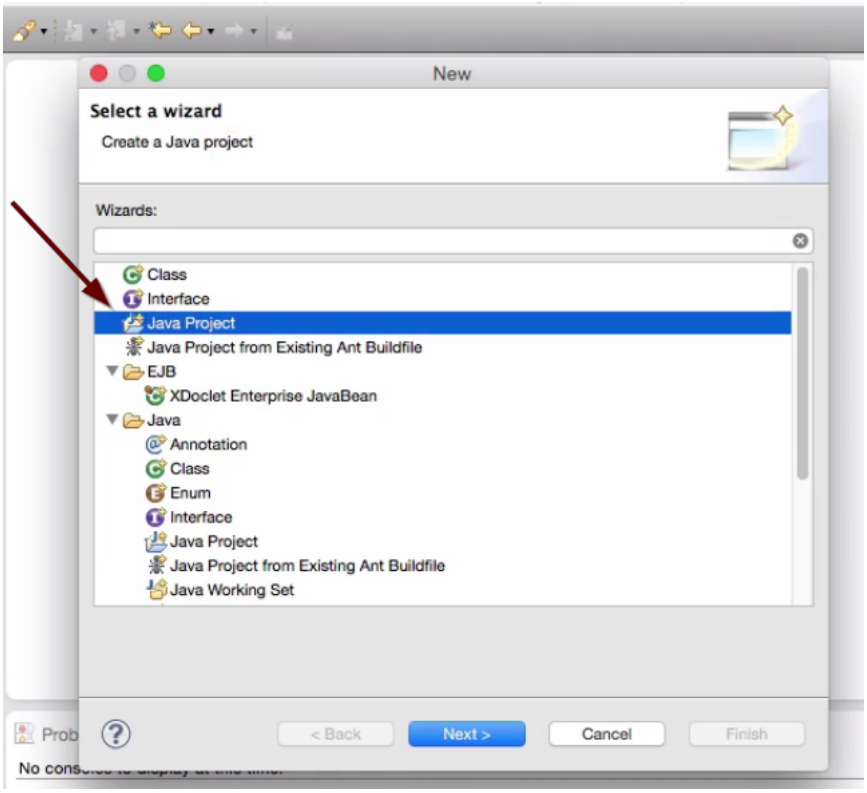
02

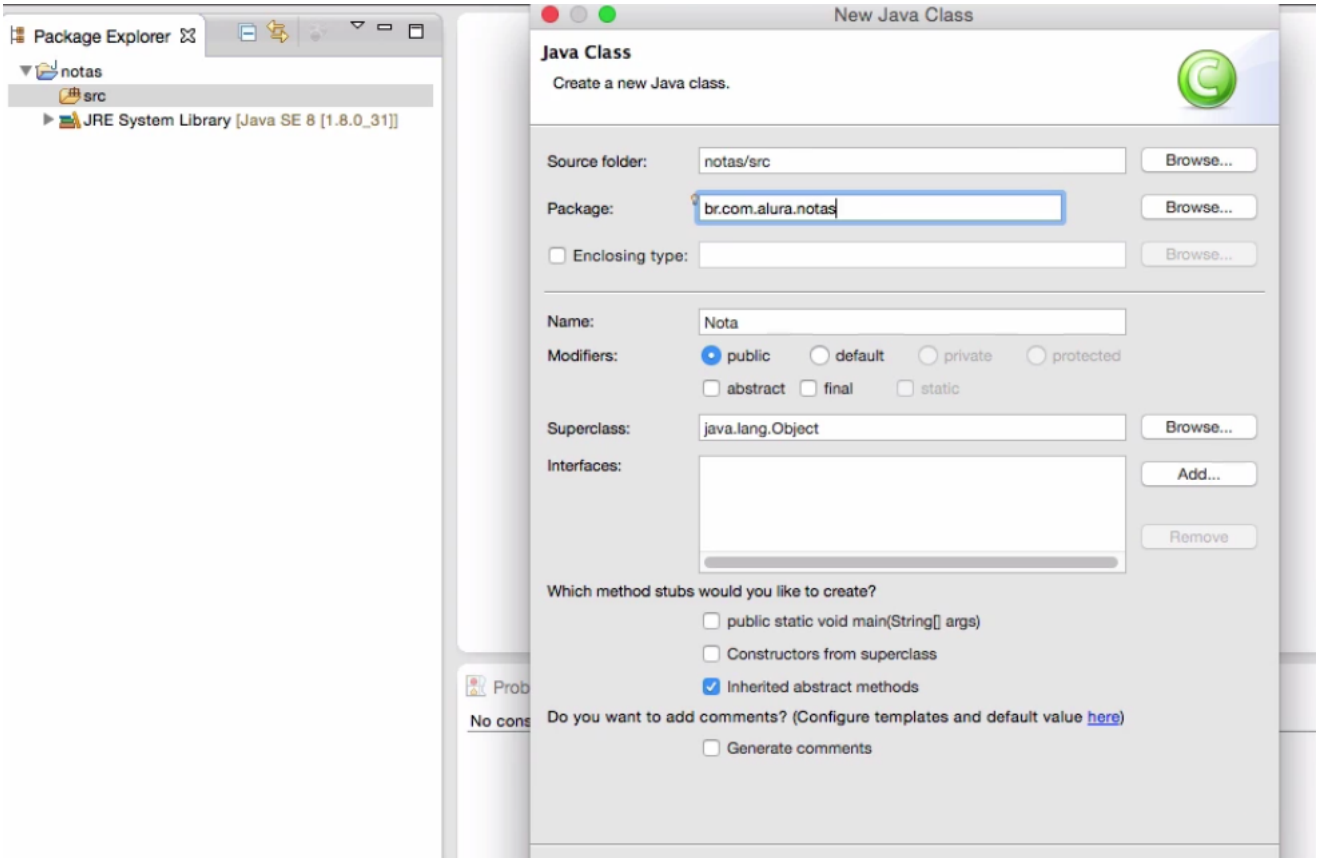
Implementando o algoritmo que junta

Transcrição

Vamos criar o nosso projeto, que será um *Java project*.



Ele será chamado **Notas**. Dentro, teremos um classe para representar cada uma das notas que foram tiradas pelos alunos. A classe irá receber o nome **Nota**, no pacote **br.com.alura.notas**.



Esta nota tem o nome do aluno (`String aluno`) e tem o valor da nota, que pode sr quebrado (`valor`), como 7,5 ou 9,7, por exemplo. O valor será entre 0 e 10.

```
package br.com.alura.notas;

public class Nota {

    private String aluno;
    private double valor;
}
```

Quero adicionar um construtor que receba o o nome do aluno e o valor das notas. Ele aceitará tanto o aluno, como o valor:

```
public Nota(String aluno, double valor) {
    this.aluno = aluno;
    this.valor = valor;
}
```

Iremos criar o *getter* apenas quando for necessário. Momentaneamente, é o suficiente. Fecharemos esta classe e em seguida, criaremos um arquivo de teste, porque criar diversas notas, já ordenadas por dois professores. Para depois pedir para o programa reunir estas notas. Iremos implementar a **junção**, quando fazemos o ***merge***. Vamos juntar, fundir os dois grupos de notas ordenadas.

Iremos fazer o `TestaMerge` , vamos testar fundir as listas, com o método `main` . Vamos trabalhar dentro desta classe, digitando diversas notas de alunos.

```
public class TestaMerge {
```

```
    public static void main(String [] args) {
        mariana 5
        andre 4
        paulo 9
        carlos 8.5
        juliana 6.7
        jonas 3
        lucia 9.3
        ana 10
        guilherme 7
    }

}
```

Temos disponíveis as notas que iremos trabalhar. Os alunos estão divididos em dois grupos. No primeiro, teremos: Mariana, André, Paulo e Carlos. No segundo, teremos: Juliana, Jonas, Lucia, Ana e Guilherme. O que queremos fazer é juntar os elementos em uma única lista.

