04

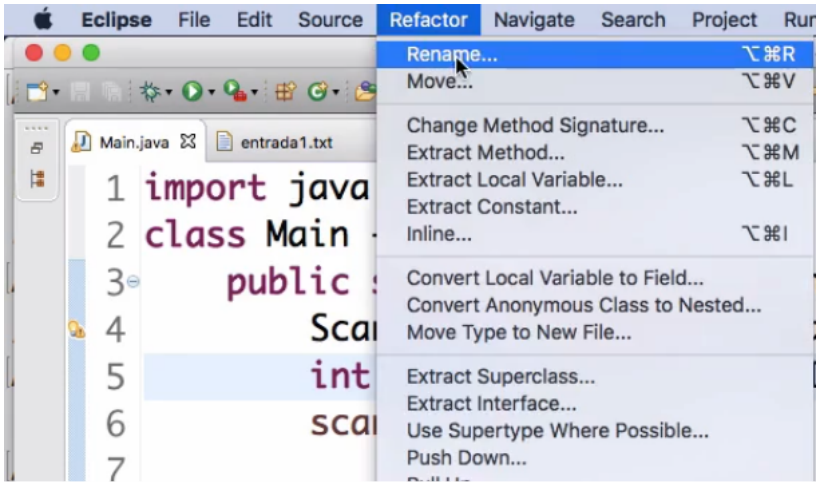
Reduzindo o código - Parte 2

Transcrição

A primeira coisa que faremos é cortar os enters que estão sobrando no final do código, bem como as linhas em branco entre as linhas de código. Além disso, em nenhum momento do desafio diz-se que a classe precisa ser pública. Assim, tiraremos o `public` também.

```
import java.util.Scanner;
class Main {
    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        scanner.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " ");
            String palavra = scanner.nextLine();
            if(ehPalindromo(palavra)) {
              System.out.println("\"YES\"");
            } else {
              System.out.println("\"NO\"");
            }
        }
    }
    private static boolean ehPalindromo(String palavra) {
      String invertido = new StringBuilder(palavra).reverse().toString;
      return palavra.equals(invertido);
    }
}
```

Lembra-se da variável `casos` ? No problema, é citada como variável `t` . Vamos renomeá-la, para economizar mais alguns caracteres, selecionando a variável e clicando em `Refactor > Rename...` , no menu superior do Eclipse.



De pouco em pouco vamos ganhando caracteres. Vamos ver quantos?

```
Alura-Azul:bin alura$ wc ../src/Main.java
      34      64     600 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            22      63     573 ../src/Main.java
```

Conseguimos diminuir mais `27` caracteres, quase 5% a menos. Precisamos diminuir ainda mais. A próxima coisa na qual podemos mexer é no primeiro `sysout` . Já podemos incluir as primeiras aspas do `"YES"` e do `"NO"` , e tirar essa pequena repetição dos outros `sysout` s.

```
import java.util.Scanner;
class Main {
    public static void main (String [] args) {
        Scanner scanner = new Scanner(System.in);
        int casos = scanner.nextInt();
        scanner.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String palavra = scanner.nextLine();
            if(ehPalindromo(palavra)) {
              System.out.println("YES\"");
            } else {
              System.out.println("NO\"");
            }
        }
    }
    private static boolean ehPalindromo(String palavra) {
      String invertido = new StringBuilder(palavra).reverse().toString;
      return palavra.equals(invertido);
    }
}
```

Vamos ver quantos caracteres ainda precisamos tirar?

```
Alura-Azul:bin alura$ wc ../src/Main.java
            22      63     573 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            20      63     566 ../src/Main.java
```

Podemos abreviar mais variáveis para diminuir ainda mais o código. Vamos abreviar `scanner` para `s` , novamente usando `Refactor > Rename...` .

```
import java.util.Scanner;
class Main {
    public static void main (String [] args) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String palavra = scanner.nextLine();
            if(ehPalindromo(palavra)) {
              System.out.println("YES\"");
```

```
            } else {
              System.out.println("NO\"");
            }
        }
    }
    private static boolean ehPalindromo(String palavra) {
      String invertido = new StringBuilder(palavra).reverse().toString;
      return palavra.equals(invertido);
    }
}
```

O código começa a ficar menos legível. Novamente reforço: esse tipo de estratégia não serve para a vida real. Não é útil ter códigos confusos. Nós temos um método extraído direitinho, o `ehPalindromo`. Mas, infelizmente, o `StringBuilder` não tem um método que verifique se uma palavra é palíndromo. Mas ele tem o `equals()` e o `reverse()`. O que precisamos fazer é pegar a `palavra`, inverter, transformar em string e comparar com seu inverso. E isso já está escrito, vamos apenas subir para o meio do restante do código, em uma só linha, que será:

```
if(new StringBuilder(palavra).reverse(). toString().equals(palavra))
```

E podemos adicionar essa linha que substitui todo o `ehPalindromo` no primeiro bloco de código.

```
import java.util.Scanner;
class Main {
    public static void main (String [] args) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String palavra = scanner.nextLine();
            if(new StringBuilder(palavra).reverse(). toString().equals(palavra)) {
              System.out.println("YES\"");
            } else {
              System.out.println("NO\"");
            }
        }
    }
}
```

Essa uma prática nojenta, que ninguém deveria usar. Tiramos o método, que deixava claro o que deveria acontecer. Se um usuário novo bater o olho nesse código, boa sorte para ele. Vamos ver quanto avançamos no desafio?

```
Alura-Azul:bin alura$ wc ../src/Main.java
            20      63     566 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     420 ../src/Main.java
```

Ainda precisamos tirar quase metade disso. Podemos renomear `palavra` como `p`.

```
import java.util.Scanner;
class Main {
```

```
    public static void main (String [] args) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String p = scanner.nextLine();
            if(new StringBuilder(p).reverse().toString().equals(p)) {
              System.out.println("YES\"");
            } else {
              System.out.println("NO\"");
            }
        }
    }
}
```

Testando novamente, temos:

```
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     420 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     402 ../src/Main.java
```

Diminuímos um pouco mais.Será que o programa ainda funciona?

```
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     402 ../src/Main.java
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
```

Será que o `toString` depois do `StringBuilder` é realmente necessário? Tentaremos tirá-lo.

```
import java.util.Scanner;
class Main {
    public static void main (String [] args) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String p = scanner.nextLine();
            if(new StringBuilder(p).reverse().equals(p)) {
              System.out.println("YES\"");
            } else {
              System.out.println("NO\"");
            }
        }
    }
}
```

Será que o programa ainda funciona? Testando no terminal:

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "NO"
2 "NO"
3 "NO"
```

Deu errado. Precisaremos voltar esse trecho. Relendo o código, tudo parece bem essencial.

```
import java.util.Scanner;
class Main {
    public static void main (String [] args) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String p = scanner.nextLine();
            if(new StringBuilder(p).reverse().toString().equals(p)) {
              System.out.println("YES\"");
            } else {
              System.out.println("NO\"");
            }
        }
    }
}
```

Podemos não receber o `String [] args` ? Existem padrões para criar o método `Main` que nos impedem de tirar esse pedaço. Se o tirarmos, dá erro no terminal

```
Alura-Azul:bin alura$ java Main < entrada2.txt
Error: Main method not foung in class Main, please define the main method as:
    public static void main(String [] args)
or a JavaFX application cass must extend javafx.application.Application
```

O `args` precisa ser recebido. Mas não precisar ser chamado de `args` . Assim, mudaremos para apenas `a` .

```
import java.util.Scanner;
class Main {
    public static void main (String [] a) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            System.out.print(i + " /"");
            String p = scanner.nextLine();
            if(new StringBuilder(p).reverse().toString().equals(p)) {
              System.out.println("YES\"");
            } else {
              System.out.println("NO\"");
            }
        }
```

```
    }
}
```

Testaremos novamente no terminal.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
```

Todas as variáveis estão com apenas um caractere; é o máximo que conseguimos comprimir o código em caracteres. Vamos ver quantos ainda precisamos eliminar?

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     399 ../src/Main.java
```

Para tirar mais caracteres, vamos tirar todos os que estão em branco nos tabs. Ficará horrível, mas deve ajudar bastante.

