

Áreas de la memoria

- 1 Áreas de la memoria
- 2 Valores y referencias
- 3 Ocultamiento

Temas

- 1 Áreas de la memoria
 - Especificación de la máquina virtual JVM
 - El recolector de basura

Temas

- 1 Áreas de la memoria
 - Especificación de la máquina virtual JVM
 - El recolector de basura

Liberación de la memoria

- Cuando un programa solicita hacer uso de memoria con el operador `new`, por ejemplo para crear un objeto, después debe liberarla, cuando ya no la necesite.

```
1 Objeto o = new Objeto();
```

- En lenguajes como C++ las clases tiene destructores para cumplir con esta función.

```
1 ~Objeto() {...} // Declaración  
2 delete o;      // Llamando un destructor
```

- La JVM tiene, en su lugar, un *recolector de basura*.
 - Éste es un programa encargado de detectar los objetos que **ya no son accesibles** por el usuario y por consiguiente los elimina, liberando la memoria.
 - Se dice que un objeto no es accesible cuando ninguna variable accesible directa o indirectamente desde la pila contiene la dirección del objeto.

Ejemplo

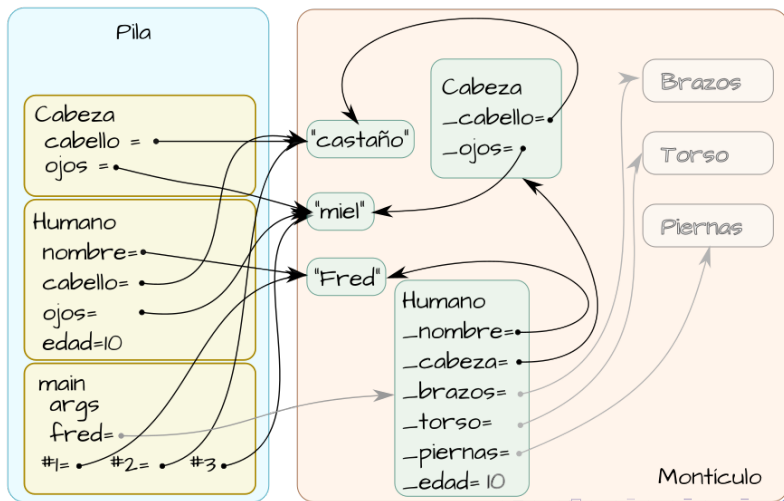
Código: "Humano"

```
1 public class Humano {
2     private String nombre;
3     private Cabeza cabeza;
4     private Brazos brazos;
5     private Torso torso;
6     private Piernas piernas;
7     private int edad;
8     public Humano(String nombre,
9                     String cabello,
10                    String ojos,
11                    int edad) {
12         this.nombre = nombre;
13         cabeza = new Cabeza(cabello, ojos);
14         brazos = new Brazos();
15         torso = new Torso();
16         piernas = new Piernas();
17         this.edad = edad;
18     }
19 }
```

```
1 public class Cabeza {
2     private String cabello;
3     private String ojos;
4     public Cabeza(String cabello,
5                    String ojos) {
6         this.cabello = cabello;
7         this.ojos = ojos;
8     }
9 }
10 public class Brazos {}
11 public class Torso {}
12 public class Piernas {}
```

Acceso a objetos

```
public static void main(String
    ➔ [] args) {
    Humano fred =
        new Humano("Fred",
                    "castaño",
                    "miel",
                    10)
}
```



Valores y referencias

- 1 Áreas de la memoria
- 2 Valores y referencias
- 3 Ocultamiento

Temas

- 2 Valores y referencias
 - Tipos primitivos y objetos

Valores y referencias

Java distingue dos tipos de variables:

- Tipos **primitivos**: las variables de tipos primitivos almacenan directamente el *valor*.
- **Clases**: las variables de objetos (instancias de clases) no contienen al objeto, si no una *referencia* a ellos.
 - Los objetos siempre están almacenados en el montículo.
 - La variable contiene la dirección del objeto y sólo podemos acceder a sus atributos y métodos mediante el uso del operador punto (.)... si tenemos permiso.

Código: "Valores y referencias"

```
1 int x = 8;  
2 String s = "¡Hola!";  
3 Persona carlos = new Persona("Carlos");
```

Paso por valor

- Cuando se llama una función o método Java pasa copias de los valores almacenados.
 - Si se trata de valores primitivos se copia el valor.
 - Si se trata de objetos se copia la dirección almacenada en la variable. Dado que es una dirección, es frecuente que este comportamiento se confunda con *paso por referencia*.

Código: "Paso por valor"

```
1 Calculadora calculadora = new Calculadora();  
2 int n = calculadora.suma(8, 2);  
3 Tocino tocino = new Tocino();  
4 Comida c = cocinero.hazDesayuno(tocino);
```

Ocultamiento

- 1 Áreas de la memoria
- 2 Valores y referencias
- 3 **Ocultamiento**

Temas

3 Ocultamiento

- Atributos y variables locales

Atributos y variables locales

- Java distingue dos tipos de variables:

- 1 Los atributos de los objetos o las variables de la clase.

```
1 public class Ejemplo {  
2     private static int cuentaObjjs = 0; // Variable de la clase  
3     private int id;                      // Atributo de objeto  
4 }
```

- 2 Las variables locales, que sólo existen dentro de los métodos o funciones, incluyendo sus parámetros.

```
1 public class Ejemplo {  
2     public Ejemplo() {  
3         int idNuevo = cuentaObjjs;      // Variable local  
4         id = idNuevo;                   // Atributo  
5         cuentaObjjs++;  
6     }  
7 }
```

Bloques en funciones


- Un *bloque* es una unidad en el lenguaje de programación donde se define un ambiente.
- En Java, los bloques se delimitan por un par de llaves {}.
- Lenguajes como Java permiten tener ambientes anidados, es decir, uno dentro de otro.
- En un ambiente se pueden declarar variables que serán visibles dentro de él y en todos los bloques dentro él.
- En un bloque interno, no se pueden declarar variables que se llamen igual que variables en bloques que los contienen.
- Si una variable local se llama igual que un atributo lo **oculta**.

Ejemplo

```
1 package ambientes;
2
3 public class Ambientes {
4     private String val = "Hola";
5     private String única = "Nadie_¡la_!tapa";
6
7     public Ambientes(int val) {
8         System.out.println(val);
9         val = 1;
10        System.out.println(val);
11        System.out.println(this.val);           // Para distinguir al atributo
12        System.out.println(única);
13        {
14            int val1 = 2;
15            System.out.println(val1);
16        }
17        // for(int val = 0; false;) {           // ¡Mira! No se puede xD
18        for(int val2 = 0; val < 2; val++) {     // Sí funciona, pero es peligroso
19            System.out.println(val2);
20        }
21        // System.out.println(val1);           // No se puede.
22    }
23    public static void main(String[] args) {
24        Ambientes a = new Ambientes(0);
25    }
26 }
```

Bibliografía I



 Lindholm, Tim et al. (26 de nov. de 2020). *Run-Time Data Areas in The Structure of the Java Virtual Machine (Java 11)*. Ed. por Oracle. URL: <https://docs.oracle.com/javase/specs/jvms/se11/html/jvms-2.html#jvms-2.5>.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual

