04

Busca do menor valor e um trecho específico no array

Bem-Vindo

Oi, pessoal. Neste curso nós vamos falar sobre soluções para diversos problemas. Eu sempre quero resolver os meus problemas, mas que tipo de problemas quero resolver aqui?

Vou resolver problemas com ordem, problemas com quem é mais caro, quem é mais barato. Qual o menor caminho? Qual é o melhor produto?

Todos esses problemas envolvem dizer "que coisa é maior do que outra coisa". Qual é o hotel mais próximo? Qual é o cinema mais próximo? Qual o horário do próximo trem que irá sair? Tudo isto envolve uma pergunta do tipo **"quem"** ou **"algo"**, que é **"mais"** ou **"menos"**. Envolve **ordenação**.

- Quem ganhou o jogo? Quem fez mais gols.
- Quem ganhou o campeonato? Quem fez mais pontos.
- Quem ganhou o campeonato e foi rebaixado? Quem fez menos pontos.

Tudo isso envolve um conjunto de itens, um monte de coisa, um monte de times de futebol, um monte de produtos que estão à venda, vários corredores em uma maratona. Envolve pegar todos esses itens e ordenar... Definir quem ficou em primeiro, quem ficou em segundo, quem ficou em terceiro, quem ficou em último...

Quem são as pessoas que passaram e nao passaram no vestibular? Quem foi reprovado? Quem passou e não passou na prova? Tudo isso envolve dizer **algo que é maior* e** algo que é menor**. Envolve ordenar as pessoas, os produtos, as coisas. Dar uma ordem para tudo.

Muitas perguntas que fazemos estão ligadas a uma **ordem**... Se você precisa buscar um hotel, quer encontrar um hotel que seja próximo de uma determinada região da cidade. Você irá buscar primeiro os que são mais baratos, depois os que são mais caros... Ou, se você procura aqueles que têm uma determinada característica... Em seguida você procura os que são mais baratos. Ou, se eu quero saber quem ganhou e quem perdeu o campeonato... São tantas coisas que queremos ordenar: quem nós queremos, quem nós não queremos; quem passou, quem não passou;

Pare para pensar: ao nosso redor, tudo tem uma questão de **ordem**. Se você quer ir para o trabalho, você quer o ônibus que vai chegar mais rápido, em menos tempo. Ou se você está em casa... Quer pegar o primeiro ônibus, que está mais próximo da saída. Tudo isso envolve ordenar um monte de coisas. **Isso é o que faremos aqui: aprender a ordenar os maiores, os menores, os melhores**. Não importa... Vamos aprender a encontrar aquilo que procuramos, com uma ordem. O mais rápido,o melhor, o com maior nota, o com menor e maior tempo... Tudo isto conseguimos fazer com o que vamos aprender aqui no curso.

Vamos aprender maneiras de encontrar **os melhores, os piores, os maiores, os menores**. Os **Algoritmos** são soluções, maneiras de resolver esses problemas.

É isso que veremos em seguida.

O Produto mais barato

Primeiro problema do meu dia-a-dia: eu quero comprar um carro. Vou entrar em um site para pesquisar preços de carros usados e ele me trouxe esses preços:

- Uma Lamborghini por R$ 1.000.000
- Um Jipe por R$ 46.000
- Uma Brasília por R$ 16.000
- Um Smart por R$ 46.000
- Um Fusca por R$ 17.000

Imagine que está difícil procurar um carro neste site, pois tenho muitas opções, com diferentes faixas de preço... No meu caso, estou procurando o carro mais barato. O que tem o **menor número possível**.

Em diversas situações, eu vou procurar este número. Por exemplo, quando eu quero encontrar outro produto **mais barato**, como um livro. Três livrarias diferentes vendem este livro e eu quero comprar o mais barato das três.

Outros exemplos:

1. Eu quero comprar um brinquedo e entro em um site que compara os preços do produto em diversas lojas... Eu quero saber: qual é o mais barato deles?
2. Quantos erros cada aluno cometeu? Quem cometeu menos erros?
3. Qual ônibus chegará mais rápido no meu destino?

Nestes casos, eu quero saber qual é o **menor número possível**...

No exemplo dos carros, a primeira coisa que quero saber é "qual é o mais barato de todos?" Não adianta sugerir uma **Lamborghini**, se o carro custa R$ 1 milhão e eu quero o mais barato.

Nós temos os dados dos carros: cada um tem "nome" e "preço". E temos cinco opções de carros... Qual tem o menor preço? Dê uma olhada e me responda.

Você já viu os preços e sabe qual é o carro mais barato. A Brasília tem o menor preço:

É bem provável que você tenha respondido rápido... Agora, eu quero saber: o que você pensou para conseguir responder bem rápido? Pense como foi e escreva (também, vale dizer em voz alta). Responda e eu já vou te contar o que fiz para concluir qual era o mais barato dos cinco carros na lista.

Encontrando o mais barato: na minha cabeça

Como você resolveu o problema: **Qual é o carro mais barato?**

Quer saber como eu resolvi?

Eu olhei todos os carros e vi qual era o mais barato? Não foi assim. Meu pensamento passou por etapas antes disso. Houve um processo, um algoritmo que rodamos na nossa cabeça. Por exemplo: eu olhei para todos os carros (sem isto, eu não descubro qual é o mais barato):

Depois, olho um de cada vez.

- Primeiro carro: a Lamborghini custa R$ 1.000.000. É o carro mais barato que eu conheço até este momento.

- Segundo carro: o Jipe custa R$ 46.000. É mais barato que a opção anterior.

- Terceiro carro: a Brasília custa R$ 16 000. Ela é mais barata? É mais barata que o carro até agora, de R$ 46.000.

- Quarto carro: o Smart custa R$ 46. 000. Este carro é mais caro que a Brasília, de R$ 16.000, o carro mais barato até agora.

- Quinto carro: o Fusca custa R$ 17.000. Também é mais caro que a Brasília.

Então, eu olhei todos os carros, sempre comparando o atual com o mais barato até o momento.

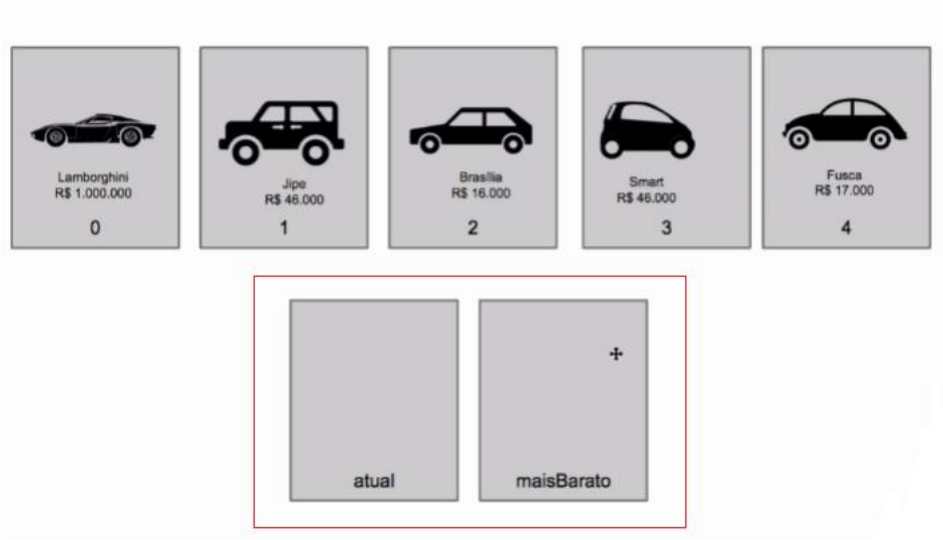
Por exemplo: Quando analiso o Jipe, que custa R$ 46.000, e comparo com a Lamborghini, que custa R$ 1.000.000, o Jipe será o carro mais barato.

Quando eu faço a comparação com outro carro *mais caro*, eu ignoro e escolho o carro *mais barato*. O processo é feito rapidamente. Eliminei as opções mais caras e fiquei apenas com a Brasília, de R$ 16.000.

As etapas do processo ficam ainda mais claras quando aumentamos o número de opções. Se compararmos 50 ou 100 carros, teremos maior dificuldade em memomizar qual é o mais barato até agora.

