**equals(), hashCode(), ==, compareTo()**

***equals()*** и ***compareTo()*** делают одно и тоже, только *compareTo()* возвращает int (1, -1, 0), а *equals()* возвращает Boolean (true, false).

**==** проверяет на равенство ссылок на один и тот-же объект, а ***equals()*** проверяет на равенство значений.

Когда прописывается собственный ***equals()***, то через if-ы в нем прописываются, обязательные к выполнению , условия.

При переопределении ***equals()*** обязательно должен быть переопределен ***hashCode()***.

Условия, которые должны быть прописаны при переопределении **equals()**:

1. Рефлексивно: для любого ненулевого ссылочного значения x, x.equals (x) должен возвращать true.

2. Симметричено: для любых ненулевых ссылочных значений x и y, x.equals (y) должен возвращать true тогда и только тогда, когда y.equals (x) возвращает true.

3. Транзитивно: для любых ненулевых ссылочных значений X, Y и Z, если x.equals(Y) возвращает True и Y.equals(Z) возвращает True, то x.equals(Z) должен возвращать True.

4. Непротиворечиво: для любых ненулевых значений x и y множественные вызовы x.equals (y) последовательно возвращается true или последовательно возвращается false при условии, что информация, используемая в сравнениях equals для объектов не изменяется.

5. Для любого ненулевого значения x, x.equals (null) должен возвращать false.

@Override

public boolean equals(Object o) {

// self check

if (this == o)

return true;

// null check

if (o == null)

return false;

// type check and cast

if (getClass() != o.getClass())

return false;

Person person = (Person) o;

// field comparison

return Objects.equals(firstName, person.firstName)

&& Objects.equals(lastName, person.lastName);

}

Также при написании можно объединить два условия в одно :

if (o == null || getClass() != o.getClass())

return false;

***hashCode()*** может быть переопределен так:

@Override

public int hashCode() {

return Objects.hash(age, name, sex);

}