

**Bloque I:**

***Introducción a la Programación***

***1.3 Variables y tipos en Java***

*1r DAW Semipresencial*

*Programación*

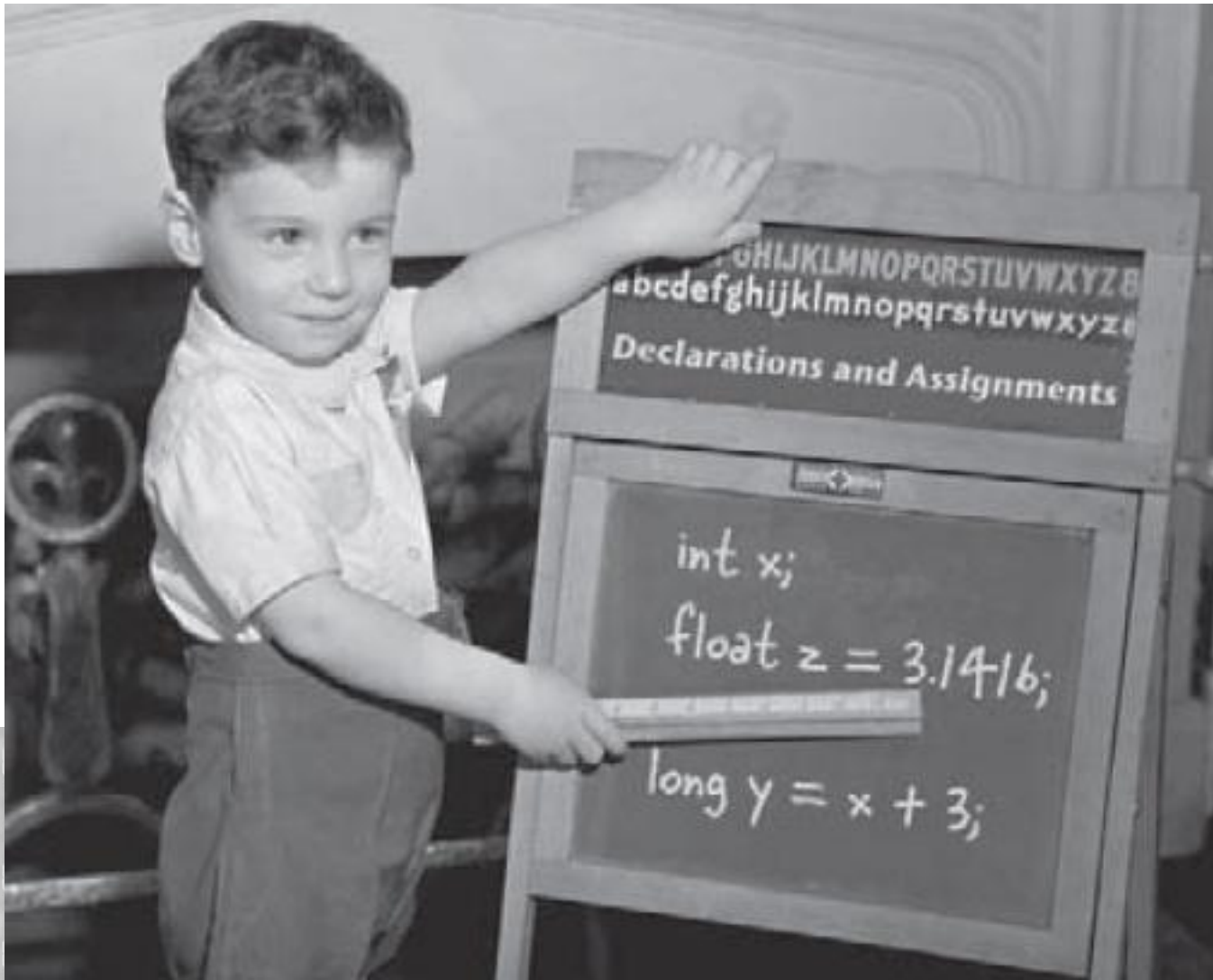
*Maria José Lozano Pérez*

*IES Pere Maria Orts i Bosch*

# ***Tema 1.3: Variables y tipos en Java***

- ▣ Declaración de variables. Tipos.
- ▣ Tipos de primitivas
- ▣ Palabras clave en Java
- ▣ Variables de referencia
- ▣ Declaración y asignación de objetos
- ▣ Objetos en el montón de basura
- ▣ Una primera mirada a los arrays
- ▣ Ejemplos y prácticas

# Variables y tipos



# Declarar una variable

- Java se preocupa por los **tipos** de las variables

- Si hacemos esto:

```
Conejo c = new Jirafa();
```

- No esperes a que funcione

- Java siempre obliga a definir el tipo de cualquier variable



# Declarar una variable

Se han de cumplir **2 reglas**:

Las variables deben tener un ***tipo***

Las variables deben tener un ***nombre***

`int count;`

↑  
tipo

←  
nombre

Podrán ser definidas en cualquier punto del programa, no necesariamente al inicio.

# Las variables y sus tipos

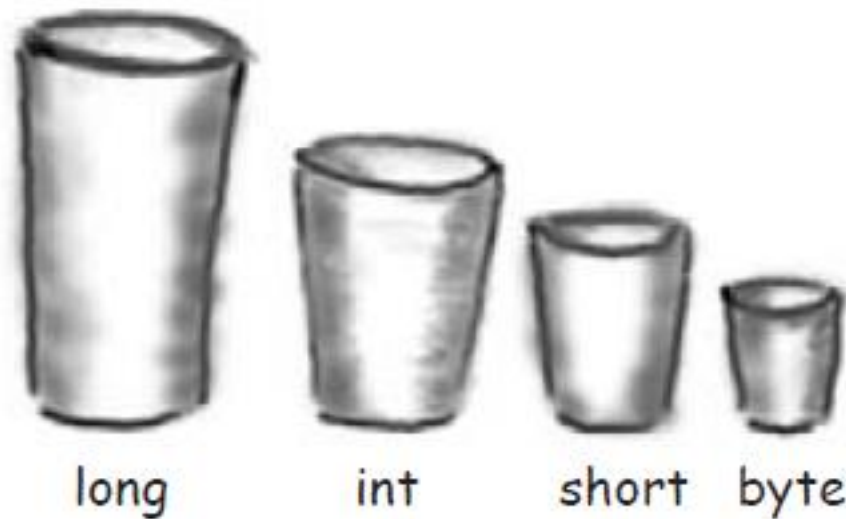
- Una variable es como una **taza**. Un contenedor, que contiene algo.
- Tiene un **tamaño** y un **tipo**, como las tazas de café de una cafetería:





# Las variables y sus tipos

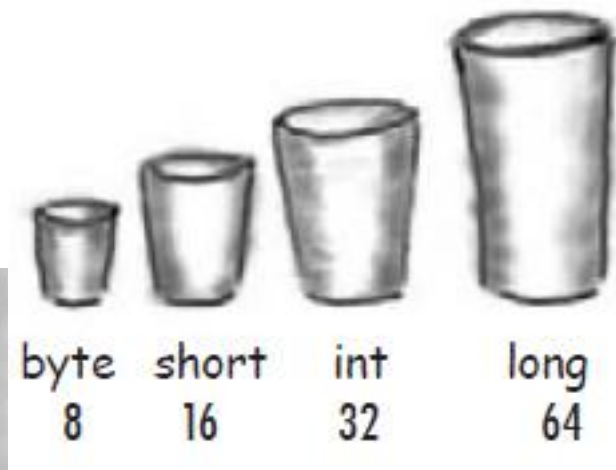
■ Aquí tenemos los 4 tamaños de **variables enteras** en Java, de mayor a menor tamaño:



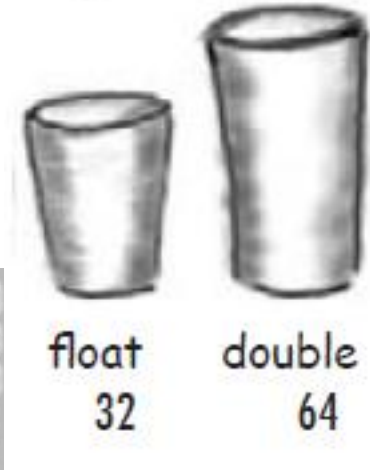
■ Igual que pedimos un **café solo corto** en la cafetería, a Java le pedimos una variable **entera** con el valor **25**, con el nombre **contador**.