```
import java.util.Scanner;
class Main {
public static void main (String [] a) {
Scanner s = new Scanner(System.in);
int casos = s.nextInt();
s.nextLine();
for(int i = 1; i < casos+1; i++){
System.out.print(i + " /"");
String p = scanner.nextLine();
if(new StringBuilder(p).reverse().toString().equals(p)) {
System.out.println("YES\"");
} else {
System.out.println("NO\"");
}
}
}
}
```

Agora contaremos quantos caracteres ficaram.

```
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     399 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            16      50     363 ../src/Main.java
```

Agora tiraremos alguns caracteres pontuais. Por exemplo, o `else` e o `if` não precisam de chaves se estão em uma única linha.

```
import java.util.Scanner;
class Main {
public static void main (String [] a) {
Scanner s = new Scanner(System.in);
int casos = s.nextInt();
s.nextLine();
for(int i = 1; i < casos+1; i++){
System.out.print(i + " /"");
String p = scanner.nextLine();
if(new StringBuilder(p).reverse().toString().equals(p)) System.out.println("YES\"");
else System.out.println("NO\"");
}
}
}
```

Vamos conferir se ainda está funcionando.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
Alura-Azul:bin alura$ wc ../src/Main.java
            13      45     355 ../src/Main.java
```

Ainda falta tirar `105` caracteres. Podemos ir eliminando espaços entre elementos do código, como em `System.out.print(i + " /"");` . Ficará `System.out.print(i+" /"");` . Pense que os 7 caracteres dentro dos parênteses rodam para os dois casos. Porém, se eles fossem impressos direto nos dois casos, poderemos eliminar essa linha inteira. Vamos ver como fica?

```
import java.util.Scanner;
class Main {
public static void main (String [] a) {
Scanner s = new Scanner(System.in);
int casos = s.nextInt();
s.nextLine();
for(int i = 1; i < casos+1; i++){
String p = scanner.nextLine();
if(new StringBuilder(p).reverse().toString().equals(p)) System.out.println(i+"\ "YES\"");
else System.out.println(i+" \"NO\"");
}
}
}
```

Embora tenhamos duplicado um pedaço do código, diminuímos o número de caracteres, pois repetimos apenas 5 caracteres em cada ocasião (`i+" \`), e a linha que eliminamos tinha mais que isso. Vamos testar novamente se está funcionando e o número de caracteres.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
```

```
Alura-Azul:bin alura$ wc ../src/Main.java
            12      43     336 ../src/Main.java
```

Ainda faltam `86` caracteres. Vamos eliminar os demais espaços desnecessários.

```
import java.util.Scanner;
class Main{
public static void main(String[]a){
Scanner s=new Scanner(System.in);
int casos=s.nextInt();
s.nextLine();
for(int i=1;i<casos+1;i++){
String p=scanner.nextLine();
if(new StringBuilder(p).reverse().toString().equals(p))System.out.println(i+"\ "YES\"");
else System.out.println(i+" \"NO\"");
}
}
}
```

Tiramos todos os espaços que podíamos sem prejudicar o código. Vamos ver o nosso avanço.

```
Alura-Azul:bin alura$ wc ../src/Main.java
            12      43     336 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            12      27     320 ../src/Main.java
```

Estamos nos aproximando dos `250` , mas ainda falta eliminar `70` caracteres. Do jeito que o nosso código está, fica um pouco difícil eliminá-los. Podemos remover as quebras de linha e deixar tudo em uma linha só.

```
import java.util.Scanner;class Main{public static void main(String[]a){Scanner s=new Scanner(Sy
```

Agora está tudo **muito** compactado. Será que conseguimos?

```
Alura-Azul:bin alura$ wc ../src/Main.java
            12      27     320 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
            0      15     307 ../src/Main.java
```

Ainda temos `57` caracteres a mais do que o desafio pede. Vamos ver se o programa ainda roda.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
```

Continua funcionando, mas esse é o mínimo de caracteres que conseguimos nessa abordagem em Java. Se dermos `Ctrl + Shift + F` vamos desfazer todo o trabalho que tivemos com o espaço, mas ao menos teremos o código legível.

```
import java.util.Scanner;
class Main {
    public static void main (String [] a) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            String p = scanner.nextLine();
            if(new StringBuilder(p).reverse().toString().equals(p)) {
              System.out.println(i + " \"YES\"");
            } else {
              System.out.println(i + " \"NO\"");
            }
        }
    }
}
```

Está legível apenas em relação à formatação, porque quanto ao nome das variáveis está terrível. Tiramos muitas coisas que contribuíam com a manutenção do código a longo prazo.