Se você precisar descobrir qual é o produto mais barato entre 100 produtos, é provável que queira fazer anotações. Isto porque é difícil processar rapidamente quando o número de dados é muito grande.

Então, o que podemos fazer? Podemos anotar quais são as nossas opções e escrever qual carro é o mais barato até o momento.



Eu irei anotar qual é a melhor opção **atual** (entre os carros 0, 1, 2, 3 ou 4) e qual é o carro **mais barato** até o momento. Se o número de itens que vou comparar é pequeno, eu não preciso anotar. Mas se o número for grande, eu escolho fazer anotações.

Eu denominei os carros como "0, 1, 2, 3, 4", porque, em programação, nós sempre começamos uma coleção (uma *array* de elementos) com a **posição 0**.

Experimente fazer esse processo visualmente. Escreva as cinco opções de carro, no papel. Compare os elementos entre si e anote qual possui o menor preço. Por exemplo:

- Entre o carro 1 e 2, o mais barato é...

O algoritmo do mais barato

Quando executamos o processo com calma, no papel, percebemos que nosso pensamento é bem rápido. Quando nós executamos o processo,como por exemplo: soma, multiplicação, divisão simples ou qualquer tipo de algoritmo simples, não percebemos que, mentalmente, estamos fazendo um monte de coisas malucas. Sem perceber, fazemos várias contas. Por exemplo:

- O número 17 é par ou ímpar?
- O número 30 é divisível por 10?

Nós respondemos automaticamente, mas na nossa cabeça, nossos pensamentos estão passando por vários processos.

A mesma coisa acontece no exemplo dos carros.

Quando eu quero encontrar o maior e o menor preço (um **algoritmo**), faço várias contas, mentalmente. Quando eu transcrevo um programa para o algoritmo, eu transcrevo o processo passo a passo para o programa. É o que faremos agora: vamos simular com imagens o processo e depois faremos a transcrição para o código.

Vamos começar com o primeiro carro da lista: o `carro 0`. Como ele é o primeiro, o **carro atual** será também o **mais barato**.

O próximo elemento da lista é o `carro 1`. O Jipe (que custa R$ 46.000) é mais barato que o `carro 0` (que custa R$ 1.000.000)? Sim, é mais barato. Então, vamos anotar que o `carro 1` é o mais barato.

Vamos analisar a Brasília, que custa R$16.000... O `carro 2` é o atual. Ele é mais barato que o anterior (que custa R$46.000)? Sim. Vamos anotar que o `carro 2` é o mais barato.

Agora, vamos para o `carro 3`. O "carro atual" (que custa R$46.000) é mais barato que o `carro 2`? Não. Então, não faremos alterações.

O próximo elemento é o `carro 4`. Ele é mais barato que o `carro 2`? Não. Então, não faremos alterações.

Vamos para o `carro 5`. Não temos mais carros para analisar! Logo, acabou.

Temos, então, que o `carro 2` é o mais barato!

O que nós fizemos no exemplo? Nós começamos com as variáveis **"atual"** e **"mais barato"** sendo **0** (a posição do primeiro carro). Em seguida, analisamos cada elemento, ou seja, fomos com o contador atual de **0** até o número total de carros. Verificamos: "Esse carro é mais barato, sim ou não?" Nos casos em que era, anotamos qual era o mais barato.

Este é o processo que fizemos graficamente. Agora, vamos transformá-lo em código?

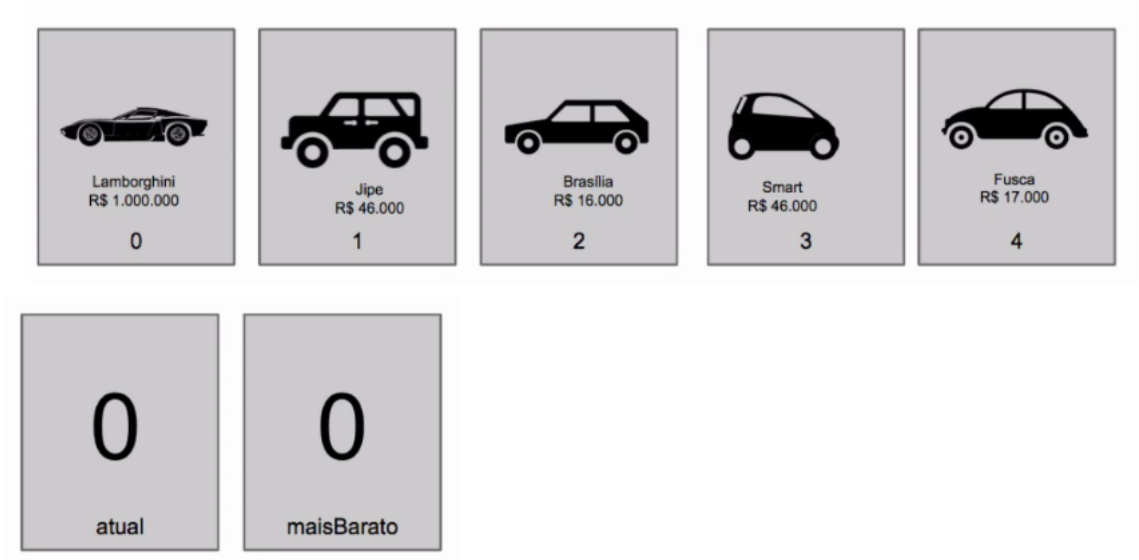
Será o nosso próximo passo...

Escrevendo o algoritmo

Vamos passar a limpo o processo que executamos na nossa mente e que criamos graficamente... Como faremos isto?

Primeiro, você lembra o que definimos? Nós definimos dois valores: o do produto **atual** que estamos analisando no nosso *array* (que iniciamos com **0**) e do **mais barato** até agora (que também é o produto na posição **0**). Isto é, tanto `atual` como `maisBarato` são iguais a `0`.

```
maisBarato = 0
atual = 0
```



Na nossa memória, nós analisamos exatamente assim: `atual` é igual a `0` e `maisBarato` é igual a `0`. Este foi o início do nosso processo (o nosso **algoritmo**).

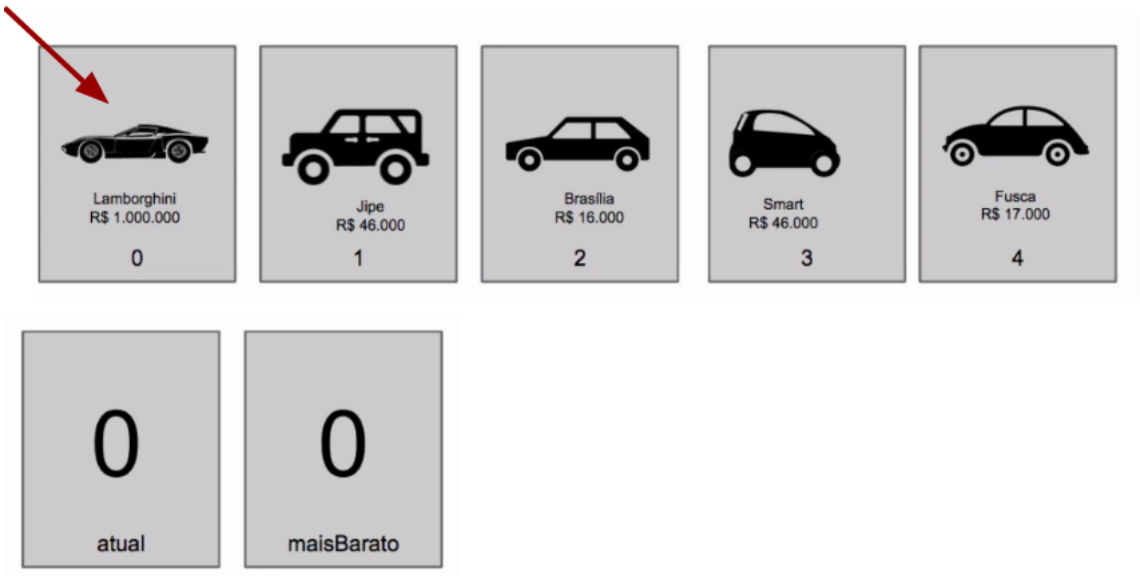
Depois, o que fizemos? Nós começamos a analisar cada produto dentro do *array*. Isto significa que, se eu tenho cinco produtos, eu vou do `0 até 4 inclusive` ou do `0 até 5 exclusive`. Então, **para atual** vou do `0 até 4 inclusive`.

```
maisBarato = 0
atual = 0

para atual = 0 até 4 inclusive {

}
```