# Las variables y sus tamaños

- ❏ Cada variable en Java ocupa un *nº fijo de bits* o tamaño de la taza.
- ❏ Estas son las **6 primitivas numéricas de Java** y sus tamaños (*enteras y decimales*):



**Variables de tipo entero**



**Variables de tipo decimal**



# Primitivas: sus tipos y ejemplos

## Tipos de primitivas

Tipo      nº de bit      Rango de valores

### boolean y char

boolean (JVM-specific) **true** o **false**

char      16 bits      0 a 65535

### numeric (todos con signo)

#### enteros

byte      8 bits      -128 a 127

short      16 bits      -32768  
a 32767

int      32 bits      -2147483648  
a 2147483647

long      64 bits      -enorme a enorme

#### decimales

float      32 bits      **varía**

double      64 bits      **varía**

## Declaración y asignación de primitivas

```
int x;
```

```
x = 234;
```

```
byte b = 89;
```

```
boolean isFun = true;
```

```
double d = 3456.98;
```

```
char c = 'f';
```

```
int z = x;
```

```
boolean isPunkRock;
```

```
isPunkRock = false;
```

```
boolean powerOn;
```

```
powerOn = isFun;
```

```
long big = 3456789;
```

```
float f = 32.5f;
```

Para indicar que se trata de un float, porque Java si encuentra un punto decimal, se piensa que es un doble, a no ser que encuentre la 'f'.

# ¿Qué pasa si el valor es mayor?

- ❑ No podemos poner un valor grande en una taza pequeña
- ❑ Sí podemos, pero se derrama.

❑ Por ejemplo:

```
int x = 24;
```

```
byte b = x;
```

```
// no funciona!!
```



❑ El valor de x no será el exacto

❑ Se perderán bits y precisión

# Asignación de variables

## Casos posibles:

 Asignando un valor literal detrás del signo =

```
x = 12;
```

```
isGood = true;
```

 Asignando el valor de una variable a otra:

```
x = y;
```

 Usando una expresión combinación de las dos:

```
x = y + 43;
```

# Asignación de variables: Ejemplos

```
int size = 32;
```

declara un entero llamado **size** asignándole el valor 32

```
char initial = 'j';
```

declara un carácter llamado **initial** asignándole el valor 'j'

```
double d = 456.709;
```

declara un doble llamado **d** asignándole el valor 456.709

```
boolean isCrazy;
```

declara un boolean llamado **isCrazy** sin asignarle valor

```
isCrazy = true;
```

asigna el valor true a la variable **isCrazy**, antes definida

```
int y = x + 456;
```

declara un entero llamado **y** y le asigna el valor de la suma del valor de x más el valor 456

# Práctica P3.0: Haz de compilador

De las siguientes asignaciones, indica cuáles son legales y el compilador las dará por buenas y cuáles no.

Primero, piensa qué pasará.

Después pruébalo, para confirmar el compilador qué hace realmente.

Explícalo caso por caso, mediante un programa, informando con cuadros de diálogo gráficos.

Se ha publicado un ejemplo.



¡Afila tu lápiz!

```
1. int x = 34.5;
2. boolean boo = x;
3. int g = 17;
4. int y = g;
5. y = y + 10;
6. short s;
7. s = y;
8. byte b = 3;
9. byte v = b;
10. short n = 12;
11. v = n;
12. byte k = 128;
```



# Variables y palabras reservadas

❏ ¿Qué nombres pueden tener las variables?

❏ Debe comenzar con una **letra**, subrayado (\_) o el carácter \$. No puede comenzar con un número.

❏ Después del primer carácter, sí que se pueden usar números. Pero el primero, no.

❏ No debe coincidir con ninguna **palabra reservada**

❏ El % no está permitido, sí el \$ y la ç



## Lista de palabras reservadas

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum



# Identificadores

- Los únicos caracteres especiales permitidos:
  - `_`, `$` y `ç`
- No permitidos: el resto, por ejemplo `%`
- Es ***case sensitive***: distingue entre mayúsculas y minúsculas
- Java permite **acentos** y la **ñ**, pero no es recomendable
- Para separar palabras, se utiliza:
  - `abrirPuerta()`
  - `cerrarPuerta()`

# *Variables referencia: Objetos*

- ❑ Las variables **objeto** realmente no existen
- ❑ Son variables que hacen referencia a objetos
- ❑ Éstas almacenan los bits que representan el camino para acceder al objeto
- ❑ No guardan el objeto en sí, pero sí guardan un puntero al objeto
- ❑ Sólo sabemos que apunta a un único objeto
- ❑ El **JVM** sabe cómo usar ese puntero para acceder al objeto

# Variables referencia: Objetos

```
Dog d = new Dog();  
d.bark();
```

piensa que esto  
es lo mismo  
que esto



Piensa que la  
variable de referencia  
Dog es como un control  
remoto a Dog.  
Lo usamos para que el  
objeto haga algo  
(invocando métodos)

# Variables referencia: Objetos

- ❑ Los objetos no se almacenan en tazas
- ❑ Viven en un lugar que es el **montón (heap)**
- ❑ Usamos el **operador punto (.)** con la variable de referencia, para decirle algo al objeto:

