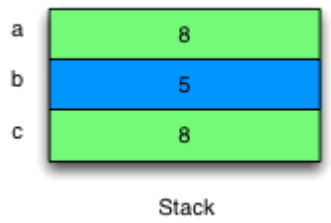
java == vs equal es una de las preguntas más habituales para los nuevos programadores cuando trabajan con el lenguaje Java . Vamos a ver a detalle este concepto en este articulo.

```
package com.arquitecturajava;

public class Principal3 {

    public static void main(String[] args) {

        int a = 8;
        int b = 5;
        int c = 8;

        // Comparación de valores con el operador ==
        System.out.println(a == b); // false
        System.out.println(a == c); // true
    }
}
```

En este caso estamos comparando tipos básicos los cuales se almacenaran en el stack o pila.



Compararlos es sencillo ya que estamos comparando valores por lo tanto a==c devolverá true. Otra cosa muy distinta ocurre si lo que nos creamos son objetos. Vamos a ver un ejemplo partiendo de la clase Persona.

```
package com.arquitecturajava;
```

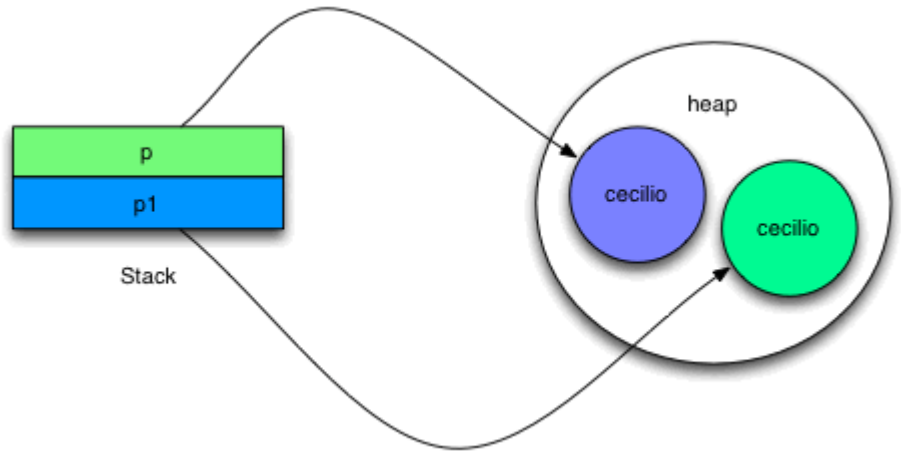
```
public class Persona {

    private String nombre;

    // Constructor de la clase Persona
    public Persona(String nombre) {
        super();
        this.nombre = nombre;
    }

    // Método getter para el atributo nombre
    public String getNombre() {
        return nombre;
    }

    // Método setter para el atributo nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Una vez creada la clase vamos a crear dos objetos:

```
package com.arquitecturajava;

public class Principal4 {

    public static void main(String[] args) {

        Persona p = new Persona("cecilio");
        Persona p1 = new Persona("cecilio");
```

```
        // Esto compara las referencias de los objetos, no su
contenido
        System.out.println(p == p1); // false, ya que son diferentes
instancias

        // Si quieres comparar el contenido (el nombre), usa equals()
        System.out.println(p.equals(p1)); // Esto devolverá false a
menos que sobrescribas equals()
    }
}
```

El resultado de comprar p y p1 devolverá false algo que a mucha gente le sorprende en un primer momento. Esto se debe a que he construido dos objetos y las variables p y p1 definen dos direcciones de memoria diferentes donde estos objetos están ubicados (heap). Así pues en el stack o pila de invocación entremos algo como lo siguiente.

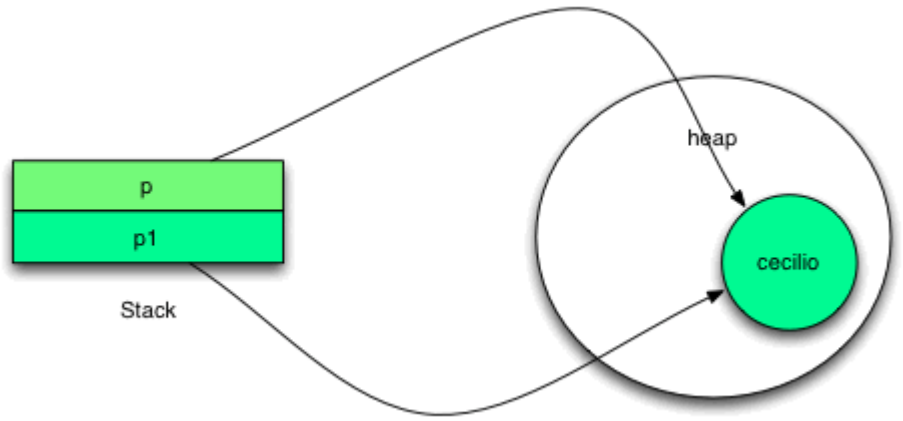


Así pues es lógico que si comparamos p y p1 nos devuelva false ya que apuntamos a dos objetos diferentes. ¿Cómo podemos asegurarnos que p==p1 devuelve true?. Muy sencillo podemos crear un único objeto y hacer que dos variables o punteros apunten a él.

