Aula 14 – Classes

Norton Trevisan Roman

9 de maio de 2013

Construtores

- Repare no construtor padrão de *AreaPiscina*:

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this.precos = aux;

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this.precos = aux;

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização
  - Do contrário, teríamos que abastecer elemento por elemento

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this.precos = aux;

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização
  - Do contrário, teríamos que abastecer elemento por elemento
  - Atribuímos diretamente o novo arranjo a *precos*

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this.precos = aux;

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização
  - Do contrário, teríamos que abastecer elemento por elemento
  - Atribuímos diretamente o novo arranjo a *precos*
- Não poderíamos ter feito algo assim?

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this(aux);

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização
  - Do contrário, teríamos que abastecer elemento por elemento
  - Atribuímos diretamente o novo arranjo a *precos*
- Não poderíamos ter feito algo assim?

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this(aux);

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

```
call to this must be first statement in constructor
        this(aux);
            ^
1 error
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização
  - Do contrário, teríamos que abastecer elemento por elemento
  - Atribuímos diretamente o novo arranjo a *precos*
- Não poderíamos ter feito algo assim?
  - *this*, se usado para referenciar um construtor, deve sempre ser o primeiro comando no construtor

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this(aux);

    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

```
call to this must be first statement in constructor
        this(aux);
            ^
1 error
```

Construtores

- Repare no construtor padrão de *AreaPiscina*:
  - Tivemos que criar um auxiliar, para usar o atalho da inicialização
  - Do contrário, teríamos que abastecer elemento por elemento
  - Atribuímos diretamente o novo arranjo a *precos*
- Não poderíamos ter feito algo assim?
  - *this*, se usado para referenciar um construtor, deve sempre ser o primeiro comando no construtor
  - Solução:

```
class AreaPiscina {
    ...

    double[] precos;

    ...

    AreaPiscina() {

        this(new double[] {1500, 1100, 750, 500});
    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
    }

    ...
}
```

```
call to this must be first statement in constructor
        this(aux);
            ^
1 error
```

Construtores

- Podemos simplificar o uso dos métodos de *AreaPiscina*

```
class AreaPiscina {
    /* 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 */
    double[] precos;
    /* nomes dos materiais */
    static char[][] nomes = {{'A','l','v','e','n','a',
                              'r','i','a'},
                             {'V','i','n','i','l'},
                             {'F','i','b','r','a'},
                             {'P','l','á','s','t','i',
                              'c','o'}};
    /* raio da piscina*/
    double raio;

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this.precos = aux;
        this.raio = 10;
    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
        this.raio = 10;
    }

    AreaPiscina(double raio) {
        this();
        this.raio = raio;
    }

    AreaPiscina(double[] precos, double raio) {
        this.precos = precos;
        this.raio = raio;
    }

    double area() {
        return((this.raio >= 0) ? Math.PI *
                        Math.pow(this.raio,2) : -1);
    }

    ...
}
```

Construtores

- Podemos simplificar o uso dos métodos de *AreaPiscina*
  - tornando suas dimensões parte do objeto

```
class AreaPiscina {
    /* 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 */
    double[] precos;
    /* nomes dos materiais */
    static char[][] nomes = {{'A','l','v','e','n','a',
                              'r','i','a'},
                             {'V','i','n','i','l'},
                             {'F','i','b','r','a'},
                             {'P','l','á','s','t','i',
                              'c','o'}};
    /* raio da piscina*/
    double raio;

    AreaPiscina() {
        double[] aux = {1500, 1100, 750, 500};
        this.precos = aux;
        this.raio = 10;
    }

    AreaPiscina(double[] precos) {
        this.precos = precos;
        this.raio = 10;
    }

    AreaPiscina(double raio) {
        this();
        this.raio = raio;
    }

    AreaPiscina(double[] precos, double raio) {
        this.precos = precos;
        this.raio = raio;
    }

    double area() {
        return((this.raio >= 0) ? Math.PI *
                        Math.pow(this.raio,2) : -1);
    }

    ...
}
```

Construtores

- Façamos o mesmo com AreaCasa:

Construtores

- Façamos o mesmo com AreaCasa:
  - Podemos inicializar os atributos com os valores padrão, em vez de fazermos isso no objeto

Construtores

- Façamos o mesmo com AreaCasa:
  - Podemos inicializar os atributos com os valores padrão, em vez de fazermos isso no objeto

```
class AreaCasa {
    /* valor do metro quadrado da casa */
    double valorM2 = 1500;

    /* comprimento da lateral da sala */
    double lateral = 10;

    /* comprimento da lateral maior do quarto */
    double cquarto = 10;

    AreaCasa() {}

    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }

    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    AreaCasa(double lateral, double cquarto,
                                    double valorM2) {
        this(lateral, cquarto);
        this.valorM2 = valorM2;
    }

    double area() {
        double areat=-1; // área total

        if (this.lateral>=0 && this.cquarto>=0) {
            areat = this.lateral*this.lateral;
            areat += this.cquarto*this.lateral;
        }
        return(areat);
    }

    double valor(double area) {
        if (area >= 0) return(this.valorM2*
                                        area);
        return(-1);
    }
}
```

Classes em Java

- Vimos objetos diferentes para a mesma classe

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes

```
class Projeto {



    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);


    }
}
```

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                               AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                       AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

- Mas será que devemos sempre passar tudo para objeto?

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                               AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

- Mas será que devemos sempre passar tudo para objeto?
  - "Tirar os *static*" do código?

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                       AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

- Mas será que devemos sempre passar tudo para objeto?
  - "Tirar os *static*" do código?
- Não, apenas os métodos e atributos específicos do objeto

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                       AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

- Mas será que devemos sempre passar tudo para objeto?
  - "Tirar os *static*" do código?
- Não, apenas os métodos e atributos específicos do objeto
  - Ou seja, da instância particular, e não da classe

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                       AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

- Mas será que devemos sempre passar tudo para objeto?
  - "Tirar os *static*" do código?
- Não, apenas os métodos e atributos específicos do objeto
  - Ou seja, da instância particular, e não da classe
  - Do ator, não do papel

Classes em Java

- Vimos objetos diferentes para a mesma classe
  - Podemos também ter objetos para classes diferentes
  - Usados como bem entendermos

```
class Projeto {

    static double area(AreaCasa casa,
                       AreaPiscina piscina) {
        return(casa.area() + piscina.area());
    }

    public static void main(String[] args) {
        AreaCasa casaPrinc = new AreaCasa(10,5);
        AreaPiscina piscina1 = new AreaPiscina(10);