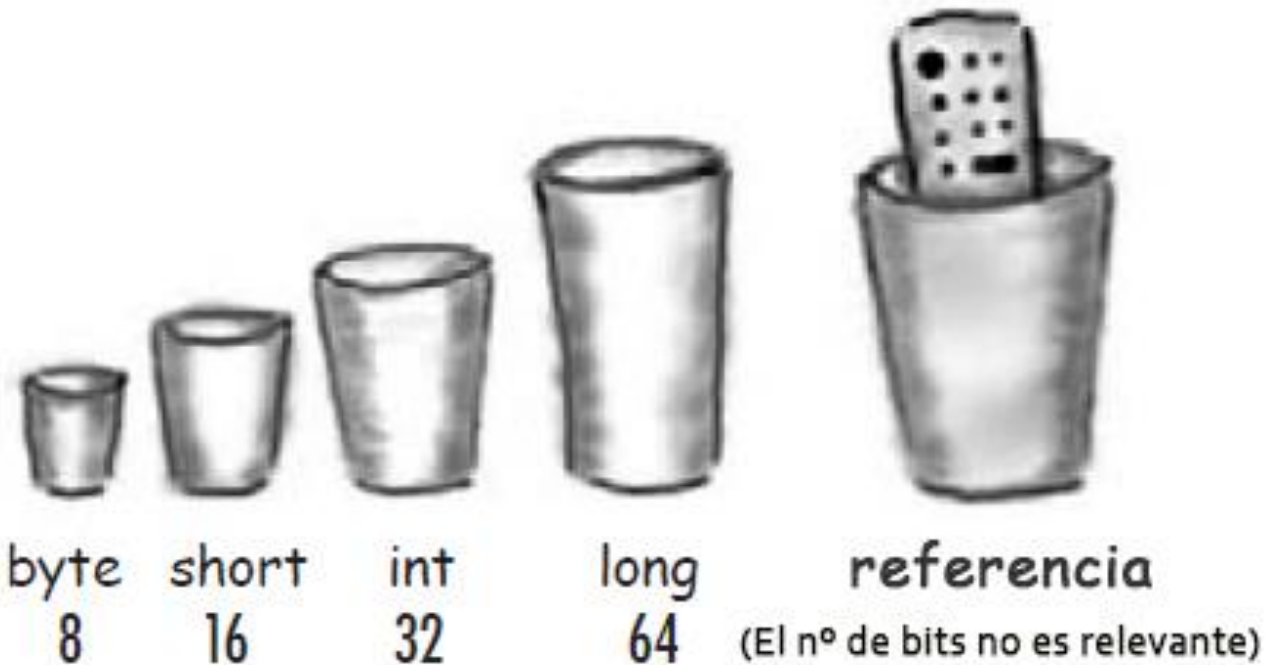
**myDog.bark ( ) ;**

que significa: *usa el objeto al que apunta la variable de referencia **myDog** para invocar al método **bark()**.*

- ❑ Usar el operador punto (.) es como pulsar el botón de un mando a distancia

# Variables referencia: Objetos

Una referencia a un objeto es otra variable:





# Variables referencia: Objetos

## Variable primitiva

```
byte x = 7;
```

Los bits representan al 7  
dentro de la variable  
(00000111)



## Variable de referencia

```
Dog myDog = new Dog();
```

Los bits representan la forma de llegar  
al objeto para acceder a la variable.

**¡El objeto Dog no va directamente dentro  
de la variable!**





# Los 3 pasos con un objeto

**1** **2**  
**3**  
`Dog myDog = new Dog();`

- ❶ Declarar la variable de referencia
- ❷ Crear el objeto
- ❸ Enlazar el objeto y la referencia

# Los 3 pasos con un objeto

## 1 Declarar la variable de referencia

```
Dog myDog = new Dog();
```

Esto es decirle al **JVM** que reserve espacio para la variable llamada **myDog**, que será del tipo **Dog**. En otras palabras, un control remoto que tiene botones para controlar un **Dog**, pero no otro tipo de objeto, como un gato o un ratón.

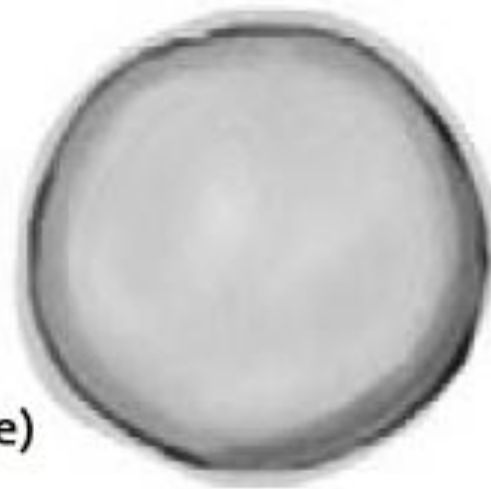


# Los 3 pasos con un objeto

## 2 Crear el objeto

```
Dog myDog = new Dog();
```

Le indica a la JVM que asigne espacio para un nuevo objeto **Dog** en el montón (ya hablaremos del montón más adelante)



Objeto Dog

# Los 3 pasos con un objeto

## 3 Enlazar el objeto y la referencia

```
Dog myDog = new Dog();
```

Asigna el nuevo objeto **Dog** a la variable de referencia myDog. En otras palabras, programa el control remoto.



# *Variables de referencia y objetos*

- ❏ Las variables de referencia pueden ser programadas para controlar diferentes objetos del mismo tipo
- ❏ Cuando una variable de referencia no se asigna a ningún objeto, apunta a ***null***
- ❏ Varias variables de referencia pueden apuntar al mismo objeto



# La vida en el montón

```
Book b = new Book();
```

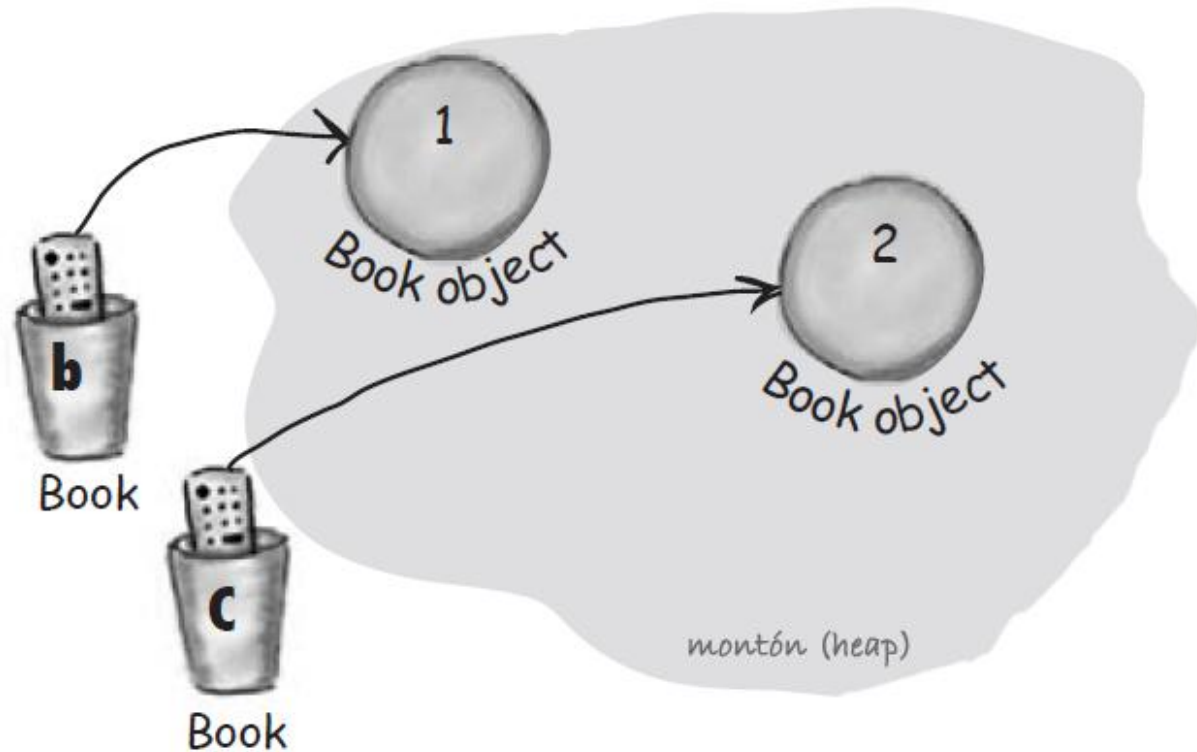
```
Book c = new Book();
```

Declaramos dos variables de referencia a Book. Creamos dos nuevos objetos Book. Asignamos los objetos Book a las variables de referencia b y c.

Los dos objetos Book ahora viven en el montón (heap).

Referencias: 2

Objetos: 2





# La vida en el montón

```
Book d = c;
```

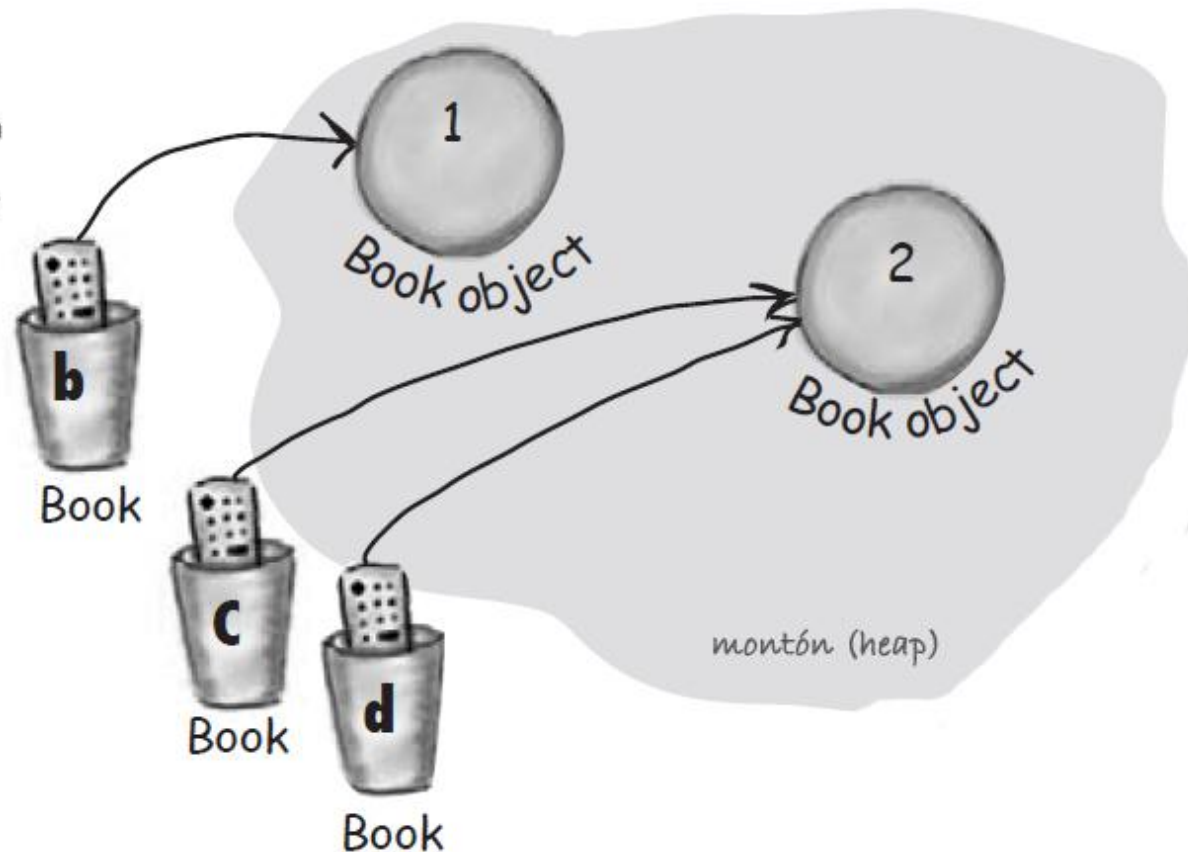
Declaramos una nueva variable de referencia a **Book**, llamada **d**. Asignamos el valor de la variable **c** a la variable **d**. ¿Esto qué significa? Es como decir: "Toma los bits de **c**, haz una copia de ellos y pega la copia en **d**."

**Ambas variables de referencia, d y c, apuntan al mismo objeto.**

**Las variables c y d almacenan dos copias diferentes del mismo valor. Dos mandos a distancia a la misma televisión.**

Referencias: 3

Objetos: 2



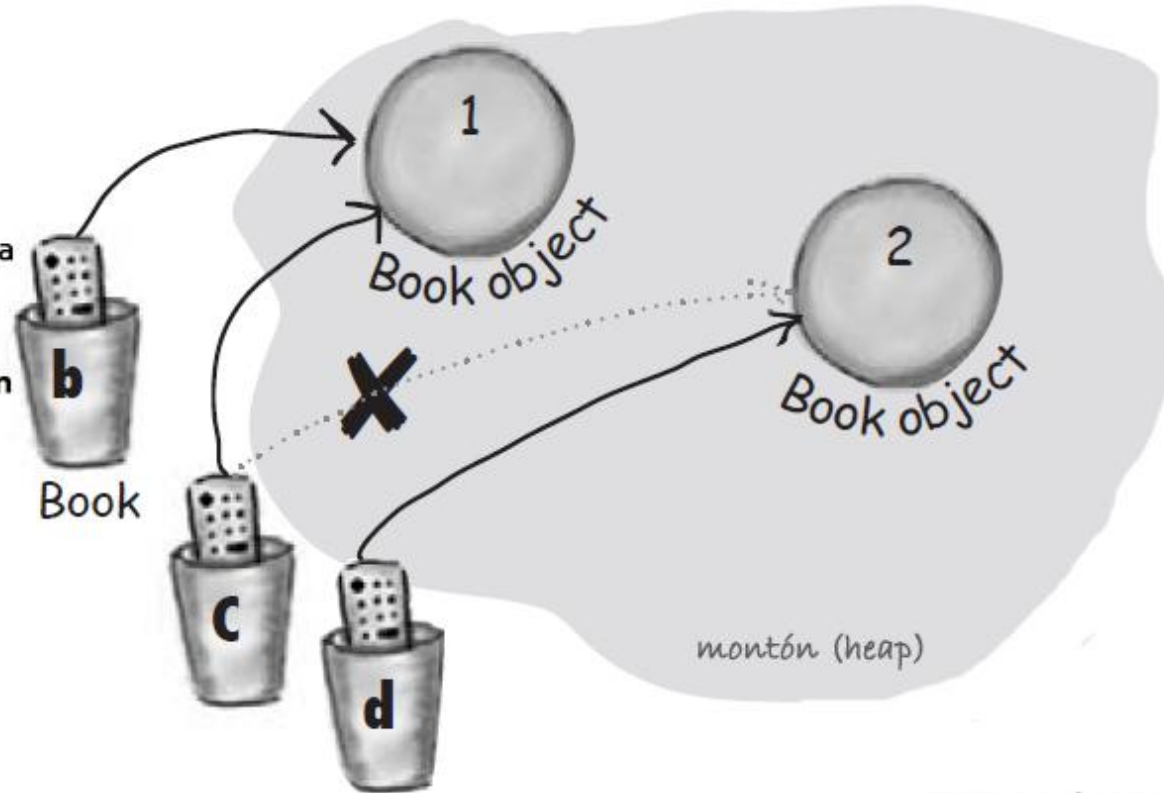
# La vida en el montón

```
c = b;
```

Asigna el valor de la variable **b** a la variable **c**.  
Los bits de la variable **b** son copiados y la nueva copia es metida en la variable **c**.

