Привет! Сегодня мы поговорим об очень важной и интересной теме, а именно — сравнении объектов между собой **equals()** в Java. И действительно, в каких случаях в Java Объект **А** будет равен Объекту **Б**? Давай попробуем написать пример:

**public** **class** Car {

String model;

**int** maxSpeed;

**public** **static** **void** main(String[] args) {

Car car1 = **new** Car();

car1.model = "Ferrari";

car1.maxSpeed = 300;

Car car2 = **new** Car();

car2.model = "Ferrari";

car2.maxSpeed = 300;

System.out.println(car1 == car2);

}

}

Вывод в консоль:

false

Так, стоп. А почему, собственно, эти две машины не равны? Мы задали им одинаковые свойства, но результат сравнения — false. Ответ прост. **Оператор == сравнивает не свойства объектов, а ссылки.** Будь у двух объектов даже 500 одинаковых свойств, результатом сравнения все равно будет false. Ведь ссылки car1 и car2 **указывают на два разных объекта**, на два разных адреса. Представь себе ситуацию со сравнением людей. В мире наверняка есть человек, у которого одинаковые с тобой имя, цвет глаз, возраст, рост, цвет волос и т.д. То есть вы во многом похожи, но все-таки вы не близнецы, и тем более не один и тот же человек. Примерно такую логику применяет оператор ==, когда с его помощью мы пытаемся сравнить два объекта. Но что, если в твоей программе тебе нужна другая логика? Например, если твоя программа симулирует анализ ДНК. Она должна сравнить код ДНК двух людей, и определить, что это близнецы.

**public** **class** Man {

**int** dnaCode;

**public** **static** **void** main(String[] args) {

Man man1 = **new** Man();

man1.dnaCode = 1111222233;

Man man2 = **new** Man();

man2.dnaCode = 1111222233;

System.out.println(man1 == man2);

}

}

Вывод в консоль:

false

Логично, что результат получился тот же самый (ведь мы особо ничего не меняли), но теперь он нас не устраивает! Ведь в реальной жизни анализ ДНК — стопроцентная гарантия того, что перед нами близнецы. Но наша программа и оператор == говорят нам об обратном. Как нам изменить это поведение и сделать так, чтобы в случае совпадения анализов ДНК программа выдавала правильный результат? Для этого в Java был создан специальный метод — **equals()**.

**Метод Equals() в Java**

Как и метод toString(), который мы разбирали ранее, **equals()** принадлежит классу **Object** самому главному классу в Java, от которого происходят все остальные классы. Однако сам по себе equals() никак не изменит поведение нашей программы:

**public** **class** Man {

String dnaCode;

**public** **static** **void** main(String[] args) {

Man man1 = **new** Man();

man1.dnaCode = "111122223333";

Man man2 = **new** Man();

man2.dnaCode = "111122223333";

System.out.println(man1.equals(man2));

}

}

Вывод в консоль:

false

Точно такой же результат, ну и зачем тогда нужен этот метод? :/ Все просто. Дело в том, что сейчас мы использовали этот метод так, как он реализован в самом классе Object. И если мы зайдем в код класса Object и посмотрим, как в нем реализован данный метод и что он делает, то увидим:

**public** **boolean** equals(Object obj) {

**return** (**this** == obj);

}

Вот и причина, почему поведение нашей программы не изменилось! Внутри метода equals() класса Object лежит то же самое сравнение ссылок, ==. Но фишка этого метода в том, что мы можем его переопределить. **Переопределить** — значит написать свой метод equals() в нашем классе Man и сделать его поведение таким, какое нам нужно! Сейчас нас не устраивает, что проверка man1.equals(man2), по сути, делает то же, что и man1 == man2. Вот что мы сделаем в такой ситуации:

**public** **class** Man {

**int** dnaCode;

**public** **boolean** equals(Man man) {

**return** **this**.dnaCode == man.dnaCode;

}

**public** **static** **void** main(String[] args) {

Man man1 = **new** Man();

man1.dnaCode = 1111222233;