O primeiro grupo é do professor Maurício. Então, iremos criar um *array* indicando isto:

```
public class TestaMerge {

    public static void main(String [] args) {
        Nota[] notasDoMauricio = {
            new Nota("mariana", 5),
            new Nota("andre", 4),
            new Nota("paulo", 9),
            new Nota("carlos", 8.5)
        };
    }
}
```

O professor Maurício foi gentil e já ordenou as notas, do menor para o maior.

```
public class TestaMerge {

    public static void main(String [] args) {
        Nota[] notasDoMauricio = {
        new Nota("andre", 4),
            new Nota("mariana", 5),
            new Nota("carlos", 8.5),
            new Nota("paulo", 9)

        };
    }
}
```

Recebemos também as notas do segundo grupo, que é do professor Alberto. Vamos indicar no código, que as notas são Alberto (`notasDoAlberto`) :

```
    Nota[] notasDoAlberto = {
    juliana 6.7
    jonas 3
    lucia 9.3
    ana 10
    guilherme 7
}
```

As notas do Alberto também já foram entregues em ordem. O professor já fez um *Insertion Sort* ou um *Selection Sort* e recebemos a ordenação pronta.

```
    Nota[] notasDoAlberto = {
        new Nota("jonas", 3),
        new Nota("juliana", 6.7),
        new Nota("guilherme", 7),
        new Nota("lucia", 9.3),
        new Nota("ana", 10)

    };
```

Então, temos os dois grupos de nota ordenados. O faremos agora? Iremos juntar os dois grupos, fundiremos as duas coleções de itens ordenados. Vamos mostrar que somos capazes de unir todos os elementos em uma única lista ordenada. Isto significa que quero criar uma lista com um ranking final, ordenado pelas notas dos alunos, do menor para o maior.

Queremos ter um rank que irá fundir todas as notas:

```
Nota[] rank = junta(notasDoMauricio, notasDoAlberto);
```

Depois que o programa criar meu rank, quero que ele imprima todos os valores ordenados corretamente. Isto é: para cada nota dentro do rank, vou querer imprimir a nota e o nome do aluno (`nota.getAluno()`).

```
for(Nota nota ) : rank) {
    System.out.println(nota.getAluno());
}
```

Será o `getAluno` que irá devolver o próprio `aluno` :

```
public String getAluno() {
    return aluno;
}
```

Nós precisamos implementar o `junta` . Isto é: dado duas coleções, com os elementos já ordenados, como fazemos para juntar os dois *arrays* e resolver rapidamente esta ordenação? Se já paralelizamos, e dividimos o trabalho entre as pessoas, vamos unir as diversas partes ordenadas. Iremos juntar todos em um.

Este método `junta` recebe as primeiras (`notasDoMauricio`) e segundas (`notasDoAlberto`) notas. É o método que iremos implementar.

Implementando o junta/intercala

Temos que implementar agora a função `junta()` que, dados dois *array*s ordenados, irá uni-los em uma única lista com todos os elementos. Ele irá juntar os dois *arrays* e reorganizar os itens.

Vamos juntar os *arrays*?

```
private static Nota[] junta(Nota[] notasDoMauricio, Nota[] notasDoAlberto) {
    return null;
}
```

Precisamos identificar o número de elementos dos dois grupos, para criar um novo *array* que caiba todos eles. Isto significa que:

```
int total = notasDoMauricio.length + notasDoAlberto.length;
```

Iremos também criar o resultado, que é a lista final.

```
Nota[] resultado = new Nota[total];
```

Nós iremos devolver o `resultado`. O nosso código ficará assim:

```
private static Nota[] junta(Nota[] notasDoMauricio, Nota[] notasDoAlberto) {
    int total = notasDoMauricio.length + notasDoAlberto.length;
    Nota[] resultado = new Nota[total];
    return resultado;
}
```

Ainda falta inserir as notas. Tanto `notasDoMauricio` como `notasDoAlberto` já estão ordenados. Vamos incluir os elementos de ambos no nosso *array*.

