Métodos comunes de todos los objetos

equals hashcode comparable

Método equals, cuando NO:

- Cada instancia de la clase es propiamente única.
- No hay necesidad para la clase de proveer una igualdad lógica.
- Una superclase ya implementó este método y el mismo es apropiado para la clase.
- La clase es privada y estamos seguros que el método equals nunca será invocado.

Método equals, cuando SI:

- Cuando una clase tiene una igualdad lógica que va más allá de la identidad del objeto.
  Y una superclase no haya implementado este método.

Que establece su contrato?

El método equals implementa una relación de equivalencia. Con las siguientes propiedades:

- Reflexivo: Para cada valor de X no nulo.

  X.equals(X) es **true**.

- Simétrico: Para cada valor de X e Y no nulo.

  X.equals(Y) es **true** si y sólo si Y.equals(X) es **true**.

- Transitivo: Para cada valor de X, Y, Z no nulo.

  Si X.equals(Y) es **true**

  Y Y.equals(Z) es **true**

  Entonces X.equals(Z) debe ser **true**.

- Consistente: Para cada valor de X e Y. Múltiples invocaciones de X.equals(Y) debe retornar consistentemente **true** o consistentemente **false**. Siempre que no se modifique la información utilizada en ambos.

  Para cada valor de X no nulo. X.equals(null) debe retornar **false**.

Receta para un método equals de calidad:

1) Usar el operador == para chequear si el argumento es una referencia a este objeto. Si lo es retorna true.
2) Usar el operador instanceof para chequear si el argumento posee el tipo correcto. Si no, retorna false.
3) Castear el argumento al tipo correcto. Como el casteo se realiza después de un instanceof está garantizado que será satisfactorio.
4) Por cada atributo "significante" en la clase chequear si ese atributo es igual al atributo correspondiente de este objeto. Si todos estos son satisfactorios, retorna true. Caso contrario false.

   Para los atributos primitivos cuyo tipo no es float ni double usar el operador == para comparar.

   Para objetos, llamar al método equals recursivamente.

   Para atributos float usar el método estático compare de la clase Float. Float.compare(float1, float2)

   Para atributos double usar el método estático compare de la clase Double. Double.compare(double1, double2)

Ejemplo método equals

```
public class Telephone {
     private short areaCode;
     private short prefix;
     private short lineNum;
     …
     @Override
     public boolean equals(Object o) {
          if (o == this) return true;
          if (!(o instanceof Telephone)) return false;
          Telephone tp = (Telephone)o;
          return tp.lineNum == lineNum && tp.prefix == prefix && tp.areaCode ==
     areaCode;
     }
     ...
```

Método hashCode:

Siempre se debe hacer Override de hashCode cuando se hace Override de equals.

Que establece su contrato?

- Cuando el método hashCode es invocado en un objeto repetidas veces debe retornar el mismo valor consistentemente.
- Si dos objetos son iguales de acuerdo al método equals, entonces hashCode debe retornar el mismo valor para ambos.
- Si dos objetos son distintos de acuerdo al método equals, no es necesario que hashCode produzca un valor distinto. Sin embargo esto puede mejorar la performance de una hash table.

Cómo NO escribir un método hashCode

```
@Override
public int hashCode() {
    return 42;
}
```

Receta para un buen método hashCode:

1. Declarar una variable entera llamada resultado e inicializarla con el valor de hash code del primer atributo significante en nuestro objeto.
2. Para cada atributo remanente en nuestro objeto hacer lo siguiente:
    a. Si el atributo es de tipo primitivo computar Type.hashCode(atributo). Donde Type es el Tipo correspondiente a la primitiva.
    b. Si el atributo es un objeto y esa clase hace uso de equals de manera recursiva para sus atributos. Usar hashcode recursivamente en el atributo. Si el valor del atributo es null usar 0.
    c. Si el atributo es un array, tratarlo como si cada elemento fuera un atributo separado. Esto significa calcular un hashcode para cada elemento en el array y combinar los valores. Si el array no tiene elementos significantes usar una constante diferente de 0. Si TODOS los elementos son importante usar Arrays.hashCode.
3. Combinar el resultado de esta forma: resultado = 31 * resultado + atributo;

Ejemplo método hashCode

```
public class Telephone {
     private short areaCode;
     private short prefix;
     private short lineNum;
     ...
     public int hashCode() {
          int result = Short.hashCode(areaCode);
          result = 31 * result + Short.hashCode(prefix);
          result = 31 * result + Short.hashCode(lineNum);
          return result;
     }
}
```

Interfaz Comparable

```
public interface Comparable<T> {
    int compareTo(T t);
}
```

Al implementar esta interfaz en una clase estamos indicando que las instancias de dicha clase poseen un **orden natural.**

El contrato de este método especifica que debe devolver lo siguiente:

- Entero negativo si el objeto es menor al especificado por parámetro.
- Cero si el objeto es igual al especificado por parámetro.
- Entero positivo si el objeto es mayor al especificado por parámetro.

Ejemplo método compareTo

```
public class Telephone {
     private short areaCode;
     private short prefix;
     private short lineNum;
     ...
     public int compareTo(Telephone tp) {
          int result = Short.compare(areaCode, tp.areaCode);
          if (result == 0) {
               result = Short.compare(prefix, tp.prefix);
               if (result == 0) result = Short.compare(lineNum, tp.lineNum);
          }
          return result;
     }
...
```