Aula 10 – Arranjos

Norton Trevisan Roman

18 de abril de 2018

Arranjos

- Considere o código para calcular o valor da piscina:
- Qual o problema?

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA:
                    return(area*1500);
    case VINIL: return(area*1100);
    case FIBRA: return(area*750);
    case PLASTICO: return(area*500);
    default: return(-1);
  }
}
```

Arranjos

- Considere o código para calcular o valor da piscina:
- Qual o problema?
  - Todos os preços estão declarados dentro do método
  - Se o código crescer, fica mais difícil achar, em caso de mudança

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA:
                    return(area*1500);
    case VINIL: return(area*1100);
    case FIBRA: return(area*750);
    case PLASTICO: return(area*500);
    default: return(-1);
  }
}
```

Arranjos

- Considere o código para calcular o valor da piscina:
- Qual o problema?
  - Todos os preços estão declarados dentro do método
  - Se o código crescer, fica mais difícil achar, em caso de mudança
- Que fazer?

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA:
                    return(area*1500);
    case VINIL: return(area*1100);
    case FIBRA: return(area*750);
    case PLASTICO: return(area*500);
    default: return(-1);
  }
}
```

Arranjos

- Poderíamos agrupar essa informação, sob a forma de constantes

```
/* materiais da piscina */
static final int ALVENARIA = 0;
static final int VINIL = 1;
static final int FIBRA = 2;
static final int PLASTICO = 3;

/* preços dos materiais */
static final double
                  P_ALVENARIA = 1500;
static final double P_VINIL = 1100;
static final double P_FIBRA = 750;
static final double
                     P_PLASTICO = 500;
```

Arranjos

- Poderíamos agrupar essa informação, sob a forma de constantes
- Tornaria mais fácil a manutenção do código

```
/* materiais da piscina */
static final int ALVENARIA = 0;
static final int VINIL = 1;
static final int FIBRA = 2;
static final int PLASTICO = 3;

/* preços dos materiais */
static final double
                  P_ALVENARIA = 1500;
static final double P_VINIL = 1100;
static final double P_FIBRA = 750;
static final double
                       P_PLASTICO = 500;
```

Arranjos

- Poderíamos agrupar essa informação, sob a forma de constantes
- Tornaria mais fácil a manutenção do código
- Basta?

```
/* materiais da piscina */
static final int ALVENARIA = 0;
static final int VINIL = 1;
static final int FIBRA = 2;
static final int PLASTICO = 3;

/* preços dos materiais */
static final double
                  P_ALVENARIA = 1500;
static final double P_VINIL = 1100;
static final double P_FIBRA = 750;
static final double
                     P_PLASTICO = 500;
```

Arranjos

- Ainda temos que relacioná-las

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA: return(area*
                        P_ALVENARIA);
    case VINIL: return(area*
                        P_VINIL);
    case FIBRA: return(area*
                        P_FIBRA);
    case PLASTICO: return(area*
                        P_PLASTICO);
    default: return(-1);
  }
}
```

Arranjos

- Ainda temos que relacioná-las
- E como faríamos se quiséssemos calcular o preço médio dos materiais?

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA: return(area*
                          P_ALVENARIA);
    case VINIL: return(area*
                              P_VINIL);
    case FIBRA: return(area*
                              P_FIBRA);
    case PLASTICO: return(area*
                            P_PLASTICO);
    default: return(-1);
  }
}
```

Arranjos

```
(P_ALVENARIA + P_VINIL +
 P_FIBRA + P_PLASTICO)/4
```

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA: return(area*
                        P_ALVENARIA);
    case VINIL: return(area*
                        P_VINIL);
    case FIBRA: return(area*
                        P_FIBRA);
    case PLASTICO: return(area*
                        P_PLASTICO);
    default: return(-1);
  }
}
```

Arranjos

```
(P_ALVENARIA + P_VINIL +
 P_FIBRA + P_PLASTICO)/4
```

- Deve haver um meio melhor, que mantenha o agrupamento, simplifique o código e facilite esse tipo de cálculo

```
static double valorPiscina(
        double area, int material) {
  double valor;

  switch (material) {
    case ALVENARIA: return(area*
                           P_ALVENARIA);
    case VINIL: return(area*
                           P_VINIL);
    case FIBRA: return(area*
                           P_FIBRA);
    case PLASTICO: return(area*
                           P_PLASTICO);
    default: return(-1);
  }
}
```

Arranjos

- Usamos Arranjos (Array):
  - Estruturas de dados, de tamanho fixo, que permitem armazenar um conjunto de valores de um mesmo tipo

Arranjos

- Usamos Arranjos (Array):
  - Estruturas de dados, de tamanho fixo, que permitem armazenar um conjunto de valores de um mesmo tipo

Em vez de termos

```
static final double P_ALVENARIA = 1500;
static final double P_VINIL = 1100;
static final double P_FIBRA = 750;
static final double P_PLASTICO = 500;
```

Podemos fazer

```
static double[] precos = {1500, 1100, 750, 500};
```

ou

```
static double precos[] = {1500, 1100, 750, 500};
```

Arranjos

Deixou de ser constante

```
static final double P_ALVENARIA = 1500;
static final double P_VINIL = 1100;
static final double P_FIBRA = 750;
static final double P_PLASTICO = 500;
```

```
static double[] precos = {1500, 1100, 750, 500};
```

ou

```
static double precos[] = {1500, 1100, 750, 500};
```

Arranjos

Deixou de ser constante

```
static final double P_ALVENARIA = 1500;
static final double P_VINIL = 1100;
static final double P_FIBRA = 750;
static final double P_PLASTICO = 500;
```

Mas deixou o código mais enxuto

```
static double[] precos = {1500, 1100, 750, 500};
```

ou

```
static double precos[] = {1500, 1100, 750, 500};
```

Arranjos

- `static double[] precos = {1500, 1100, 750, 500};` diz ao compilador para reservar espaço na memória para 4 doubles

Arranjos

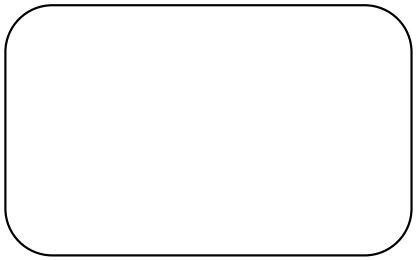
- `static double[] precos = {1500, 1100, 750, 500};` diz ao compilador para reservar espaço na memória para 4 doubles
- Armazenando os valores 1500, 1100, 750, 500 neles

Arranjos

- `static double[] precos = {1500, 1100, 750, 500};` diz ao compilador para reservar espaço na memória para 4 doubles
- Armazenando os valores 1500, 1100, 750, 500 neles
- Por enquanto, digamos que o `static` está aí por `precos` ser um atributo do programa...