        System.out.println(area(casaPrinc, piscina1));
    }
}
```

- Mas será que devemos sempre passar tudo para objeto?
  - "Tirar os *static*" do código?
- Não, apenas os métodos e atributos específicos do objeto
  - Ou seja, da instância particular, e não da classe
  - Do ator, não do papel
- Os demais (gerais), devem continuar como estão... com *static*

Classe × Objeto

- Métodos e Atributos da Classe
- Métodos e Atributos do Objeto

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
- Métodos e Atributos do Objeto

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
  - Acessados diretamente com *Nome_da_classe.método* ou *Nome_da_classe.atributo*
- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
  - Acessados diretamente com *Nome_da_classe.método* ou *Nome_da_classe.atributo*
- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*
  - Acessados somente via objeto, com *objeto.método* ou *objeto.atributo*

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
  - Acessados diretamente com *Nome_da_classe.método* ou *Nome_da_classe.atributo*
  - Cópia única na memória
- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*
  - Acessados somente via objeto, com *objeto.método* ou *objeto.atributo*

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
  - Acessados diretamente com *Nome_da_classe.método* ou *Nome_da_classe.atributo*
  - Cópia única na memória
- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*
  - Acessados somente via objeto, com *objeto.método* ou *objeto.atributo*
  - Uma cópia por objeto

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
