Aula 09 – Arranjos

Norton Trevisan Roman

12 de abril de 2013

Arranjos

- Considere o código para calcular o valor da piscina:

```
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?

```
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

```
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

- 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?
  - Poderíamos agrupar essa informação, sob a forma de constantes

```
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?
  - Poderíamos agrupar essa informação, sob a forma de constantes

```
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);
    }
}
```

```
/* 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

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

```
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);
    }
}
```

```
/* 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

- 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

- Basta?
  - Ainda temos que relacioná-las

```
/* 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

- Basta?
  - Ainda temos que relacioná-las

```
/* 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;


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

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

```
/* 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;


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

- Basta?
  - Ainda temos que relacioná-las
- E mais... como faríamos se quiséssemos calcular o preço médio dos materiais?
  - `(P_ALVENARIA + P_VINIL + P_FIBRA + P_PLASTICO)/4` ?

```
/* 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;


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

- Basta?
  - Ainda temos que relacioná-las
- E mais... como faríamos se quiséssemos calcular o preço médio dos materiais?
  - `(P_ALVENARIA + P_VINIL + P_FIBRA + P_PLASTICO)/4` ?
- Deve haver um meio melhor

```
/* 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;


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

- Basta?
  - Ainda temos que relacioná-las
- E mais... como faríamos se quiséssemos calcular o preço médio dos materiais?
  - `(P_ALVENARIA + P_VINIL + P_FIBRA + P_PLASTICO)/4` ?
- Deve haver um meio melhor
  - Que mantenha o agrupamento

```
/* 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;


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

- Basta?
  - Ainda temos que relacioná-las
- E mais... como faríamos se quiséssemos calcular o preço médio dos materiais?
  - `(P_ALVENARIA + P_VINIL + P_FIBRA + P_PLASTICO)/4` ?
- Deve haver um meio melhor
  - Que mantenha o agrupamento
  - Mas que simplifique o código em *valorPiscina*

```
/* 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;


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

- Basta?
  - Ainda temos que relacioná-las
- E mais... como faríamos se quiséssemos calcular o preço médio dos materiais?
  - `(P_ALVENARIA + P_VINIL + P_FIBRA + P_PLASTICO)/4` ?
- Deve haver um meio melhor
  - Que mantenha o agrupamento
  - Mas que simplifique o código em *valorPiscina*
  - E ainda facilite o cálculo da média

```
/* 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;


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

- Basta?
  - Ainda temos que relacioná-las
- E mais... como faríamos se quiséssemos calcular o preço médio dos materiais?
  - `(P_ALVENARIA + P_VINIL + P_FIBRA + P_PLASTICO)/4` ?
- Deve haver um meio melhor
  - Que mantenha o agrupamento
  - Mas que simplifique o código em *valorPiscina*
  - E ainda facilite o cálculo da média
- Arranjos (Array)

```
/* 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;


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

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência de valores de um mesmo tipo.

Arranjos

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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;
```

Arranjos

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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
  - Deixou de ser constante

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

ou

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

Arranjos

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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
  - Deixou de ser constante
  - Mas deixou o código mais enxuto

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

ou

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

Arranjos

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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
  - Deixou de ser constante
  - Mas deixou o código mais enxuto

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

ou

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

- Isso diz ao compilador para reservar espaço na memória para 4 doubles

Arranjos

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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
  - Deixou de ser constante
  - Mas deixou o código mais enxuto

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

ou

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

- Isso diz ao compilador para reservar espaço na memória para 4 doubles
  - Armazenando os valores 1500, 1100, 750, 500 neles

Arranjos

Arranjos

Estruturas de dados, de tamanho fixo, que permitem armazenar uma sequência 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
  - Deixou de ser constante
  - Mas deixou o código mais enxuto

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

ou

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

- Isso 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 este 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

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

- 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
- Endereço?

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

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

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

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

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

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
  - Normalmente em hexadecimal
    - ★ Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Em binário, de 0000 a 1001

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

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
  - Normalmente em hexadecimal
    - ★ Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Em binário, de 0000 a 1001
    - ★ Binário: 0, 1.

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

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
  - Normalmente em hexadecimal
    - ★ Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Em binário, de 0000 a 1001
    - ★ Binário: 0, 1.
    - ★ Octal: 0, 1, 2, 3, 4, 5, 6, 7. Em binário, de 000 a 111.