**Ambas variables de referencia, c y d, apuntan al mismo objeto.**

Referencias: 3  
Objetos: 2



# Vida y muerte en el montón

```
Book b = new Book();
```

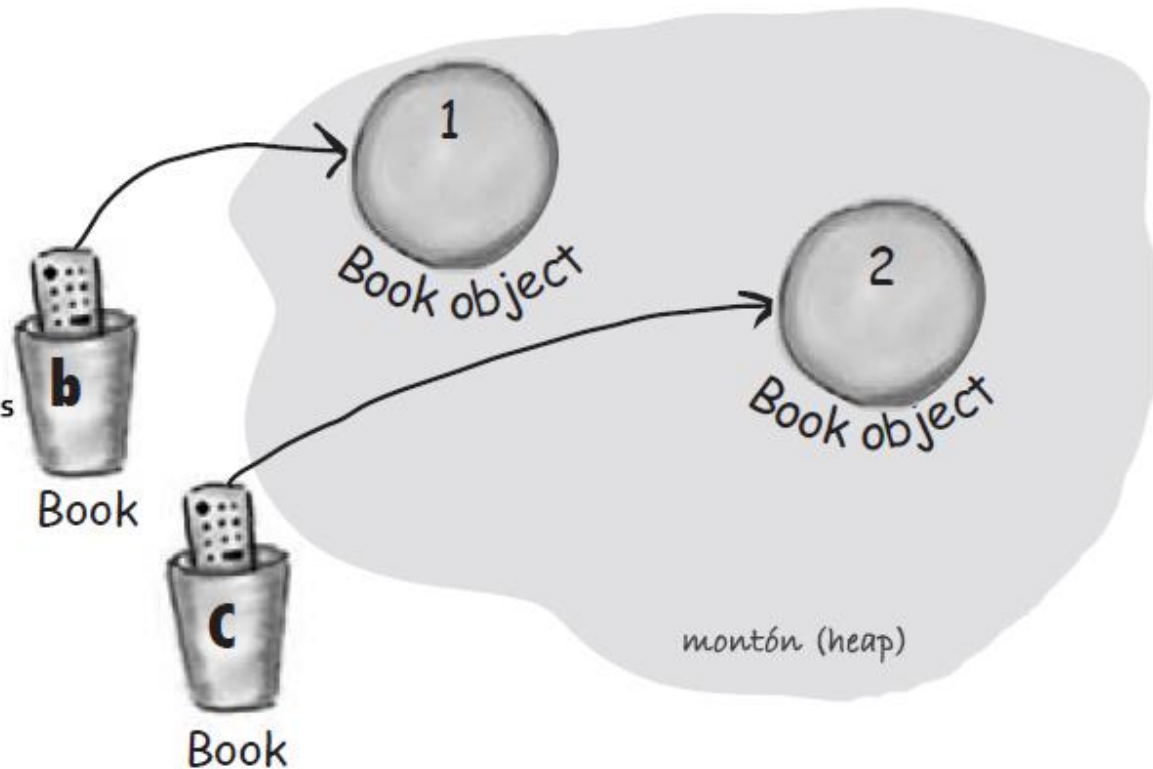
```
Book c = new Book();
```

Declaramos dos variables de referencia.  
Creamos dos nuevos objetos **Book**. Asignamos  
los objetos a las variables de referencia.

Los dos nuevos objetos viven ahora en el  
montón.

Referencias activas: 2

Objetos alcanzables: 2



# Vida y muerte en el montón

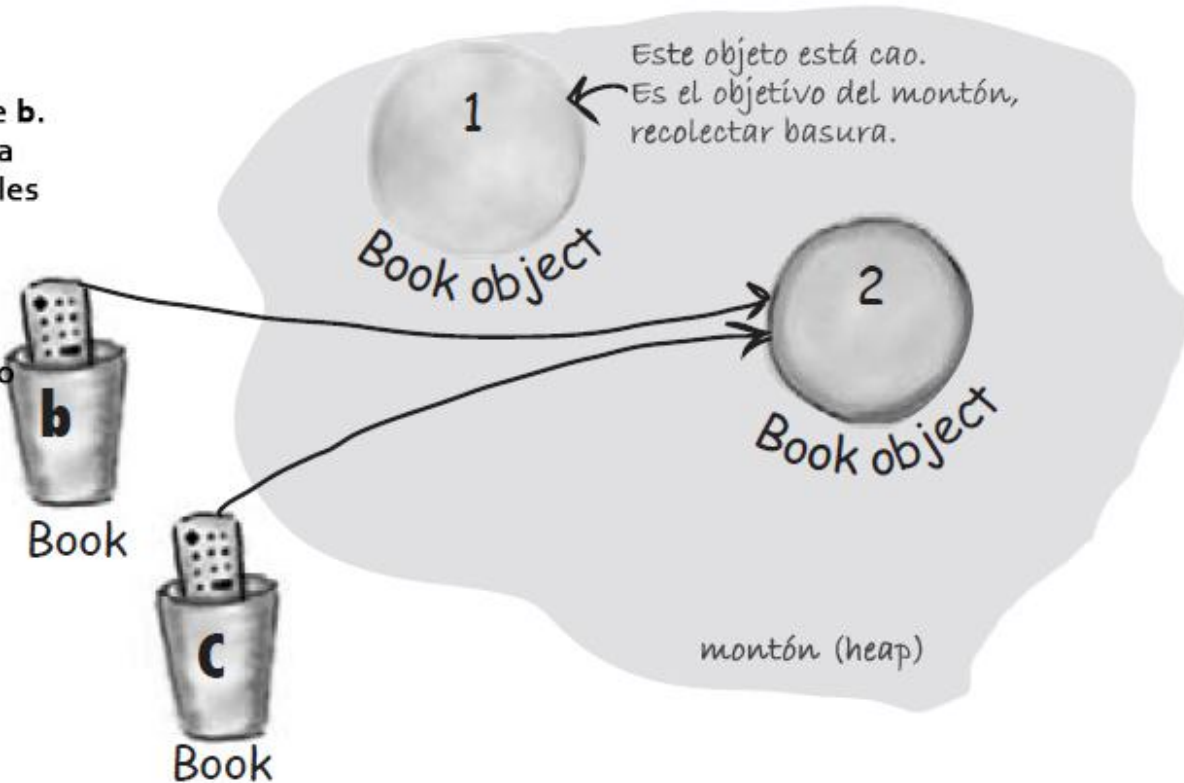
**b = c;**

Asignamos el valor de la variable **c** a la variable **b**. Los bits de la variable **c** son copiados y la nueva copia es metida en la variable **b**. Ambas variables almacenan valores idénticos.

Ambas variables de referencia, **b** y **c**, apuntan al mismo objeto. El objeto 1, al que no apunta ninguna variable, ha sido abandonado y ha sido elegido por el recolector de basura (**garbage collection, GC**).

Referencias activas: 2  
Objetos alcanzables: 1  
Objetos abandonados: 1

El primer objeto que referenció **b**, el objeto 1, no tiene más referencias. Es *inalcanzable*.





# Vida y muerte en el montón

```
c = null;
```

Asignamos el valor **null** a la variable **c**. Esto convierte a **c** en una referencia nula, que significa que no apunta a nada. Pero es todavía una variable de referencia y otro objeto **Book** podría ser asignado a ella.