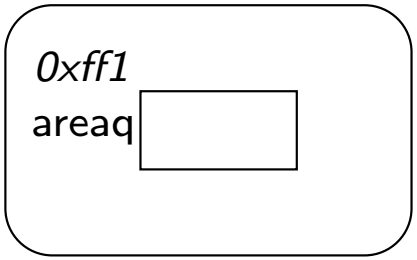
Revendo a Memória

- O que acontece ao fazermos `float areaq;`?



Revendo a Memória

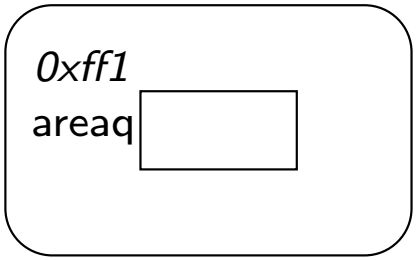
- O que acontece ao fazermos `float areaq;`?



- Alocamos um espaço para a variável `areaq` grande o suficiente para guardar um `float` (4B), e cujo endereço o compilador conhece (o 0xff1 na figura)

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?



- Alocamos um espaço para a variável `areaq` grande o suficiente para guardar um `float` (4B), e cujo endereço o compilador conhece (o 0xff1 na figura)
- Qualquer valor para `areaq` é armazenado diretamente nesse espaço – Armazena o valor

Revendo a Memória

- Endereço?

Revendo a Memória

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço

Revendo a Memória

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
- Normalmente em hexadecimal

Revendo a Memória

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
- Normalmente em hexadecimal
  - Decimal: 0, 1, . . . , 9. Em binário, de 0000 a 1001

Revendo a Memória

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
- Normalmente em hexadecimal
  - Decimal: 0, 1, . . . , 9. Em binário, de 0000 a 1001
  - Binário: 0, 1

Revendo a Memória

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
- Normalmente em hexadecimal
  - Decimal: 0, 1, . . . , 9. Em binário, de 0000 a 1001
  - Binário: 0, 1
  - Octal: 0, 1, . . . , 7. Em binário, de 000 a 111

Revendo a Memória

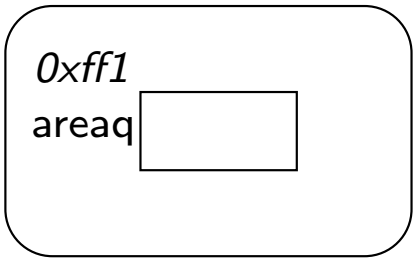
- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
- Normalmente em hexadecimal
  - Decimal: 0, 1, . . . , 9. Em binário, de 0000 a 1001
  - Binário: 0, 1
  - Octal: 0, 1, . . . , 7. Em binário, de 000 a 111
  - Hexadecimal: 0, 1, . . . , 9, A, B, . . . , F. Em binário, de 0000 a 1111

Revendo a Memória

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
- Normalmente em hexadecimal
  - Decimal: 0, 1, . . . , 9. Em binário, de 0000 a 1001
  - Binário: 0, 1
  - Octal: 0, 1, . . . , 7. Em binário, de 000 a 111
  - Hexadecimal: 0, 1, . . . , 9, A, B, . . . , F. Em binário, de 0000 a 1111
  - Note que tanto Octal quanto Hexa usam todos os bits a eles alocados

Revendo a Memória

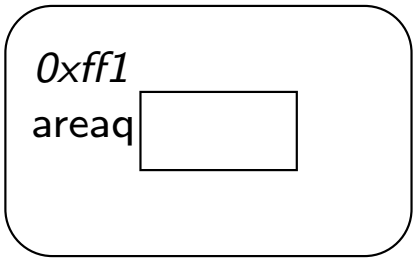
- No caso de `float areaq`, são alocados 4B contíguos na memória, sendo *0xff1* o endereço do primeiro deles



- O compilador, para sua facilidade, deixa você dar nomes a esses endereços $\rightarrow$ são as variáveis
- Os nomes das variáveis são, então, o mapeamento feito pelo compilador a esses endereços
  - Um nome ou rótulo dado a esse local de memória

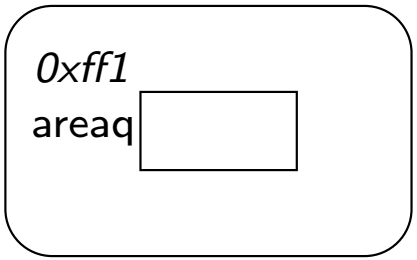
Revendo a Memória

- O programador não precisa saber qual é esse endereço



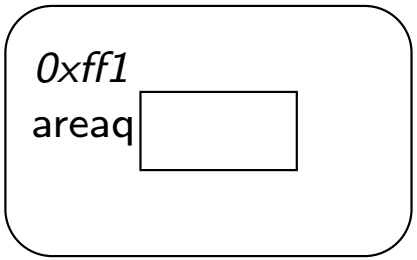
Revendo a Memória

- O programador não precisa saber qual é esse endereço
- Diz-se que a informação foi abstraída:



Revendo a Memória

- O programador não precisa saber qual é esse endereço
- Diz-se que a informação foi abstraída:

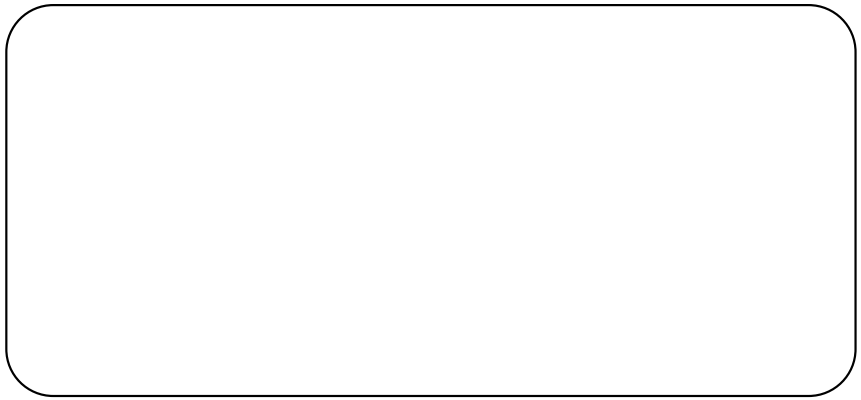


Olha-se o problema sob um ângulo em que não há a necessidade de se saber o valor desse endereço

