Aula 18 – Ordenação e Herança

Norton Trevisan Roman

24 de maio de 2018

Ordenação por Inserção (Insertion sort)

Algoritmo

Ordenação por Inserção (Insertion sort)

Algoritmo

- Percorremos o arranjo e, a cada novo elemento:

Ordenação por Inserção (Insertion sort)

Algoritmo

- Percorremos o arranjo e, a cada novo elemento:
  - Procuramos onde, à esquerda desse elemento, ele se encaixa

Ordenação por Inserção (Insertion sort)

Algoritmo

- Percorremos o arranjo e, a cada novo elemento:
  - Procuramos onde, à esquerda desse elemento, ele se encaixa
  - Abrimos espaço para o elemento lá, deslocando para a direita todos os elementos que estão entre essa posição e a original do elemento

Ordenação por Inserção (Insertion sort)

Algoritmo

- Percorremos o arranjo e, a cada novo elemento:
  - Procuramos onde, à esquerda desse elemento, ele se encaixa
  - Abrimos espaço para o elemento lá, deslocando para a direita todos os elementos que estão entre essa posição e a original do elemento
  - Inserimos o elemento nesse espaço assim aberto

Ordenação por Inserção (Insertion sort)

- Ou seja, aumentamos a parte ordenada do arranjo em uma posição, inserindo um novo elemento na posição correta e deslocando os demais para a direita

Ordenação por Inserção (Insertion sort)

- Ou seja, aumentamos a parte ordenada do arranjo em uma posição, inserindo um novo elemento na posição correta e deslocando os demais para a direita
- Semelhante ao modo como ordenamos cartas de baralho

Ordenação por Inserção (Insertion sort)

- Ou seja, aumentamos a parte ordenada do arranjo em uma posição, inserindo um novo elemento na posição correta e deslocando os demais para a direita
- Semelhante ao modo como ordenamos cartas de baralho
  - Percorremos da esquerda para a direita e, à medida que avançamos vamos deixando as cartas mais à esquerda ordenadas

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente

9 8 4 10 6

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado

9 8 4 10 6

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):

9	8	4	10	6
9	8	4	10	6

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 9 & \underline{8} & 4 & 10 & 6 \\ \uparrow & & & & \end{array}$$

  - Identificamos onde ele deve estar, na parte ordenada

Ordenação por Inserção (Insertion sort)

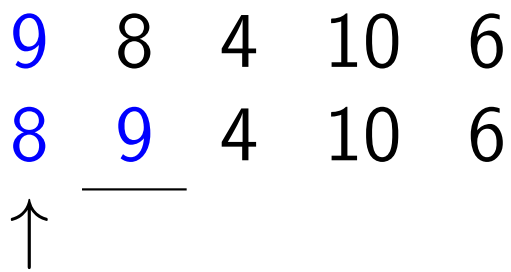
- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 9 & 9 & 4 & 10 & 6 \\ \uparrow & \rightarrow & & & \end{array}$$

  - Deslocamos a parte ordenada a partir dessa posição

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):



  - Inserimos o 8 na posição correta

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

9	8	4	10	6
8	9	4	10	6
8	9	4	10	6

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 8 & 9 & \underline{\quad 4 \quad} & 10 & 6 \\ \uparrow & & & & \end{array}$$

  - Identificamos onde ele deve estar, na parte ordenada

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 8 & 8 & 9 & 10 & 6 \\ \uparrow & \rightarrow & \_\_\_ & & \end{array}$$

  - Deslocamos a parte ordenada a partir dessa posição

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
↑		———		

  - Inserimos o 4 na posição correta

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o terceiro elemento a ser inserido (10):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o terceiro elemento a ser inserido (10):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6

  - Identificamos onde ele deve estar, na parte ordenada

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o terceiro elemento a ser inserido (10):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6

  - Não há necessidade de deslocamento e inserção. Já está na posição correta

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6
4	8	9	10	6

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6
4	8	9	10	6
	↑			

  - Identificamos onde ele deve estar, na parte ordenada

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6
4	8	8	9	10
	↑	→		

  - Deslocamos a parte ordenada a partir dessa posição

Ordenação por Inserção (Insertion sort)

- Ex: ordene em ordem crescente
  - Em azul está o sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
4	8	9	10	6
4	6	8	9	10

  - Inserimos o 6 na posição correta

Ordenação por Inserção (Insertion sort)

- Note que, a cada passo:

Ordenação por Inserção (Insertion sort)

- Note que, a cada passo:
  - Encontrávamos a posição em que o valor deveria estar

Ordenação por Inserção (Insertion sort)

- Note que, a cada passo:
  - Encontrávamos a posição em que o valor deveria estar
  - Deslocávamos os elementos necessários

Ordenação por Inserção (Insertion sort)

- Note que, a cada passo:
  - Encontrávamos a posição em que o valor deveria estar
  - Deslocávamos os elementos necessários
  - Inseríamos o valor nessa posição

Ordenação por Inserção (Insertion sort)

- Uma melhoria direta seria:

Ordenação por Inserção (Insertion sort)

- Uma melhoria direta seria:
  - Para cada passo do vetor original, a partir da posição 1 até o final

Ordenação por Inserção (Insertion sort)

- Uma melhoria direta seria:
  - Para cada passo do vetor original, a partir da posição 1 até o final
    - Deslocar o arranjo ordenado para direita (uma posição por vez) até encontrar local adequado para o valor a ser inserido

Ordenação por Inserção (Insertion sort)

- Uma melhoria direta seria:
  - Para cada passo do vetor original, a partir da posição 1 até o final
    - Deslocar o arranjo ordenado para direita (uma posição por vez) até encontrar local adequado para o valor a ser inserido
    - Inserir esse valor na posição correta

Ordenação por Inserção (Insertion sort)

- Uma melhoria direta seria:
  - Para cada passo do vetor original, a partir da posição 1 até o final
    - Deslocar o arranjo ordenado para direita (uma posição por vez) até encontrar local adequado para o valor a ser inserido
    - Inserir esse valor na posição correta
- Evita assim a necessidade de se buscar a posição de antemão

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente

9 8 4 10 6

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado

9 8 4 10 6

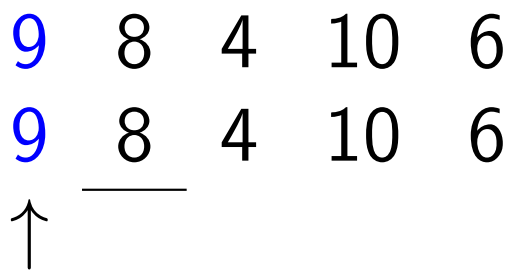
Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 9 & \underline{8} & 4 & 10 & 6 \end{array}$$

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):



  - Menor que o elemento atual

Ordenação por Inserção (Melhoria)

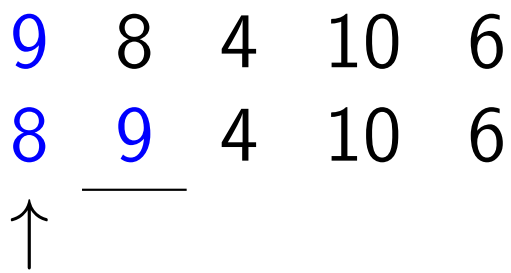
- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):

9	8	4	10	6
9	9	4	10	6
↑	→			

  - Deslocamos esse elemento

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o primeiro elemento a ser inserido (8):



  - Inserimos o 8 na posição correta

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

9	8	4	10	6
8	9	4	10	6
8	9	4	10	6

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 8 & 9 & \underline{4} & 10 & 6 \\ & & \uparrow & & \end{array}$$

  - Menor que o elemento atual

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 8 & 9 & 9 & 10 & 6 \\ & \uparrow & \rightarrow & & \end{array}$$

  - Deslocamos esse elemento

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

9	8	4	10	6
8	9	4	10	6
8	9	9	10	6
↑		———		

  - Menor que o elemento atual

Ordenação por Inserção (Melhoria)

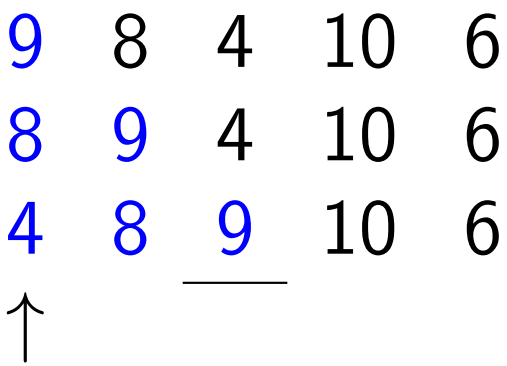
- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 8 & 8 & 9 & 10 & 6 \\ \uparrow & \rightarrow & \multicolumn{3}{c}{\multirow{2}{*}{\_\_\_\_}} \end{array}$$

  - Deslocamos esse elemento

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o segundo elemento a ser inserido (4):



  - Inserimos o 4 na posição correta

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o terceiro elemento a ser inserido (10):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o terceiro elemento a ser inserido (10):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
		↑		

  - Maior que o elemento atual

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o terceiro elemento a ser inserido (10):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
		↑		

  - Já está na posição correta

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	6
			↑	

  - Menor que o elemento atual

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 4 & 8 & 9 & 10 & 10 \\ & & & \uparrow & \rightarrow \end{array}$$

  - Deslocamos esse elemento

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	10	10
		↑		

  - Menor que o elemento atual

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 4 & 8 & 9 & 9 & 10 \\ & & \uparrow & \rightarrow & \end{array}$$

  - Deslocamos esse elemento

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

9	8	4	10	6
8	9	4	10	6
4	8	9	9	10
	↑			

  - Menor que o elemento atual

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 4 & 8 & 8 & 9 & 10 \\ & \uparrow & \rightarrow & & \end{array}$$

  - Deslocamos esse elemento

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 4 & 8 & 8 & 9 & 10 \\ \uparrow & & & & \end{array}$$

  - Maior que o elemento atual

Ordenação por Inserção (Melhoria)

- Ex: ordene em ordem crescente
  - Sub-arranjo já ordenado
  - Analisando o quarto elemento a ser inserido (6):

$$\begin{array}{ccccc} 9 & 8 & 4 & 10 & 6 \\ 8 & 9 & 4 & 10 & 6 \\ 4 & 6 & 8 & 9 & 10 \\ \uparrow & & & & \end{array}$$

  - Inserimos o 6 na posição correta

Ordenação por Inserção (Insertion sort)

- Então...

Ordenação por Inserção (Insertion sort)

- Então...