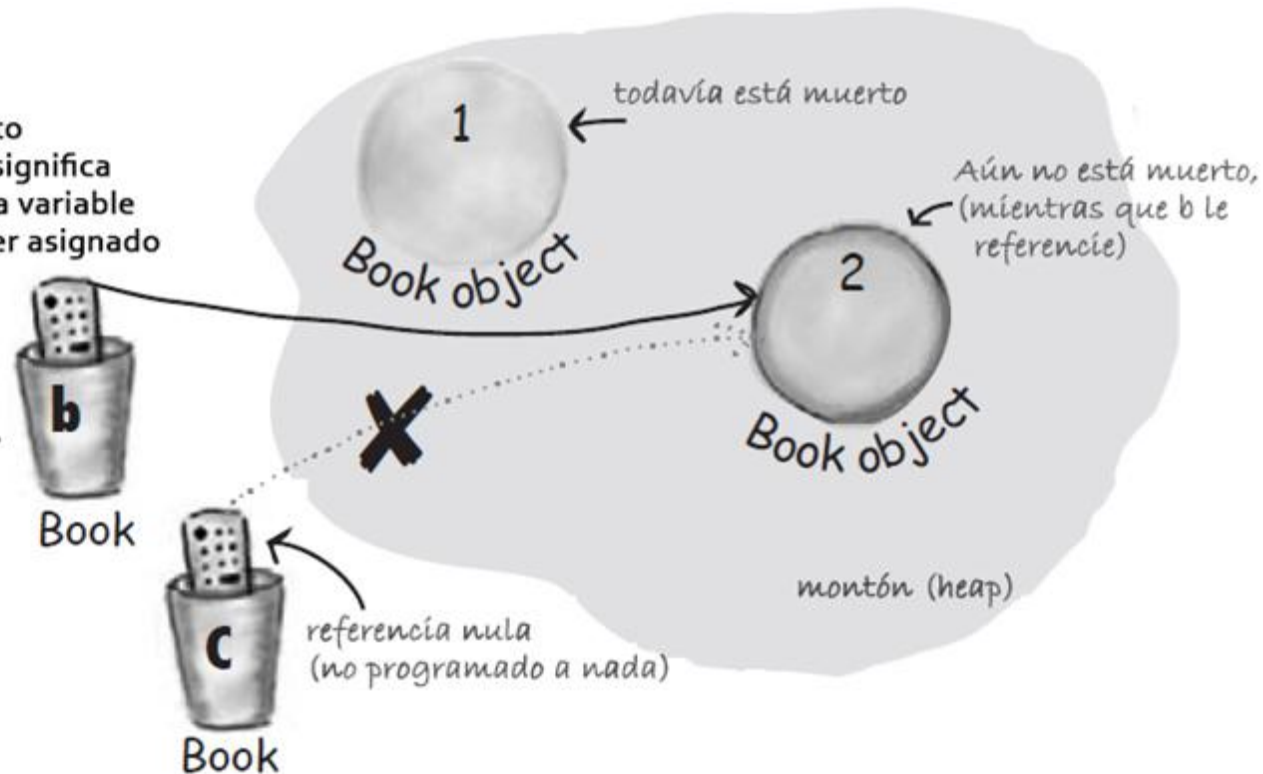
El objeto 2 todavía tiene una referencia activa (**b**) y, mientras tanto la tenga, el objeto no será elegido por la **basura** (GC).

Referencias activas: 1

Referencias nulas: 1

Objetos alcanzables: 1

Objetos abandonados: 1

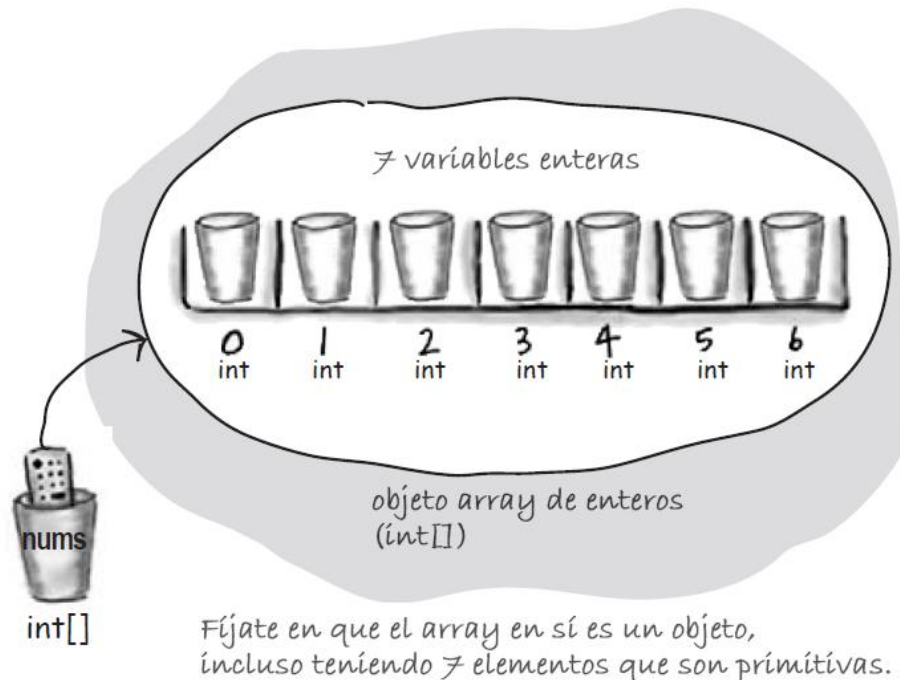


# *Array = objeto = bandeja de copas*

- ❶ Declaramos una variable **array de enteros**:

```
int[] nums;
```

- ❷ Creamos un nuevo array de enteros con la longitud de 7 y se lo asignamos a la variable **nums**: **nums = new int[7];**





# *Array = objeto = bandeja de copas*

- ③ Asignamos a cada elemento del array un valor (entero, al ser un array de enteros). Siempre empezaremos por el índice 0:

7 int variables {

```
nums[0] = 6;  
nums[1] = 19;  
nums[2] = 44;  
nums[3] = 42;  
nums[4] = 10;  
nums[5] = 20;  
nums[6] = 1;
```

# *Array de objetos Dog*

Vamos a crear un array de objetos tipo Dog:

❶ Declaramos una variable **array de Dog**:

```
Dog[] pets;
```

❷ Creamos un nuevo array de **Dog** con la longitud de 7 y se lo asignamos a la variable **pets**:

```
pets = new Dog[7];
```

El array no es realmente un array de objetos, es un array de referencias a objetos.