Como fizemos a ordenação antes? Verificava qual era a menor nota do grupo do Maurício e depois, do grupo do Alberto. Comparava os dois elementos e colocava o menor na lista geral. Seguimos para o próximo elemento de cada grupo, e depois para o próximo, até não ser mais possível comprar os grupos. Então, precisamos começar com a primeira posição de cada lista. Isto significa que:

```
int atualDoMauricio = 0;
int atualDoAlberto = 0;
```

Assim vamos começar pela primeira posição dos dois grupos. Observo a primeira posição de cada um, quem está lá? Será a nota do Maurício (`nota1`) é a `notasDoMauricio` que está na posição `atualDoMauricio`. A linha ficará assim:

```
Nota nota1 = notasDoMauricio[atualDoMauricio];
```

Qual será a nota 2? Será `notasDoAlberto` que está na posição `atualDoAlberto`.

```
Nota nota2 = notasDoAlberto[atualDoAlberto];
```

Vamos descobrir qual das duas notas é a menor. Se (`if`) a nota 1 (nota1.getValor) for menor do que a nota 2 (`nota2.getValor`), a menor nota será a do Maurício. Senão (`else`), será a do Alberto.

```
Nota nota1 = notasDoMauricio[atualDoMauricio];
Nota nota2 = notasDoAlberto[atualDoAlberto];
if(nota1.getValor() < nota2.getValor()) {
    // mauricio
} else {
    // alberto
}
```

É isto que queremos comparar. Iremos identificar quando uma ou outra é a menor, para então colocá-la no *array* geral. Vamos criar o método `getValor()` , que vai devolver um `double` que é o `valor` e salvamos a classe `Nota` :

```
public double getValor() {
    return valor;
}
```

Temos nosso `getValor` . Se a `nota1` for menor que a `nota2` , usaremos a `nota1` . Caso a menor seja a `nota2` , ela que será usada. Temos que fazer isto dentro do laço. Precisamos ir andando dentro do *array*. Isto significa que todo o código deverá ser executado, enquanto pudermos comparar as notas dos dois grupos. Enquanto (`while`) o `atualDoMauricio` não ultrapassar o total do grupo (`notasDoMauricio.length`) e o `atualDoAlberto` não ultrapassar o total do outro grupo (`notasDoAlberto.length`), podemos seguir colocando itens na nova lista.

```
while(atualDoMauricio < notasDoMauricio.length &&
      atualDoAlberto < notasDoAlberto.length) {
}
```

Enquanto o `while` for verdadeiro, devo continuar analisando. Vou recortar parte do `if` do código e colá-lo dentro do `while` :

```
while(atualDoMauricio < notasDoMauricio.length &&
      atualDoAlberto < notasDoAlberto.length) {

    Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    if(nota1.getValor() < nota2.getValor()) {
        // mauricio
    } else {
        // alberto
    }

}
```

Enquanto tivermos elementos para serem analisados, seguimos com o processo de comparação.

Qual é o primeiro elementos de cada grupo? Temos o André, com 4, e o Jonas, com 3. Então, ele irá cair no caso do Alberto. A `nota2.getValor` é referente ao Jonas, e é a menor. Então, vamos colocá-la no resultado. Para isso vamos escrever no nosso código que o `resultado` na posição 0 é igual a `nota2` .

```
Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    if(nota1.getValor() < nota2.getValor()) {
        // mauricio
    } else {
        // alberto
        resultado[0] = nota2;
    }
```

Se o menor fosse o do Maurício, iríamos escrever que o `resultado` na posição 0 é a `nota1` .

```
Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    if(nota1.getValor() < nota2.getValor()) {
        // mauricio
        resultado[0] = nota1;
    } else {
        // alberto
        resultado[0] = nota2;
    }
```

