEnlazada {  
    public LetraEnlazada (char  
    letra) {  
        this.letra = letra;  
    }  
  
    char letra;  
  
    LetraEnlazada siguiente;  
}
```

```
LetraEnlazada palabra = new LetraEnlazada('h');  
palabra.siguiente = new LetraEnlazada('o');  
palabra.siguiente.siguiente = new LetraEnlazada('l');  
palabra.siguiente.siguiente.siguiente = new  
LetraEnlazada('a');
```

LetraEnlazada

Letra = 'h'

Siguiente =

LetraEnlazada

Letra = 'o'

Siguiente =

LetraEnlazada

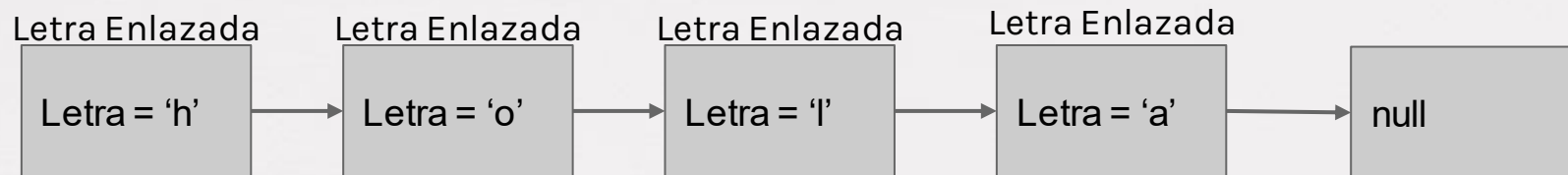
Letra = 'l'

Siguiente =

LetraEnlazada

Letra = 'a'

Siguiente = null



```
LetraEnlazada letra = palabra;  
while (letra != null) {  
    System.out.print(letra.letra);  
    letra = letra.siguiente;  
}
```

PASO POR REFERENCIA EN FUNCIONES

Técnicamente no existe, tu pasas un valor, el valor a la referencia a un objeto en memoria


```
public static void main(String args[]) {  
    ArrayList<String> lista = new ArrayList<String>();  
    System.out.println(lista.size()); // 0  
    addToList(lista);  
    System.out.println(lista.size()); // 1  
}  
  
public static void addToList(ArrayList<String> lista){  
    lista.add("hola");  
}
```

```
public static void main(String[] args) {  
    Dog dog = new Dog("Max");  
    Dog oldDog = dog;
```

```
    foo(dog);  
    // dog aún está apuntando al perro Max  
    dog.getName().equals("Max"); // true  
    dog.getName().equals("Fifi"); // false  
    dog == oldDog; // true  
}
```

```
public static void foo(Dog d) {  
    d.getName().equals("Max"); // true  
    // Cambiamos el valor del puntero, no el valor al que apunta  
    // el puntero  
    d = new Dog("Fifi");  
    d.getName().equals("Fifi"); // true  
}
```

VARIABLES

dog
foo(Dog d)
foo.d

MEMORIA



The diagram illustrates the relationship between variables and memory. On the left, under the heading 'VARIABLES', are three identifiers: 'dog', 'foo(Dog d)', and 'foo.d'. On the right, under the heading 'MEMORIA', is a large gray rectangular box representing memory. Inside the top-left corner of this box is the text 'Object1'. Two arrows originate from the variable names: one from 'dog' and one from 'foo.d', both pointing to the 'Object1' label within the memory box. The 'foo(Dog d)' variable has no arrow pointing to it.

Object1

```
public static void foo(Dog d) {  
    d.getName().equals("Max"); // true  
    // Cambiamos el valor del puntero, no el valor al que apunta  
    // el puntero  
    d = new Dog("Fifi");  
    d.getName().equals("Fifi"); // true  
}
```

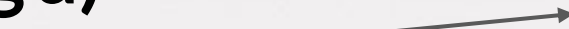
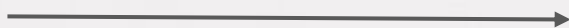
VARIABLES

dog
foo(Dog d)
foo.d

MEMORIA

Object1

Object2



¡GRACIAS!

¿ALGUNA PREGUNTA?