# *Array de objetos Dog*

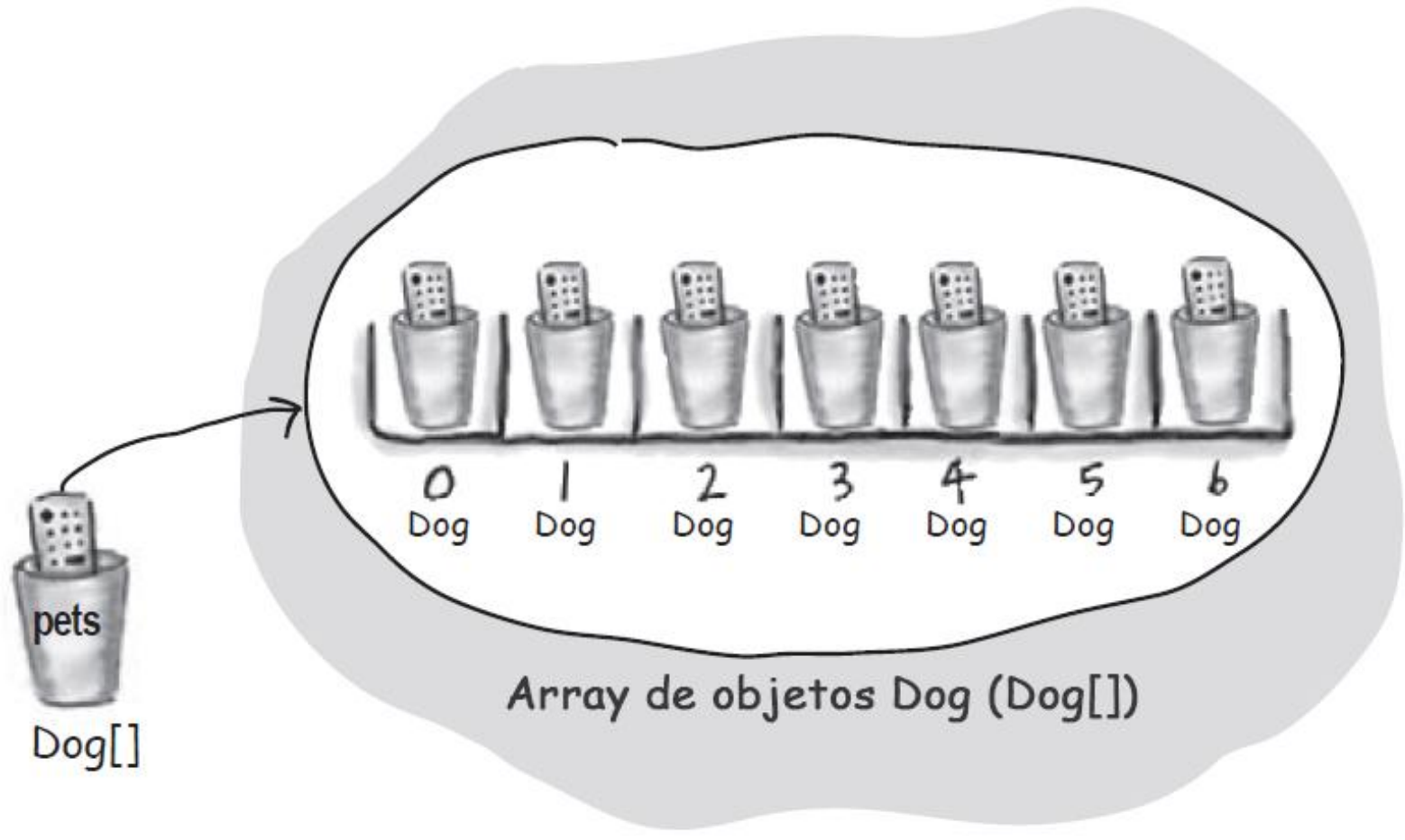
- ③ Creamos dos nuevos objetos **Dog**, y los asignamos al array de objetos creado:

```
pets[0] = new Dog();
```

```
pets[1] = new Dog();
```

Recuerda que los elementos del array no son objetos, son referencias a objetos, todavía nos faltan los objetos Dog.

# Array de referencias de objetos Dog



# Práctica P3.1: ¿Cómo lo harías?

- ❏ ¿Cuál es el valor del elemento `pets[3]`?
- ❏ ¿Cómo podrías hacer para que `pets[3]` apunte a uno de los objetos **Dog** que ya existen?
- ❏ Contesta a estas preguntas en la actividad.



# Controlando el objeto Dog (sin array)

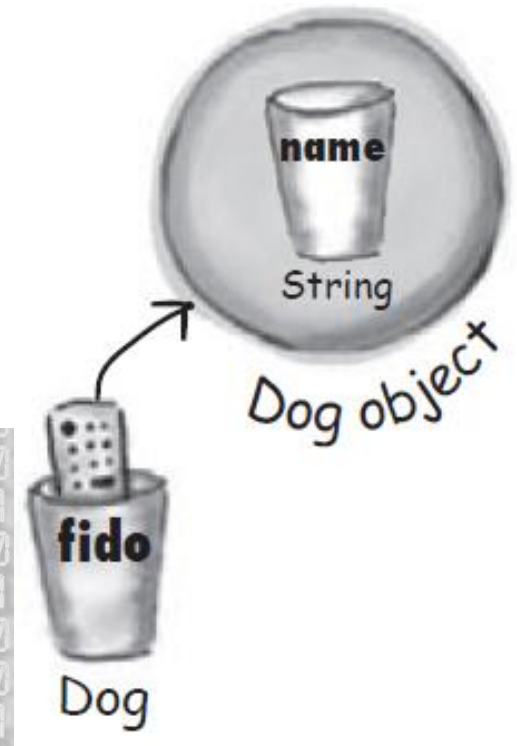
Dog
name
bark() eat() chaseCat()

```
Dog fido = new Dog();
```

```
fido.name = "Fido";
```

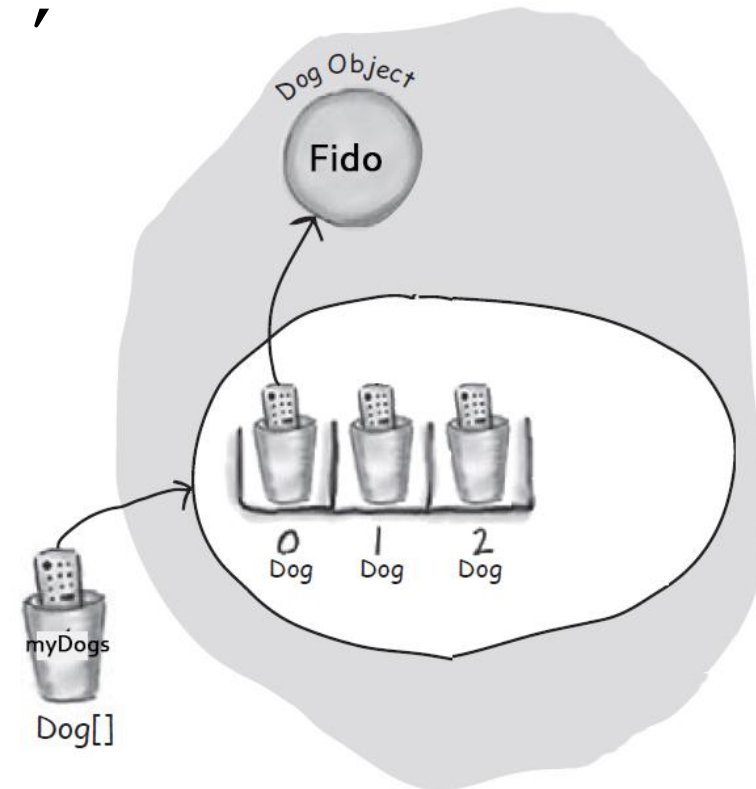
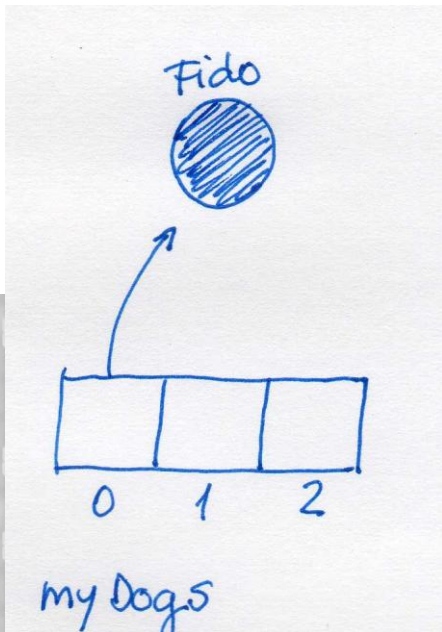
```
fido.bark();
```

```
fido.eat();
```



# Controlando el objeto Dog (en array)

```
Dog[] myDogs = new Dog[3];  
myDogs[0] = new Dog();  
myDogs[0].name = "Fido";  
myDogs[0].bark();
```



# Práctica P3.2: Adivina la salida

🖥️ ¿Cuál será la salida del siguiente programa?