Arranjos na Memória

- O que acontece ao fazermos

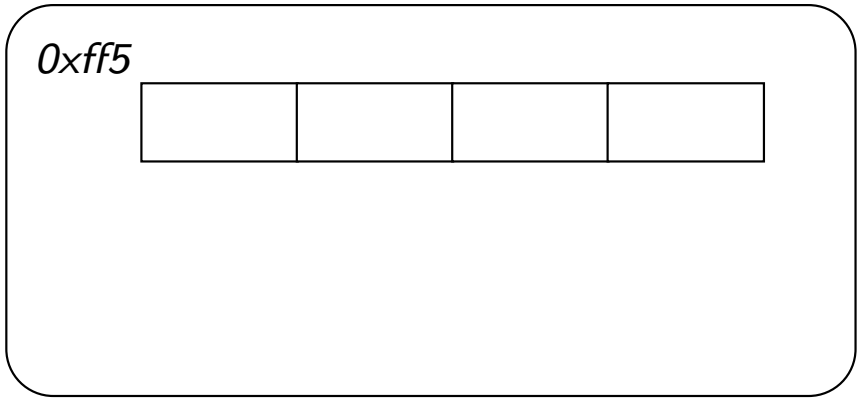
```
double[] precos =
{1500, 1100, 750, 500};?
```



Arranjos na Memória

- O que acontece ao fazermos

  `double[] precos = {1500, 1100, 750, 500};`?

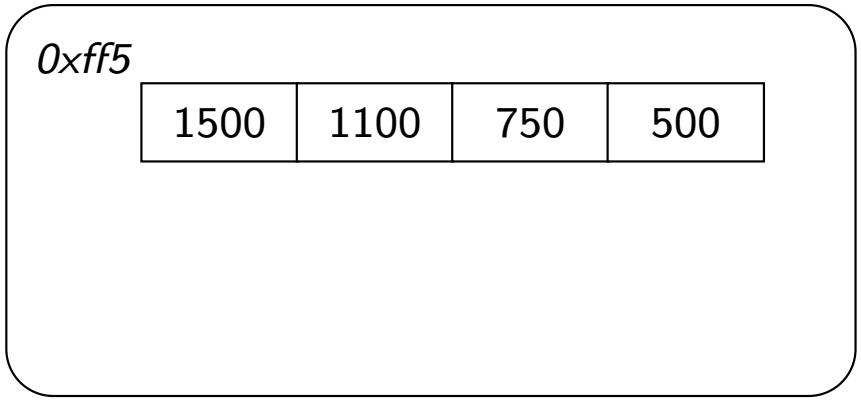


- O compilador aloca espaço suficiente para 4 doubles consecutivos (32B)

Arranjos na Memória

- O que acontece ao fazermos

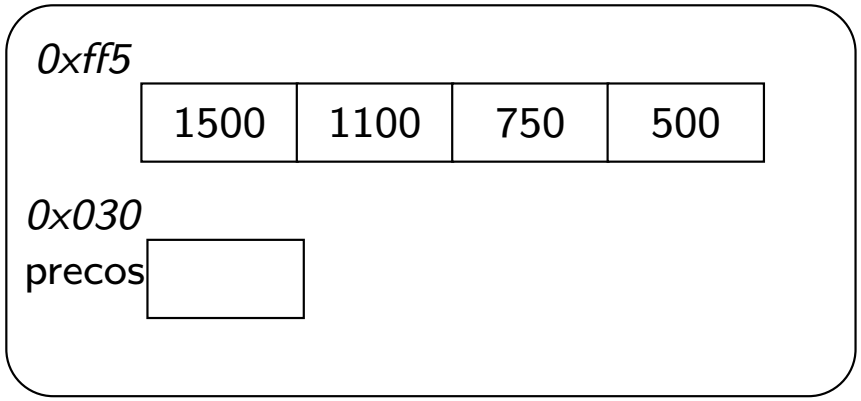
```
double[] precos =
{1500, 1100, 750, 500};?
```



- O compilador aloca espaço suficiente para 4 doubles consecutivos (32B)
- Guarda os valores da inicialização lá

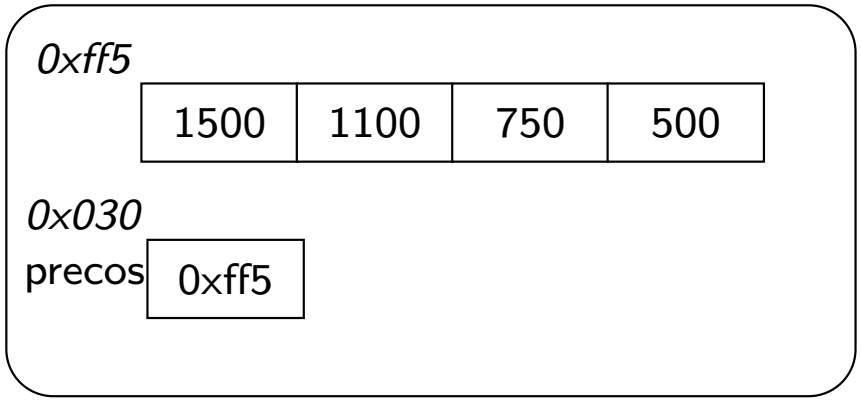
Arranjos na Memória

- Em seguida aloca memória para a variável `precos`, grande o suficiente para caber um endereço, e cujo endereço o compilador também conhece



Arranjos na Memória

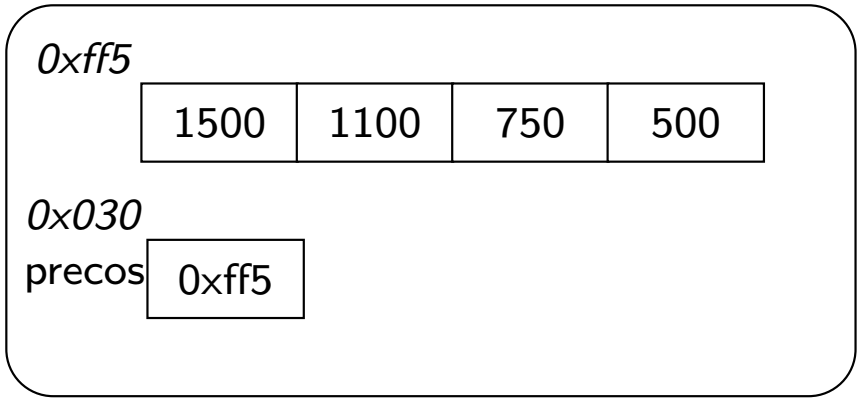
- Em seguida aloca memória para a variável `precos`, grande o suficiente para caber um endereço, e cujo endereço o compilador também conhece