  - Acessados diretamente com *Nome_da_classe.método* ou *Nome_da_classe.atributo*
  - Cópia única na memória
- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*
  - Acessados somente via objeto, com *objeto.método* ou *objeto.atributo*
  - Uma cópia por objeto

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) ...

    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }

    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

Classe × Objeto

- Métodos e Atributos da Classe
  - Utilizam a palavra reservada *static*
  - Acessados diretamente com *Nome_da_classe.método* ou *Nome_da_classe.atributo*
  - Cópia única na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) ...

    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }

    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

- Métodos e Atributos do Objeto
  - Não utilizam a palavra reservada *static*
  - Acessados somente via objeto, com *objeto.método* ou *objeto.atributo*
  - Uma cópia por objeto

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) ...

    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }

    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

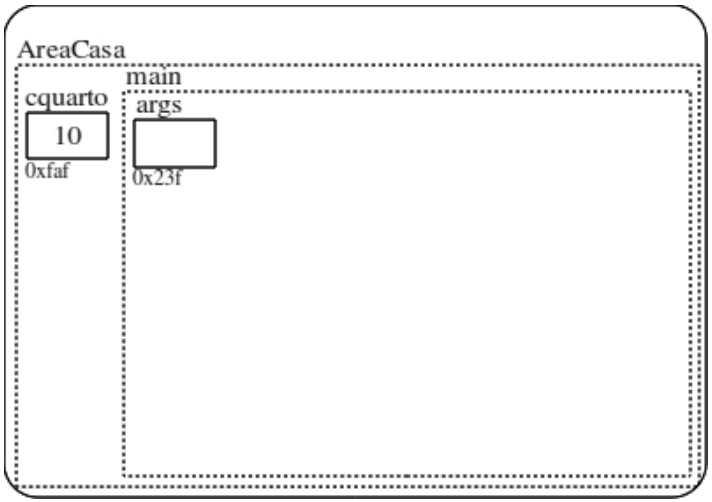
Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

Classe × Objeto: Funcionamento na memória

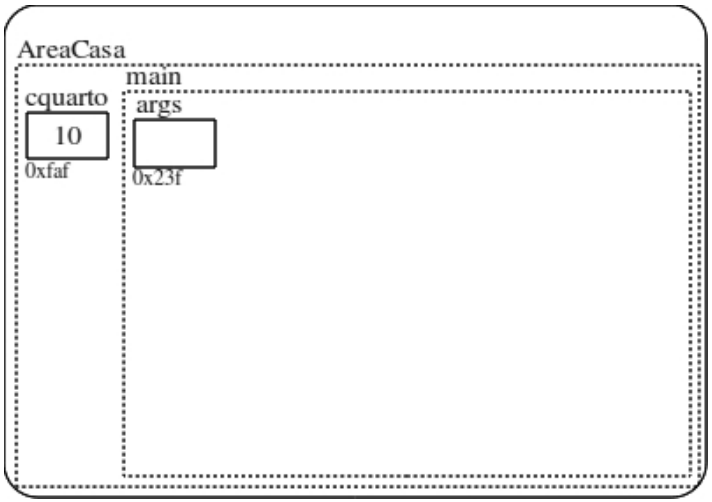
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```



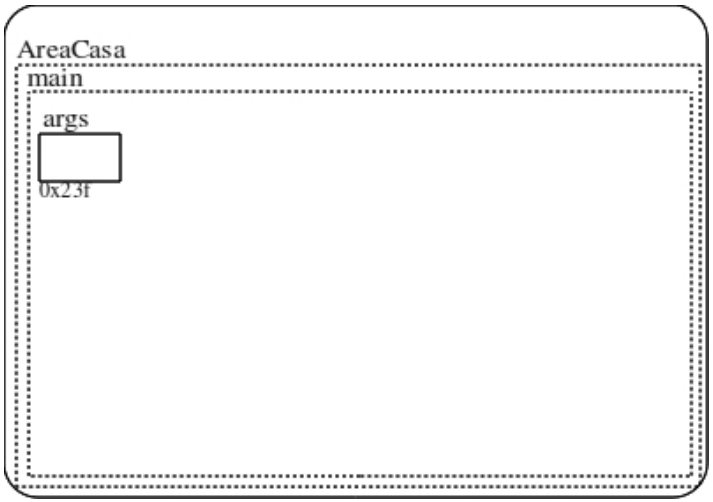

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

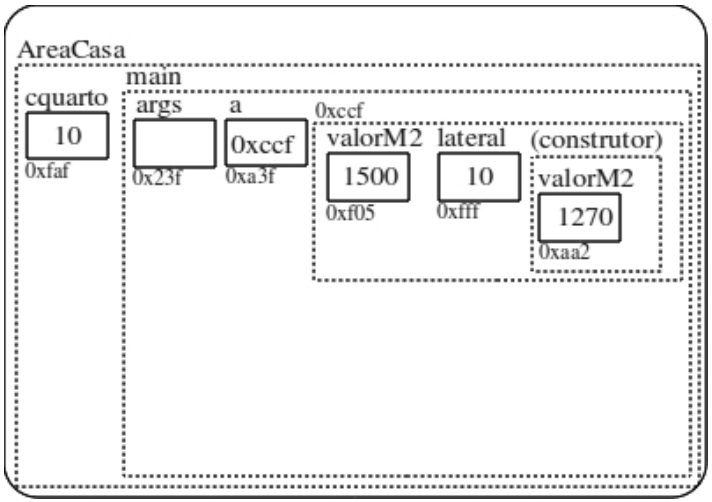



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

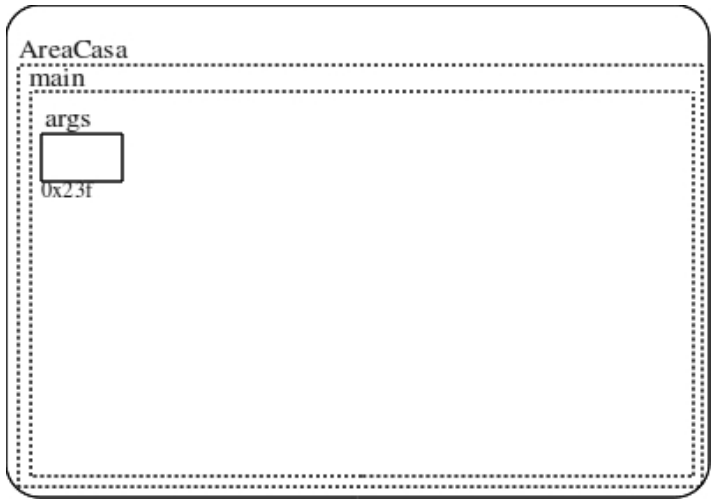