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

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
  - Normalmente em hexadecimal
    - ★ Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Em binário, de 0000 a 1001
    - ★ Binário: 0, 1.
    - ★ Octal: 0, 1, 2, 3, 4, 5, 6, 7. Em binário, de 000 a 111.
    - ★ Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Em binário, de 0000 a 1111.

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

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
  - Normalmente em hexadecimal
    - ★ Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Em binário, de 0000 a 1001
    - ★ Binário: 0, 1.
    - ★ Octal: 0, 1, 2, 3, 4, 5, 6, 7. Em binário, de 000 a 111.
    - ★ Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Em binário, de 0000 a 1111.
  - Note que tanto Octal quanto Hexa usam todos os bits a eles alocados.

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

- Endereço?
  - Os bytes na memória são numerados de 0 ao máximo de memória que há – seu endereço
  - Normalmente em hexadecimal
    - ★ Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Em binário, de 0000 a 1001
    - ★ Binário: 0, 1.
    - ★ Octal: 0, 1, 2, 3, 4, 5, 6, 7. Em binário, de 000 a 111.
    - ★ Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Em binário, de 0000 a 1111.
  - Note que tanto Octal quanto Hexa usam todos os bits a eles alocados.
    - ★ Por isso usados. Não desperdiçam espaço

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços
  - São as variáveis

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços
  - São as variáveis
  - Os nomes das variáveis são, então, o mapeamento feito pelo compilador a esses endereços

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços
  - 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 que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços
  - 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
  - O programador não precisa saber qual é esse endereço

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços
  - 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
  - O programador não precisa saber qual é esse endereço
    - ★ Diz-se que a informação foi abstraída

Revendo a Memória

- O que acontece ao fazermos `float areaq;`?
  - Nesse caso, como *areaq* tem 4B, são alocados contíguos 4B na memória, sendo *0xff1* o endereço do primeiro deles

- O compilador, para sua facilidade, deixa você dar nomes a esses endereços
  - 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
  - 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

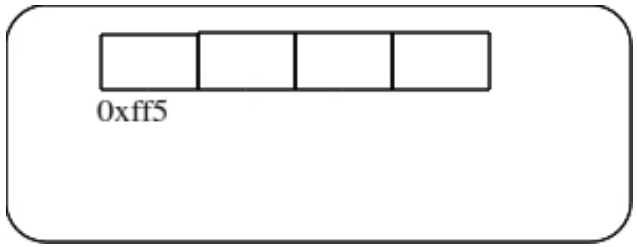
Arranjos na Memória

- O que acontece ao fazermos

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

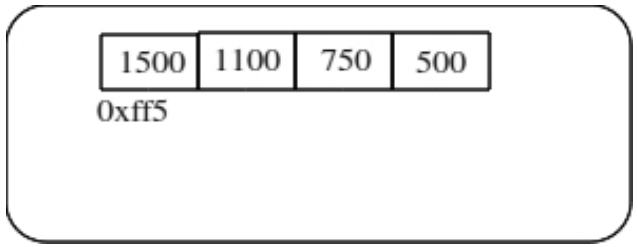
Arranjos na Memória

- O que acontece ao fazermos
  `double[] precos = {1500, 1100, 750, 500};`?
  - O compilador aloca espaço suficiente para 4 doubles consecutivos (32B)



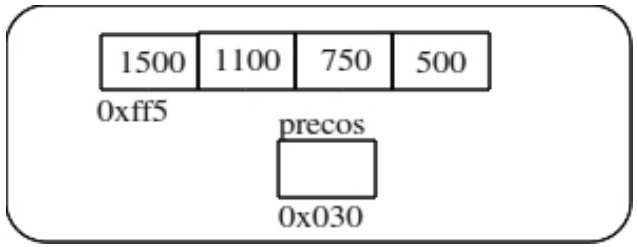
Arranjos na Memória

- 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



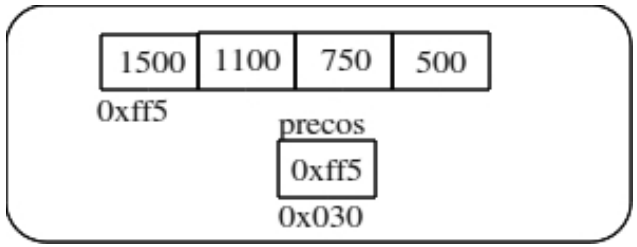
Arranjos na Memória

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



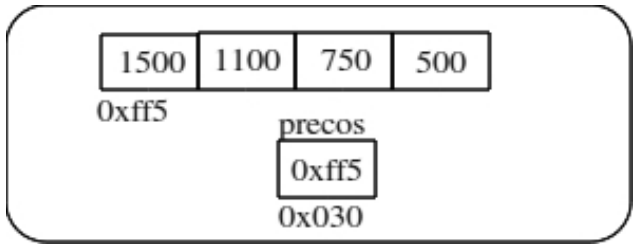
Arranjos na Memória

- 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
  - 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