```
package com.arquitecturajava;
```

```
public class Principal5 {

    public static void main(String[] args) {

        Persona p = new Persona("cecilio");
        Persona p1 = p;  // p1 apunta al mismo objeto que p

        // Esto comparará las referencias de los objetos, no el
contenido
        System.out.println(p == p1); // true, ya que ambos apuntan al
mismo objeto en memoria
    }
}
```

De esta manera como los dos punteros apuntan a la misma dirección de memoria.



El resultado del programa será true.

java == vs equals (usando equals)

Mientras que es muy habitual usar el operador == para comparar tipos básicos no es tan habitual usarlo a la hora de comparar objetos ya que no tiene sentido a nivel de reglas de negocio. Por ejemplo en este caso a nivel de negocio consideraremos dos objetos iguales si

tienen el mismo nombre. Por lo tanto deberemos sobreescribir el método equals y hashcode de la clase Persona y hacer que dos objetos Persona sean iguales si su nombre coincide :

```
package com.arquitecturajava;

import java.util.Objects;

public class Persona {

    private String nombre;

    // Constructor
    public Persona(String nombre) {
        this.nombre = nombre;
    }

    // Getter para nombre
    public String getNombre() {
        return nombre;
    }

    // Setter para nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Sobrescritura de hashCode
    @Override
    public int hashCode() {
        return Objects.hash(nombre); // Usar Objects.hash() para
manejar posibles null
```

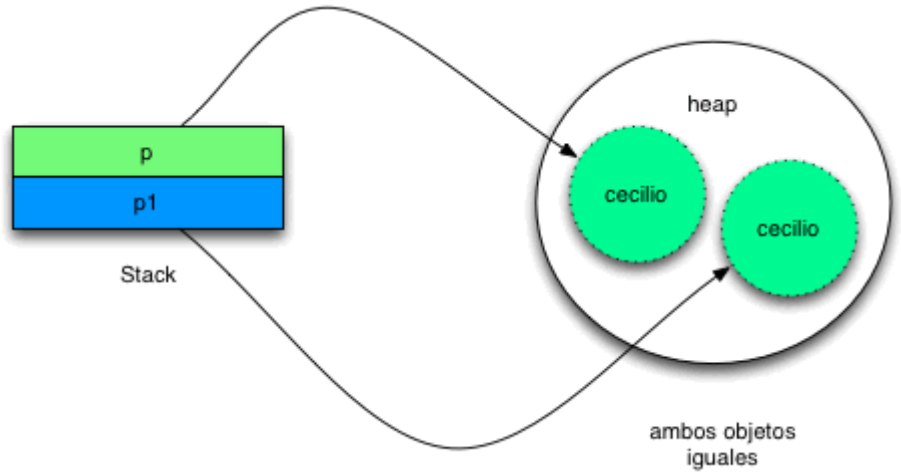
```
    }

    // Sobrescritura de equals
    @Override
    public boolean equals(Object obj) {
        // Verificar si es el mismo objeto
        if (this == obj) {
            return true;
        }
        // Verificar si el objeto es nulo o no es de la misma clase
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        // Realizar el casting seguro y comparar los nombres
        Persona p = (Persona) obj;
        return Objects.equals(nombre, p.getNombre()); // Usar
Objects.equals() para manejar nulls
    }
}
```

Acabamos de sobreescribir los métodos (de forma simplificada) . Ahora podemos usar el método equals para comparar los siguientes objetos y nos devolverá true.

```
package com.arquitecturajava;

public class Principal4 {

    public static void main(String[] args) {

        Persona p = new Persona("cecilio");
        Persona p1 = new Persona("cecilio");
```

```
        // Comparación utilizando el método equals()
        System.out.println(p.equals(p1)); // Devolverá true si
equals() está bien implementado
    }
}
```

Aunque tenemos dos variables que apuntan a objetos distintos ambos se consideran iguales a nivel semántico o de reglas de negocio que es lo que importa.



Existen clases que tienen el método equals ya sobrecargado por defecto. Un ejemplo clásico es la clase String. Dos cadenas son iguales si su texto es idéntico.

Otros artículos relacionados:

- ¿Qué es un Java Lambda?
- Java equals y hashcode ¿Como funcionan?
- Comparando java == vs equals
- Generando un Java HashCode correcto
- Java Comparable Interface y Ordenaciones