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

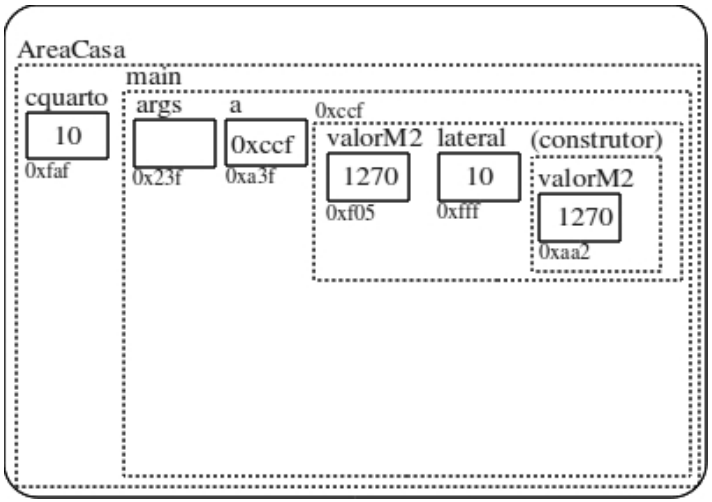



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

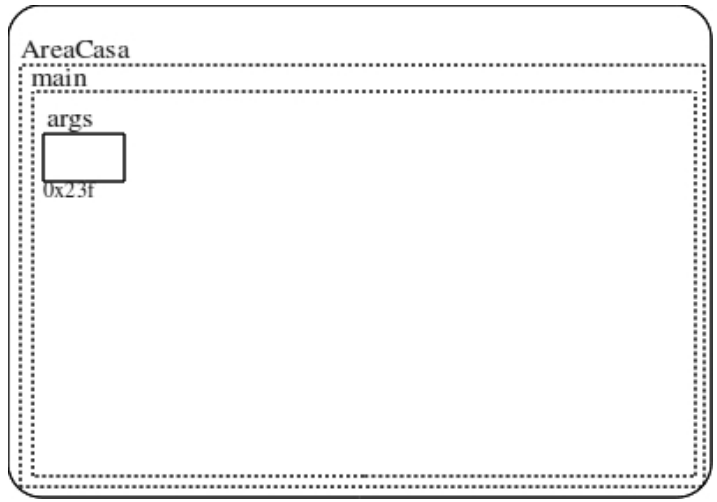


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

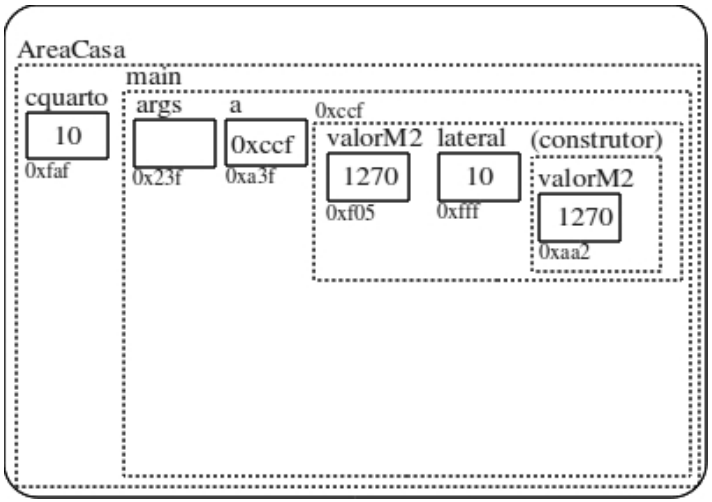



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

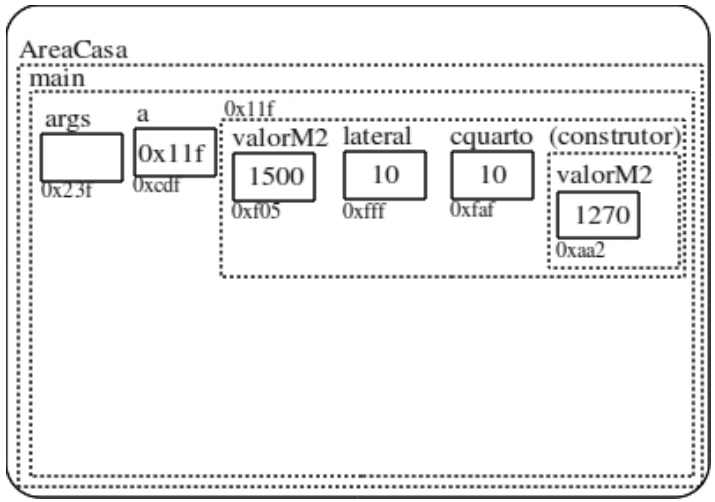


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

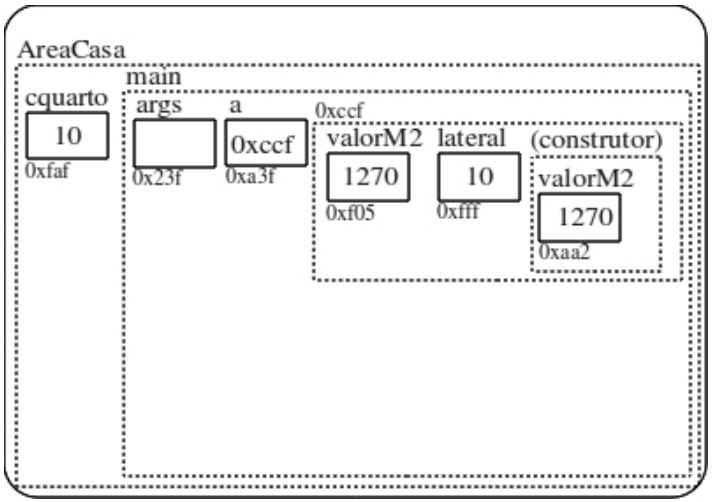



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

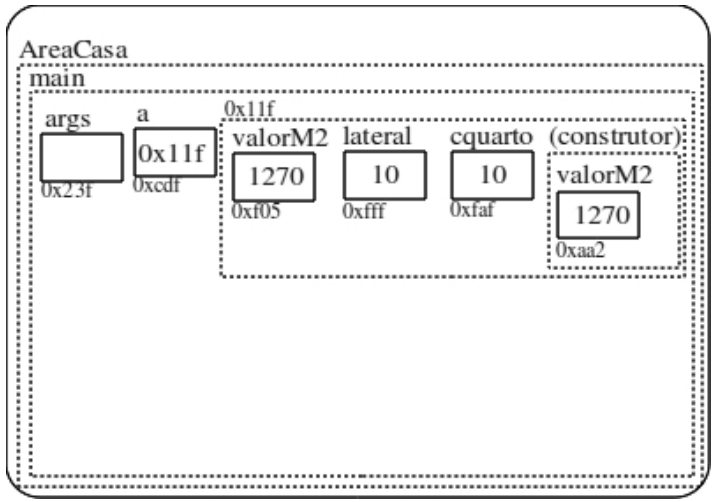


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

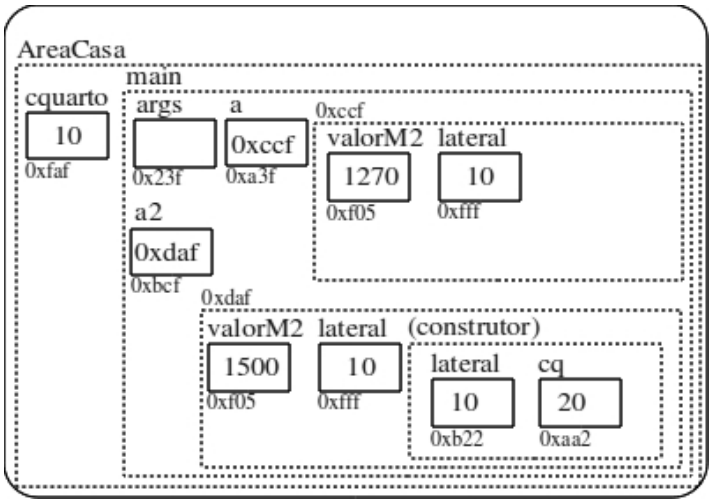



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

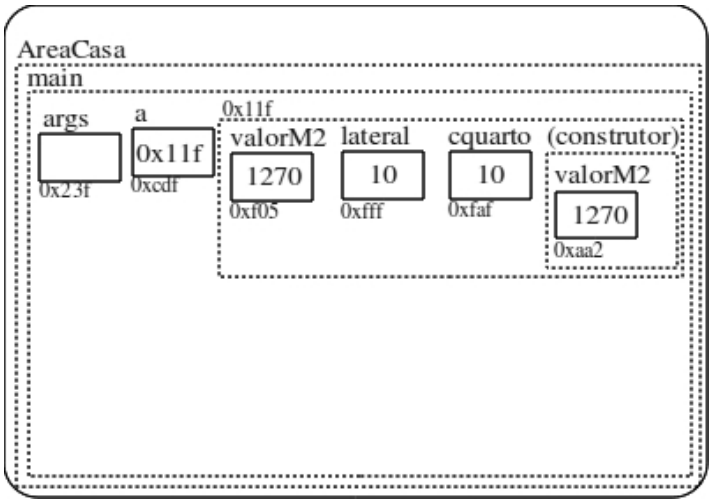


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

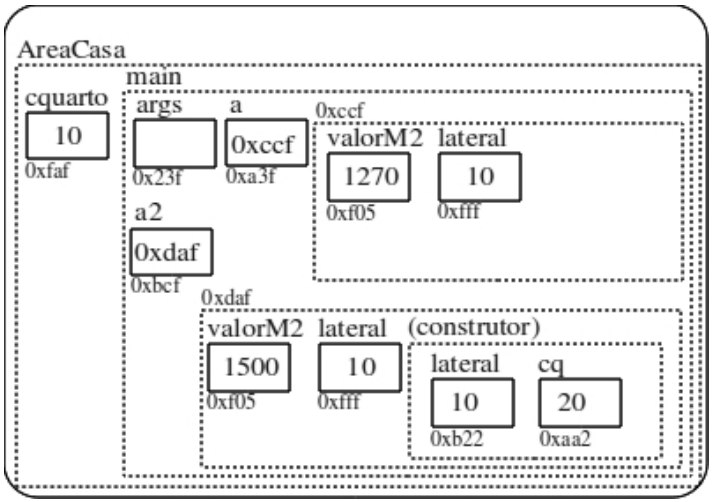



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

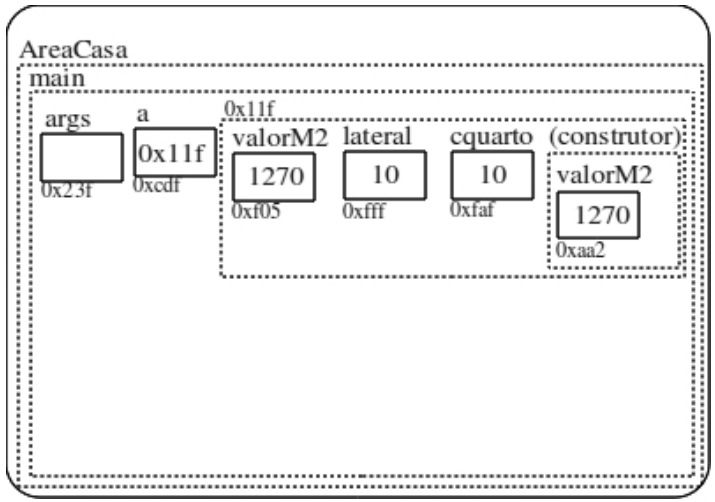


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```



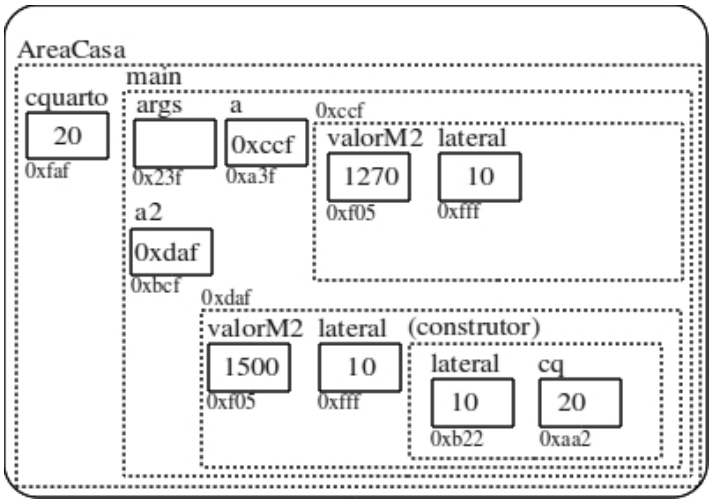

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

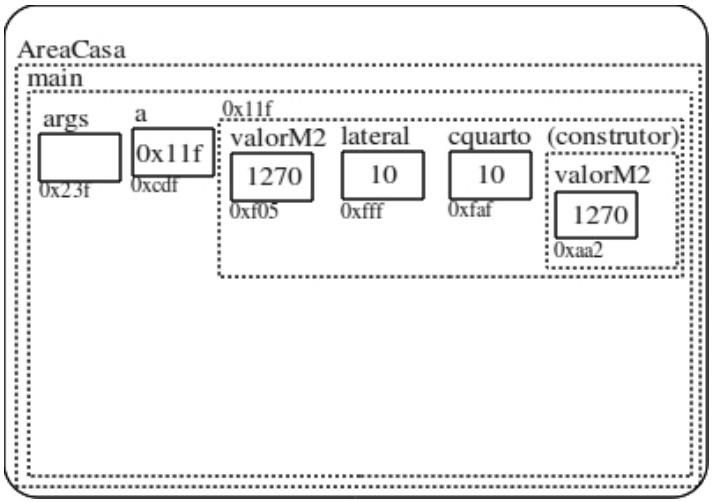


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

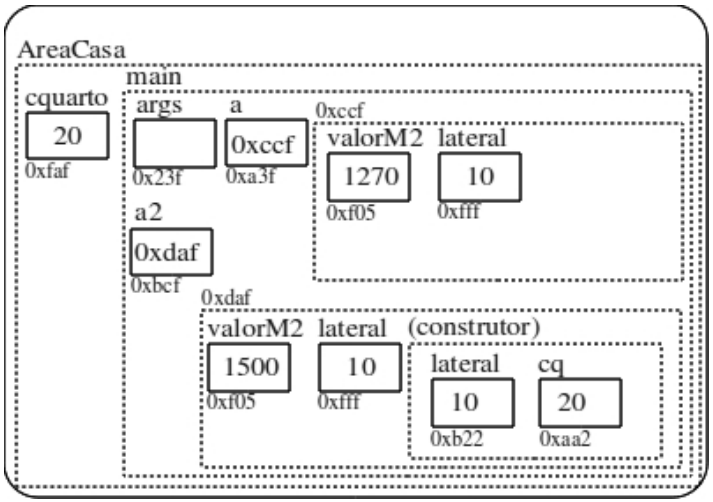


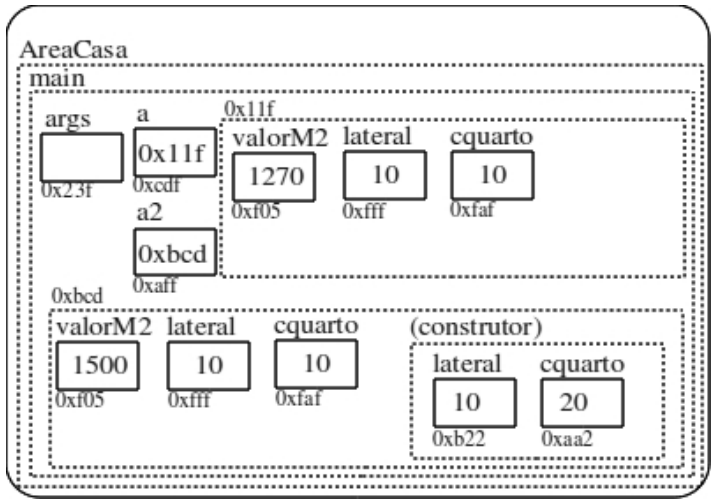


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

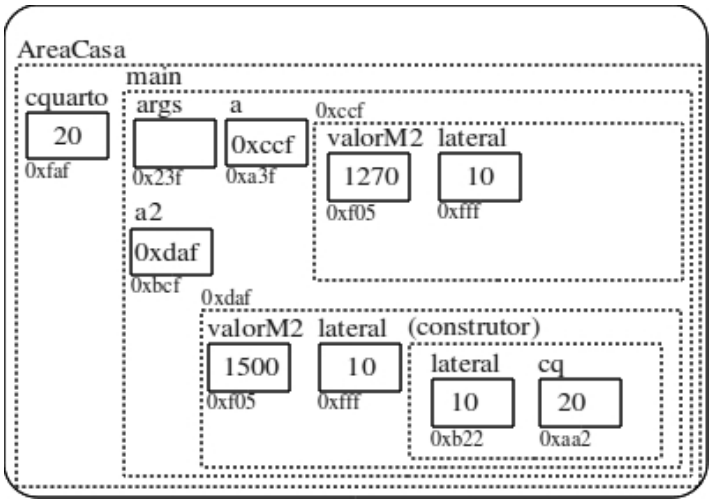
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```



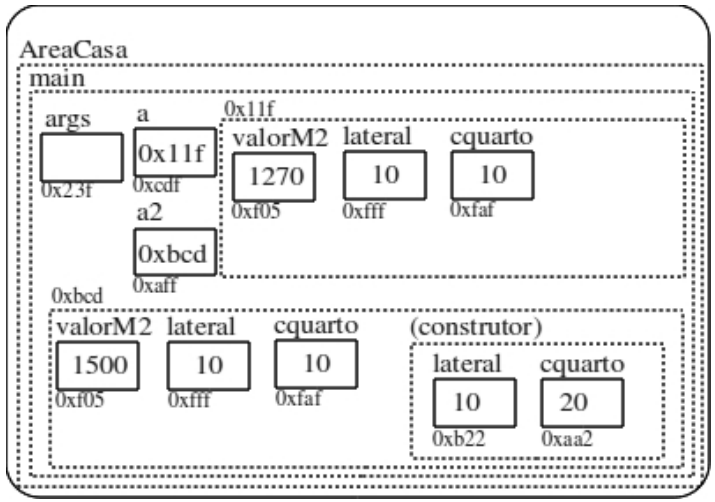


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

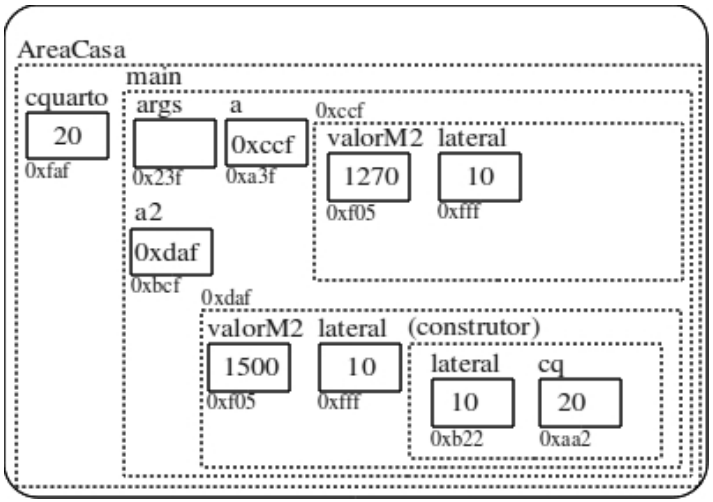