- Então guarda em `precos` o endereço na memória do primeiro byte do arranjo

Arranjos na Memória

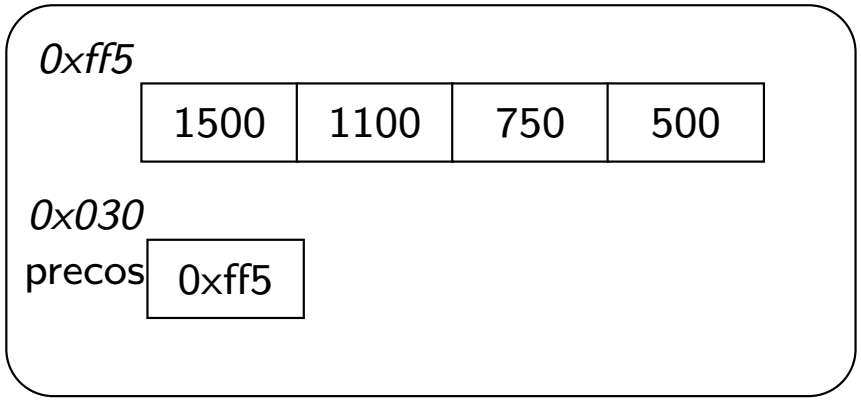
- Em seguida aloca memória para a variável `precos`, grande o suficiente para caber um endereço, e cujo endereço o compilador também conhece



- Então guarda em `precos` o endereço na memória do primeiro byte do arranjo
- Guarda o endereço $\rightarrow$ armazena uma referência ao início do arranjo

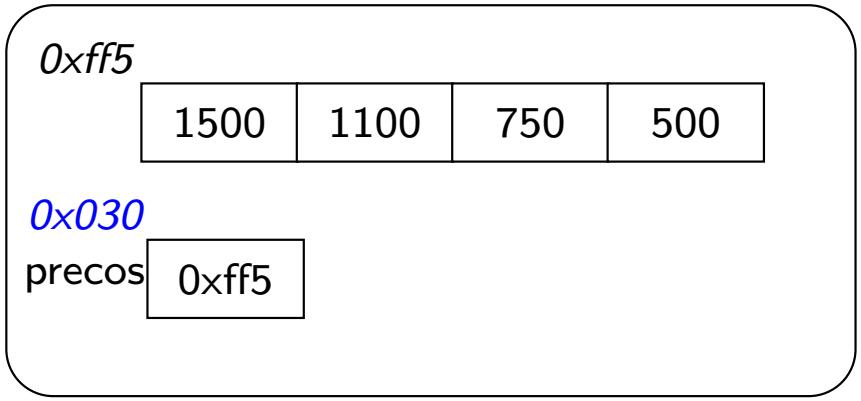
Arranjos na Memória

- Para chegar ao primeiro elemento do arranjo, o computador:



Arranjos na Memória

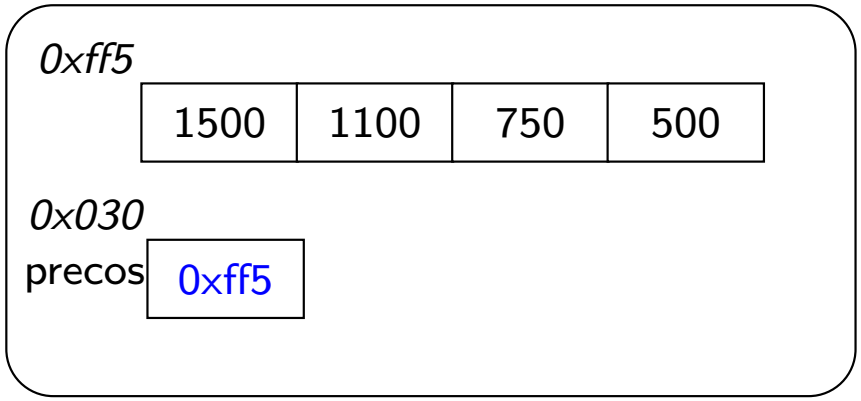
- Para chegar ao primeiro elemento do arranjo, o computador:



- Vai à região da memória correspondente a `preços`

Arranjos na Memória

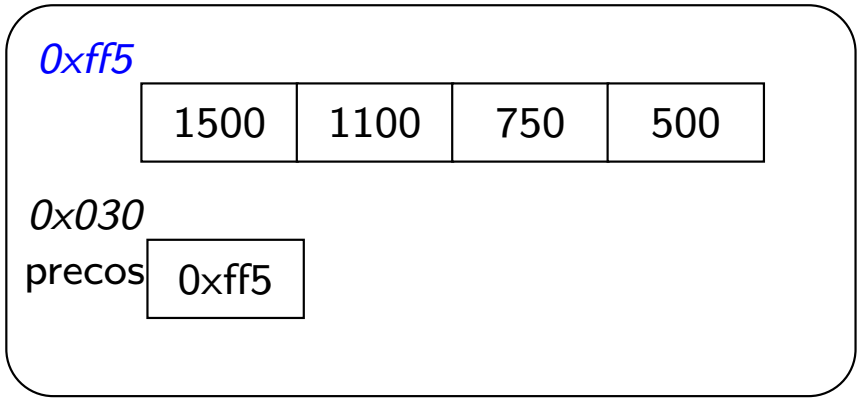
- Para chegar ao primeiro elemento do arranjo, o computador:



- Vai à região da memória correspondente a `preços`
- Lê seu conteúdo – endereço do primeiro byte do arranjo

Arranjos na Memória

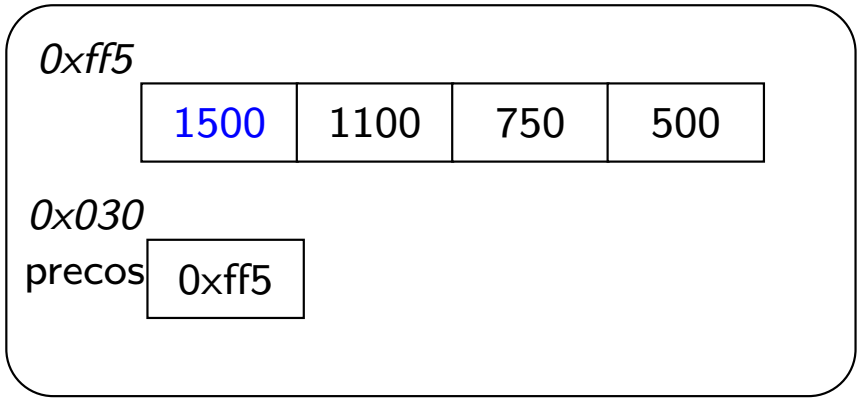
- Para chegar ao primeiro elemento do arranjo, o computador:



- Vai à região da memória correspondente a `preços`
- Lê seu conteúdo – endereço do primeiro byte do arranjo
- Vai então a essa região da memória

Arranjos na Memória

- Para chegar ao primeiro elemento do arranjo, o computador:



- Vai à região da memória correspondente a `preços`
- Lê seu conteúdo – endereço do primeiro byte do arranjo
- Vai então a essa região da memória e lê o conteúdo

Arranjos na Memória

- Em java, como podemos ler um elemento do arranjo?

Arranjos na Memória

- Em java, como podemos ler um elemento do arranjo?
- Fazendo `arranjo[indice]`
  - Onde índice é um inteiro de 0 a $n - 1$, com n sendo o número de elementos do arranjo
  - 0 corresponde ao primeiro elemento, 1 ao segundo, etc

Arranjos na Memória – Exemplo

```
public static void main(String[] args) {
  System.out.println(precos[0]);
  System.out.println(precos[1]);
  System.out.println(precos[2]);
  System.out.println(precos[3]);
}
```

Arranjos na Memória – Exemplo

```
public static void main(String[] args) {
  System.out.println(precos[0]);
  System.out.println(precos[1]);
  System.out.println(precos[2]);
  System.out.println(precos[3]);
}
```

Ou:

```
public static void main(String[] args) {
  for (int i=0; i<4; i++) {
    System.out.println(precos[i]);
  }
}
```

Arranjos na Memória

- Uma vez que o índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código

```
public static void main(String[]
                                  args) {
  for (int i=0; i<4; i++) {
    System.out.println(precos[i]);
  }
}
```

```
public static void main(String[]
                                    args) {
  for (int i=ALVENARIA; i<=PLASTICO;
                                    i++) {
    System.out.println(precos[i]);
  }
}
```

Arranjos na Memória

- Uma vez que o índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código
- Repare no `main`... `String[] args`

```
public static void main(String[]
                                 args) {
  for (int i=0; i<4; i++) {
    System.out.println(precos[i]);
  }
}
```

```
public static void main(String[]
                                   args) {
  for (int i=ALVENARIA; i<=PLASTICO;
                                     i++) {
    System.out.println(precos[i]);
  }
}
```

Arranjos na Memória

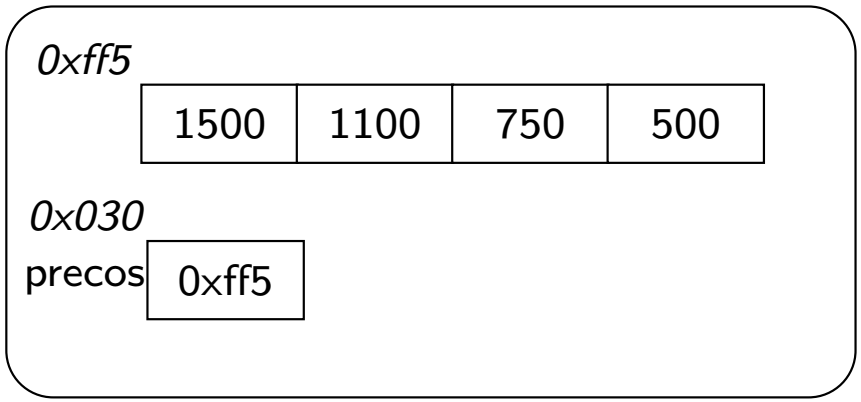
- Uma vez que o índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código
- Repare no `main`... `String[] args`
- `args` também é um arranjo

```
public static void main(String[]
                                 args) {
  for (int i=0; i<4; i++) {
    System.out.println(precos[i]);
  }
}
```

```
public static void main(String[]
                                   args) {
  for (int i=ALVENARIA; i<=PLASTICO;
                                     i++) {
    System.out.println(precos[i]);
  }
}
```

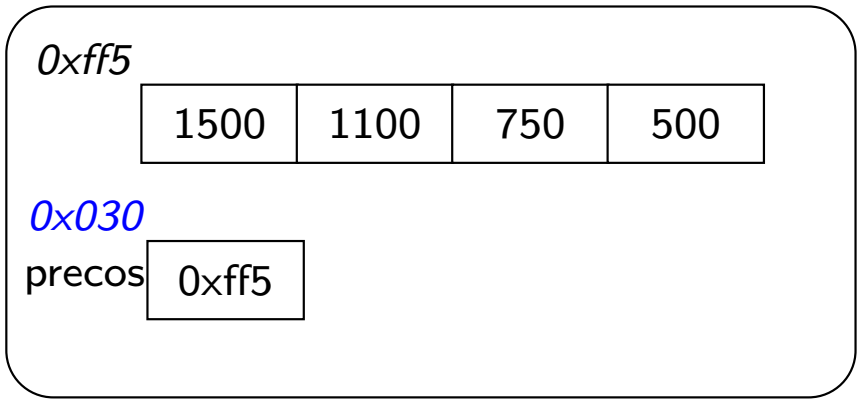
Índices na Memória

- Como o computador faz para achar o elemento de índice `i` do arranjo `precos`?



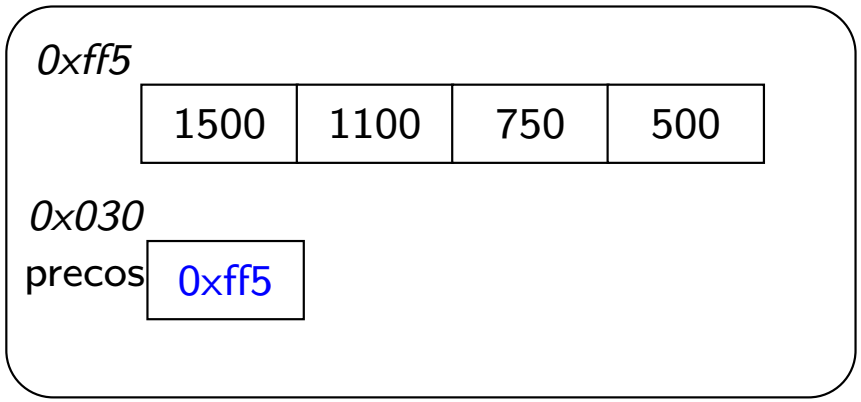
Índices na Memória

- Como o computador faz para achar o elemento de índice `i` do arranjo `precos`?
- Primeiro, vai à região da memória correspondente a `precos`



Índices na Memória

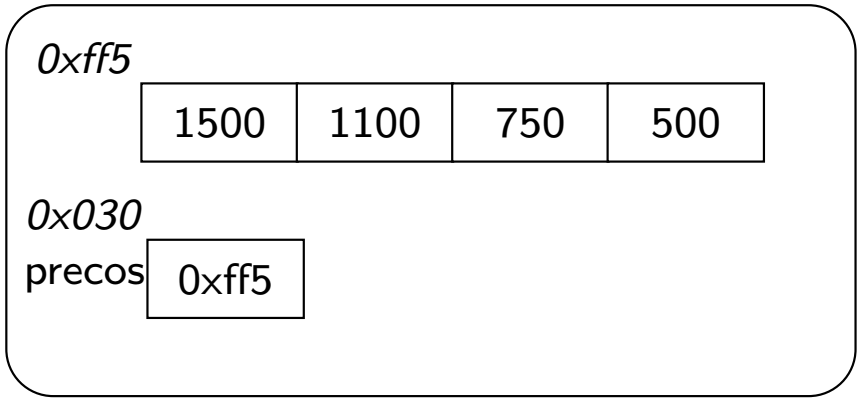
- Como o computador faz para achar o elemento de índice `i` do arranjo `precos`?
- Primeiro, vai à região da memória correspondente a `precos`



