Sobreescribir equals(Object o)

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

1 equals(Object o)

.equals(Object o)

Cuando creamos un objeto o instancia, lo que tenemos es una referencia a la memoria (RAM), es decir, no se almacenan datos directamente en la variable de tipo objeto, solo la referencia al lugar donde están los valores de los atributos del objeto. Es por eso que no podemos utilizar el operador “==” para comparar la igualdad entre dos objetos porque estaríamos comparando referencias.

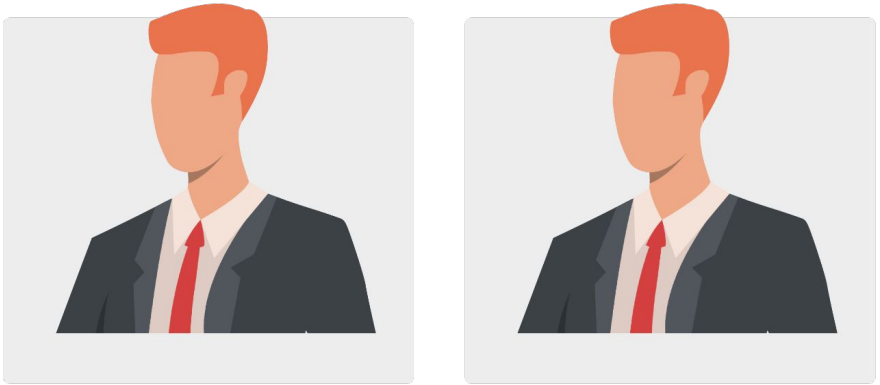
Para comparar correctamente debemos usar el **método equals()**, el cual lo heredamos de **Object**, pero no siempre funciona correctamente el equals que obtenemos por defecto, por eso, es necesario sobreescribirlo. El método equals() recibe como parámetro un objeto Object, esto nos dará una dificultad adicional a la hora de sobreescribirlo.

Recordemos que no podemos cambiar la firma del método si queremos sobreescribirlo.

¿Qué quiere decir que dos objetos son iguales?

Antes de comenzar a trabajar con el equals, debemos pensar qué quiere decir que dos objetos son iguales, por ejemplo, si comparamos dos empleados, podríamos definir que son iguales si sus legajos son iguales.

De esta forma cuando escribimos una clase, una de las cosas que debemos determinar es cómo vamos a comprobar la igualdad de dos instancias de esa clase.



2 | Object puede ser cualquier cosa

La clase Empleado

Como ejemplo vamos a trabajar con la clase empleado y, tal como mencionamos, dos empleados son iguales si sus legajos también lo son.

```
public class Empleado{
      private   String nombre;
      private   String legajo;
      protected double sueldo;
      protected double descuentos;

      }
}
```

.equals(Object o)

Vamos a sobreescribir el .equals(Object o). Nuestro primer paso es recordar cómo es la firma de este método, debemos respetar la firma del equals() heredado de Object

```
public class Empleado{
      private String nombre;
      private String legajo;
      private double sueldo;
      private double descuentos;

      @Override
      public boolean equals(Object o){
      }
}
```

.equals(Object o)

Para comenzar a escribir nuestro equals, debemos considerar que el parámetro que me está llegando es un Object, no dice que sea un Empleado, entonces, lo primero a verificar es si realmente es un Empleado, si no lo fuera ya podemos decir que no son iguales.

Vamos a ver dos formas de comprobarlo:

- **instanceof**
- **getClass()**

```
@Override
public boolean equals(Object o){

}
```

instanceof

Una forma de comparar dos instancias. **instanceof**

```
@Override
public boolean equals(Object o){
      if (o==null)
            return false;
      if (!(o instanceof Empleado))
            return false;
      else{

      }
}
```

instanceof

```
@Override
public boolean equals(Object o)
      if (o==null)
            return false;
      if (!(o instanceof Empleado))
            return false;
      else{

      }
}
```

Por ser un objeto podría tener valor null, es lo primero que comprobamos.

instanceof

```
@Override
public boolean equals(Object o)
     if (o  null)
          return false;
     if (!(o instanceof Empleado))
          return false;
     else{

     }
}
```

instanceof es un operador que nos permite comprobar si o es una instancia de Empleado. En este caso, si no lo es devolvemos falso, los objetos no pueden ser iguales.

instanceof

```
@Override
public boolean equals(Object o)
     if (o  null)
          return false;
     if (!(o instanceof Empleado))
          return false;
     else{

     }
}
```

Si es una instancia de Empleado, sigue la comprobación.

.getClass()

Con getClass() también podemos comparar la clase a la que pertenecen los objetos.

```
@Override
public boolean equals(Object o){
      if (o==null)
            return false;
      if (this.getClass()==o.getClass())
            return false;
      else{

      }
}
```

.getClass()

```
@Override
public boolean equals(Object o)
     if (o==null)
          return false;
     if (this.getClass()==o.getClass())
          return false;
     else{

     }
}
```

Comprobamos si la clase de la instancia es la misma clase del objeto recibido como parámetro.

3 | Casting

Casting

Ahora nos restaría comprobar la igualdad (tener el mismo legajo). Para hacer esta comprobación, vamos a necesitar pedirle a “o”, el legajo para compararlo con el de la instancia. Pero “o” es un Object, o sea, no “sabe” que tiene legajo.

```
@Override
public boolean equals(Object o){
      if (o==null)
            return false;
      if (!(o instanceof Empleado))
            return false;
      else{
      }
}
```

Casting

Asi quedaria el método terminado, pero para mayor comodidad usamos un casteo. Si bien no es necesario, crear un nuevo objeto, es más cómodo para una posterior lectura.

```
@Override
public boolean equals(Object o){
      if (o==null)
            return false;
      if (!(o instanceof Empleado))
            return false;
      else{
            Empleado empleadoAux=(Empleado)o;
            return
this.getLegajo().equals(empleadoAux.getLegajo());
            }
      }
}
```

.getClass()

```
@Override
public boolean equals(Object o)
     if (o==null)
          return false;
     if (this.getClass()==o.getClass())
          return false;
     else{
          Empleado empleadoAux=(Empleado)o;

          if(this.getLegajo().equals(empleadoAux.getLegajo()));
               return true;
          }
     return false;
     }
}
```

Casteamos “o” y lo asignamos a un objeto de tipo Empleado. Con el casting lo que logramos es transformar, para poder asignarlo a un objeto de tipo Empleado y de esta forma usar sus métodos.

.getClass()

```
@Override
public boolean equals(Object o)
      if (o==null)
            return false;
      if (this.getClass()==o.getClass())
            return false;
      else{
            Empleado empleadoAux=(Empleado)o;
            if (this.getLegajo().equals(empleadoAux.getLegajo()));
                  return true;
            }
      return false;
      }
}
```

Comprobamos que tiene el mismo legajo, lo hacemos con equals. Legajo es un atributo de tipo String, por eso, no podemos usar el operador “==”.


DigitalHouse>
Coding School