Java HashCode es uno de los métodos que mas quebraderos de cabeza genera a los programadores.

```
public boolean hashCode();
```

El método se usa en combinación con el método equals para comparar a nivel de negocio cuando dos objetos de una clase son idénticos. La implementación de equals siempre es mucho más sencilla de entender.

Java HashCode

Este método existe para mejorar el rendimiento de Java a la hora de comparar dos objetos. Ya que si dos objetos tienen hashCodes diferentes ya no pueden ser iguales y se trata de hacer una comparación sencilla entre números enteros. Los problemas surgen en como generar los hashCodes, ya que el framework de colecciones lo usa para diseñar las estructuras internas de sus tipos abstractos de datos. En el siguiente ejemplo se muestra la clase Persona con el método equals sobreescrito (override) para que dos objetos sean iguales si su nombre y apellidos coinciden . El método es mas o menos entendible.

```
package com.arquitecturajava;

public class Persona {

private String nombre;
private String apellidos;
```

```
public String getNombre() {
return nombre;
}
public void setNombre(String nombre) {
this.nombre = nombre;
}
public String getApellidos() {
return apellidos;
}
public void setApellidos(String apellidos) {
this.apellidos = apellidos;
}

@Override
public boolean equals(Object obj) {
if (this == obj)
return true;
if (obj == null)
return false;
if (getClass() != obj.getClass())
return false;
Persona other = (Persona) obj;
if (apellidos == null) {
if (other.apellidos != null)
return false;
} else if (!apellidos.equals(other.apellidos))
return false;
if (nombre == null) {
if (other.nombre != null)
return false;
} else if (!nombre.equals(other.nombre))
```

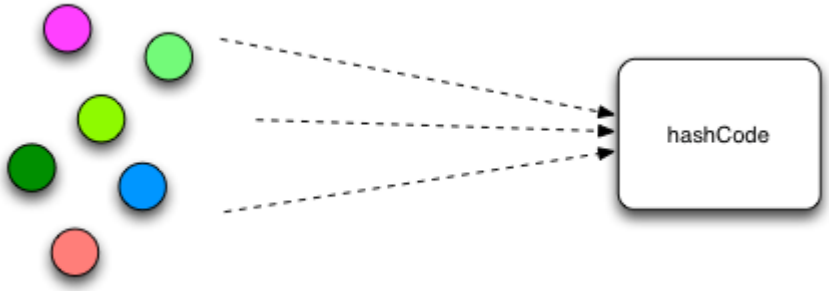
```
return false;
return true;
}

}
```

Eclipse Java HashCode

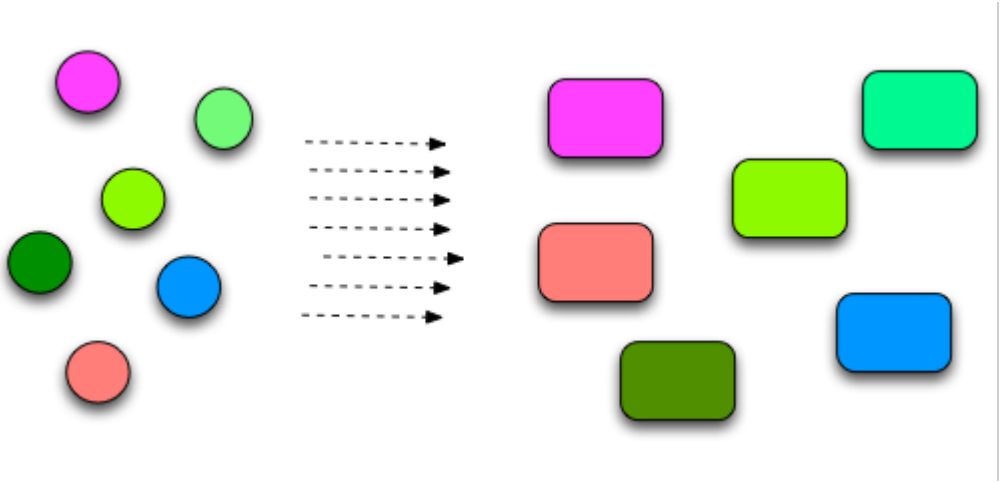
El método anterior ha sido generado con los refactorings de Eclipse . De igual forma se puede generar el hashCode.

```
public int hashCode() {
final int prime = 31;
int result = 1;
result = prime * result
+ ((apellidos == null) ? 0 : apellidos.hashCode());
result = prime * result + ((nombre == null) ? 0 : nombre.hashCode());
return result;
}
```

Este método ya no es tan sencillo de entender . Esto es debido a que los hashCodes deben tener una cierta variabilidad . Es decir si todos los objetos generan el mismo hashcode() , esto no valdrá para mejorar el rendimiento en las comparaciones ya que todos generan el mismo y habrá que utilizar siempre el método equals a la hora de realizarlas.



Por el otro lado si todos los objetos generan hashCodes diferentes , será imposible agruparlos dentro de los distintos tipos de colecciones de una forma óptima.



Java HashCode Correctos

De ahí que normalmente Eclipse se encargue de diseñar su propio método hashCode a través del uso de números primos. Sin embargo existe otra opción mucho mas elegante y sencilla para generarlos. El API de Java dispone de una clase Objects en el paquete de

utilidades que genera hashCodes.

```
@Override
 public int hashCode() {
 // TODO Auto-generated method stub
 return Objects.hash(nombre,apellidos);
 }
```

De esta forma nos olvidaremos de los líos que este método genera en los programadores.

Otros artículos relacionados : Entendiendo Equals y HashCode, Java == vs Equals