- Lê seu conteúdo – endereço do primeiro byte do arranjo

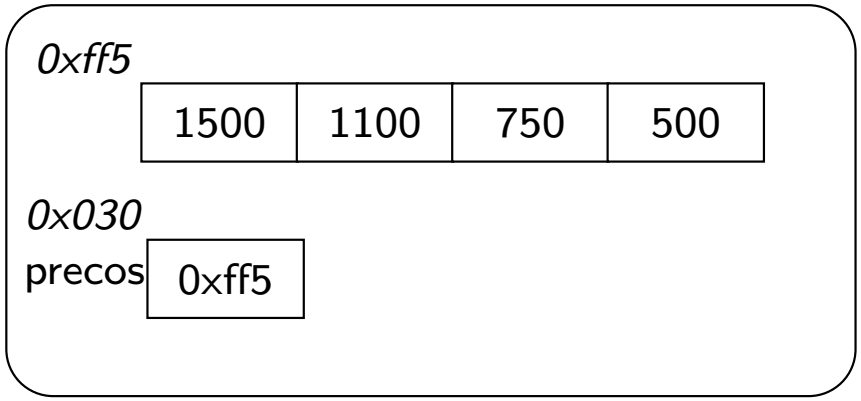
Índices na Memória

- Calcula então a posição do elemento de índice `i`:



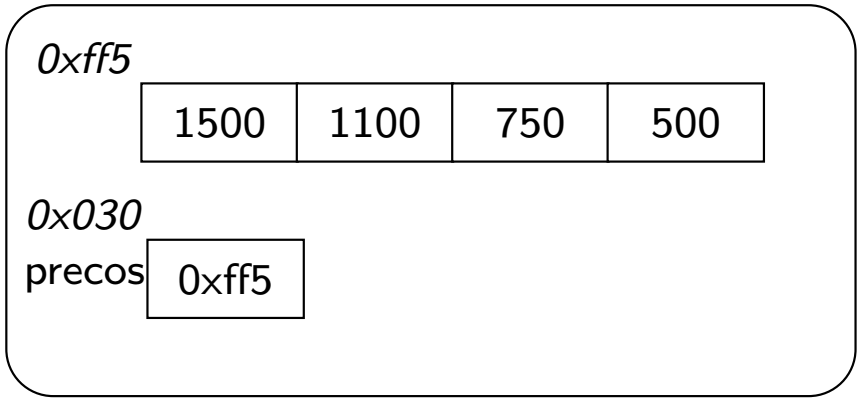
Índices na Memória

- Calcula então a posição do elemento de índice `i`:
- Sabendo que cada elemento tem 8B (double), e que $0 \leq \texttt{i} \leq \texttt{n} - 1$



Índices na Memória

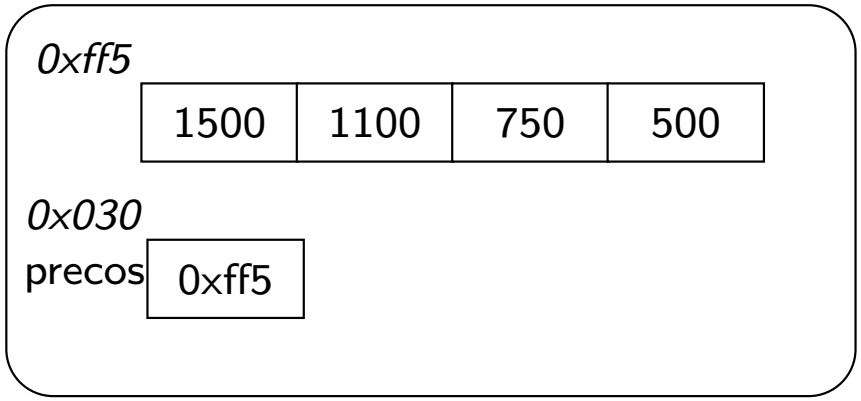
- Calcula então a posição do elemento de índice `i`:
- Sabendo que cada elemento tem 8B (double), e que $0 \leq \texttt{i} \leq \texttt{n} - 1$



- O elemento estará a $8 \times \texttt{i}$ bytes do início do arranjo

Índices na Memória

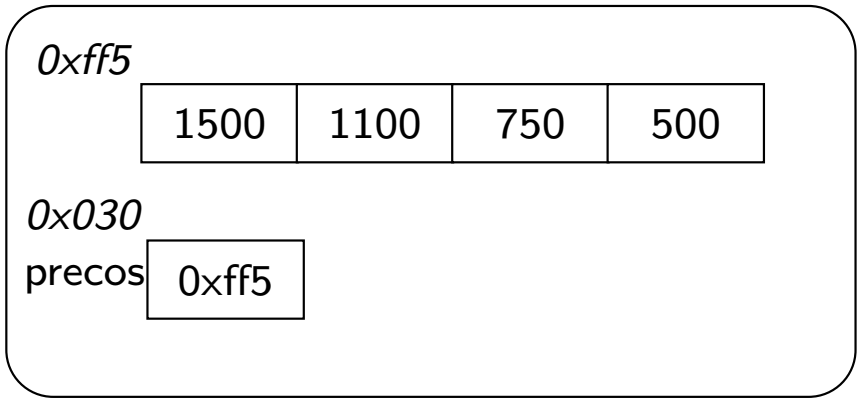
- Calcula então a posição do elemento de índice `i`:
- Sabendo que cada elemento tem 8B (double), e que $0 \leq \texttt{i} \leq \texttt{n} - 1$



- O elemento estará a $8 \times \texttt{i}$ bytes do início do arranjo
- O endereço será $\texttt{0xff5} + 8 \times \texttt{i}$

Índices na Memória

- Vai à região da memória correspondente a esse endereço e lê seu conteúdo



- Por isso o índice começa no 0. Se `i` $= 0$, o endereço visitado será o do início do arranjo

Arranjos

- Voltemos agora a `valorPiscina`...

```
static double valorPiscina(double
                    area, int material) {
  switch (material) {
    case ALVENARIA: return(area*
                          P_ALVENARIA);
    case VINIL: return(area*
                               P_VINIL);
    case FIBRA: return(area*
                               P_FIBRA);
    case PLASTICO: return(area*
                            P_PLASTICO);
    default: return(-1);
  }
}
```