Agora que analisamos a `nota1` e a `nota2` , temos que ir para o próximo do Alberto (atualDoAlberto++). Se o do Mauricio for o menor, também seguiremos para o próximo (atualDoMauricio++)

```
    Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    if(nota1.getValor() < nota2.getValor()) {
        // mauricio
        resultado[0] = nota2;
        atualDoMauricio++;
    } else {
        // alberto
        resultado[0] = nota2;
        atualDoAlberto++;
    }
```

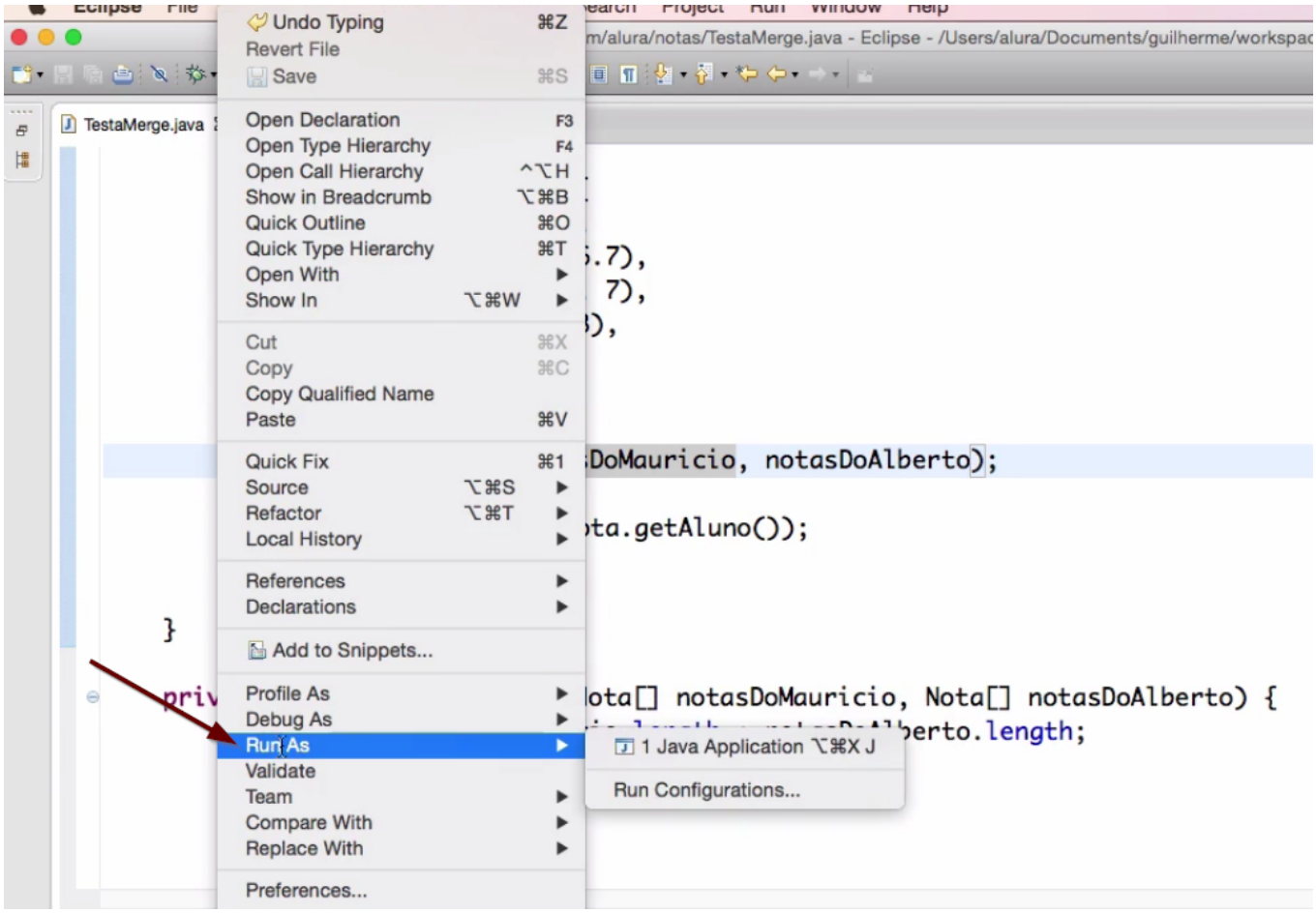
Depois, voltamos para o laço e comparamos novamente os elementos para identificar qual é o menor elemento. Quando descobrimos qual é o menor, o que fazemos com ele? Colocamos na posição 0? Não. Vamos colocando os itens em um pouco mais para frente no *array*. Esta posição onde coloco o elemento tem que ir se deslocando. Logo, iremos precisar de uma posição (`int atual`) que comece no 0. Vamos incluir o atual no resultado também. E indiferente se é o Mauricio ou o Alberto usaremos `atual++` . Vamos sempre passar para a próxima casa...

```
int atual = 0;

while(atualDoMauricio < notasDoMauricio.length &&
      atualDoAlberto < notasDoAlberto.length) {
```

```
        Nota nota1 = notasDoMauricio[atualDoMauricio];
        Nota nota2 = notasDoAlberto[atualDoAlberto];
        if(nota1.getValor() < nota2.getValor()) {
            // mauricio
            resultado[atual] = nota2;
            atualDoMauricio++;
        } else {
            // alberto
            resultado[atual] = nota2;
            atualDoAlberto++;
        }
        atual++;

    }
```

