Quando vamos comparar textos no Java (Strings), é inevitável pensarmos que isso é feito da mesma forma que comparar números. E nessa hora você certamente fica em tentado em fazer algo desse tipo:

1. String s1 = "a";
2. String s2 = "a";
3. **if** (s1 == s2) {
4. // faça algo
5. } else {
6. // faça outra coisa
7. }

Só que, por mais estranho que possa parecer, **isso não funciona**. Ou melhor, não funciona às vezes, o que é ainda pior…

Bom, mas deixa eu te explicar o que está por trás de tudo isso…

Dá só uma olhada neste código:

1. String s1 = "abc";
2. String s2 = "abc";
3. System.out.**println**(s1 == s2);

Ao executar o código acima, o resultado é true (verdadeiro), como é de se esperar. Afinal de contas, estamos pensando na comparação do texto *“abc”* com o texto *“abc”*, e eles são iguais…

Agora olha só o que acontece se a gente fizer uma pequena modificação:

1. String s1 = "abc";
2. String s2 = new **String**("abc");
3. System.out.**println**(s1 == s2);

Ao executar, você vai perceber que o resultado agora é false (falso).

Isso mesmo: **aos olhos do Java, as strings *“abc”* e *“abc”* são diferentes**.

Isso realmente parece não fazer sentido, mas agora eu vou te mostrar por que o Java tira essa conclusão.

Conheça o Pool de Strings do Java!

Em algumas linguagens de programação, o tipo String faz parte do conjunto de tipos primitivos da linguagem.

Mas no Java String é uma classe, o que significa que os objetos de String que são criados seguem a mesma regra de armazenamento em memória das outras classes do Java. Eles são armazenadas no *heap* e as variáveis do tipo String referenciam essas áreas de memória no *heap* onde ficam os objetos (não vou aprofundar essa questão aqui pra não perder o foco, então fica pra outro post).

Mas isso tem um **efeito colateral**…

Como strings são MUITO usadas em qualquer tipo de aplicação, isso pode causar um DESPERDÍCIO ABSURDO de memória.

Imagine se o seu sistema usa a string *“A”* em 20 partes do código. Será que é preciso existir 20 objetos de String na memória pra guardar a mesma informação? Pois é, totalmente desnecessário…

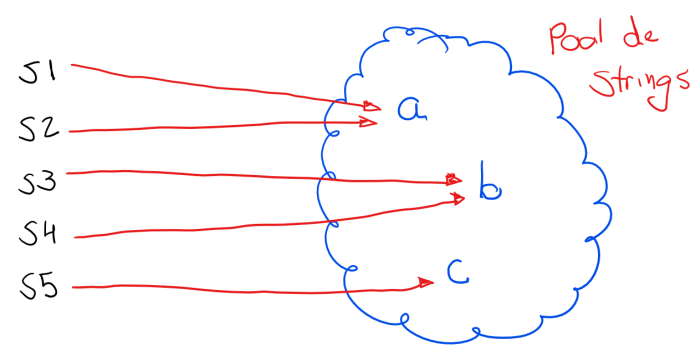
Pensando nisso, os projetistas do Java tomaram uma decisão: criaram um **mecanismo de compartilhamento de strings**, que eles chamaram de ***pool* de strings**.

O *pool* é uma área de memória que armazena strings únicas. Quando você faz algo como isso aqui…

1. String s1 = "a";
2. String s2 = "a";
3. String s3 = "b";
4. String s4 = "b";
5. String s5 = "c";

… são criadas 3 strings no *pool*: *“a”*, *“b”* e *“c”*. E cada variável (s1, s2, s3, etc.) referencia um desses objetos que estão no *pool*.

Graficamente, seria algo como o desenho abaixo:



*(Aliás, o fato de objetos String no pool serem compartilhados é o motivo pelo qual strings em Java são imutáveis. Mas isso é uma discussão pra outro momento…)*

Quando o operador de igualdade funciona

O operador de igualdade do Java (*==*) **compara o conteúdo** de duas variáveis.

Quando usamos tipos primitivos (int, double, boolean, etc.) ele se comporta sempre como o esperado, pois nos tipos primitivos o que está dentro da variável é realmente o valor dela.

Mas no caso de classes (como a String), o conteúdo da variável é uma **referência pra um objeto na memória** (que está no *heap*).

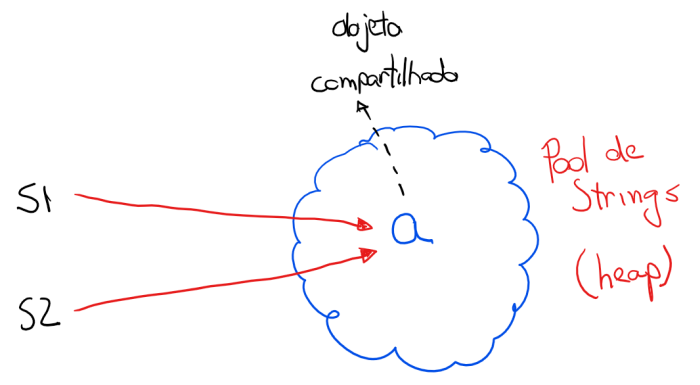
Portanto, a conclusão que chegamos é que, no caso de strings, **o operador de igualdade não compara o conteúdo das strings, mas sim as referências de memória** (que é o que está efetivamente sendo armazenado pelas variáveis).