Arranjos

- Voltemos agora a `valorPiscina`...
- Como ficaria usando o arranjo `precos`?

```
static double valorPiscina(double
                  area, int material) {
  switch (material) {
    case ALVENARIA: return(area*
                          P_ALVENARIA);
    case VINIL: return(area*
                               P_VINIL);
    case FIBRA: return(area*
                               P_FIBRA);
    case PLASTICO: return(area*
                            P_PLASTICO);
    default: return(-1);
  }
}
```

Arranjos

```
static double valorPiscina(double
                  area, int material) {
  if (material<ALVENARIA ||
                   material>PLASTICO ||
                    area<0) return(-1);

  return(area*precos[material]);
}
```

Arranjos

- Incluímos o teste para a área

```
static double valorPiscina(double
                    area, int material) {
  if (material<ALVENARIA ||
                     material>PLASTICO ||
                      area<0) return(-1);

  return(area*precos[material]);
}
```

Arranjos

- Incluímos o teste para a área
- Usamos o código do tipo do material como índice em `precos`

```
static double valorPiscina(double
                    area, int material) {
  if (material<ALVENARIA ||
                     material>PLASTICO ||
                      area<0) return(-1);

  return(area*precos[material]);
}
```

Arranjos

- Incluímos o teste para a área
- Usamos o código do tipo do material como índice em `precos`

```
static double valorPiscina(double
                    area, int material) {
  if (material<ALVENARIA ||
                     material>PLASTICO ||
                      area<0) return(-1);

  return(area*precos[material]);
}
```

- É importante certificar-se que `precos[0]` tem o preço de `ALVENARIA`, que `precos[1]` tem o preço de `VINIL` etc

Arranjos

- Elementos em arranjos podem ser atribuídos a outras variáveis, como uma variável comum:
  - `double x = precos[0];`
- Desde que o tipo da variável seja o mesmo do tipo armazenado no arranjo
  - Do contrário, um type cast será necessário

Arranjos

- Elementos em arranjos podem ser atribuídos a outras variáveis, como uma variável comum:
  - `double x = precos[0];`
- Desde que o tipo da variável seja o mesmo do tipo armazenado no arranjo
  - Do contrário, um type cast será necessário
- E o que acontece se tentamos acessar um elemento fora dos limites do arranjo?

Arranjos

Código

```
public static void main(String[]
                                args) {
  double x = precos[-1];
}
```

Saída

Arranjos

Código

```
public static void main(String[]
                                args) {
  double x = precos[-1];
}
```

Saída

```
$ javac AreaCasa
$ java AreaCasa
Exception in thread "main"
  java.lang.ArrayIndexOutOfBounds
                              Exception: -1
at AreaCasa.main(AreaCasa.java:73)
```

Arranjos

Código

```
public static void main(String[]
                                args) {
  double x = precos[-1];
}
```

Saída

```
$ javac AreaCasa
$ java AreaCasa
Exception in thread "main"
  java.lang.ArrayIndexOutOfBounds
                              Exception: -1
at AreaCasa.main(AreaCasa.java:73)
```

```
public static void main(String[]
                                args) {
  double x = precos[4];
}
```

Arranjos

Código

```
public static void main(String[]
                                args) {
  double x = precos[-1];
}
```

Saída

```
$ javac AreaCasa
$ java AreaCasa
Exception in thread "main"
  java.lang.ArrayIndexOutOfBounds
                                Exception: -1
at AreaCasa.main(AreaCasa.java:73)
```

```
public static void main(String[]
                                args) {
  double x = precos[4];
}
```

```
$ javac AreaCasa
$ java AreaCasa
Exception in thread "main"
  java.lang.ArrayIndexOutOfBounds
                                 Exception: 4
at AreaCasa.main(AreaCasa.java:73)
```

Arranjos

Código

```
public static void main(String[]
                                args) {
  double x = precos[-1];
}
```

Saída

```
$ javac AreaCasa
$ java AreaCasa
Exception in thread "main"
  java.lang.ArrayIndexOutOfBounds
                                Exception: -1
at AreaCasa.main(AreaCasa.java:73)
```

```
public static void main(String[]
                                args) {
  double x = precos[4];
}
```

```
$ javac AreaCasa
$ java AreaCasa
Exception in thread "main"
  java.lang.ArrayIndexOutOfBounds
                                 Exception: 4
at AreaCasa.main(AreaCasa.java:73)
```

Compilará... mas gerará erro ao executar

Arranjos

- E como podemos dar um novo valor a um elemento do arranjo?

Arranjos

- E como podemos dar um novo valor a um elemento do arranjo?
  - `arranjo[índice] = novo_valor`
  - `precos[2] = 500;`

Arranjos

- E como podemos dar um novo valor a um elemento do arranjo?
  - `arranjo[índice] = novo_valor`
  - `precos[2] = 500;`
- Vimos que, para criar um arranjo, basta fazer
  - `double[] precos = {1500, 1100, 750, 500};`

Arranjos

- E como podemos dar um novo valor a um elemento do arranjo?
  - `arranjo[índice] = novo_valor`
  - `precos[2] = 500;`
- Vimos que, para criar um arranjo, basta fazer
  - `double[] precos = {1500, 1100, 750, 500};`
- Seria a única maneira?

Arranjos

```
public static void main(String[]
                                  args) {
  double[] precos2 = new double[4];

  precos2[ALVENARIA] = 1500;
  precos2[VINIL] = 1100;
  precos2[FIBRA] = 750;
  precos2[PLASTICO] = 500;
}
```

Arranjos

- Criamos um arranjo de 4 elementos, todos com valor zero

```
public static void main(String[]
                                    args) {
  double[] precos2 = new double[4];

  precos2[ALVENARIA] = 1500;
  precos2[VINIL] = 1100;
  precos2[FIBRA] = 750;
  precos2[PLASTICO] = 500;
}
```

Arranjos

- Criamos um arranjo de 4 elementos, todos com valor zero
- Inicializamos então esse arranjo

```
public static void main(String[]
                                    args) {
  double[] precos2 = new double[4];

  precos2[ALVENARIA] = 1500;
  precos2[VINIL] = 1100;
  precos2[FIBRA] = 750;
  precos2[PLASTICO] = 500;
}
```

Arranjos

```
double[] precos = {1500, 1100,
                   750, 500};
```

```
double[] precos2 =
            new double[4];

precos2[ALVENARIA] = 1500;
precos2[VINIL] = 1100;
precos2[FIBRA] = 750;
precos2[PLASTICO] = 500;
```