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int j = i;
    while ((j > 0) && (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}

public static void main(String[] args) {
  int[] v = {9,8,4,10,6};

  insercao(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Ordenação por Inserção (Insertion sort)

- Então...
- Corremos todos os possíveis candidatos a inserção

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int j = i;
    while ((j > 0) && (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}

public static void main(String[] args) {
  int[] v = {9,8,4,10,6};

  insercao(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Ordenação por Inserção (Insertion sort)

- Então...
- Corremos todos os possíveis candidatos a inserção
  - Enquanto o candidato estiver fora de lugar, deslocamos os anteriores a ele, até achar o lugar certo

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int j = i;
    while ((j > 0) && (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}

public static void main(String[] args) {
  int[] v = {9,8,4,10,6};

  insercao(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Ordenação por Inserção (Insertion sort)

- Então...
- Corremos todos os possíveis candidatos a inserção
  - Enquanto o candidato estiver fora de lugar, deslocamos os anteriores a ele, até achar o lugar certo
  - E colocamos ele lá

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int j = i;
    while ((j > 0) && (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}

public static void main(String[] args) {
  int[] v = {9,8,4,10,6};

  insercao(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Ordenação por Inserção (Insertion sort)

- Então...
- Corremos todos os possíveis candidatos a inserção
  - Enquanto o candidato estiver fora de lugar, deslocamos os anteriores a ele, até achar o lugar certo
  - E colocamos ele lá

Saída

```
$ java Projeto
4 6 8 9 10
```

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int j = i;
    while ((j > 0) && (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}

public static void main(String[] args) {
  int[] v = {9,8,4,10,6};

  insercao(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Ordenação por Inserção (Insertion sort)

- E como ficaria a versão com objetos?

Ordenação por Inserção (Insertion sort)

- E como ficaria a versão com objetos?

```
static void insercao(Residencia[] v) {
  for (int i=1; i<v.length; i++) {
    Residencia aux = v[i];
    int j = i;
    while ((j > 0) &&
       (aux.comparaRes(v[j-1]) < 0)) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

Ordenação por Inserção (Insertion sort)

- E como ficaria a versão com objetos?

```
static void insercao(Residencia[] v) {
  for (int i=1; i<v.length; i++) {
    Residencia aux = v[i];
    int j = i;
    while ((j > 0) &&
       (aux.comparaRes(v[j-1]) < 0)) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);

  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(Math.
         random()*100,Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                        Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  insercao(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Ordenação por Inserção (Insertion sort)

Saída

```
$ java Projeto
6583.183940438665
1130.4506182200782
7379.352930903931
3289.6719206296757
5739.294165717424

1130.4506182200782
3289.6719206296757
5739.294165717424
6583.183940438665
7379.352930903931
```

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);

  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(Math.
        random()*100,Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                        Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  insercao(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Ordenação – Comparação

- Seleção e Inserção:

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado
  - Diferem em como fazem isso:

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado
  - Diferem em como fazem isso:
    - Seleção troca 2 elementos

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado
  - Diferem em como fazem isso:
    - Seleção troca 2 elementos
    - Inserção desloca à direita parte ou todo o subvetor

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado
  - Diferem em como fazem isso:
    - Seleção troca 2 elementos
    - Inserção desloca à direita parte ou todo o subvetor
- Bolha:

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado
  - Diferem em como fazem isso:
    - Seleção troca 2 elementos
    - Inserção desloca à direita parte ou todo o subvetor
- Bolha:
  - Garante que, no passo i, o subvetor de `tam-1-i` a `tam-1` está ordenado

Ordenação – Comparação

- Seleção e Inserção:
  - Garantem que, no passo i, o subvetor de 0 a i está ordenado
  - Diferem em como fazem isso:
    - Seleção troca 2 elementos
    - Inserção desloca à direita parte ou todo o subvetor
- Bolha:
  - Garante que, no passo i, o subvetor de `tam-1-i` a `tam-1` está ordenado
- Todos ordenam *in loco*/*in place* (no próprio vetor, sem precisar de vetor auxiliar)

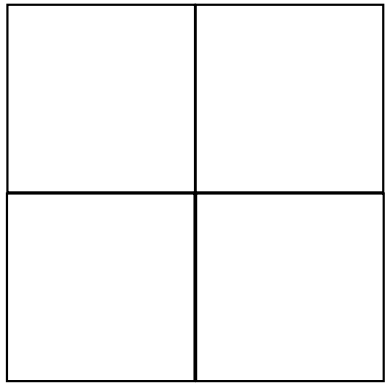
Curiosidades

- Bolha:
  - `http://www.youtube.com/watch?v=lyZQPjUT5B4`
- Seleção:
  - `http://www.youtube.com/watch?v=Ns4TPTC8whw` (versão levemente diferente do algoritmo)
- Inserção:
  - `http://www.youtube.com/watch?v=ROalU37913U`

Herança

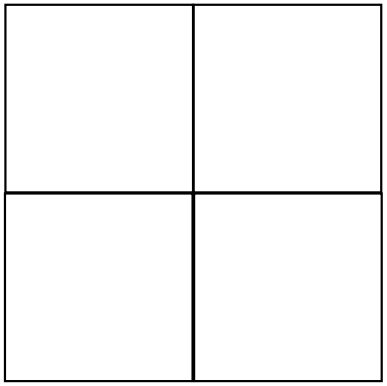
Herança

- Temos agora outro tipo de cabana, com outro formato.