```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

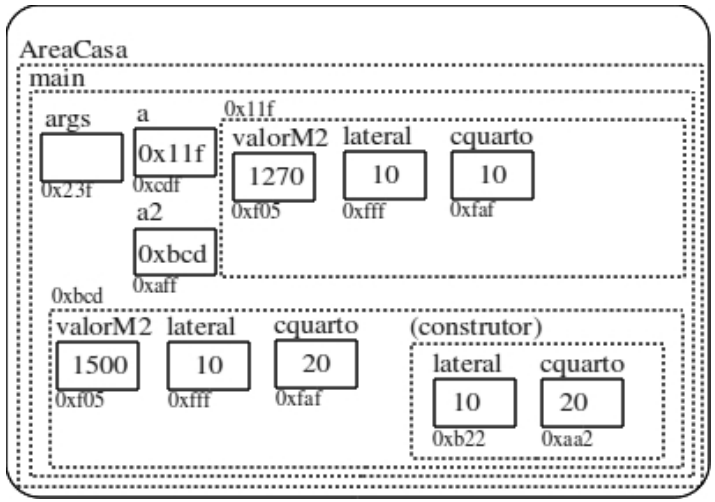


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```



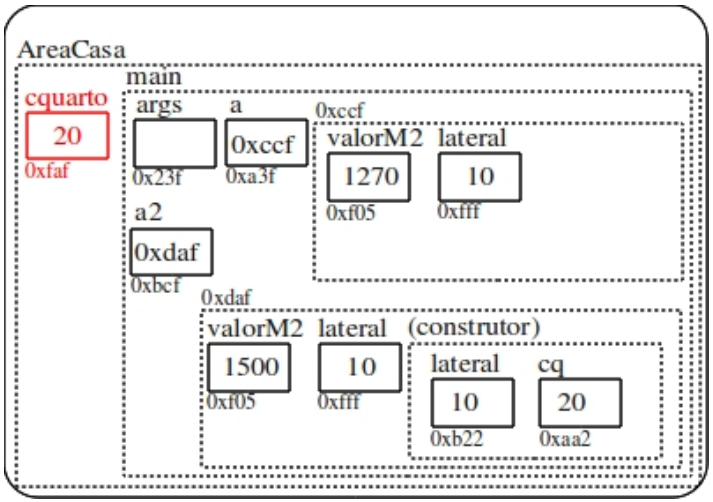

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

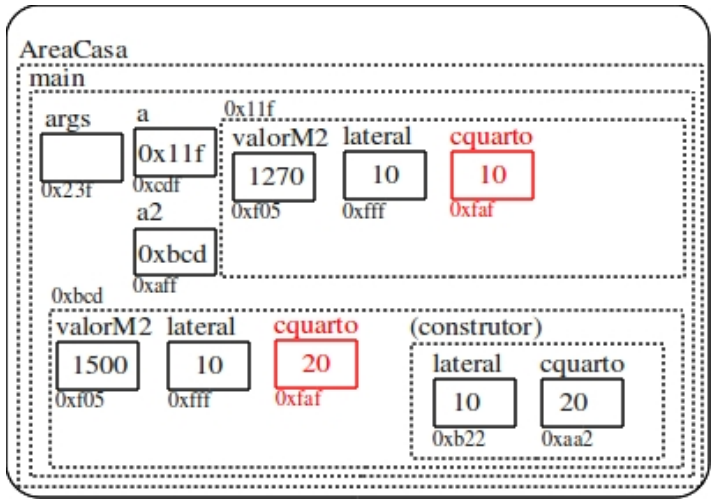


Classe × Objeto: Funcionamento na memória

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    static double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cq) {
        this.lateral = lateral;
        cquarto = cq;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
    }
}
```





Garbage Collection

- Considere o código

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);

    }
}
```

Garbage Collection

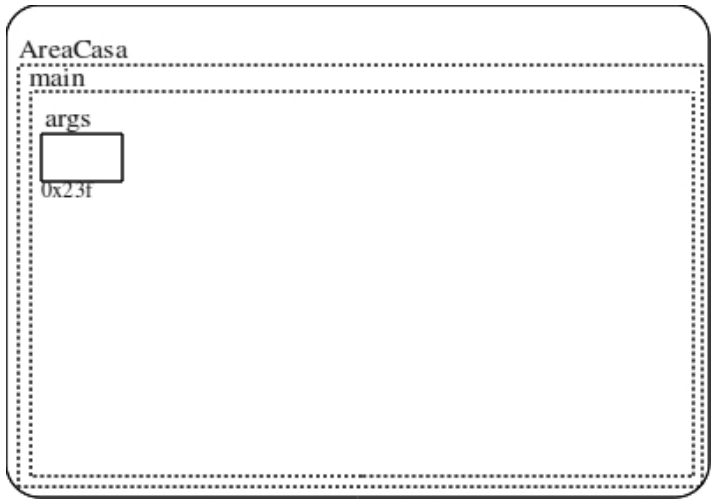
- Considere o código
- O que aconteceria se fizéssemos

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```

Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

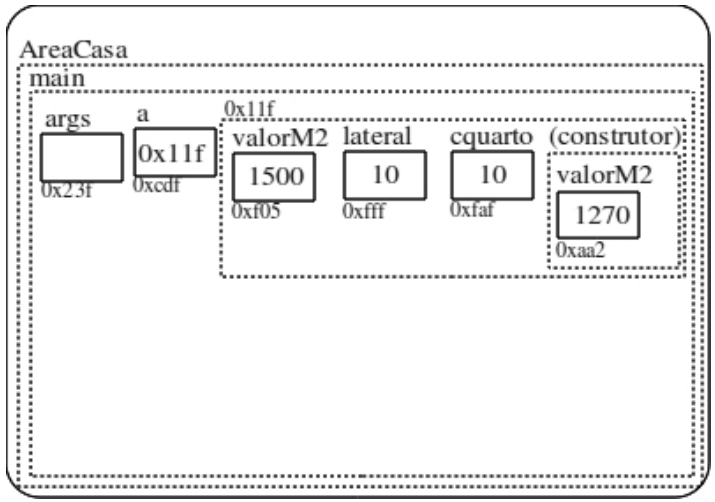
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