Arranjos

```
double[] precos = {1500, 1100,
                   750, 500};
```

```
double[] precos2 =
                new double[4];

precos2[ALVENARIA] = 1500;
precos2[VINIL] = 1100;
precos2[FIBRA] = 750;
precos2[PLASTICO] = 500;
```

- Útil se conhecemos os valores de antemão

Arranjos

```
double[] precos = {1500, 1100,
                   750, 500};
```

```
double[] precos2 =
              new double[4];

precos2[ALVENARIA] = 1500;
precos2[VINIL] = 1100;
precos2[FIBRA] = 750;
precos2[PLASTICO] = 500;
```

- Útil se conhecemos os valores de antemão
- E se esses valores são poucos

Arranjos

```
double[] precos = {1500, 1100,
                   750, 500};
```

```
double[] precos2 =
              new double[4];

precos2[ALVENARIA] = 1500;
precos2[VINIL] = 1100;
precos2[FIBRA] = 750;
precos2[PLASTICO] = 500;
```

- Útil se conhecemos os valores de antemão
- E se esses valores são poucos

- Útil se não conhecemos os valores de antemão

Arranjos

```
double[] precos = {1500, 1100,
                   750, 500};
```

```
double[] precos2 =
              new double[4];

precos2[ALVENARIA] = 1500;
precos2[VINIL] = 1100;
precos2[FIBRA] = 750;
precos2[PLASTICO] = 500;
```

- Útil se conhecemos os valores de antemão
- E se esses valores são poucos

- Útil se não conhecemos os valores de antemão
- Ou se esses valores são muitos

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - `tipo[] nomeDaVariavel = new tipo[tamanho do arranjo]`
- Em que `tamanho do arranjo` é o número de elementos que ele conterá
- Seus índices variando de 0 a `tamanho` $-$ 1

Arranjos

Ex:

- `float[] precos2 = new float[4];`
- `double[] precos2 = new double[4];`
- `int[] tamanhos = new int[10];`
- `long[] tamanhos = new long[10];`
- `boolean[] comprados = new boolean[20];`
- etc (veremos mais adiante)

Arranjos

- Vamos calcular o preço médio dos materiais de nossa piscina

Arranjos

- Vamos calcular o preço médio dos materiais de nossa piscina
- Como fazer?

Arranjos

- Vamos calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços, e dividindo pelo número deles

Arranjos

- Vamos calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços, e dividindo pelo número deles

```
public static void main(
                    String[] args) {
  double media = 0;

  for (int i=0; i<4; i++) {
    media += precos[i];
  }
  media = media/4;

  System.out.println(media);
}
```

Arranjos

- Funciona... mas e se tivermos que aumentar o tamanho do arranjo?

```
public static void main(
                  String[] args) {
  double media = 0;

  for (int i=0; i<4; i++) {
    media += precos[i];
  }
  media = media/4;

  System.out.println(media);
}
```

Arranjos

- Funciona... mas e se tivermos que aumentar o tamanho do arranjo?
- Teremos que mudar o limite do `for` também

```
public static void main(
                  String[] args) {
  double media = 0;

  for (int i=0; i<4; i++) {
    media += precos[i];
  }
  media = media/4;

  System.out.println(media);
}
```

Arranjos

- Funciona... mas e se tivermos que aumentar o tamanho do arranjo?
- Teremos que mudar o limite do `for` também
- Deve haver um meio melhor...

```
public static void main(
                    String[] args) {
  double media = 0;

  for (int i=0; i<4; i++) {
    media += precos[i];
  }
  media = media/4;

  System.out.println(media);
}
```

Arranjos

- Em java, arranjos vêm com o atributo pré-definido `length`, contendo seu comprimento
- Seu valor é definido automaticamente pelo compilador

```
public static void main(
                    String[] args) {
  double media = 0;

  for (int i=0;i<precos.length;
                              i++) {
    media += precos[i];
  }
  media = media/precos.length;

  System.out.println(media);
}
```

Arranjos

- Esse valor não pode ser alterado

Arranjos

- Esse valor não pode ser alterado
- Se fizermos `precos.length = 10;` teremos a mensagem:

```
$ javac AreaCasa.java
AreaCasa.java:82: cannot assign
      a value to final variable
      length
            precos.length = 10;
                          ^
1 error
```

For com Arranjos

- Vimos 3 tipos de laços: `while`, `do...while` e `for`
- Com isso, o cálculo da média ficava:

```
public static void main(
                    String[] args) {
  double media = 0;

  for (int i=0;i<precos.length;
                              i++) {
    media += precos[i];
  }
  media = media/precos.length;

  System.out.println(media);
}
```

For com Arranjos

- Existe, contudo, um quarto tipo em java

For com Arranjos

- Existe, contudo, um quarto tipo em java
- Na verdade, um segundo tipo de `for`, projetado para iterar, dentre outras coisas, em arranjos

For com Arranjos

- Existe, contudo, um quarto tipo em java
- Na verdade, um segundo tipo de `for`, projetado para iterar, dentre outras coisas, em arranjos

```
public static void main(
                    String[] args) {
  double media = 0;

  for (double valor : precos) {
    media += valor;
  }
  media = media/precos.length;

  System.out.println(media);
}
```

For com Arranjos

```
public static void main(String[]
                              args) {
  double media = 0;
  for (int i=0;i<precos.length;
                               i++){
    media += precos[i];
  }
  media = media/precos.length;
  System.out.println(media);
}
```

- Itera no índice do arranjo

```
public static void main(String[]
                              args) {
  double media = 0;
  for (double valor : precos) {
    media += valor;
  }

  media = media/precos.length;
  System.out.println(media);
}
```

- Itera em cada valor do arranjo

For com Arranjos

```
public static void main(String[]
                                args) {
  double media = 0;
  for (int i=0;i<precos.length;
                                i++){
    media += precos[i];
  }
  media = media/precos.length;
  System.out.println(media);
}
```

- Usa o iterador (`i`) como índice para acessar o valor

```
public static void main(String[]
                                args) {
  double media = 0;
  for (double valor : precos) {
    media += valor;
  }

  media = media/precos.length;
  System.out.println(media);
}
```

- O iterador (`valor`) contém o próprio valor

For com Arranjos

```
public static void main(String[]
                              args) {
  double media = 0;
  for (int i=0;i<precos.length;
                              i++){
    media += precos[i];
  }
  media = media/precos.length;
  System.out.println(media);
}
```

- Acessa o valor de maneira indireta

```
public static void main(String[]
                              args) {
  double media = 0;
  for (double valor : precos) {
    media += valor;
  }

  media = media/precos.length;
  System.out.println(media);
}
```

- Acessa o valor de maneira direta

Videoaula

https://www.youtube.com/watch?v=Iq50qq6QfUU
e
https://www.youtube.com/watch?v=H6kWSniXHto