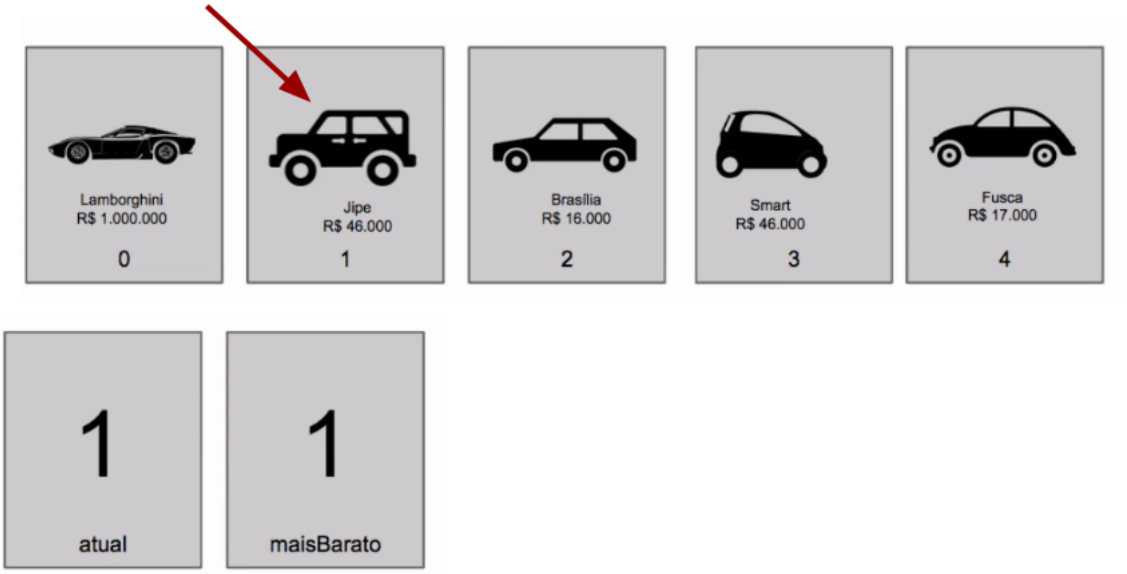
Depois disso, se o preço do **atual** for menor do que o **maisBarato**, significa que ele é o produto mais barato que encontrei até agora. Então, nós iremos trocar o **maisBarato** e especificar qual elemento irá substituí-lo. O produto **mais barato** é o elemento **atual**, que tem o menor preço. Esse é o código que eu terei:

```
maisBarato=0
atual = 0

para atual = 0 até inclusive {
   se precos[atual] < precos[maisBarato] {
          maisBarato = atual
   }
}
```

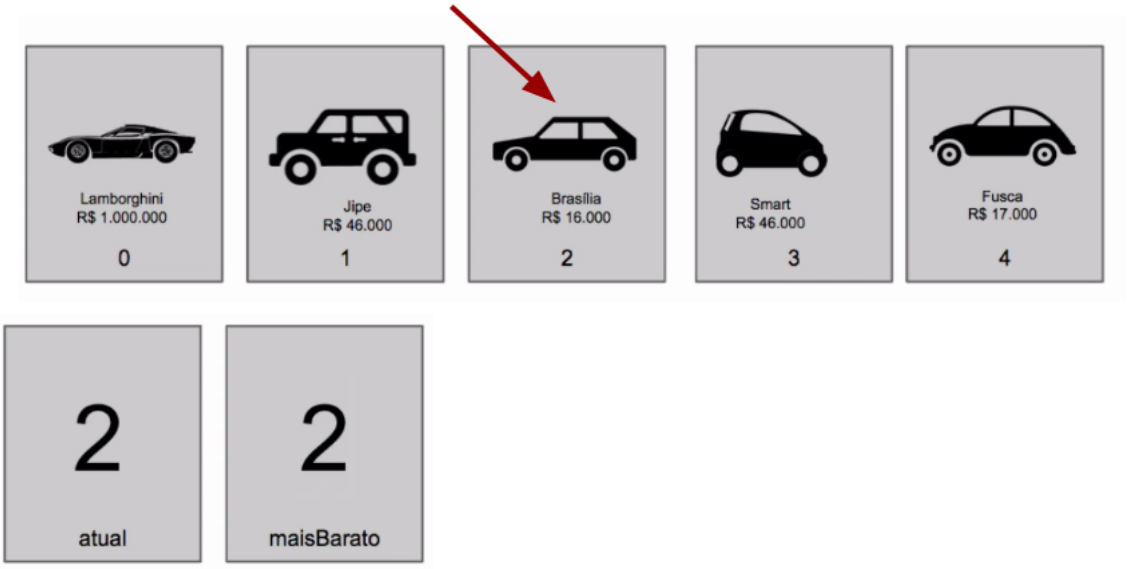
Agora, vamos comparar o preço do produto **0** (que custa R$1.000.000) com ele mesmo. Logo, não faremos alterações.



O próximo é o produto **1**. O **atual** será igual a **1**, que custa R$46.000. Este preço é menor que o **maisBarato**, que agora é igual a **0**? O valor R$ 46.000 é menor que R$ 1.000.000? Sim, entra no `maisBarato=atual`. Eu vou atribuir o valor **1** no **maisBarato**.

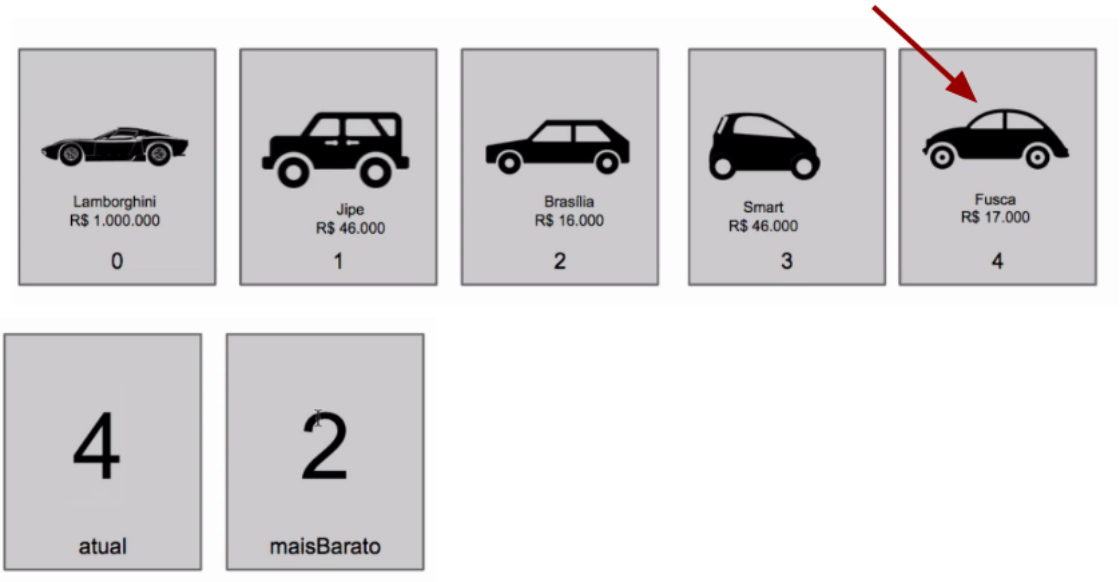


Seguimos para o produto **2**. O preço do carro 2 (que custa R$ 16.000) é menor do que o preço do carro 1 (que custa R4 46.000)? Sim, então, vamos substituir o `maisBarato` pelo o **atual**.



