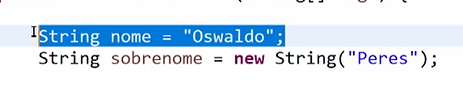
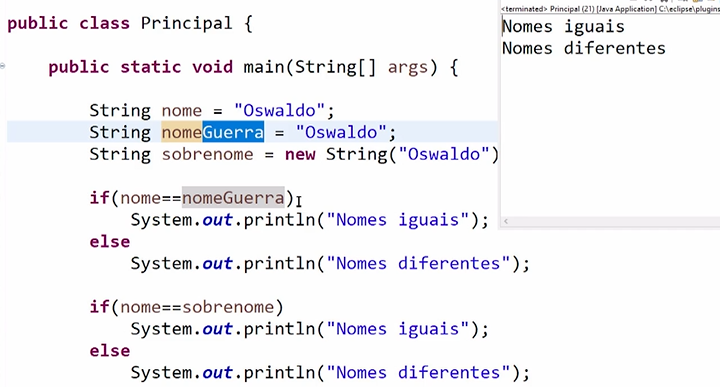
**Classe String:**

Usado quase sempre como tipo primitivo (int, double, char, boolean...) por isso não tem o New String. MAS NÃO É TIPO PRIMITIVO



Por ser uma string, ele tem vários métodos! Ex: .length() = quantidade de carecteres.

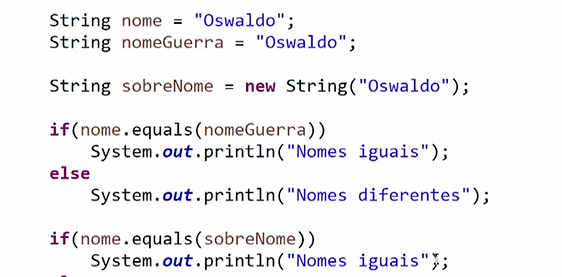


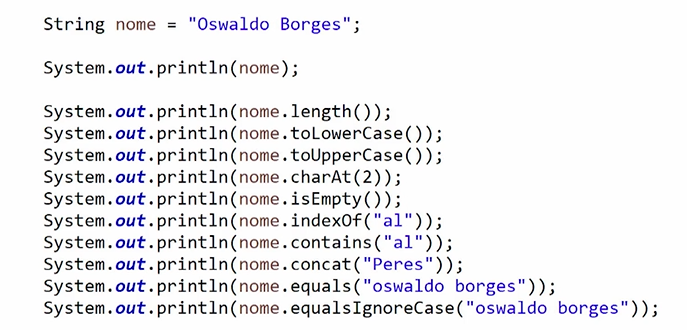
Por ser um objeto, a string é **imutável**, não mudam após a definição. O sistema operacional entende que você quer mudar, ele apaga a string anterior e cria uma nova string (quando você edita por exemplo).

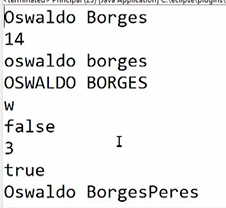
Por que nome e sobrenome deu diferente? Bom o java entende que criou o nome e no nomeguerra você ta colocando o mesmo nome? Que mane, por que não usou a mesma variável? Então ele faz a referencia, apontando para o mesmo conteúdo. Entao no nome==nomeGuerra ele **compara a localização** do conteúdo. Quando deu new String é a localização do qual você criou com mão própria e não a que o java criou.

**Métodos:**

.equals (faz comparação de CONTEUDO)







Possíveis duvidas:

.isEmpty = esta vazio?.

.indexOf = índice de X elemento.

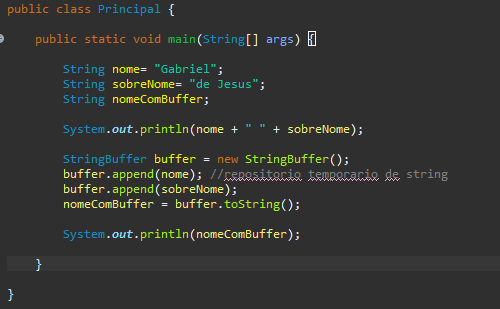
.contains = contem X?.

.concat = junta X elemento.

.equals= X é igual a Y.

**StringBuffer:**

Ao invés de ir criando strings, colocamos como num buffer e depois cria de uma vez somente. Como um vetor dinâmico na memoria.



Vantagem é que no 1º caso do nome completo eu estou concatenando com “+” e isso implica em um esforço maior de processamento, pois cria um novo objeto já que a strings são imutaveis.

**StringBuilder:**

Faz a mesma coisa que a stringBuffer mas tem uma diferença, ela não é safe trad. Meu programa (rotina) pode ter 2 programa ocorrendo entre si. E como posso resguardar? No caso de String tem essa StringBuilder.