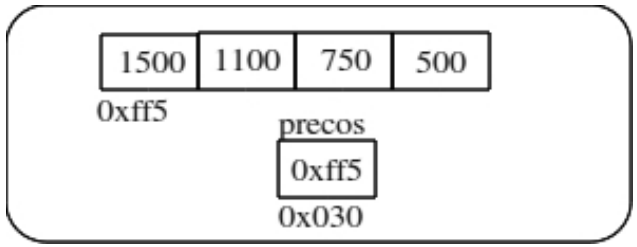
- O que acontece ao fazermos
  `double[] precos = {1500, 1100, 750, 500};`?
  - O compilador aloca espaço suficiente para 4 doubles consecutivos (32B)
  - Guarda os valores da inicialização lá



  - 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 → armazena uma referência ao início do arranjo

Arranjos na Memória

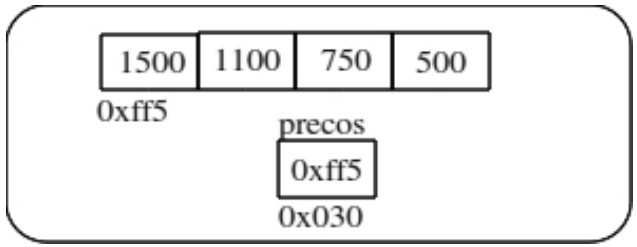
- O que acontece ao fazermos
  `double[] precos = {1500, 1100, 750, 500};`?
  - O compilador aloca espaço suficiente para 4 doubles consecutivos (32B)
  - Guarda os valores da inicialização lá
  - Em seguida aloca memória para a variável *precos*, grande o suficiente para caber um endereço, e cujo endereço o compilador também conhece
    - ★ Então guarda em *precos* o endereço na memória do primeiro byte do arranjo
    - ★ Guarda o endereço → armazena uma referência ao início do arranjo



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

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

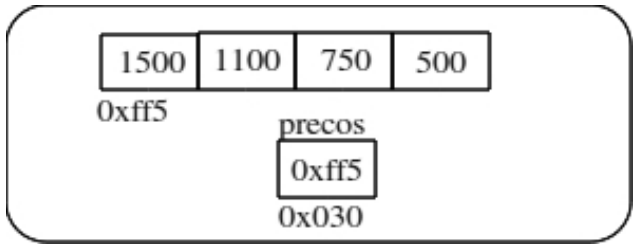


  - Em seguida aloca memória para a variável *precos*, grande o suficiente para caber um endereço, e cujo endereço o compilador também conhece
    - ★ Então guarda em *precos* o endereço na memória do primeiro byte do arranjo
    - ★ Guarda o endereço → armazena uma referência ao início do arranjo

- Para chegar ao primeiro elemento do arranjo, o computador:
  - Vai à região da memória correspondente a *preços*

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

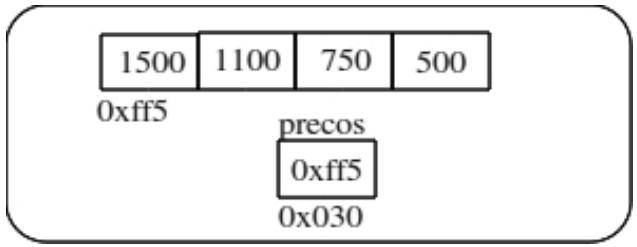


  - Em seguida aloca memória para a variável *precos*, grande o suficiente para caber um endereço, e cujo endereço o compilador também conhece
    - ★ Então guarda em *precos* o endereço na memória do primeiro byte do arranjo
    - ★ Guarda o endereço → armazena uma referência ao início do arranjo

- 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

- 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



  - 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 → armazena uma referência ao início do arranjo

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

Índices

- Como podemos ler um elemento do arranjo?