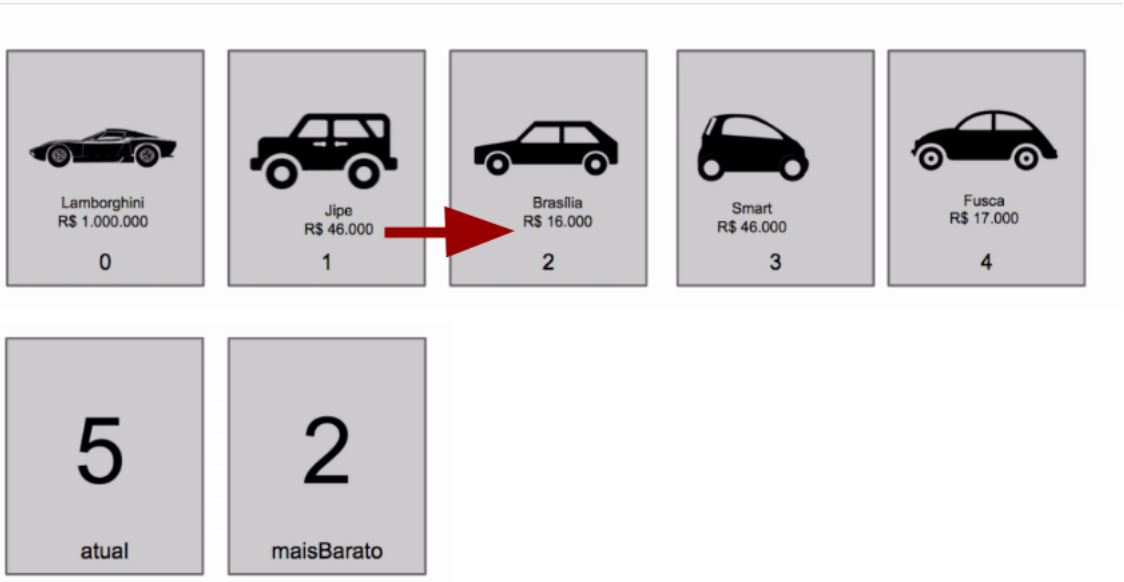
O próximo elemento é o carro 3. Vamos alterar o **atual** para **3** e comparar o valor R$ 46.000 com o valor do `maisBarato` , que custa R$ 16.000. O preço do produto **3** é menor do que **2**? Não. Vamos para outro elemento.

Agora, o **atual** é igual a **4**. O preço do produto é R$46.000. Este preço é menor do que o valor do `maisBarato` até o momento? Não. Seguimos...



E com **atual** igual a **5**? Ele não está no intervalo que estamos interessados. Nós vamos parar por aqui.

Já encontramos o produto mais barato da lista. É o elemento que está na posição **2**, a Brasília, que custa R$16.000.




```
maisBarato=0
atual = 0

para atual = 0 até inclusive {
   se precos[atual] < precos[maisBarato] {
          maisBarato = atual
   }
}
```

Este é o nosso **pseudo algoritmo** para encontrar o valor mais barato e descobrir o menor valor de uma *array*.

Criando o Projeto

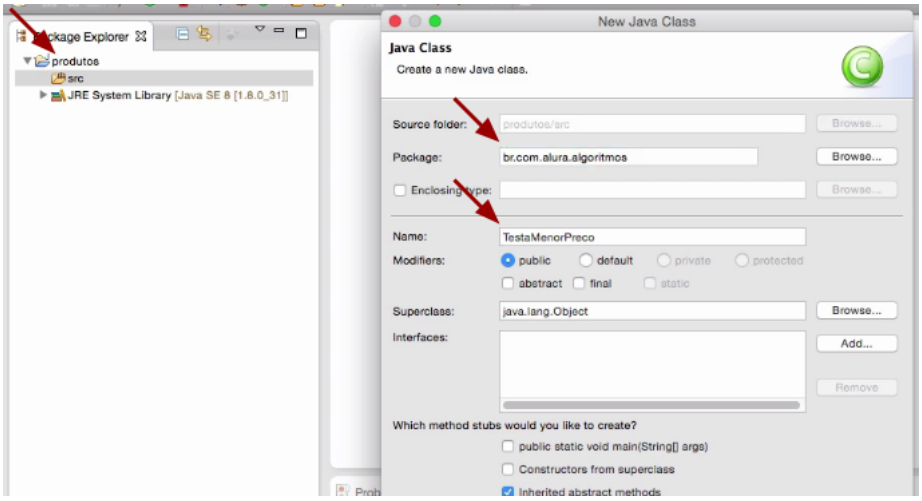
Nós já vimos como executar o processo, o **algoritmo**, que é uma sequência de tarefas para detectar o elemento mais barato dentro de uma *array* de produtos.

Meu *array* tem cinco posições, indo do `0 até 4`, criando uma variável `atual` e atualizando-a (sempre verificando se o preço do produto **atual** é menor que o **mais barato**). Agora, vamos escrever isto em *Java*?

Criarei um projeto chamado "produtos". Dentro do meu projeto, irei criar a Classe, ou *class* `TestaMenorPreco`:

O pacote que vou usar é `br.com.alura.algoritmos`.

Agora, eu tenho a minha classe:



Vou colocar o método `main()` de execução do nosso programa. Inicialmente, o que precisamos inserir dentro dele são os preços dos carros. Para isso, vou criar um *array* de preços, com cinco posições. Irei defini-las passo a passo.

Os preços dos carros são:

- `carro 0` custa R$ 1000000;
- `carro 1` custa R$ 46000;
- `carro 2` custa R$ 16000;
- `carro 3` custa R$ 46000;
- `carro 4` custa R$ 17000.

Então meu código fica assim:

```
package br.com.alura.algoritmos;
public class TestaMenorPreco {

    public static void main(String[] args) {

        double precos[] = new double[5];
        precos[0] = 1000000;
        precos[1] = 46000;
        precos[2] = 16000;
        precos[3] = 46000;
        precos[4] = 17000;
    }
}
```

O que eu desejo saber? Qual elemento tem o menor preço? Para fazer isso, nós criamos uma variável que armazena aquele que é mais barato. Tanto `maisBarato` como `atual` são variáveis inteiras que começam pelo `0`. Como esta é uma variável inteira, eu continuo com os produtos 1, 2, 3 e 4.

Em seguida, vou executar uma tarefa para os diferentes valores da variável `atual`, de `0 até 4`. O `4` será `inclusive`, porque eu quero incluir o preço do Fusca (o carro 4).

Do `0 até 4`, o que nós queremos fazer? Nós queremos criar a condição: "se o preço do `atual` for menor que o preço do `maisBarato`, o preço do `maisBarato` será igual ao preço do produto `atual`":

```
double precos[] = new double[5];
precos[0] = 1000000;
precos[1] = 46000;
precos[2] = 16000;
precos[3] = 46000;
precos[4] = 17000;

int maisBarato = 0;
int atual = 0;
executo do 0 ate 4 inclusive {
        se preco do atual < preco do mais barato
            mais barato = atual
}
```

Toda vez que nós executarmos o *loop*, precisamos atualizar o `atual` e, só então, seguimos para o próximo produto. Teremos que somar `1` no `atual`, ou seja, `atual +1`.

```
double precos[] = new double[5];
precos[0] = 1000000;
precos[1] = 46000;
precos[2] = 16000;
precos[3] = 46000;
precos[4] = 17000;

int maisBarato = 0;
int atual = 0;
executo do 0 ate 4 inclusive {
        se preco do atual < preco do mais barato {
            mais barato = atual
            }
        atual = atual + 1
}
```

Continuamos com o próximo produto, independentemente se ele é ou não o mais caro. O que fizemos até agora: definimos o `maisBarato` e o `atual` igual a `0`. Executamos o código do `0 até 4, inclusive`. Se o preço do `atual` for menor que o preço do `maisBarato`, nós iremos definir qual é o novo elemento `maisBarato`. Caso contrário, não modificamos. E seguimos para o próximo... No fim, pedimos para o programa imprimir: `imprime o maisBarato e o preço do mais barato`. Nosso código ficará assim:

```
package br.com.alura.algoritmos;

public class TestaMenorPreco {
```

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

        double precos[] = new double[5];
        precos[0] = 1000000;
        precos[1] = 46000;
        precos[2] = 16000;
        precos[3] = 46000;
        precos[4] = 17000;
        int maisBarato = 0;
        int atual = 0;
        executo do 0 ate 4 inclusive {
            se preco do atual < preco do mais barato {
                mais barato = atual
            }
            atual = atual + 1
        }
        imprime o mais barato
        imprime o preco do mais barato
    }
}
```

Vamos traduzir este código para Java? Será o nosso próximo desafio.

Transformando pseudo código em Java

Nós já escrevemos um código que é quase *Java*. Na verdade, o que fizemos é um pseudo Java. Ele ainda não funciona, mas já nos dá uma ideia do que está acontecendo...