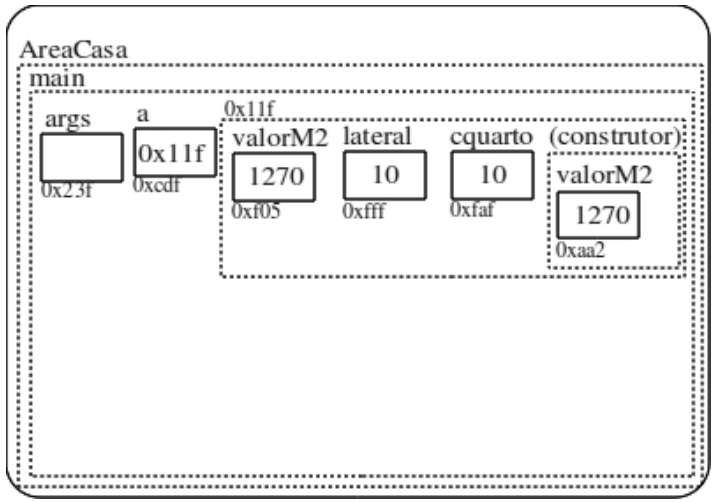
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

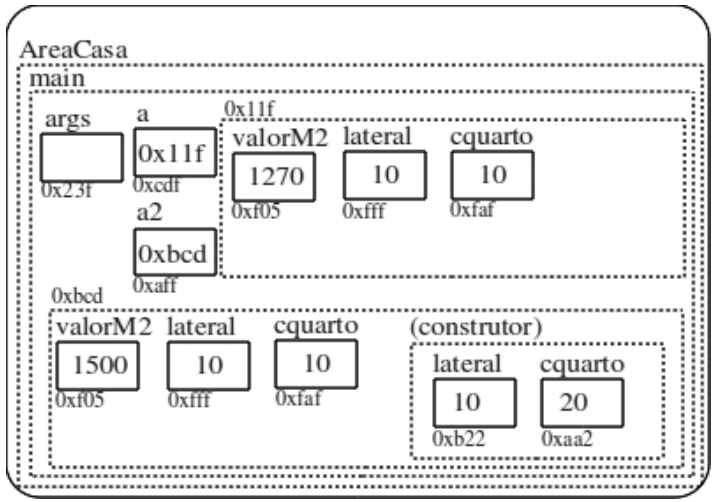
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

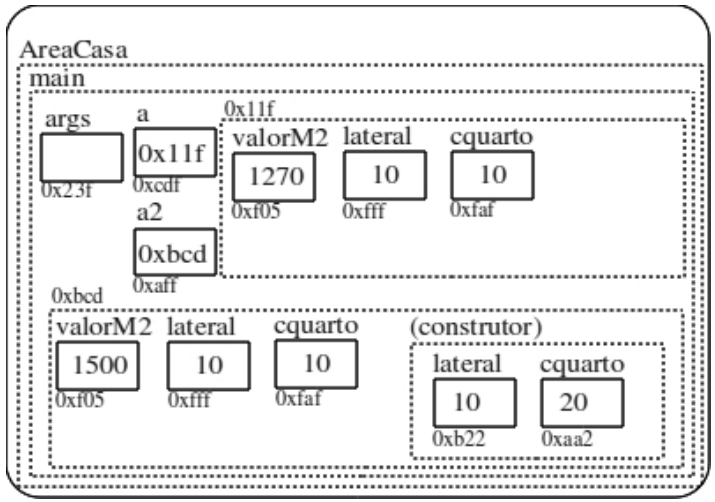
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

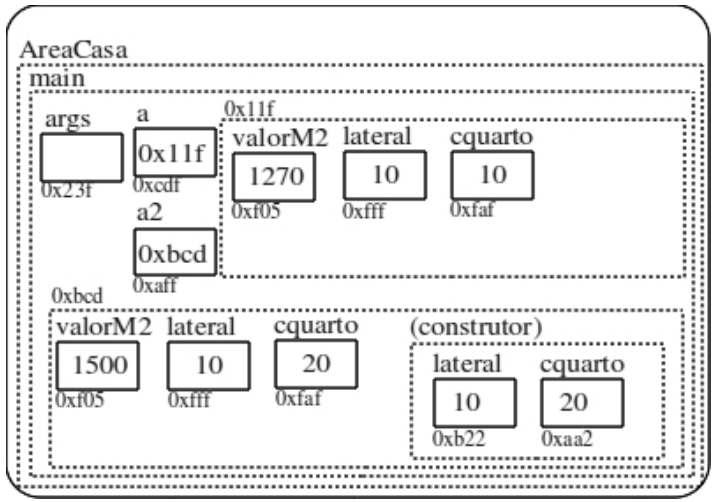
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

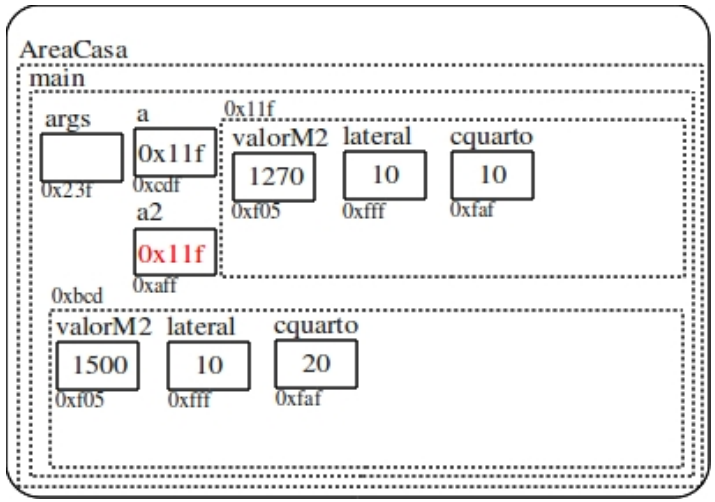
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos

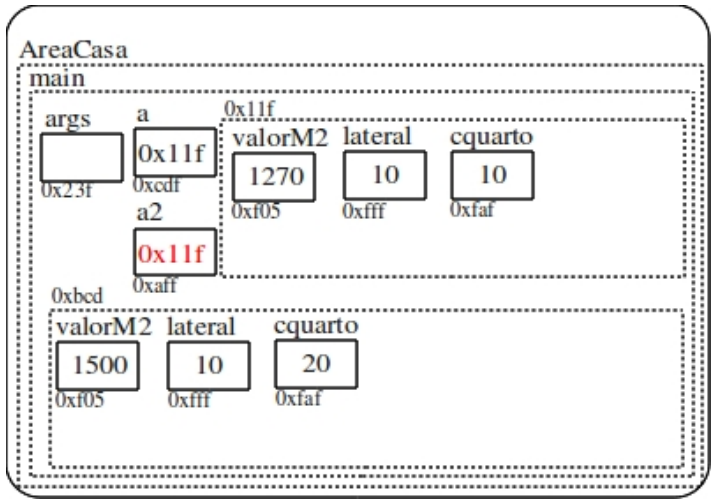
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos
  - Perderíamos o endereço do segundo objeto

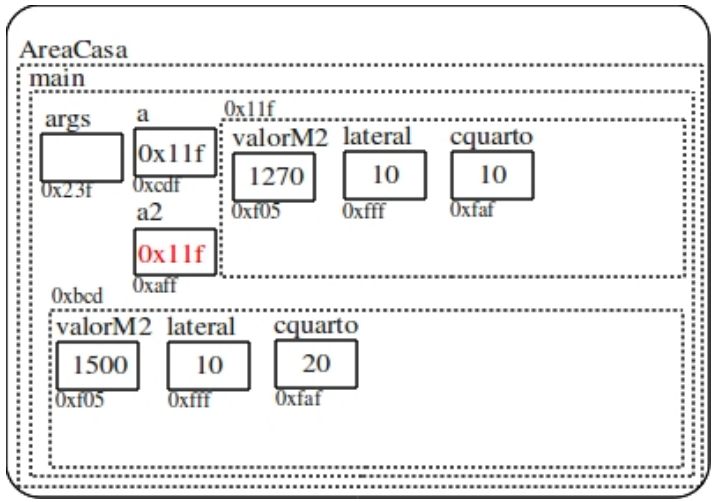
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos
  - Perderíamos o endereço do segundo objeto
  - Mas ele continua na memória...