Índices

- Como podemos ler um elemento do arranjo?
  - `arranjo[indice]`

Índices

- Como podemos ler um elemento do arranjo?
  - `arranjo[indice]`
    - Onde índice é um inteiro de 0 a $n-1$, com n sendo o número de elementos do vetor

Índices

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

Índices

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

```
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]);
    }
}
```

Índices

- Como podemos ler um elemento do arranjo?
  - `arranjo[indice]`
    - ★ Onde índice é um inteiro de 0 a $n-1$, com n sendo o número de elementos do vetor
    - ★ 0 corresponde ao primeiro elemento, 1 ao segundo, etc
  - Uma vez que índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código:

```
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]);
    }
}
```

Índices

- Como podemos ler um elemento do arranjo?
  - `arranjo[indice]`
    - ★ Onde índice é um inteiro de 0 a $n-1$, com n sendo o número de elementos do vetor
    - ★ 0 corresponde ao primeiro elemento, 1 ao segundo, etc
  - Uma vez que índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código:

```
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]);
    }
}
```

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

Índices

- Como podemos ler um elemento do arranjo?
  - `arranjo[indice]`
    - ★ Onde índice é um inteiro de 0 a $n-1$, com n sendo o número de elementos do vetor
    - ★ 0 corresponde ao primeiro elemento, 1 ao segundo, etc
  - Uma vez que índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código:

```
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]);
    }
}
```

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

Repare no main... String[] args...

Índices

- Como podemos ler um elemento do arranjo?
  - `arranjo[indice]`
    - ★ Onde índice é um inteiro de 0 a $n-1$, com n sendo o número de elementos do vetor
    - ★ 0 corresponde ao primeiro elemento, 1 ao segundo, etc
  - Uma vez que índice pode ser qualquer inteiro, podemos melhorar a legibilidade do código:

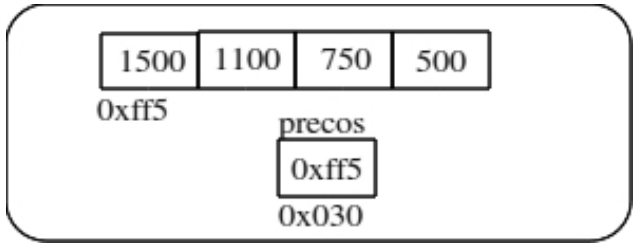
```
public static void main(String[] args) {
    System.out.println(precos[0]);
    System.out.println(precos[1]);
    System.out.println(precos[2]);
    System.out.println(precos[3]);
}

ou

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

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

Repare no main... String[] args... *args* também é um arranjo

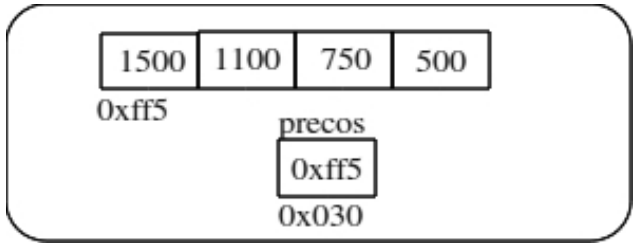
Índices na Memória

- Como o computador faz para achar o elemento na posição i do arranjo *precos*?



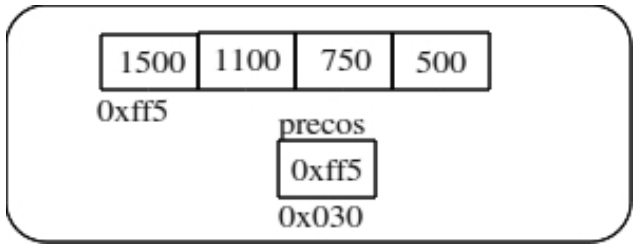
Índices na Memória

- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



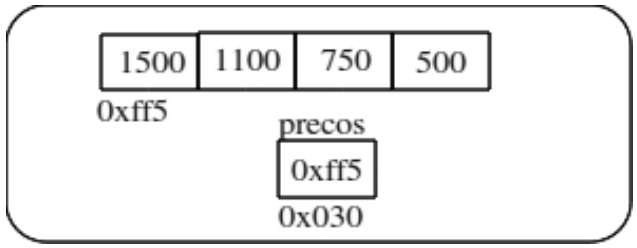
Índices na Memória

- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*
  - Lê seu conteúdo – endereço do primeiro byte do arranjo



Índices na Memória

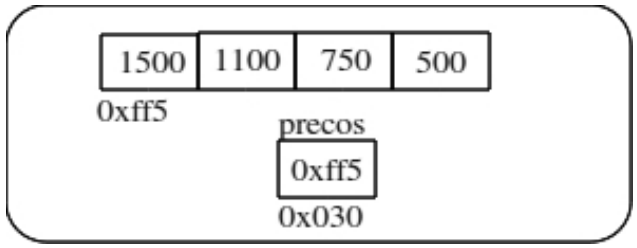
- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:

Índices na Memória

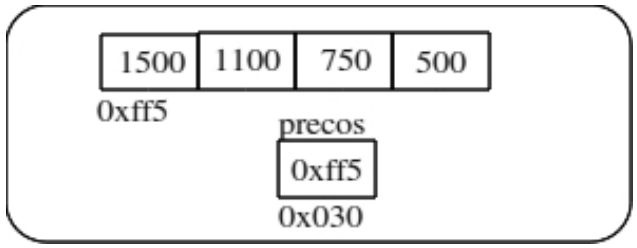
- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:
    - ★ Sabendo que cada elemento tem 8B (double)

Índices na Memória

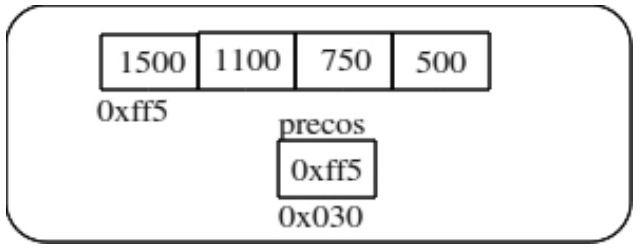
- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:
    - ★ Sabendo que cada elemento tem 8B (double)
    - ★ E que $0 \leq i \leq n - 1$

Índices na Memória

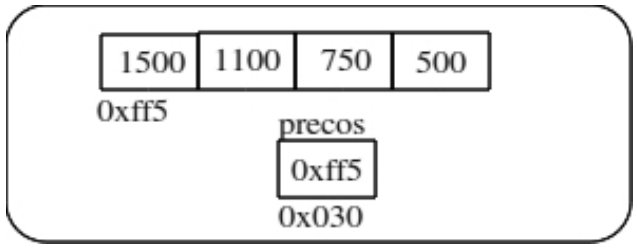
- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:
    - ★ Sabendo que cada elemento tem 8B (double)
    - ★ E que $0 \leq i \leq n - 1$
    - ★ O elemento estará a $8 \times i$ do início do arranjo

Índices na Memória

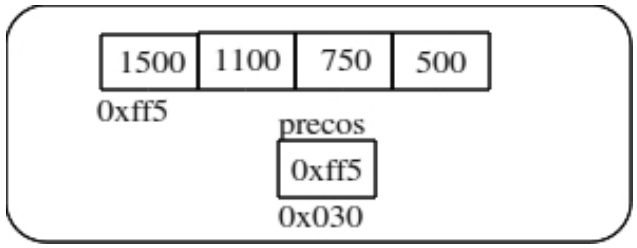
- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:
    - ★ Sabendo que cada elemento tem 8B (double)
    - ★ E que $0 \leq i \leq n-1$
    - ★ O elemento estará a $8 \times i$ do início do arranjo
    - ★ O endereço será $0xff5 + 8 \times i$

Índices na Memória

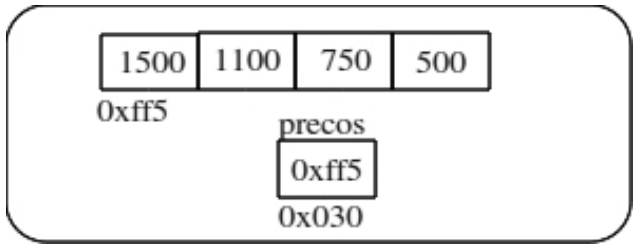
- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:
    - ★ Sabendo que cada elemento tem 8B (double)
    - ★ E que $0 \leq i \leq n-1$
    - ★ O elemento estará a $8 \times i$ do início do arranjo
    - ★ O endereço será $0xff5 + 8 \times i$
  - Vai à região da memória correspondente a esse endereço e lê seu conteúdo

Índices na Memória

- Como o computador faz para achar o elemento na posição i do arranjo *precos*?
  - Primeiro, vai à região da memória correspondente a *preços*



  - Lê seu conteúdo – endereço do primeiro byte do arranjo
  - Calcula a posição do elemento na posição i:
    - ★ Sabendo que cada elemento tem 8B (double)
    - ★ E que $0 \leq i \leq n-1$
    - ★ O elemento estará a $8 \times i$ do início do arranjo
    - ★ O endereço será $0xff5 + 8 \times i$
  - 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 array

Arranjos

- Como era a *valorPiscina*?

Arranjos

- Como era 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

- Como era 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

- Como era 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);
    }
}
```