Ok, ok… Confesso que isso tudo pode ser meio **confuso** até você se acostumar, então vou dar um exemplo.

Vamos relembrar o primeiro código que eu te mostrei no post:

1. String s1 = "a";
2. String s2 = "a";

Quando você faz isso, o que está acontecendo na memória é  isso aqui:



Note que temos **1 único objeto** String na memória (com o valor *“a”*) e **2 variáveis** que referenciam o mesmo objeto.

Portanto, quando testamos s1 == s2, o resultado é true. Não porque os textos são iguais, por porque ambas as variáveis (s1 e s2) referenciam o mesmo objeto.

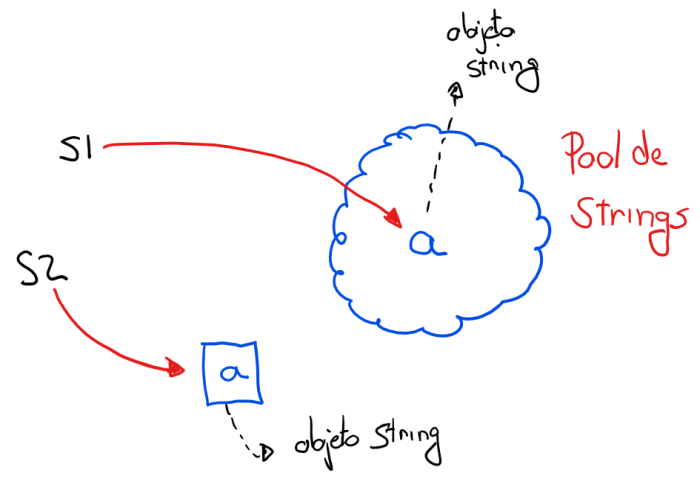
De novo: lembre que o == compara o conteúdo das variáveis, e o conteúdo aqui são referências pra objetos que estão no *pool*.

Quando o operador de igualdade NÃO funciona

Como eu já te mostrei antes, o == pode não se comportar como você espera. É o caso deste código aqui:

1. String s1 = "a";
2. String s2 = new **String**("a");

Quando esse código é executado, é assim que as coisas ficam na memória:



Note que s1 referencia a string *“a”* que está no *pool*; mas s2 referencia outro objeto String, com o conteúdo *“a”,*mas que está fora do *pool*.

Isto acontece porque a criação do objeto String, através do operador new(), faz com que seja criado um novo objeto em uma área de memória fora do *pool*.

Olhando o desenho, dá pra ver claramente que os conteúdos de s1 e s2 são diferentes (cada variável referencia um objeto diferente). Portanto a comparação s1 == s2 vai retornar, neste caso, false.

Afinal: Como Comparar Strings no Java?

Todo esse problema acontece pelo fato do == do Java comparar referências de memória ao invés de comparar o texto armazenado no objeto String.

Então a grande pergunta é: **será que existe uma forma de comparar o texto da String, ao invés de comparar referências de memória?**

E a resposta é **SIM, EXISTE**!

A partir de agora, TODAS as vezes que você precisar comparar strings no Java você vai usar o método equals().

O método equals() é definido na classe Object do Java. Como todas as classes do Java herdam diretamente ou indiretamente e de Object, significa que todas as classes do Java têm este método.

Na classe String, este método foi implementado de forma a comparar o texto, portanto ele **resolve o nosso problema**! Veja:

1. String s1 = "a";
2. String s2 = "a";
3. System.out.**println**(s1.**equals**(s2));

O resultado do código acima é true!

Agora olhe o código abaixo:

1. String s1 = "a";
2. String s2 = new **String**("a");
3. System.out.**println**(s1.**equals**(s2));

A resposta aqui também é true!

O fato do equals() comparar os textos (a comparação é feita de *“a”* com *“a”*), faz com que o resultado seja verdadeiro sempre, independentemente de como estão os objetos na memória.

*“Mas Carlos, se eu não usar o new* ***String****() não vou ter problemas e posso usar == pra comparar strings. Estou certo?”*

Não: você está **ERRADO**!

Mesmo que você não use o new **String**(), em muitos casos você não sabe como a String foi criada.

É muito comum chamarmos métodos de outras bibliotecas/*frameworks*/APIs ou até códigos que foram criados por outros desenvolvedores.

Então não pense duas vezes: **use o equals() em todas as situações!**