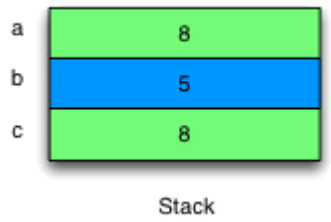
Ayer me han preguntado en twitter si podría explicar la diferencia de Java == vs equals . En vez de responder por twitter lo voy a hacer directamente en el blog ya que me ha parecido una pregunta interesante. Supongamos que tenemos el siguiente bloque de código:

Ayer me han preguntado en twitter si podría explicar la diferencia de Java == vs equals . En vez de responder por twitter lo voy a hacer directamente en el blog ya que me ha parecido una pregunta interesante. Supongamos que tenemos el siguiente bloque de código:

```
package com.arquitecturajava;

public class Principal3 {

public static void main(String args[]) {

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

System.out.println(a==b);
System.out.println(a==c);
}
```

```
}
```

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.

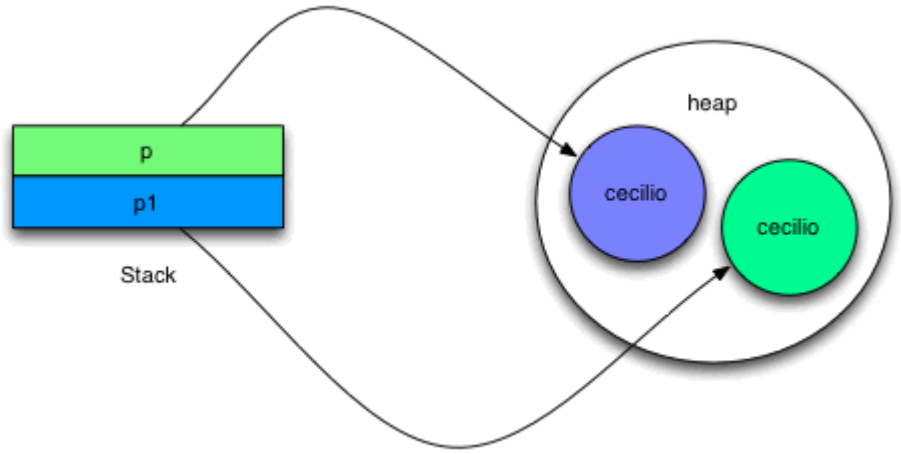
```
package com.arquitecturajava;

public class Persona {

private String nombre;

public String getNombre() {
return nombre;
}

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

public Persona(String nombre) {
```

```
super();
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");

System.out.println(p==p1);

}

}
```

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.

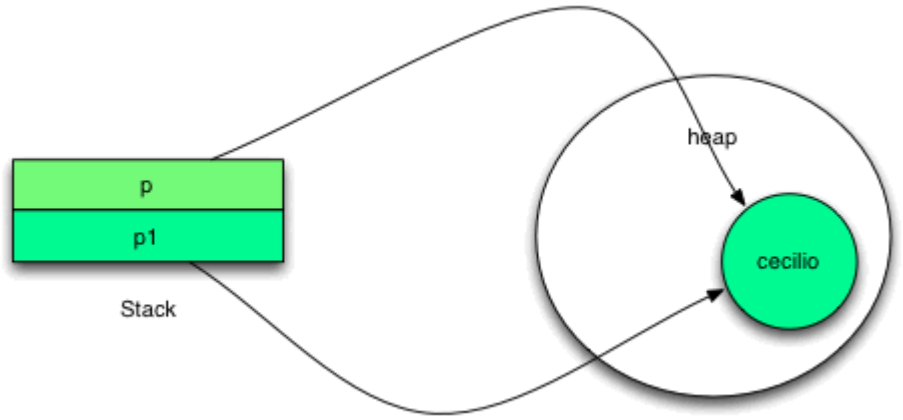


Así pues es lógico que si comparamos p y p1 nos devuelva false ya que apuntamos a dos objetos diferentes. ¿Cómo podemos asegurarnos que p==p1 devuelve true?. Muy sencillo podemos crear un único objeto y hacer que dos variables o punteros apunten a él.

```
package com.arquitecturajava;

public class Principal5 {

public static void main(String[] args) {

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

```
Persona p1=p;

System.out.println(p==p1);

}

}
```

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



El resultado del programa será true.

El método equals y hashCode

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 :

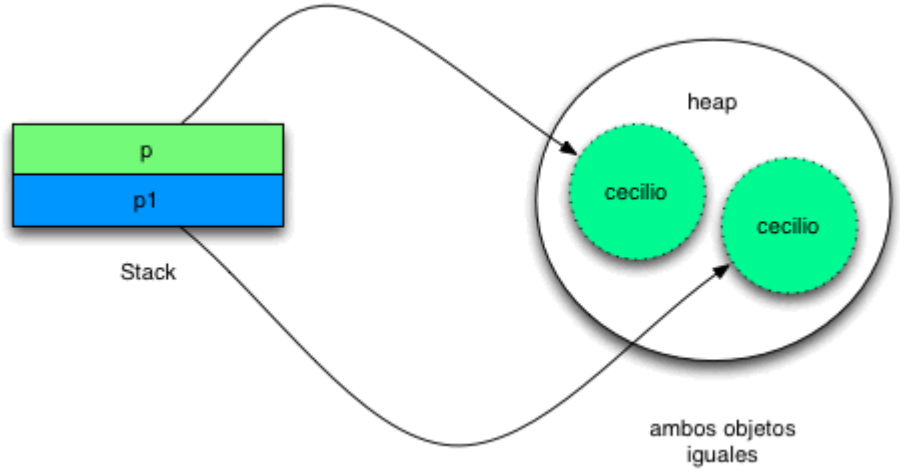
```
package com.arquitecturajava;

public class Persona {

private String nombre;

public String getNombre() {
return nombre;
}

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

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

@Override
public int hashCode() {
return nombre.hashCode();
}

@Override
public boolean equals(Object obj) {
Persona p= (Persona)obj;

return p.getNombre().equals(this.getNombre());

}
```

```
}
```

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");

System.out.println(p.equals(p1));

}

}
```

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

- Java equals y hashcode
- this() y super
- java fluid interface