## La clase Dog

Dog
name
bark() eat() chaseCat()

```
class Dog {
    String name;
    public static void main (String[] args) {
        // creamos el objeto Dog y accedemos a él
        Dog dog1 = new Dog();
        dog1.bark();
        dog1.name = "Bart";

        // ahora creamos el array de Dog
        Dog[] myDogs = new Dog[3];
        // y ponemos algunos Dog en él
        myDogs[0] = new Dog();
        myDogs[1] = new Dog();
        myDogs[2] = dog1;

        // ahora accedemos a los objetos
        // usando el array de referencias

        myDogs[0].name = "Fred";
        myDogs[1].name = "Marge";
```

```
        // ... ¿cuál es el nombre de myDogs[2]?
        System.out.print("last dog's name is ");
        System.out.println(myDogs[2].name);

        // recorremos el array y les decimos
        // a los perros (Dog) que ladren (bark)

        int x = 0;
        while(x < myDogs.length) {
            myDogs[x].bark();
            x = x + 1;
        }

        public void bark() {
            System.out.println(name + " says Ruff!");
        }
        public void eat() { }
        public void chaseCat() { }
    }
}
```

Los arrays tienen una variable 'length' que nos devuelve el nº de elementos del array

# **Puntos importantes (III)**

- ❑ La variables en Java pueden ser de dos tipos: ***primitivas y referencias.***

- ❑ Todas las variables se declaran con un ***nombre*** y un ***tipo.***

- ❑ Las ***primitivas*** contienen los bits que representan su valor.

- ❑ Las ***variables de referencia*** contienen bits que representan el camino para alcanzar el objeto en el montón.

## **Puntos importantes (III)**

- ❏ Una variable de referencia es como ***un control remoto*** o mando a distancia.
- ❏ Usar el ***operador punto (.)*** es como apretar el botón del control remoto, para acceder a los datos o los métodos del objeto.
- ❏ Cuando una variable de referencia no apunta a nada, su valor es ***null***.
- ❏ Un ***array*** es siempre un ***objeto***, incluso cuando los elementos del array almacenen primitivas.



# Práctica P3.3: Haz de compilador



De cada uno de los siguientes archivos, comprueba cuál de ellos compila, ¿Podrías identificar cuál compila y cuáles no y encontrar los errores?

A

```
class Books {
    String title;
    String author;
}

class BooksTestDrive {
    public static void main(String [] args) {

        Books [] myBooks = new Books[3];
        int x = 0;
        myBooks[0].title = "The Grapes of Java";
        myBooks[1].title = "The Java Gatsby";
        myBooks[2].title = "The Java Cookbook";
        myBooks[0].author = "bob";
        myBooks[1].author = "sue";
        myBooks[2].author = "ian";

        while (x < 3) {
            System.out.print(myBooks[x].title);
            System.out.print(" by ");
            System.out.println(myBooks[x].author);
            x = x + 1;
        }
    }
}
```

B

```
class Hobbits {

    String name;

    public static void main(String [] args) {

        Hobbits [] h = new Hobbits[3];
        int z = 0;

        while (z < 4) {
            z = z + 1;
            h[z] = new Hobbits();
            h[z].name = "bilbo";
            if (z == 1) {
                h[z].name = "frodo";
            }
            if (z == 2) {
                h[z].name = "sam";
            }
            System.out.print(h[z].name + " is a ");
            System.out.println("good Hobbit name");
        }
    }
}
```



Intenta resolver el ejercicio observando el código, y después comprueba tus resultados con el compilador de Java. Crea el diagrama de clases UML, con el DIA.

# Práctica P3.4: Ordena el código

📄 Ordena las sentencias del siguiente programa para obtener esta salida:

```
int y = 0;  
ref = index[y];  
islands[0] = "Bermuda";  
islands[1] = "Fiji";  
islands[2] = "Azores";  
islands[3] = "Cozumel";  
int ref;  
while (y < 4) {  
    System.out.println(islands[ref]);  
    index[0] = 1;  
    index[1] = 3;  
    index[2] = 0;  
    index[3] = 2;
```


```
String [] islands = new String[4];  
System.out.print("island = ");  
int [] index = new int[4];  
y = y + 1;
```

```
class TestArrays {  
    public static void main(String [] args) {
```

```
File Edit Window Help Bikini  
% java TestArrays  
island = Fiji  
island = Cozumel  
island = Bermuda  
island = Azores
```

📄 Crea el diagrama de clases con el DIA, y prueba el programa para que funcione como se espera.

## *Práctica P3.5: Puzzle en la piscina*


 Escoge los fragmentos de código de la piscina y colócalos en los espacios en blanco, de forma que compile y la ejecución produzca la salida esperada.

```

class Triangle {
    double area;
    int height;
    int length;
    public static void main(String [] args) {
        _____
        while ( _____ ) {
            _____
            _____
            _____
            System.out.print("triangle "+x+", area");
            System.out.println(" = " + _____ .area);
            _____
        }
        _____
        x = 27;
        Triangle t5 = ta[2];
        ta[2].area = 343;
        System.out.print("y = " + y);
        System.out.println(", t5 area = "+ t5.area);
    }
    void setArea() {
        _____ = (height * length) / 2;
    }
}

```

Nota: cada fragmento de código sólo se podrá utilizar una vez.



```

4, t5 area = 18.0
4, t5 area = 343.0
area
ta.area
27, t5 area = 18.0
x ta.x.area
27, t5 area = 343.0
y ta[x].area

ta[x] = setArea();
ta.x = setArea();
ta[x].setArea();

int x;
int y;
x = x + 1;
x = x + 2;
x = x - 1;
ta.x
ta(x)
ta[x]
x < 4
x < 5

Triangle [] ta = new Triangle(4);
Triangle ta = new [] Triangle(4);
Triangle [] ta = new Triangle(4);

28.0
30.0

ta = new Triangle();
ta[x] = new Triangle();
ta.x = new Triangle();

```









Salida

```
File Edit Window Help Bermuda
%java Triangle
triangle 0, area = 4.0
triangle 1, area = 10.0
triangle 2, area = 18.0
triangle 3, area = ____
y = ____
```

 Crea también el diagrama de clases.



# Práctica P3.5: Puzzle en la piscina

## Consideraciones a tener en cuenta:

-  Esta actividad es más complicada de lo que parece, no te desanimes
-  Cada fragmento de solución sólo podrá ser usado una vez
-  No será necesario utilizar todos los fragmentos
-  Cada línea representa una línea de código
-  Las líneas largas son para instrucciones largas
-  Las líneas cortas para instrucciones cortas
-  Las condiciones van dentro de bucles o tests condicionales
-  Las sentencias van sueltas



# Práctica P3.6: Empareja

-  Determina, una vez ejecutado el programa, a qué objetos apuntan cada una de las variables de referencia. No todas las variables de referencia serán usadas y algunos objetos serán apuntados por varias de ellas.
-  Empareja las variables con los objetos a los que apuntan.

```
class HeapQuiz {  
    int id = 0;  
    public static void main(String [] args) {  
        int x = 0;  
        HeapQuiz [ ] hq = new HeapQuiz(5);  
        while ( x < 3 ) {  
            hq[x] = new HeapQuiz();  
            hq[x].id = x;  
            x = x + 1;  
        }  
        hq[3] = hq[1];  
        hq[4] = hq[1];  
        hq[3] = null;  
        hq[4] = hq[0];  
        hq[0] = hq[3];  
        hq[3] = hq[2];  
        hq[2] = hq[0];  
        // do stuff  
    }  
}
```

Variables de referencia



Objetos HeapQuiz