Man man2 = **new** Man();

man2.dnaCode = 1111222233;

System.out.println(man1.equals(man2));

}

}

Вывод в консоль:

true

Совсем другой результат! Написав свой метод equals() вместо стандартного, мы добились правильного поведения: теперь если у двух людей одинаковый код ДНК, программа говорит нам: “Анализ ДНК показал, что это близнецы” и возвращает true! Переопределяя метод equals() в своих классах, ты можешь легко создавать нужную логику сравнения объектов. Мы затронули сравнение объектов только в общих чертах. Впереди у нас еще будет отдельная большая [лекция](https://javarush.ru/quests/lectures/questmultithreading.level01.lecture04) на эту тему (можешь бегло прочесть ее уже сейчас, если интересно).

**String compare в Java - Сравнение строк**

Почему мы рассматриваем сравнение строк отдельно от всего остального? Ну, на самом деле, строки в программировании — вообще отдельная песня. Во-первых, если взять все написанные человечеством программы на Java, порядка 25% объектов в них составляют именно они. Поэтому данная тема очень важна. Во-вторых, процесс сравнения строк действительно сильно отличается от остальных объектов. Рассмотрим простой пример:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = **new** String("JavaRush - лучший сайт для изучения Java!");

System.out.println(s1 == s2);

}

}

Вывод в консоль:

false

Но почему false? Строки-то ведь абсолютно одинаковые, слово-в-слово :/ Ты можешь предположить: *это потому что оператор* == *сравнивает ссылки!* Ведь у s1 и s2 разные адреса в памяти. Если тебя посетила такая мысль, то давай переделаем наш пример:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = "JavaRush - лучший сайт для изучения Java!";

System.out.println(s1 == s2);

}

}

Сейчас у нас тоже две ссылки, но вот результат изменился на противоположный: Вывод в консоль:

true

Окончательно запутался? :) Давай разбираться. Оператор == действительно сравнивает адреса в памяти. **Это правило работает всегда и в нем не надо сомневаться.** Значит, если s1 == s2 возвращает true, у этих двух строк одинаковый адрес в памяти. И это действительно так! Настало время познакомиться со специальной областью памяти для хранения строк — пулом строк (String pool) **Пул строк** — область для хранения всех строковых значений, которые ты создаешь в своей программе. Для чего он был создан? Как и говорилось раньше, строки занимают огромную часть от всех объектов. В любой большой программе создается очень много строк. С целью экономии памяти и нужен String Pool — туда помещается строка с нужным тебе текстом, и в дальнейшем вновь созданные ссылки ссылаются на одну и ту же область памяти, нет нужды каждый раз выделять дополнительную память. Каждый раз, когда ты пишешь String = “........”, программа проверяет, есть ли строка с таким текстом в пуле строк. Если есть — новая создана не будет. И новая ссылка будет указывать на тот же адрес в пуле строк, где эта строка хранится. Поэтому когда мы написали в программе

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = "JavaRush - лучший сайт для изучения Java!";

ссылка s2 указывает ровно туда же, куда и s1. Первая команда создала в пуле строк новую строку с нужным нам текстом, а когда дело дошло до второй — она просто сослалась на ту же область памяти, что и s1. Можно сделать хоть еще 500 строк с таким же текстом, результат не изменится. Стоп. Но почему тогда ранее у нас не сработал этот пример?