```
int maisBarato = 0;
int atual = 0;
executo do 0 ate 4 inclusive {
    se preco do atual < preco do mais barato {
        mais barato = atual
    }
    atual = atual + 1
}
imprime o maisbarato
imprime o preco do mais barato
```

O que estou querendo fazer aqui? Quero construir um "laço" de `0 até 4`. Comumente, nós vamos usar o laço `for`, desde `atual = 0` (e por isso, não vou precisar da definição do `atual`) até que ele seja menor ou igual a 4 e somo `+1` ou digito `atual ++`. E removo a linha: `atual = atual + 1`.

```
int maisBarato = 0;
for(int atual = 0; atual <= 4; atual++) {
    se preco da atual < preco do mais barato {
        mais barato = atual
    }
}
imprime o maisbarato
imprime o preco do mais barato
```

Agora, o que eu tenho que fazer? Aqui dentro estou verificando se o preço do `atual` é menor que o preço do `mais barato` :

```
if(precos[atual] < precos[maisBarato])`
```

Isso significa que o **mais barato** é o elemento **atual**:

```
maisBarato = atual
```

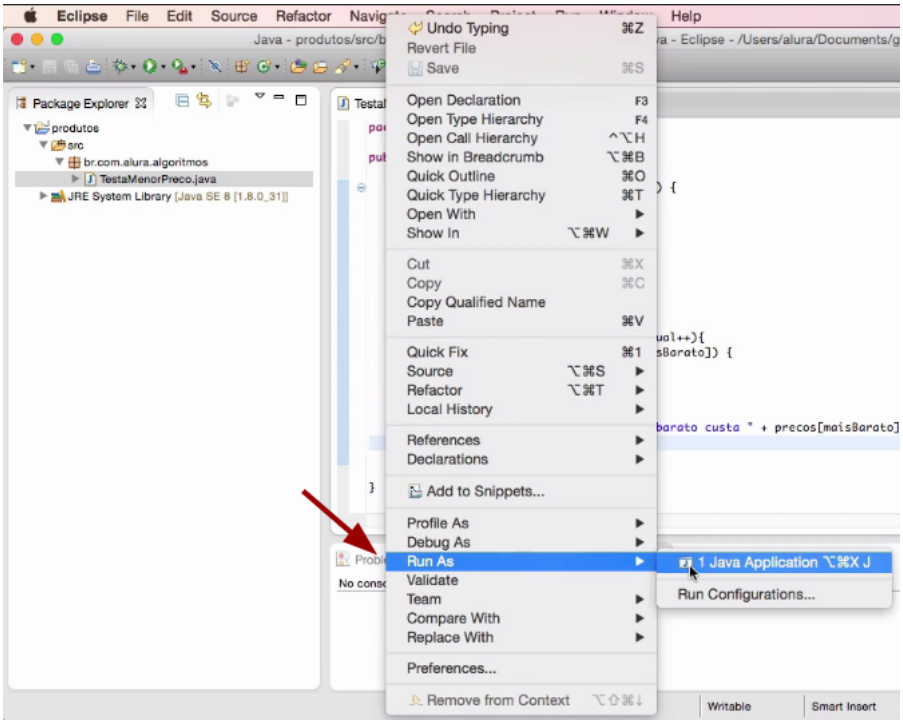
Então, nosso código fica:

```
int maisBarato = 0;
for(int atual = 0; atual <= 4; atual++){
    if(precos[atual] < precos[maisBarato]) {
        maisBarato = atual;
    }
}
imprime o maisbarato
imprime o preco do mais barato
```

Depois, eu imprimo `"O carro mais barato custa" + precos[maisBarato])` e o nosso código ficará assim:

```
int maisBarato = 0;
for(int atual = 0; atual <= 4; atual++){
    if(precos[atual] < precos[maisBarato]) {
        maisBarato = atual;
    }
}
System.out.println(maisBarato);
System.out.println("O carro mais barato custa" + precos[maisBarato]);
```

Vamos testar nosso código? Clico no botão direito do mouse e depois em "Run As" e "Java Application":



Na tela, aparecerá que o carro 2 é o produto mais barato:

```
2
O carro mais barato custa 16000.0
```

O computador consegue executar o algoritmo que passa em nossa mente e do qual somos inconscientes... Observe: quando nós executamos o algoritmo mentalmente, nós olhamos os cinco produtos e, de imediato, identificamos qual era o carro mais barato. Como? Nós comparamos cada produto e apontamos qual era o produto mais barato. O computador faz a mesma coisa: ele anota na memória qual é o produto mais barato e qual elemento foi analisado. Depois, ele armazena o produto **atual** e o **mais barato**, e segue verificando qual tem o menor preço. Nós fizemos a mesma coisa de forma instantânea, porque eram poucos produtos.

Caso trabalhássemos com 500 produtos, provavelmente, nós iríamos fazer anotações com as variáveis `atual` e `maisBarato`.

Adiante, iremos simular esse algoritmo graficamente junto com o código.

Simulando encontrar o menor valor

Agora que nós temos o nosso código Java, vamos simular o algoritmo com o programa.

```
int maisBarato = 0;
for(int atual = 0; atual <= 4; atual++){
    if(precos[atual] < precos[maisBarato]) {
        maisBarato = atual;
    }
}
```

Começaremos com as nossas variáveis `atual` e `maisBarato` iguais a **0**.

De acordo com o nosso laço: `atual <= 4`.

O produto **0** é `<=4` ? Sim. Então, ele entra no nosso laço.

```
for(int atual = 0; atual <= 4; atual++) {
    if(precos[atual] < precos[maisBarato]) {
```

Ao analisarmos o primeiro carro, **atual** é igual a **0** e o preço do produto é R$1.000.000. O carro **mais barato** também será o **atual**, então, o preço mais barato será R$ 1.000.000. Logo, não vamos modificar o `maisBarato` .

Seguimos para o próximo carro.

Quando verificamos o carro 1, o valor de **atual** será menor que **4** e continuamos dentro do nosso laço. O preço do produto (que custa R$ 46.000) é menor que o do mais barato até o momento (R$ 1.000.000)? Sim, ele é menor. Podemos entrar dentro do `if` . Vamos trocar o valor de `maisBarato` por **1**, que é o `atual` .

No carro 2, o valor de **atual** é `<= 4` ? Sim, continua dentro do laço.

O preço do carro 2 (R$ 16.000) é menor que o `maisBarato` (R$ 46.000)? Sim, então:

```
maisBarato = atual`
```

Seguimos para o carro 3. Ainda `atual <= 4` , logo

```
if(precos[atual] < precos[maisBarato])
```

O preço do `atual` (R$ 46.000) é menor que o `maisBarato` (R$16000)? Não, então não faremos modificações.

Próximo carro... `Atual` é `<=4` ? Sim.

O preço `atual` (que custa R$17000) é menor que o `maisBarato` (que custa R$16000)? Não, então, não modificamos o `maisBarato` .

`atual++` e passamos para o `carro 5` .

`atual <= 4` ? Não. Logo, acabou o laço.

O que o nosso **algoritmo** fez? Ele executou o mesmo processo que fizemos ao observar os cinco elementos e armazenou na memória "quem" estamos analisando e "qual" é o valor menor até agora. Fizemos isto na nossa mente... Enquanto estávamos observando os produtos, nosso **olho** foi a variável `atual` e o nosso **dedo** foi a variável `maisBarato`, que indicava o produto com o menor preço.

Porém, como temos poucos produtos, o processo é feito rapidamente, de forma inconsciente. Apenas observamos e, logo, identificamos o carro mais barato.

Agora o computador também identifica imediatamente qual é o mais barato pois ele já sabe qual é o processo. Qual é o **algoritmo de detecção** dentro de uma *array*.

Trabalhando com objetos

Nós sabemos que quando trabalhamos com Java não precisamos, necessariamente, utilizar tipos primitivos. Temos a opção de criar novas classes e objetos. Como nossos elementos têm nome e preço, não faz sentido que um produto seja um *double*. Por exemplo, a Lamborghini, o Smart, o Fusca, todos têm um nome...

Desejamos obter um produto com nome e com preço. Então, ao invés, de ter um *array* de *double*, eu vou ter um *array* de produtos. Vamos, então, criar uma classe de **Produtos**? A classe **Produto** terá um nome e um preço:

```
public class Produto {

    private String nome;
    private double preco;

}
```

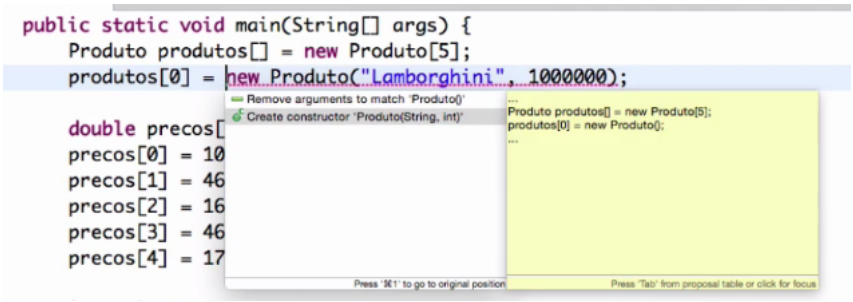
Quando construirmos um produto, vamos passar tanto o nome quanto o preço dele para a nossa classe. Isto significa que, ao invés de ter uma *array* de *double*, iremos ter um *array* de cinco produtos.

```
Produto produtos[] = new Produto[5];
```

O primeiro item do *array* será uma Lamborghini, que custa R$ 1.000.000:

```
    Produto produtos[] = new Produto[5];
    produtos[0] = new Produto("Lamborghini", 1000000);
```

Ainda não existe o construtor que recebe o **nome** e o **preço**. Clico em "Ctrl + 1" e depois, em *create constructor*, e o programa irá criar o construtor para mim:




```
public class Produto {

    private String nome;
    private double preco;

    public Produto(String nome, double preco ) {
        this.nome = nome;
        this.preco = preco;

    }
}
```

A Lamborghini ocupará a posição `0` na lista. Nas posições seguintes teremos:

- posição `1` : `Jipe` que custa R$ 46.000;
- posição `2` : `Brasília` que custa R$ 16.000;
- posição `3` : `Smart` que custa R$ 46.000;
- posição `4` : `Fusca` que custa R$ 17.000;

E também podemos remover o *array* de *doubles*:

```
public static void main(String[] args) {
    Produto produtos[] = new Produto[5];
    produtos[0] = new Produto ("Lamborghini", 1000000);
    produtos[1] = new Produto("Jipe", 46000);
    produtos[2] = new Produto("Brasília", 16000);
    produtos[3] = new Produto("Smart", 46000);
    produtos[4] = new Produto("Fusca", 17000);

    int maisBarato = 0;
    for(int atual = 0; atual <= 4; atual++){
        if(precos[atual] < precos[maisBarato]) {
            maisBarato = atual;
        }
    }
    // ...
}
```

Fizemos um *array* de produtos. Agora, que não temos mais o preço do `atual` e do `maisBarato`, usaremos o `produto` do `atual` e o seu `getPreco`. Em seguida, vamos inserir o `produto` do `maisBarato` e o `getPreco`. Nosso código ficará assim:

```
public static void main(String[] args) {
    Produto produtos[] = new Produto[5];
    produtos[0] = new Produto ("Lamborghini", 1000000);
    produtos[1] = newProduto("Jipe", 46000);
    produtos[2]= new Produto("Brasília", 16000);
    produtos[3] = new Produto("Smart", 46000);
    produtos[4] = new Produto("Fusca", 17000);

    int maisBarato = 0;
    for(int atual = 0; atual <= 4; atual++){
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
        }
    }
```

É possível que você pense: "Ainda não foi criado o `getPreco` ... E quando eu crio uma variável membro, deveria já criar o *getter* e o *setter*". No entanto, vamos criá-los apenas quando for preciso. Se for desnecessário, não temos razão para criá-los.

No nosso exemplo, precisaremos do `getPreco` , porque vou usar o produto `atual` e seu preço. Para em seguida, comparar com o produto `maisBarato` e seu preço.

Vamos criar o `getPreco` ? Entramos na classe **Produto**, digitamos `getPreco` + Ctrl + barra de espaço e o programa irá criar o *getter*.

```
public class Produto {

    private String nome;
    private double preco;

    public Produto(String nome, double preco ) {
        this.nome = nome;
        this.preco = preco;

    }

    public double getPreco() {
        return preco;
    }

}
```

Após fechar a classe **Produto**, vamos comparar os dois preços.

Também tenho um `System.out` que imprime o **número** do `maisBarato` e o `O carro mais barato custa` o valor do `produtos[maisBarato].getPreco`

```
public static void main(String[] args) {
    Produto produtos[] = new Produto[5];
    produtos[0] = new Produto ("Lamborghini", 1000000);
    produtos[1] = newProduto("Jipe", 46000);
    produtos[2]= new Produto("Brasília", 16000);
    produtos[3] = new Produto("Smart", 46000);
    produtos[4] = new Produto("Fusca", 17000);
```

```
        int maisBarato = 0;
        for(int atual = 0; atual <= 4; atual++){
            if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
                maisBarato = atual;
            }
        }
        System.out.println(maisBarato);
        System.out.println("O carro mais barato custa"
                                    + produtos[maisBarato].getPreco());

    }
```

Esse será o preço do carro...

Na verdade, já temos o nome do produto e podemos também imprimi-lo. O carro `produtos[maisBarato].getNome()` é o **mais barato**, e **custa**...

```
        System.out.println(maisBarato);
        System.out.println("O carro" + produtos[maisBarato].getNome()
                                    + " é o mais barato, e custa"
                                    + produtos[maisBarato].getPreco());
```

Vamos criar o `getNome` ? Entramos no `produto` ... Com o comando "get + Ctrl + barra de espaço", o programa irá sugerir um `getNome : public String getNome()`

```
  public class Produto {

      private String nome;
      private double preco;

      public Produto(String nome, double preco ) {
          this.nome = nome;
          this.preco = preco;

      }

      public double getPreco() {
          return preco;
      }

      public String getNome() {
          return nome;
      }

  }
```

Quando colocamos o `get.Nome` , o programa irá imprimir o nome e o preço do produto.

Revisando o que fizemos até agora: Criamos uma classe `Produto` e um *array* com todos os elementos. Em seguida, quando fazemos a comparação de preços listados, comparamos os preços dos produtos entre si. Quando quisermos imprimir,

necessito do nome e o preço do produto. Paramos de usar um *array* de double e começamos a usar um *array* de produtos. Porém, a comparação continuou sendo de elementos numéricos.

Vamos testar o código? Quando nós rodarmos o programa, irá aparecer na tela que o `carro 2` é o mais barato e custa R$16.000.

```
2
O carro Brasília é o mais barato, e custa 16000.0
```

No código, passamos a usar uma classe e objetos para armazenar os dados relativos aos meus produtos.

Detalhes de implementação na linguagem

Nós temos nosso código Java que cria um *array* de produtos, faz a busca do menor de todos os valores e nos mostra o menor de todos os preços. O algoritmo está implementado. Agora, o que queremos saber é: como eu posso melhorar o nosso código? Vamos aperfeiçoá-lo, antes de continuar.