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

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

Arranjos

- Como era a *valorPiscina*?
- Como ficaria usando o arranjo *precos*?

- Incluímos o teste para a área

```
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);
    }
}
```

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

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

Arranjos

- Como era 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);
    }
}
```

- 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

- Como era 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);
    }
}
```

- Incluímos o teste para a área
- Usamos o código do tipo do material como índice em *precos*
  - É importante certificar-se que *preco[0]* tem o preço de *ALVENARIA*, que *preco[1]* tem o preço de *VINIL* etc

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

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

Arranjos

- Arranjos podem ser atribuídos a outras variáveis, como uma variável comum:

```
double x = precos[0];
```

Arranjos

- 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

Arranjos

- 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

- 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

- Arranjos podem ser atribuídos a outras variáveis, como uma variável comum:
  - Desde que o tipo da variável seja o mesmo do tipo armazenado no arranjo
  - Do contrário, um type cast será necessário

```
double x = precos[0];
```

- E o que acontece se tentamos acessar um elemento fora dos limites do arranjo?

Código

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

Saída

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

Arranjos

- 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?

Código

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

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

Saída

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

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

Arranjos

- 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?

Código

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

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

Saída

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

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

- Compilará...

Arranjos

- 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?

Código

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

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

Saída

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

```
$ java AreaCasa
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: 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`

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

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
  - Seria a única maneira?

```
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
  - Seria a única maneira?

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

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

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
  - Seria a única maneira?
    - ★ Cria um arranjo de 4 elementos

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

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

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
  - Seria a única maneira?
    - ★ Cria um arranjo de 4 elementos
    - ★ Todos com valor zero

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

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

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
  - Seria a única maneira?
    - ★ Cria um arranjo de 4 elementos
    - ★ Todos com valor zero
    - ★ Há então que inicializá-lo:

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

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

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
  - Seria a única maneira?
    - ★ Cria um arranjo de 4 elementos
    - ★ Todos com valor zero
    - ★ Há então que inicializá-lo:

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

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

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

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

Inicialização de Arranjos

- Qual seria o melhor meio de inicializar um arranjo?

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

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

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

Inicialização de Arranjos

- Qual seria o melhor meio de inicializar um arranjo?

```
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 conhecermos os valores de antemão

Inicialização de Arranjos

- Qual seria o melhor meio de inicializar um arranjo?

```
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 conhecermos os valores de antemão
- E se esses valores forem poucos

Inicialização de Arranjos

- Qual seria o melhor meio de inicializar um arranjo?

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

- Útil se conhecermos os valores de antemão
- E se esses valores forem poucos

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

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

- Útil se não conhecermos os valores antes

Inicialização de Arranjos

- Qual seria o melhor meio de inicializar um arranjo?

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

- Útil se conhecermos os valores de antemão
- E se esses valores forem poucos

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

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

- Útil se não conhecermos os valores antes
- Ou se esses valores forem muitos

Inicialização de Arranjos

- Qual seria o melhor meio de inicializar um arranjo?