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = **new** String("JavaRush - лучший сайт для изучения Java!");

System.out.println(s1 == s2);

}

}

Думаю, интуитивно ты уже догадываешься в чем причина :) Попробуй предположить, прежде чем читать дальше. Ты видишь, что эти две строки были созданы по-разному. Одна — с помощью оператора new, а вторая без него. Именно в этом кроется причина. Оператор new при создании объекта **принудительно выделяет для него новую область в памяти**. И строка, созданная с помощью new, не попадает в String Pool: она становится отдельным объектом, даже если ее текст полностью совпадает с такой же строкой из String Pool’a. То есть если мы напишем такой код:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = "JavaRush - лучший сайт для изучения Java!";

String s3 = **new** String("JavaRush - лучший сайт для изучения Java!");

}

}

В памяти это будет выглядеть вот так: И каждый раз при создании нового объекта через new в памяти будет выделяться новая область, даже если текст внутри новых строк будет одинаковым! С оператором == вроде разобрались, а что с нашим новым знакомым — методом equals()?

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = **new** String("JavaRush - лучший сайт для изучения Java!");

System.out.println(s1.equals(s2));

}

}

Вывод в консоль:

true

Интересно. Мы точно знаем, что s1 и s2 указывают на разные области в памяти. Но, тем не менее, метод equals() говорит, что они равны. Почему? Помнишь, выше мы говорили о том, что метод equals() можно переопределить в своем классе, чтобы он сравнивал объекты так, как тебе нужно? С классом String так и поступили. У него есть переопределенный метод equals(). И сравнивает он не ссылки, а именно последовательность символов в строках. И если текст в строках одинаковый, неважно, как они были созданы и где хранятся: в пуле строк, или в отдельной области памяти. Результатом сравнения будет true. Кстати, Java позволяет корректно сравнивать строки без учета регистра. В обычной ситуации, если написать одну из строк, например, капсом, то результатом сравнения будет false:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = **new** String("JAVARUSH - ЛУЧШИЙ САЙТ ДЛЯ ИЗУЧЕНИЯ JAVA!");

System.out.println(s1.equals(s2));

}

}

Вывод в консоль:

false

Для этого случая в классе String имеется метод equalsIgnoreCase(). Если в твоем сравнении главным является именно последовательность конкретных символов, а не их регистр, можно применить его. Например, это будет полезно при сравнении двух почтовых адресов:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String address1 = "г. Москва, ул. Академика Королева, дом 12";

String address2 = **new** String("Г. МОСКВА, УЛ. АКАДЕМИКА КОРОЛЕВА, ДОМ 12");

System.out.println(address1.equalsIgnoreCase(address2));

}

}

В данном случае очевидно, что речь идет об одном и том же адресе, поэтому использование метода equalsIgnoreCase() будет верным решением.

**Метод String.intern()**

У класса String есть еще один хитрый метод — intern(); Метод intern() напрямую работает со String Pool’ом. Если ты вызываешь метод intern() у какой-то строки, он:

* Смотрит, есть ли строка с таким текстом в пуле строк
* Если есть — возвращает ссылку на нее в пуле
* Если же нет — помещает строку с этим текстом в пул строк и возвращает ссылку на нее.

Применив метод intern() к ссылке на строку, которая создавалась через new, мы можем сравнивать ее со ссылкой на строку из String Pool’a через оператор ==.

**public** **class** Main {

**public** **static** **void** main(String[] args) {

String s1 = "JavaRush - лучший сайт для изучения Java!";

String s2 = **new** String("JavaRush - лучший сайт для изучения Java!");

System.out.println(s1 == s2.intern());

}

}

Вывод в консоль:

true

Раньше, когда мы сравнивали их без intern(), результат был равен false. Теперь же метод intern() проверил, есть ли строка с текстом "JavaRush — лучший сайт для изучения Java!" в пуле строк. Разумеется, она там есть: мы создали ее, когда написали

String s1 = "JavaRush — лучший сайт для изучения Java!";

Была проведена проверка, что ссылка s1 и ссылка, возвращенная методом s2.intern() указывают на одну область в памяти, и, конечно, так оно и есть:) Подводя итоги, запомни и используй главное правило: Для сравнения строк **ВСЕГДА** используй метод equals()! Сравнивая строки, ты почти всегда имеешь в виду сравнение их текста, а не ссылок, областей памяти и прочего. Метод equals() делает именно то, что тебе нужно.