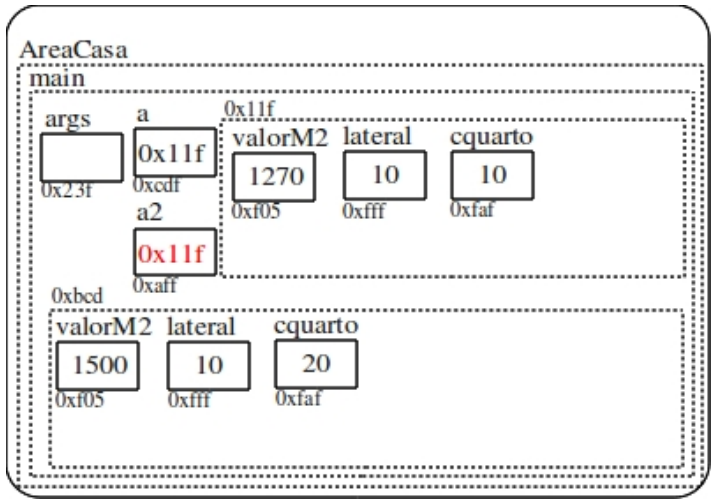
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos
  - Perderíamos o endereço do segundo objeto
  - Mas ele continua na memória...
- Ficará assim eternamente?

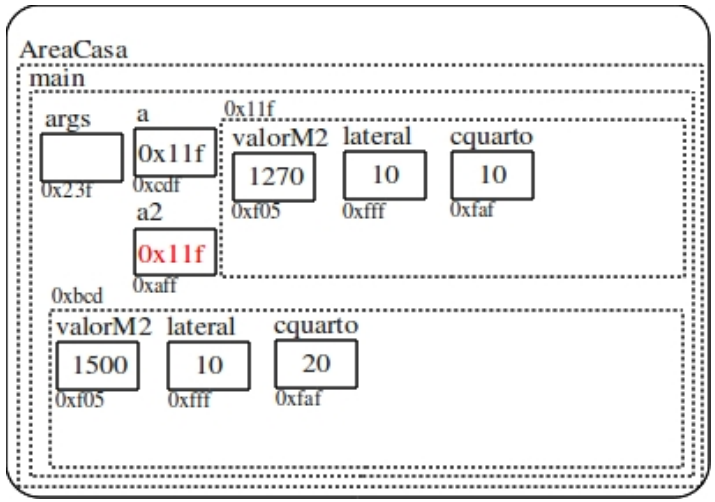
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos
  - Perderíamos o endereço do segundo objeto
  - Mas ele continua na memória...
- Ficará assim eternamente?
  - Não. Será levado pelo lixeiro

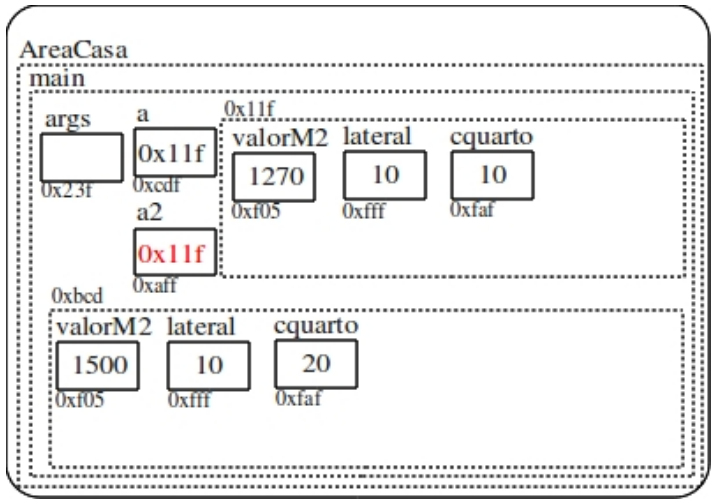
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos
  - Perderíamos o endereço do segundo objeto
  - Mas ele continua na memória...
- Ficará assim eternamente?
  - Não. Será levado pelo lixeiro
- Garbage Collector

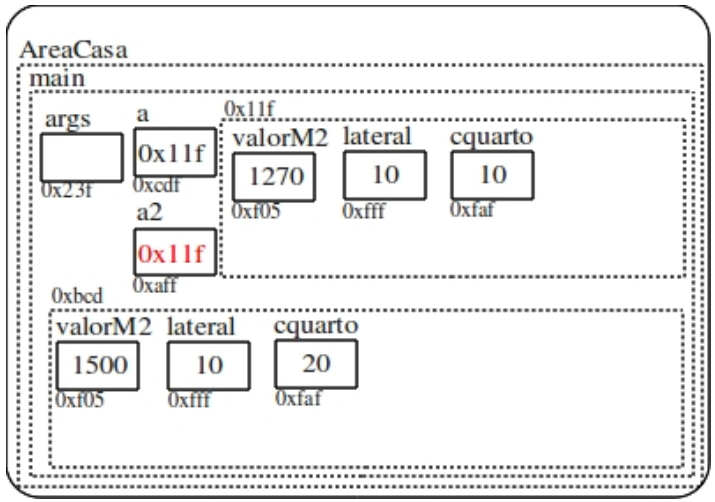
```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- Considere o código
- O que aconteceria se fizéssemos
  - Perderíamos o endereço do segundo objeto
  - Mas ele continua na memória...
- Ficará assim eternamente?
  - Não. Será levado pelo lixeiro
- Garbage Collector
  - Serviço provido pela JVM, com o objetivo de limpar a memória de referências perdidas

```
class AreaCasa {
    double valorM2 = 1500; double lateral = 10;
    double cquarto = 10;
    ...
    AreaCasa(double valorM2) {
        this.valorM2 = valorM2;
    }
    AreaCasa(double lateral, double cquarto) {
        this.lateral = lateral;
        this.cquarto = cquarto;
    }
    ...
    public static void main(String[] args) {
        AreaCasa a = new AreaCasa(1270);
        AreaCasa a2 = new AreaCasa(10,20);
        a2 = a;
    }
}
```



Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!
    - ★ Não há ninguém referenciado ele

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!
    - ★ Não há ninguém referenciado ele
- O Garbage Collector é nativo – varia de JVM para JVM

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!
    - ★ Não há ninguém referenciado ele
- O Garbage Collector é nativo – varia de JVM para JVM
- Não pode ser invocado

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!
    - ★ Não há ninguém referenciado ele
- O Garbage Collector é nativo – varia de JVM para JVM
- Não pode ser invocado
  - Não podemos obrigá-lo a rodar

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!
    - ★ Não há ninguém referenciado ele
- O Garbage Collector é nativo – varia de JVM para JVM
- Não pode ser invocado
  - Não podemos obrigá-lo a rodar
  - No máximo fazer uma requisição para que rode

Garbage Collection

- De tempos em tempos, o Garbage Collector varre os objetos na memória
  - Anotando quantas referências existem para cada objeto, em contadores
  - Com isso, sabe quantas referências há para aquele pedaço da memória
  - Quando um contador atinge zero, o objeto correspondente é varrido da memória!
    - ★ Não há ninguém referenciado ele
- O Garbage Collector é nativo – varia de JVM para JVM
- Não pode ser invocado
  - Não podemos obrigá-lo a rodar
  - No máximo fazer uma requisição para que rode
    - ★ *System.gc();*

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.
    - ★ Por isso, diz-se que variáveis associadas a objetos são referências a esses objetos.

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.
    - ★ Por isso, diz-se que variáveis associadas a objetos são referências a esses objetos.
  - Por exemplo, o que significa...

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.
    - ★ Por isso, diz-se que variáveis associadas a objetos são referências a esses objetos.
  - Por exemplo, o que significa...
    - ★ a == b (ambos inteiros)?

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.
    - ★ Por isso, diz-se que variáveis associadas a objetos são referências a esses objetos.
  - Por exemplo, o que significa...
    - ★ a == b (ambos inteiros)? Compara os valores

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.
    - ★ Por isso, diz-se que variáveis associadas a objetos são referências a esses objetos.
  - Por exemplo, o que significa...
    - ★ a == b (ambos inteiros)? Compara os valores
    - ★ obj1 == obj2 ?

Garbage Collection

- O GC limpa apenas tipos abstratos (Arranjos, Objetos etc), não primitivos (int, float etc)
  - A menos que estejam dentro de um abstrato
- Lembrando...
  - Variáveis de tipo primitivo armazenam o próprio conteúdo.
  - Isso não acontece com tipos abstratos (objetos complexos)!
    - ★ Como o nome diz, podem ser complexos e ocupar muito espaço de memória.
    - ★ Então, Java não guarda nas variáveis uma cópia do objeto, mas o endereço de memória no qual o objeto está armazenado.
    - ★ Por isso, diz-se que variáveis associadas a objetos são referências a esses objetos.
  - Por exemplo, o que significa...
    - ★ a == b (ambos inteiros)? Compara os valores
    - ★ obj1 == obj2 ? Compara as referências (endereços)

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina
- O que fazer?

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina
- O que fazer?
  - Conjuntos de casa e piscina – esse é o bloco básico

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina
- O que fazer?
  - Conjuntos de casa e piscina – esse é o bloco básico
  - Então...

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina
- O que fazer?
  - Conjuntos de casa e piscina – esse é o bloco básico
  - Então... fazemos disso uma classe