```
public static void main(String[] args) {
   Produto produtos[] = new Produto[5];
   produtos[0] = new Produto ("Lamborghini", 1000000);
   produtos[1] = new Produto("Jipe", 46000);
   produtos[2] = new Produto("Brasília", 16000);
   produtos[3] = new Produto("Smart", 46000);
   produtos[4] = new Produto("Fusca", 17000);

   int maisBarato = 0;
   for(int atual = 0; atual <= 4; atual++){
       if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
           maisBarato = atual;
       }
   }
   System.out.println(maisBarato);
   System.out.println("O carro" + produtos[maisBarato].getNome()
                               + " é o mais barato, e custa"
                               + produtos[maisBarato].getPreco());

 }
```

Quando nós criamos um *array* nas versões mais recentes do Java, em vez de falar o tamanho do *array*, podemos usar `double precos[] = {1.3, 4.4}`

Podemos colocar os valores desde o princípio, entre as chaves. Então, vamos declarar o *array* de produtos dessa maneira:

```
public static void main(String[] args) {
   Produto produtos[] = {
           new Produto ("Lamborghini", 1000000),
               new Produto("Jipe", 46000),
               new Produto("Brasília", 16000),
               new Produto("Smart", 46000),
               new Produto("Fusca", 17000)
   };
```

```
        int maisBarato = 0;
        for(int atual = 0; atual <= 4; atual++) {
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
            }
        }
        System.out.println(maisBarato);
        System.out.println("O carro" + produtos[maisBarato].getNome()
                                + " é o mais barato, e custa"
                                + produtos[maisBarato].getPreco());

    }
```

A definição de *array* de produtos fica um pouco mais simples no Java desta maneira. E para separar cada um dos elementos, nós usamos a **vírgula** (,).

Temos o mesmo código, a mesma equivalência, só que conseguimos deixá-lo mais simples: criamos esses cinco produtos e um *array* baseado nestes elementos. Agora, temos um código mais bonito!

Refatoração: extraindo um função

Considerando que nós já temos a criação de um *array* e nosso algoritmo que detecta o produto com o menor valor, queremos extrair o código:

```
int maisBarato = 0;
for(int atual = 0; atual <= 4; atual ++){
    if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
        maisBarato = atual;
    }
}
```

Este código é bastante utilizado para detectar qual preço é o mais barato ou qual é o menor valor de um *array*. É possível, daqui, extrair um método (uma **função**) deste recorte.

Moveremos uma parte do código e depois, vamos substuí-la pela função `int maisBarato = buscaMenor(produtos)` dentro do *array*. Em seguida, iremos criar o método.

```
public class TestaMenorPreco {

    public static void main(String[] args) {
        Produto produtos[] = {
                new Produto ("Lamborghini", 1000000),
                new Produto("Jipe", 46000),
                new Produto("Brasília", 16000),
                new Produto("Smart", 46000),
                new Produto("Fusca", 17000)
        };

        int maisBarato = buscaMenor(produtos);
        System.out.println(maisBarato);
        System.out.println("O carro" + produtos[maisBarato].getNome()
                                    + " é o mais barato, e custa"
```

```
                                        + produtos[maisBarato].getPreco());

    }

    private static int buscaMenor(Produto[] produtos) {
        return 0;
  }

}
```

Observe que o **método** recebe o *array* de produtos, então, apenas daremos um *paste* na parte retirada do código . No fim, é claro que não iremos retornar `0` . Não faria sentido... Retornaremos o `maisBarato` .

```
  private static int buscaMenor(Produto[] produtos) {
       int maisBarato = 0;
       for(int atual = 0; atual <= 4; atual++) {
           if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
               maisBarato = atual;
           }
        }
    return maisBarato;
  }
```

Depois disso, vamos salvar e rodar o código. O programa irá imprimir:

```
2
O carro Brasília é o mais barato, e custa 16000.0
```

Nós extraímos uma função que é um algoritmo de busca do menor valor dentro do *array*.

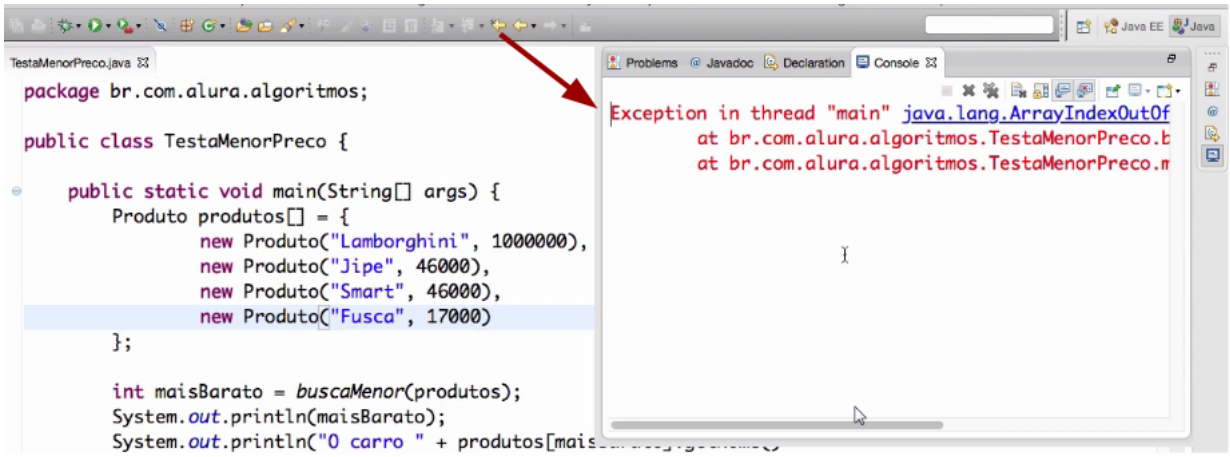
Buscando do início ao fim de um *array*

Vamos ser cuidadosos... O que acontece se tirarmos um produto da lista de cinco carros? Por exemplo, a Brasília...

```
        Produto produtos[] = {
                new Produto ("Lamborghini", 1000000),
                new Produto("Jipe", 46000),
                new Produto("Smart", 46000),
                new Produto("Fusca", 17000)
        };
```

Nossa lista fica com quatro produtos. Quando tentarmos rodar o código, o que aparecerá na tela?

```
Exception in thread "main"java.lang.ArrayIndexOutOfBoundsException: 4
        at br.com.alura.algoritmos.TestaMenorPreco.buscaMenor(TestaMenorPreco.java:24)
        at br.com.alura.algoritmos.TestMenorPreco.main(TestaMenorPreco.java:13)
```



O programa tentou acessar o produto na posição 4, porque o nosso `for` vai até a posição 4, inclusive. Porém, não temos mais cinco produtos. Agora, temos quatro elementos, que vão até a posição `3` . Temos um problema...

Nosso *array* tem um número fixo e o ideal é que o número seja do **tamanho** da nossa lista, menos `um` :

```
for(int atual = 0; atual <= produtos.length - 1); atual ++)
```

Considerando que agora temos apenas 4 elementos... Por isso, vou incluir a variável `termino = (produtos. length - 1);`

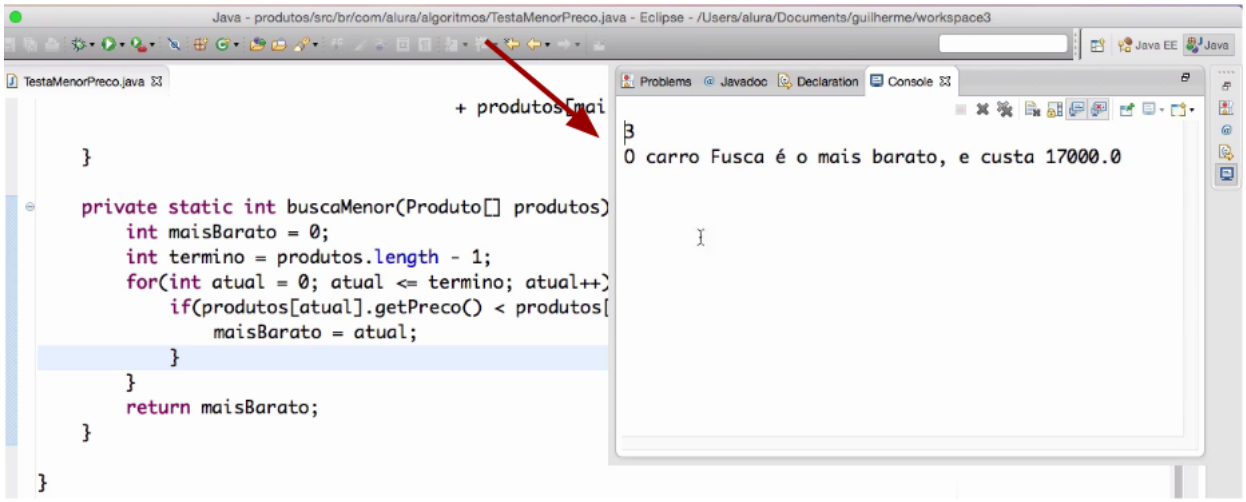
```
    private static int buscaMenor(Produto[] produtos) {
        int maisBarato = 0;
        int termino = produtos. length - 1;
        for(int atual = 0; atual <= termino; atual ++){
            if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
                maisBarato = atual;
          }
        }
        return maisBarato;
    }
```

Vamos testar nosso código? Quando todos os cinco produtos estiverem listados, nós vamos executar o programa e aparecerá que o carro mais barato é a Brasília.

Em seguida, apagamos a Brasília da lista, que ficará com quatro elementos. Vou rodar o código, que irá indicar o Fusca como o carro mais barato.