Fizemos o nosso código, a parte inicial do `junta()`. O que ele fará? Irá criar um *array* que caiba todos os elementos. Depois, vamos analisar o primeiro de cada grupo e identificar qual é o menor dos dois, para então colocá-lo na primeira posição do resultado. Temos duas notas para comparar? Sim. Após compará-los e descobrir qual é o menor, vamos colocá-lo na posição adequada. Em seguida, avançamos para o próximo. Repetimos o processo até analisarmos o total de elemento. Quando acabar, devolve para o resultado.

Aparentemente, tudo está pronto. Vamos testar o código? Iremos rodá-lo e o programa deve imprimir o nome de todos os alunos. Clicamos em *Run As* e depois em *Java Aplication*



O resultado no console será:

```
jonas
andre
```

```
mariana
juliana
guilherme
carlos
paulo
```

No resultado, além de faltar a Ana, apareceu uma mensagem de erro: `NullPointerException` . Nós quase acertamos, mas ficou faltando algum detalhe.

Iremos descobrir qual é este detalhe, em seguida. Por que a Ana não apareceu?

Logando informações para procurar o erro em um algoritmo

Nós vimos que ao rodar o nosso programa, aconteceu algum erro no fim. Ele imprimiu no resultado sete alunos, porém deixou de fora dois. Quais ficaram faltando? Vamos conferir detalhadamente qual informação estamos imprimindo? Não nos limitaremos aos nomes (`nota.getAluno`), mas também ao valor das notas (`nota.getValor`).

```
Nota[] rank = junta(notasDoMauricio, notasDoAlberto);
for(Nota nota : rank) {
    System.out.println(nota.getAluno() +" " + nota.getValor());
}
```

Vamos rodar de novo o programa e ele irá imprimir novamente a lista de alunos do Jonas até o Paulo e suas notas.

```
- jonas 3.0
- andre 4.0
- mariana 5.0
- juliana 6.7
- guilherme 7.0
- carlos 8.5
- paulo 9.0
```

Quais alunos ficaram faltando no resultado? Do primeiro *array* não faltou nenhum elemento. Porém, do segundo, faltaram a Lúcia e a Ana. Por que será que elas não apareceram? Vamos tentar imprimir quem passou pelo processo de comparação. Para cada uma das comparações, iremos imprimir os resultados com o texto:

```
System.out.println("Estou comparando " + nota1.getAluno() + " com " + nota2.getAluno());
```