```
class Residencia {
    AreaCasa casa;
    AreaPiscina piscina;

    Residencia(AreaCasa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina
- O que fazer?
  - Conjuntos de casa e piscina – esse é o bloco básico
  - Então... fazemos disso uma classe
- E agora?

```
class Residencia {
    AreaCasa casa;
    AreaPiscina piscina;

    Residencia(AreaCasa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

Arranjos de Objetos

- Suponha que nosso projeto agora envolva um condomínio de residências
  - Cada uma com uma casa e uma piscina
- O que fazer?
  - Conjuntos de casa e piscina – esse é o bloco básico
  - Então... fazemos disso uma classe
- E agora?
  - Nosso projeto conterá um arranjo de objetos dessa classe

```
class Residencia {
    AreaCasa casa;
    AreaPiscina piscina;

    Residencia(AreaCasa casa, AreaPiscina piscina) {
        this.casa = casa;
        this.piscina = piscina;
    }
}
```

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }
}
```

Arranjos de Objetos

- E como usamos isso?

Arranjos de Objetos

- E como usamos isso?

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                    re.piscina.area());
        }

    }
}
```

Arranjos de Objetos

- E como usamos isso?
- Repare que reusamos as variáveis para os objetos

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                    re.piscina.area());
        }

    }
}
```

Arranjos de Objetos

- E como usamos isso?
- Repare que reusamos as variáveis para os objetos
- Com isso, não iremos perder o endereço na memória?

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                    re.piscina.area());
        }

    }
}
```

Arranjos de Objetos

- E como usamos isso?
- Repare que reusamos as variáveis para os objetos
- Com isso, não iremos perder o endereço na memória?
  - Via essa variável, sim

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                    re.piscina.area());
        }

    }
}
```

Arranjos de Objetos

- E como usamos isso?
- Repare que reusamos as variáveis para os objetos
- Com isso, não iremos perder o endereço na memória?
  - Via essa variável, sim
  - Mas antes passamos esse endereço a outras variáveis, via parâmetros

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                    re.piscina.area());
        }

    }
}
```

Arranjos de Objetos

- E como usamos isso?
- Repare que reusamos as variáveis para os objetos
- Com isso, não iremos perder o endereço na memória?
  - Via essa variável, sim
  - Mas antes passamos esse endereço a outras variáveis, via parâmetros
    - ★ *c* e *p* em *r* = *new Residencia(c, p)*

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                      re.piscina.area());
        }

    }
}
```

Arranjos de Objetos

- E como usamos isso?
- Repare que reusamos as variáveis para os objetos
- Com isso, não iremos perder o endereço na memória?
  - Via essa variável, sim
  - Mas antes passamos esse endereço a outras variáveis, via parâmetros
    - ★ *c* e *p* em *r = new Residencia(c, p)*
    - ★ *r* em *proj.adicionaRes(r)*

```
class Projeto {
    ...

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        AreaCasa c = new AreaCasa(10,5);
        AreaPiscina p = new AreaPiscina(5);
        Residencia r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(12,7);
        p = new AreaPiscina(6);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        c = new AreaCasa(10,6);
        p = new AreaPiscina(3.5);
        r = new Residencia(c, p);
        proj.adicionaRes(r);

        for (Residencia re : proj.condominio) {
            System.out.println("Área da Casa: "+
                                        re.casa.area());
            System.out.println("Área da Psicina: "+
                                      re.piscina.area());
        }

    }
}
```

Arranjos: Recapitulando

- Se o array é definido como tipo[]

Arranjos: Recapitulando

- Se o array é definido como tipo[]
  - Se tipo é um tipo primitivo (int, double, char...)

Arranjos: Recapitulando

- Se o array é definido como tipo[]
  - Se tipo é um tipo primitivo (int, double, char...)
    - ★ cada elemento tem o tamanho deste tipo primitivo

Arranjos: Recapitulando

- Se o array é definido como tipo[]
  - Se tipo é um tipo primitivo (int, double, char...)
    - ★ cada elemento tem o tamanho deste tipo primitivo
    - ★ cada elemento armazena um valor deste tipo

Arranjos: Recapitulando

- Se o array é definido como tipo[]
  - Se tipo é um tipo primitivo (int, double, char...)
    - ★ cada elemento tem o tamanho deste tipo primitivo
    - ★ cada elemento armazena um valor deste tipo
  - Se tipo é um tipo abstrato (uma classe, por exemplo)

Arranjos: Recapitulando

- Se o array é definido como tipo[]
  - Se tipo é um tipo primitivo (int, double, char...)
    - ★ cada elemento tem o tamanho deste tipo primitivo
    - ★ cada elemento armazena um valor deste tipo
  - Se tipo é um tipo abstrato (uma classe, por exemplo)
    - ★ cada elemento tem o tamanho de uma referência (endereço)

Arranjos: Recapitulando

- Se o array é definido como tipo[]
  - Se tipo é um tipo primitivo (int, double, char...)
    - ★ cada elemento tem o tamanho deste tipo primitivo
    - ★ cada elemento armazena um valor deste tipo
  - Se tipo é um tipo abstrato (uma classe, por exemplo)
    - ★ cada elemento tem o tamanho de uma referência (endereço)
    - ★ cada elemento armazena a referência para um objeto de tipo tipo

Arranjos de Objetos

- Voltemos então ao código de *Projeto*

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }

}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }

}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos
  - Arranjo de endereços de memória

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }



}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos
  - Arranjo de endereços de memória
- O que acontece quando o alocamos sem inicializar?

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }









}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos
  - Arranjo de endereços de memória
- O que acontece quando o alocamos sem inicializar?
  - O compilador irá inicializá-lo com um valor prévio

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }




}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos
  - Arranjo de endereços de memória
- O que acontece quando o alocamos sem inicializar?
  - O compilador irá inicializá-lo com um valor prévio
  - *null* – o endereço nulo

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }



}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos
  - Arranjo de endereços de memória
- O que acontece quando o alocamos sem inicializar?
  - O compilador irá inicializá-lo com um valor prévio
  - *null* – o endereço nulo
- Então a saída do *main* será

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        for (Residencia re : proj.condominio) {
            System.out.println("Endereço do objeto: "
                                                +re);
        }
    }
}
```

Arranjos de Objetos

- Voltemos então ao código de *Projeto*
- *condominio* é arranjo de objetos
  - Arranjo de endereços de memória
- O que acontece quando o alocamos sem inicializar?
  - O compilador irá inicializá-lo com um valor prévio
  - *null* – o endereço nulo
- Então a saída do *main* será

```
$ java Projeto
Endereço do objeto: null
Endereço do objeto: null
Endereço do objeto: null
```

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1; // último alocado

    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }

    Projeto(int tam) {
        condominio = new Residencia[tam];
    }

    public static void main(String[] args) {
        Projeto proj = new Projeto(3);

        for (Residencia re : proj.condominio) {
            System.out.println("Endereço do objeto: "
                                                +re);
        }
    }
}
```