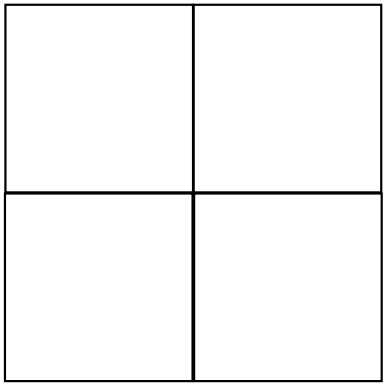
Herança

- Temos agora outro tipo de cabana, com outro formato.
- Queremos calcular também a área e custo



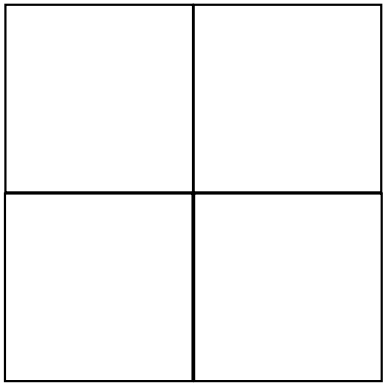
Herança

- Temos agora outro tipo de cabana, com outro formato.
- Queremos calcular também a área e custo
- Quais os dados necessários?



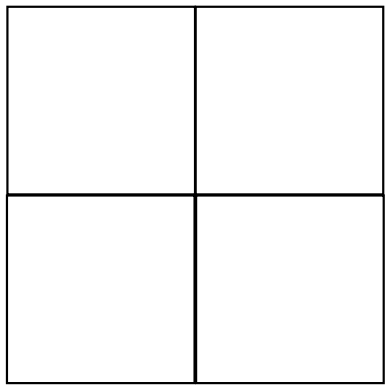
Herança

- Temos agora outro tipo de cabana, com outro formato.
- Queremos calcular também a área e custo
- Quais os dados necessários?
  - Lateral da cabana



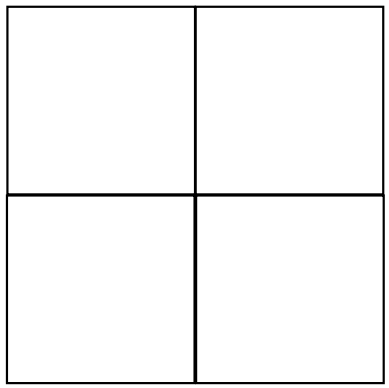
Herança

- Temos agora outro tipo de cabana, com outro formato.
- Queremos calcular também a área e custo
- Quais os dados necessários?
  - Lateral da cabana
  - Preço do m^2



Herança

- Temos agora outro tipo de cabana, com outro formato.
- Queremos calcular também a área e custo
- Quais os dados necessários?
  - Lateral da cabana
  - Preço do m^2
- Que fazer?



Herança

- Podemos construir uma classe para essa casa

Herança

- Podemos construir uma classe para essa casa
  - CasaQuad

Herança

- Podemos construir uma classe para essa casa
  - CasaQuad

```
class CasaQuad {
  double valorM2 = 1500;
  double lateral = 10;

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

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

Herança

- Podemos construir uma classe para essa casa
  - CasaQuad
- Note que não há construtor do tipo:

```
CasaQuad(double valorM2) {
  this.valorM2 = valorM2;
}
```

```
class CasaQuad {
  double valorM2 = 1500;
  double lateral = 10;

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

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

Herança

- Se houvesse, ao compilarmos teríamos:

Linha de Comando

```
$ javac CasaQuad.java
CasaQuad.java:24: CasaQuad(double) is
          already defined in CasaQuad
    CasaQuad(double valorM2) {
    ^
1 error
```

```
class CasaQuad {
  double valorM2 = 1500;
  double lateral = 10;

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

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

Comparando AreaCasa e CasaQuad

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

    ... (construtores) ...

    double area() {
        double areat=-1;

        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);
    }
}
```

```
class CasaQuad {
    double valorM2 = 1500;
    double lateral = 10;


    ... (construtores) ...

    double area() {
        return(this.lateral>=0 ?
                    this.lateral*this.lateral :
                                            -1);
    }


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

Comparando AreaCasa e CasaQuad

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

    ... (construtores) ...

    double area() {
        double areat=-1;

        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);
    }
}
```

```
class CasaQuad {
    double valorM2 = 1500;
    double lateral = 10;


    ... (construtores) ...

    double area() {
        return(this.lateral>=0 ?
                    this.lateral*this.lateral :
                                            -1);
    }


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

- Há vários pontos em comum

Comparando AreaCasa e CasaQuad

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

    ... (construtores) ...

    double area() {
        double areat=-1;

        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);
    }
}
```

```
class CasaQuad {
    double valorM2 = 1500;
    double lateral = 10;


    ... (construtores) ...

    double area() {
        return(this.lateral>=0 ?
                    this.lateral*this.lateral :
                                        -1);
    }


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

- E pontos bastante semelhantes

Comparando AreaCasa e CasaQuad

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

    ... (construtores) ...

    double area() {
        double areat=-1;

        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);
    }
}
```

```
class CasaQuad {
    double valorM2 = 1500;
    double lateral = 10;


    ... (construtores) ...

    double area() {
        return(this.lateral>=0 ?
                    this.lateral*this.lateral :
                                        -1);
    }


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

- E pontos bastante semelhantes
  - Mesma assinatura, mas comportamento ou interpretação diferentes

Herança

- Quase todo o código das classes ou está repetido ou muito semelhante

Herança

- Quase todo o código das classes ou está repetido ou muito semelhante
  - Repetição do trabalho

Herança

- Quase todo o código das classes ou está repetido ou muito semelhante
  - Repetição do trabalho
  - Desperdício de tempo de desenvolvimento

Herança

- Quase todo o código das classes ou está repetido ou muito semelhante
  - Repetição do trabalho