```
3
O carro Fusca é o mais barato, e custa 17000.0
```

Isto significa que o nosso código funcionou.



Independentemente do tamanho do *array*, o programa irá verificá-lo por completo, para então encontrar o menor valor possível. Nós **generalizamos** nossa função de busca de menor valor.

Buscando somente em um trecho específico de um *array*

Nós aprendemos a criar a função que verifica **todos** os elementos de um *array*, para em seguida, encontrar aquele com o menor valor. No exemplo dos carros, nós detectamos o menor preço. Se trabalhássemos com um *array* de *double*, nós compararíamos os valores. Se fôssemos analisar os dados sobre provas de alunos, faríamos uma comparação das notas.

Nós comparamos o que queremos e encontramos o menor valor de todos.

O que acontece se a nossa lista termina com diversos valores inválidos? Por exemplo, é comum criarmos um *array* com vários elementos, mas nem todos são válidos:

```
public static void main(String[] args) {
    Produto produtos[] = {
        new Produto ("Lamborghini", 1000000),
                new Produto("Jipe", 46000),
                new Produto("Brasília", 16000),
                new Produto("Smart", 46000),
                new Produto("Fusca", 17000)
                null,
                null,
                null,
                null,
                null
    };
```

Nós já temos cinco itens válidos, porém podemos incluir outros elementos sem preencher. Em Java, isto é bem incomum, porque na prática nós podemos usar um *arraylist* (outros elementos que representem um *array*). Mas em diversas linguagens, podemos não saber o tamanho exato (válido) da nossa lista.

No exemplo dos carros, nosso *array* tem nove elementos, porém apenas cinco são válidos. O programa irá comparar `null` com `null`, o que vai resultar em `Nullpointer` e isto não é bom. Por isso, de alguma maneira nós temos que indicar para a nossa função onde ela deve terminar.

O término do código é `produtos.length - 1`, isto significa que quando o produto for `5`, o processo deve parar no `4`. Mas podemos encontrar casos diferentes, em que um *array* pode conter valores vazios, além de linguagens em que não temos

como saber o seu tamanho ou quando ele acaba... Se não temos estes dados, não teremos `length`. Sendo assim, como iremos calcular o `termino`? É impossível calculá-lo.

```
 private static int buscaMenor(Produto[] produtos) {
    int maisBarato = 0;
    int termino = produtos.legth - 1;
    for(int atual= 0; atual <= termino; atual++) {
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
        }
    }
    return maisBarato;
}
```

Em todas estas situações, quando mandamos fazer uma busca dentro de uma lista, devemos indicar quantos elementos ela dispõe. No nosso caso, preciso dizer que temos `5` elementos.

```
int maisBarato = buscaMenor(produtos, 5)
```

Com as alterações, nosso código ficará assim:

```
    int maisBarato = buscaMenos(produtos, 5)
    System.out.println(maisBarato);
    System.out.println("O carro" + produtos[maisBarato].getNome()
                                 + " é o mais barato, e custa"
                                 + produtos[maisBarato].getPreco());

 private static int buscaMenor(Produto[] produtos) {
    int maisBarato = 0;
    int termino = produtos.legth - 1;
    for(int atual= 0; atual <= termino; atual++) {
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
        }
    }
    return maisBarato;
}
```

Isto significa que nós estamos informando para a função (o nosso método) qual é o tamanho do *array*.

```
private static int buscaMenor(Produto[] produtos, int termino) {
```

Com isso, eliminamos a linha:

```
int termino = produtos.legth - 1;
```

Na verdade, já podemos indicar o `termino` direto. Como o termino é `4`, podemos inseri-lo diretamente no meu código.

```
    int maisBarato = buscaMenos(produtos, 4)
    System.out.println(maisBarato);
    System.out.println("O carro" + produtos[maisBarato].getNome()
                                 + " é o mais barato, e custa"
                                 + produtos[maisBarato].getPreco());
```

No Java, nós podíamos usar `produtos.length`. Porém, em diversas linguagens nós não podemos fazer isso, porque não sabemos o `length` do nosso *array*. O responsável pela busca precisa nos informar o tamanho da lista... É o que alteramos no nosso código: informamos quantos produtos têm dentro do *array*. Isto é, estou falando para minha função (meu método) qual é o tamanho do meu do *array*.

```
    int maisBarato = buscaMenos(produtos, 4)
    System.out.println(maisBarato);
    System.out.println("O carro" + produtos[maisBarato].getNome()
                                 + " é o mais barato, e custa"
                                 + produtos[maisBarato].getPreco());

private static int buscaMenor(Produto[] produtos, int termino) {
    int maisBarato = 0;
    for(int atual = 0; atual <= termino; atual++) {
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
        }
    }
    return maisBarato;
}
```

Como temos cinco produtos, irá da posição `0` até a posição `4`. Vamos indicar isto no código?

```
int maisBarato = buscaMenor(produtos, 0, 4)
```

Vamos falar tudo de uma vez e indicar também o início e o termino:

```
private static int buscaMenor(Produto[] produtos, int inicio, int termino) {
    int maisBarato = inicio;
    for(int atual = inicio; atual <= termino; atual++){
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
        }
    }
    return maisBarato;
}
```

Fizemos uma busca do menor valor possível, de uma determinada posição à outra do *array*, do inicio até o fim, ou do `0` ao `length - 1`. Ele irá começar em uma parte do nosso *array* e vai terminar em outra. Do `int inicio` até o `int termino`.

```
int maisBarato = buscaMenos(produtos, 0, 4)
System.out.println(maisBarato);
System.out.println("O carro" + produtos[maisBarato].getNome()
                                 + " é o mais barato, e custa"
                                 + produtos[maisBarato].getPreco());
```

```
+ produtos[maisBarato].getPreco());

private static int buscaMenor(Produto[] produtos, int inicio, int termino) {
    int maisBarato = inicio;
    for(int atual = inicio; atual <= termino; atual++) {
        if(produtos[atual].getPreco() < produtos[maisBarato].getPreco()) {
            maisBarato = atual;
        }
    }
    return maisBarato;
}
```

Por que vamos específicar o `inicio` e o `termino`? Porque queremos generalizar nossa função e fazer com que ela funcione em diversas linguagens (incluindo aquelas que não suportam os objetos de um *array* ou quando não sabemos quantos são os elementos). A nossa função irá funcionar em todas essas situações, porque o programa irá varrer exatamente entre as posições determinadas, em busca do menor valor de todos. Se estamos verificando do `0` até `4 inclusive`, vamos levar em consideração da Lamborghini até o Fusca. Quando rodarmos o programa, vamos obter o mesmo resultado: o carro 2 é o mais barato. Se retirarmos os elementos `null`, o resultado será sempre o mesmo.

A minha função é genérica o suficiente para funcionar em diversas linguagens e em diversas situações. Tanto se estamos programando em C ou em Java e tenho diversos `nulls` (valores que quero ignorar) no meu *array*... independentemente se estão no começo ou no fim. Descartamos os elementos desnecessários e analisamos apenas um determinado pedaço para descobrir o menor valor de todos.

A função está mais poderosa agora.