Vamos voltar a rodar o código com a nova linha e ver o que acontece?

```
while(atualDoMauricio < notasDoMauricio.length &&
      atualDoAlberto < notasDoAlberto.length) {

    Nota nota1 = notasDoMauricio[atualDoMauricio];
    Nota nota2 = notasDoAlberto[atualDoAlberto];
    System.out.println("Estou comparando " + nota1.getAluno() + " com " + nota2.getAluno());

    if(nota1.getValor() < nota2.getValor()) {
```

```
            // mauricio
            resultado[atual] = nota2;
            atualDoMauricio++;
        } else {
            // alberto
            resultado[atual] = nota2;
            atualDoAlberto++;
        }
        atual++;

    }
```

O programa irá imprimir diversas comparações:

```
Estou comparando andre com jonas
Estou comparando andre com juliana
Estou comparando mariana com juliana
Estou comparando carlos com juliana
Estou comparando carlos com guilherme
Estou comparando carlos com lucia
Estou comparando paulo com lucia
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
```

Ele comparou o Paulo com a Lúcia e depois interrompeu o processo. O Paulo está no primeiro *array*, enquanto a Lúcia está no *segundo*. Ele comparou as duas notas e identifico que a do Paulo era a menor. Se ele escolheu o Paulo, ele aumentou +1 no `atualDoMauricio` . Com isto, acabaram os elementos do *array* do Mauricio e assim, ele interrompeu as comparações. Vamos ver se foi isto que realmente aconteceu? Iremos incluir um `System.out.println` no fim do código.

```
System.out.println("Estou saindo");
```

O resultado foi:

```
Estou comparando andre com jonas
Estou comparando andre com juliana
Estou comparando mariana com juliana
Estou comparando carlos com juliana
Estou comparando carlos com guilherme
Estou comparando carlos com lucia
Estou comparando paulo com lucia
Estou saindo
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
```

```
carlos 8.5
paulo 9.0
```

Por que ele saiu? Porque estas duas condições do `while` precisavam ser verdadeiras.

```
while(atualDoMauricio < notasDoMauricio.length &&
      atualDoAlberto < notasDoAlberto.length) {
}
```

Vamos imprimir cada uma dela e conferir se as duas condições são verdadeiras? A primeira será:

```
System.out.println("Estou saindo");
System.out.println(atualDoMauricio.length);
```

Para a segunda faremos a mesma coisa para o Alberto:

```
System.out.println("Estou saindo");
System.out.println(atualDoMauricio.length);
System.out.println(atualDoAlberto.length);
```

Isto significa estamos perguntando: "sobrou alguém no Mauricio?" e "Sobrou alguém no Alberto?"

Vamos rodar e o resultado será:

```
Estou comparando andre com jonas
Estou comparando andre com juliana
Estou comparando mariana com juliana
Estou comparando carlos com juliana
Estou comparando carlos com guilherme
Estou comparando carlos com lucia
Estou comparando paulo com lucia
Estou saindo
false
true
jonas 3.0
andre 4.0
mariana 5.0
juliana 6.7
guilherme 7.0
carlos 8.5
paulo 9.0
```

O `false` indica que não sobrou ninguém no Maurício. Já o `true` nos diz que sobraram alunos no Alberto.

Qual foi o nosso problema? Qual foi o *bug*? Nós intercalamos todos os elementos, um por um. Quando chegamos no fim de um dos *array*s, ainda existia elementos no outro. Com esta condição pode ter sobrado diversos elementos em cada um dos grupos. Mas nós precisamos incluir estas pessoas, afinal se só restaram elementos do grupo do Maurício, vamos colocá-los no *array*. Se sobraram alunos do Alberto, vamos incluí-los lá também. Em breve, é isto o que iremos fazer.