  - Desperdício de tempo de desenvolvimento
  - Desperdício de tempo de validação do código (teste das mesmas coisas....)

Herança

- Quase todo o código das classes ou está repetido ou muito semelhante
  - Repetição do trabalho
  - Desperdício de tempo de desenvolvimento
  - Desperdício de tempo de validação do código (teste das mesmas coisas....)
  - Mudanças em uma parte comum a elas devem ser feitas (e testadas novamente) nas duas classes!

Herança

Há uma solução?

Herança

Há uma solução?

- Agrupar o que for comum (exatamente igual) em ambas em uma nova classe

Herança

Há uma solução?

- Agrupar o que for comum (exatamente igual) em ambas em uma nova classe
- Ambas então compartilhariam das definições (não necessariamente memória) contidas nessa nova classe

Herança

Há uma solução?

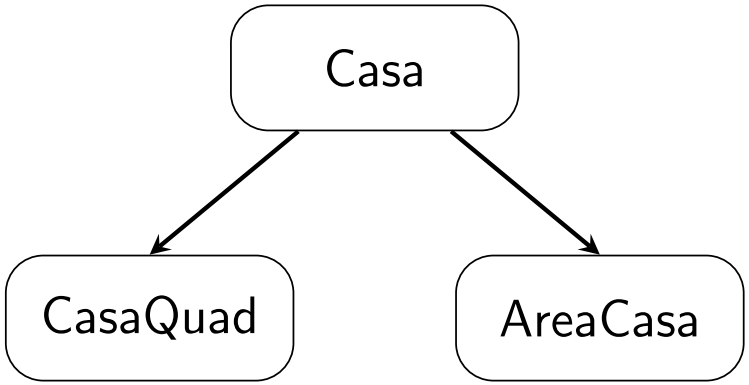
- Agrupar o que for comum (exatamente igual) em ambas em uma nova classe
- Ambas então compartilhariam das definições (não necessariamente memória) contidas nessa nova classe
- Ambas herdariam essas definições

Herança

- Esse esquema dá origem a uma hierarquia de classes

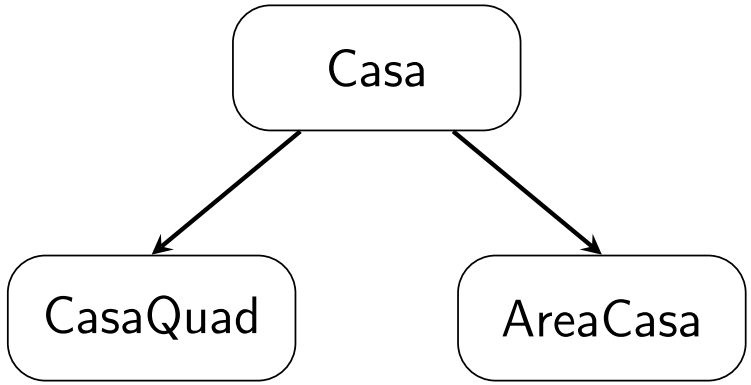
Herança

- Esse esquema dá origem a uma hierarquia de classes



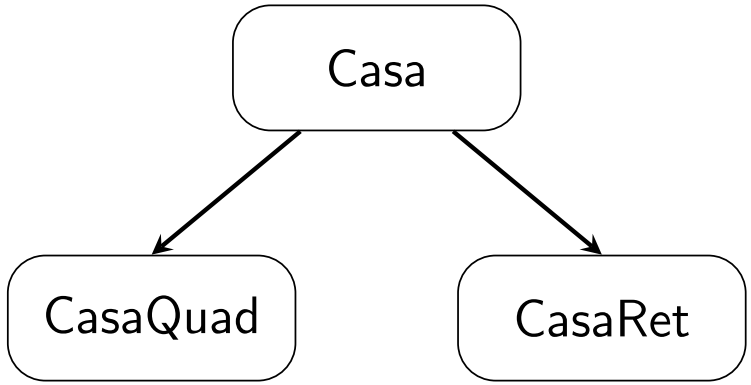
Herança

- Esse esquema dá origem a uma hierarquia de classes
  - Nela, a nova classe – `Casa` – está acima de `CasaQuad` e `AreaCasa`



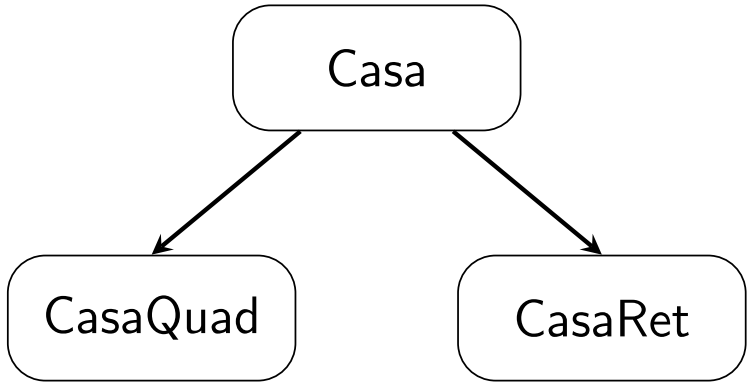
Herança

- Esse esquema dá origem a uma hierarquia de classes
  - Nela, a nova classe – `Casa` – está acima de `CasaQuad` e `AreaCasa`
  - Para manter o padrão, podemos rebatizar `AreaCasa` para `CasaRet`



Herança

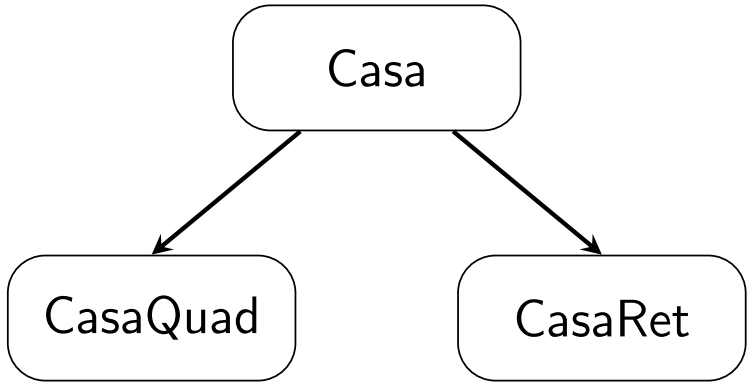
- Esse esquema dá origem a uma hierarquia de classes
  - Nela, a nova classe – `Casa` – está acima de `CasaQuad` e `AreaCasa`
  - Para manter o padrão, podemos rebatizar `AreaCasa` para `CasaRet`



- Diz-se que `CasaQuad` e `CasaRet` são subclasses de `Casa`

Herança

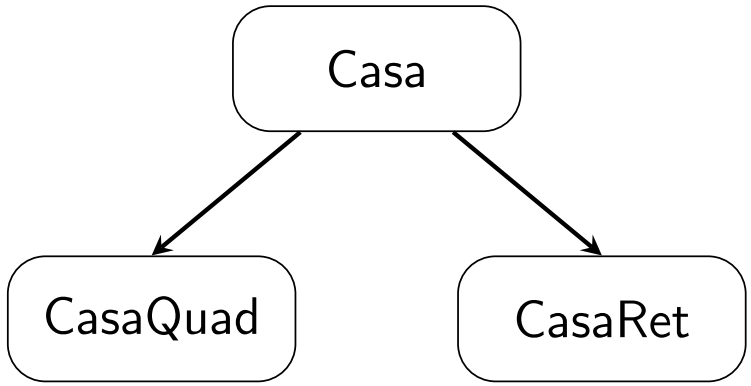
- Esse esquema dá origem a uma hierarquia de classes
  - Nela, a nova classe – `Casa` – está acima de `CasaQuad` e `AreaCasa`
  - Para manter o padrão, podemos rebatizar `AreaCasa` para `CasaRet`



- Diz-se que `CasaQuad` e `CasaRet` são subclasses de `Casa`
  - E que `Casa` é superclasse de `CasaQuad` e `CasaRet`

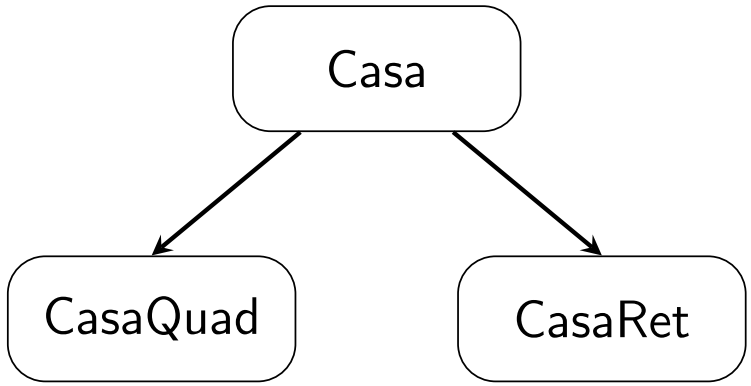
Herança

- Classes mais especializadas (subclasses) herdam propriedades (atributos e código) da classe mais geral (superclasse)



Herança

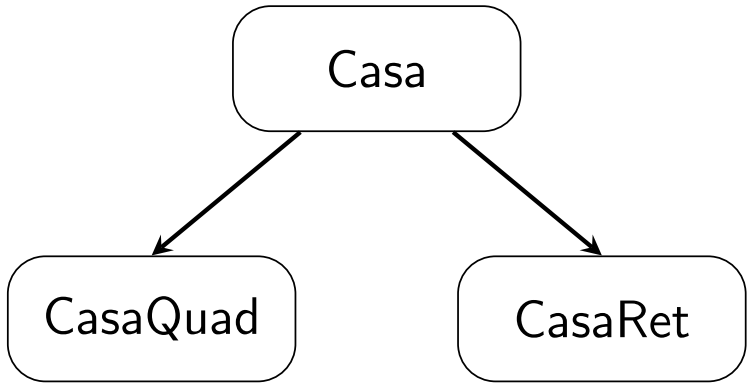
- Classes mais especializadas (subclasses) herdam propriedades (atributos e código) da classe mais geral (superclasse)



- Cria-se uma subclasse inserindo somente as diferenças desta para sua superclasse

Herança

- Classes mais especializadas (subclasses) herdam propriedades (atributos e código) da classe mais geral (superclasse)



- Cria-se uma subclasse inserindo somente as diferenças desta para sua superclasse
- Identifica-se a possibilidade de herança por meio da expressão "é um tipo de"

Herança

- Como criamos `Casa`?

Herança

- Como criamos `Casa`?
  - Separando os atributos e métodos idênticos em `CasaQuad` e `CasaRet`