Não conseguimos cumprir o desafio. E o que aprendemos com isso? Que o objetivo de ninguém na vida real é digitar o mínimo possível. Porque, mesmo se escrevermos o mínimo mantendo a formatação, o código fica ilegível. Não conseguimos saber facilmente o que ele fez, é preciso analisar linha a linha.

Se tentássemos mais um pouco, poderíamos diminuir o `sysout` do nosso algoritmo usando um operador ternário, e deixando o código ainda mais nojento. O código que construiríamos para isso dependeria do trecho abaixo ser verdadeiro (`?`):

```
new StringBuilder(p).reverse().toString().equals(p)
```

Se ele for verdadeiro, imprimiremos `" \"YES\""` , senão (`:`) `" \"NO\""` .

```
new StringBuilder(p).reverse().toString().equals(p) ? " \"YES\"" : " \"NO\""
```

Imprimiremos um `sysout` , acrescentando o `i +` .

```
System.out.println(i +
  (new StringBuilder(p).reverse().toString().equals(p) ? " \"YES\"" : " \"NO\"")
  );
```

Esse código é ainda mais feio que o anterior, mas o substituirá por ser mais curto.

```
import java.util.Scanner;
class Main {
    public static void main (String [] a) {
        Scanner s = new Scanner(System.in);
        int casos = s.nextInt();
        s.nextLine();
        for(int i = 1; i < casos+1; i++){
            String p = scanner.nextLine();
```

```
            System.out.println(i +
              (new StringBuilder(p).reverse().toString().equals(p) ? " \"YES\"" : " \"NO\"")
              );
        }
    }
}
```

Será que o programa ainda funciona? Vamos conferir no terminal, já verificando o número de caracteres.

```
Alura-Azul:bin alura$ java Main < entrada2.txt
1 "YES"
2 "YES"
3 "NO"
Alura-Azul:bin alura$ wc ../src/Main.java
           12      48     332 ../src/Main.java
```

Conseguimos diminuir mais um pouco graças a um operador ternário nojento, que é fim de carreira por ser muito difícil de entender. Podemos tentar apagar os espaços novamente, e isso precisa nos economizar `82` caracteres. Tirarei primeiro os espaços, e depois os enters, conferindo se não deixei nenhum espaço passar. Ficará assim:

```
import java.util.Scanner;class Main{public static void main(String[]a){Scanner s=new Scanner(Sy
```

Será que finalmente conseguimos?

```
Alura-Azul:bin alura$ wc ../src/Main.java
           12      48     332 ../src/Main.java
Alura-Azul:bin alura$ wc ../src/Main.java
           0      14     279 ../src/Main.java
```

Ainda temos `29` caracteres a mais, em um código que é obviamente inviável, mesmo que formatado. Supondo que você está em uma maratona, quer escrever o mais rápido possível e já escreveu esse código 300 vezes, é aceitável escrevê-lo direto assim. Do contrário, as chances de você errar fazendo dessa maneira é infinita, ou seja aproxima-se de um (100%). Na prática, não escrevemos esse código, o escrevemos um pouco mais bonito. Lembrando que esse código avacalhado não serve como código de produção no mundo real, o que serve é o primeiro que fizemos, com as variáveis devidamente nomeadas. Sem operador ternário em uma linha de `sysout`, sem variáveis abreviadas.

O que eu queria passar para você nesse problema é o limite da "otimização". Para um programador, otimização não é escrever o mínimo possível, como o desafio nos pede. Otimização é escrever um código que será compreensível a longo prazo. Tenho minhas dúvidas se esse código é compreensível agora. Mostre-o para um amigo programador e veja se ele consegue entender o que ele faz. Não explique o que é um palíndromo e veja quanto tempo ele precisará para entender o que o código faz.

E na outra variação, em que o código estava extraído bonito, com as variáveis nomeadas? De quanto tempo precisamos para entender aquele código? No dia a dia, um programador mais pensa que escreve. É pensar para fazer direito, não escrever o mínimo. É essa a minha dica.

Não conseguiremos enviar esse exercício para o Spoj com Java, porque o mínimo de caracteres nessa abordagem é `279`. Até a próxima!