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

- Útil se conhecermos os valores de antemão
- E se esses valores forem poucos

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

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

- Útil se não conhecermos os valores antes
- Ou se esses valores forem muitos
- Ex:

```
int[] v = new int[1000];

for (int i=0; i<1000; i++) {
 v[i] = -1;
}
```

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `tipo[] nomeDaVariavel = new tipo[tamanho do arranjo]`

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `tipo[] nomeDaVariavel = new tipo[tamanho do arranjo]`
  - Em que *tamanho do arranjo* é o número de elementos que ele conterá

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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 $tamanho - 1$
- Ex:
  - float[] precos2 = new float[4];

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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

- Ex:
  - float[] precos2 = new float[4];
  - double[] precos2 = new double[4];

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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

- Ex:
  - float[] precos2 = new float[4];
  - double[] precos2 = new double[4];
  - int[] tamanhos = new int[10];

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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

- Ex:
  - float[] precos2 = new float[4];
  - double[] precos2 = new double[4];
  - int[] tamanhos = new int[10];
  - long[] tamanhos = new long[10];

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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

- 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];

Arranjos

- Arranjos podem ser criados de qualquer tipo no java:
  - Fazendo: `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

- 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 então calcular o preço médio dos materiais de nossa piscina

Arranjos

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

Arranjos

- Vamos então calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços

Arranjos

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

Arranjos

- Vamos então 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

- Vamos então calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços
  - E dividindo pelo número deles
- Funciona... mas e se tivermos que aumentar o tamanho do arranjo (por conta de um novo material)?

```
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

- Vamos então calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços
  - E dividindo pelo número deles
- Funciona... mas e se tivermos que aumentar o tamanho do arranjo (por conta de um novo material)?
  - 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

- Vamos então calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços
  - E dividindo pelo número deles
- Funciona... mas e se tivermos que aumentar o tamanho do arranjo (por conta de um novo material)?
  - 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

- Vamos então calcular o preço médio dos materiais de nossa piscina
- Como fazer?
  - Somando todos os preços
  - E dividindo pelo número deles
- Funciona... mas e se tivermos que aumentar o tamanho do arranjo (por conta de um novo material)?
  - 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);
}
```

```
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);
}
```

Length

- Em java, arranjos vêm com o atributo pré-definido length, contendo seu comprimento

```
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);
}
```

Length

- 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);
}
```

Length

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

```
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);
}
```

Length

- Em java, arranjos vêm com o atributo pré-definido length, contendo seu comprimento
  - Seu valor é definido automaticamente pelo compilador
  - Não pode ser alterado
  - Se fizermos:

```
precos.length = 10;
```

```
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);
}
```

Length

- Em java, arranjos vêm com o atributo pré-definido length, contendo seu comprimento
  - Seu valor é definido automaticamente pelo compilador
  - Não pode ser alterado
  - Se fizermos:

    ```
    precos.length = 10;
    ```

  - Teremos a mensagem:

```
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);
}
```

Length

- Em java, arranjos vêm com o atributo pré-definido length, contendo seu comprimento
  - Seu valor é definido automaticamente pelo compilador
  - Não pode ser alterado
  - Se fizermos:

    `precos.length = 10;`
  - Teremos a mensagem:

```
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);
}
```

```
$ 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

For com Arranjos

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

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

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

- Existe, contudo, um quarto tipo em java

```
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

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

- Existe, contudo, um quarto tipo em java
  - Na verdade, um segundo tipo de for

```
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

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

- Existe, contudo, um quarto tipo em java
  - Na verdade, um segundo tipo de for
  - Projetado para iterar, dentre ontras coisas, em 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);
}
```

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);
}
```

- Existe, contudo, um quarto tipo em java
  - Na verdade, um segundo tipo de for
  - Projetado para iterar, dentre ontras coisas, em arranjos
  - Além de tornar os laços mais compactos e fáceis de ler:

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);
}
```

- Existe, contudo, um quarto tipo em java
  - Na verdade, um segundo tipo de for
  - Projetado para iterar, dentre ontras coisas, em arranjos
  - Além de tornar os laços mais compactos e fáceis de ler:

```
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);
}
```

```
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);
}
```

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

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

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

- Itera no índice 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);
}
```

- 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);
}
```

- Itera no índice do arranjo
- 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);
}
```

- 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);
}
```

- Itera no índice do arranjo
- Usa o iterador (i) como índice para acessar o valor
  - 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);
}
```

- 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);
}
```

- Itera no índice do arranjo
- Usa o iterador (i) como índice para acessar o valor
  - 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);
}
```

- Itera em cada valor do arranjo
- 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);
}
```

- Itera no índice do arranjo
- Usa o iterador (i) como índice para acessar o valor
  - 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);
}
```

- Itera em cada valor do arranjo
- O iterador (valor) contém o próprio valor
  - Acessa o valor de maneira direta