```
class Casa {
  double valorM2 = 1500;

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

Herança

- Como criamos `Casa`?
  - Separando os atributos e métodos idênticos em `CasaQuad` e `CasaRet`
- E `CasaQuad`?

```
class Casa {
  double valorM2 = 1500;

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

Herança

- Inserindo somente as diferenças

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- Inserindo somente as diferenças
- E dizendo ao compilador quem é a classe mãe (superclasse)

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- Esse procedimento não cria um objeto extra para a superclasse

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- Esse procedimento não cria um objeto extra para a superclasse
  - É como se copiasse os atributos desta para dentro da memória da subclasse

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- E por que lateral, embora comum a `CasaRet` e `CasaQuad`, foi colocada na subclasse?

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- E por que lateral, embora comum a `CasaRet` e `CasaQuad`, foi colocada na subclasse?
  - Porque tem significados diferentes: Lateral da casa e da sala

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- E por que lateral, embora comum a `CasaRet` e `CasaQuad`, foi colocada na subclasse?
  - Porque tem significados diferentes: Lateral da casa e da sala
  - Poderia também haver uma casa redonda, em que esse atributo não faria sentido

```
class CasaQuad extends Casa {
  double lateral = 10;

  CasaQuad() {}

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

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

  double area() {
    return(this.lateral>=0 ?
        this.lateral*this.lateral : -1);
  }
}
```

Herança

- CasaRet sai de maneira semelhante:

```
class CasaRet extends Casa {
  double cquarto = 10;
  double lateral = 10;

  CasaRet() {}

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

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

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

  double area() {
    double areat=-1;
    if (this.lateral>=0 &&
                        this.cquarto>=0) {
      areat = this.lateral*this.lateral;
      areat += this.cquarto*this.lateral;
    }
    return(areat);
  }
}
```

Herança

Lembrando que...

Herança

Lembrando que...

- `CasaQuad` é um tipo de `Casa`

Herança

Lembrando que...

- `CasaQuad` é um tipo de `Casa`
- `CasaRet` é um tipo de `Casa`

Herança

Lembrando que...

- `CasaQuad` é um tipo de `Casa`
- `CasaRet` é um tipo de `Casa`
- Podemos identificar a possibilidade de herança por meio da expressão "é um tipo de"

Herança

- E como usamos essa nova versão?

Herança

- E como usamos essa nova versão?
  - Como antes:

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Área: "+cr.area());
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("M2: "+cr.valorM2);
  System.out.println("Valor: "+cr.valor(
                                   cr.area());
  System.out.println();
  System.out.println("Área: "+cq.area());
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("M2: "+cq.valorM2);
  System.out.println("Valor: "+cq.valor(
                                   cq.area());
}
```

Herança

- E como usamos essa nova versão?
  - Como antes:
- A saída será:

```
$ java Projeto
Quarto: 5.0
Área: 150.0
Lateral da sala: 10.0
M2: 1320.0
Valor: 198000.0

Área: 100.0
Lateral da casa: 10.0
M2: 1523.0
Valor: 152300.0
```

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Área: "+cr.area());
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("M2: "+cr.valorM2);
  System.out.println("Valor: "+cr.valor(
                                   cr.area());
  System.out.println();
  System.out.println("Área: "+cq.area());
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("M2: "+cq.valorM2);
  System.out.println("Valor: "+cq.valor(
                                   cq.area());
}
```

Herança

- Temos acesso a todos os métodos e atributos tanto da subclasse

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Área: "+cr.area());
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("M2: "+cr.valorM2);
  System.out.println("Valor: "+cr.valor(
                                   cr.area());
  System.out.println();
  System.out.println("Área: "+cq.area());
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("M2: "+cq.valorM2);
  System.out.println("Valor: "+cq.valor(
                                   cq.area());
}
```

Herança

- Temos acesso a todos os métodos e atributos tanto da subclasse quanto da superclasse

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Área: "+cr.area());
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("M2: "+cr.valorM2);
  System.out.println("Valor: "+cr.valor(
                                   cr.area());
  System.out.println();
  System.out.println("Área: "+cq.area());
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("M2: "+cq.valorM2);
  System.out.println("Valor: "+cq.valor(
                                   cq.area());
}
```

Herança

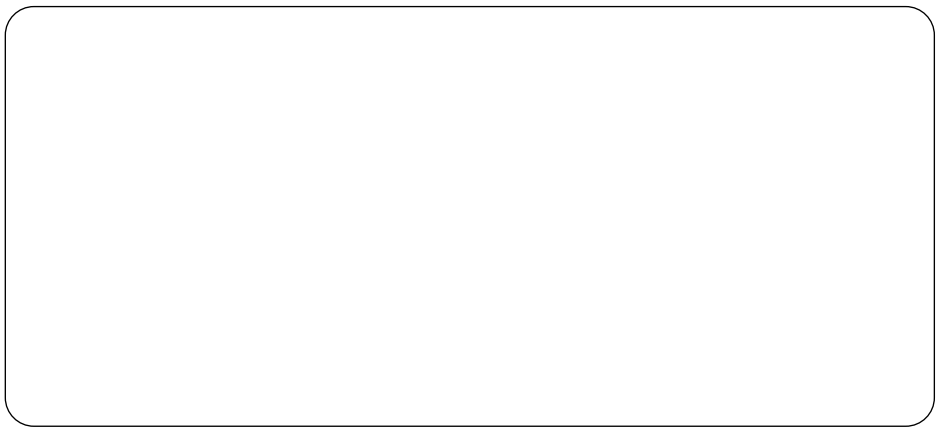
- Temos acesso a todos os métodos e atributos tanto da subclasse quanto da superclasse
- Não há compartilhamento de memória (pois não há `static`) – cada um tem o seu espaço

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Área: "+cr.area());
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("M2: "+cr.valorM2);
  System.out.println("Valor: "+cr.valor(
                                   cr.area());
  System.out.println();
  System.out.println("Área: "+cq.area());
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("M2: "+cq.valorM2);
  System.out.println("Valor: "+cq.valor(
                                   cq.area());
}
```

Herança e Memória

- Como se dá o funcionamento da memória nesse caso?

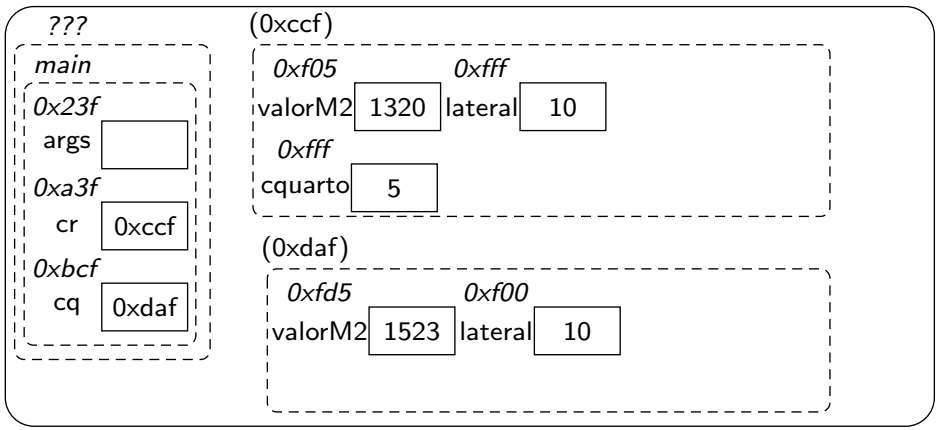
```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Herança e Memória

- Como se dá o funcionamento da memória nesse caso?
- Ao criarmos os objetos, alocamos espaço para atributos tanto de sua classe quanto superclasse

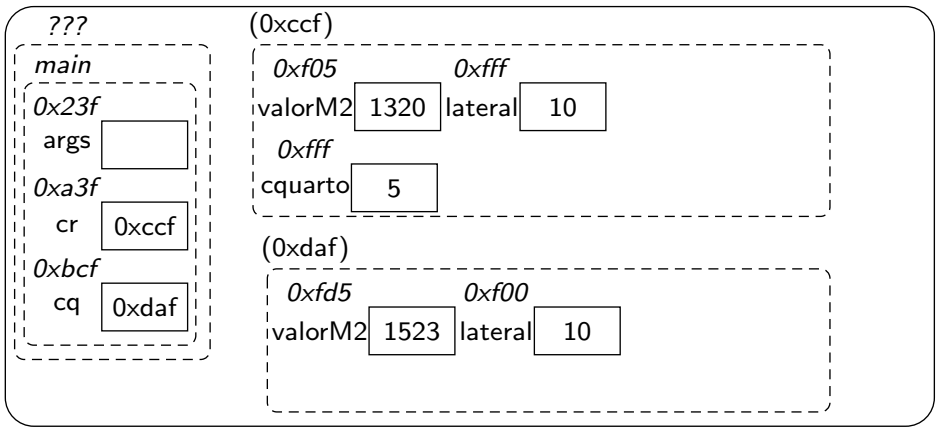
```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Herança e Memória

- Como se dá o funcionamento da memória nesse caso?
- Ao criarmos os objetos, alocamos espaço para atributos tanto de sua classe quanto superclasse
  - O funcionamento dos construtores foi omitido para simplificar

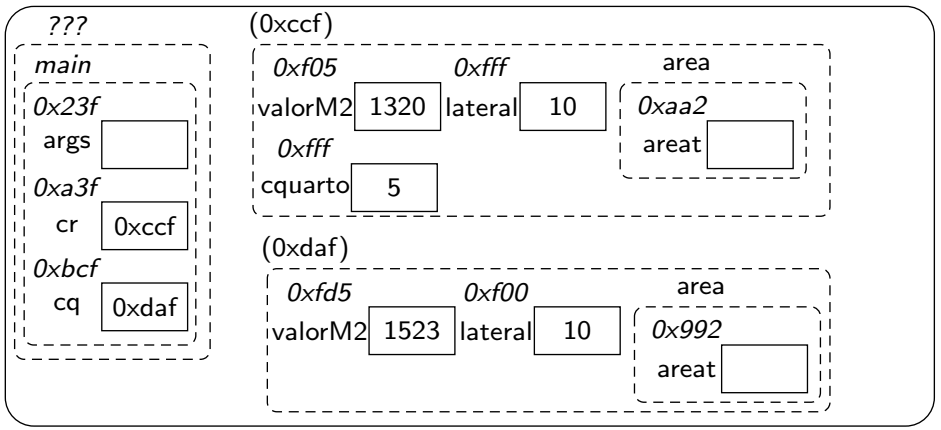
```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Herança e Memória

- Cada objeto tem seus próprios atributos e métodos

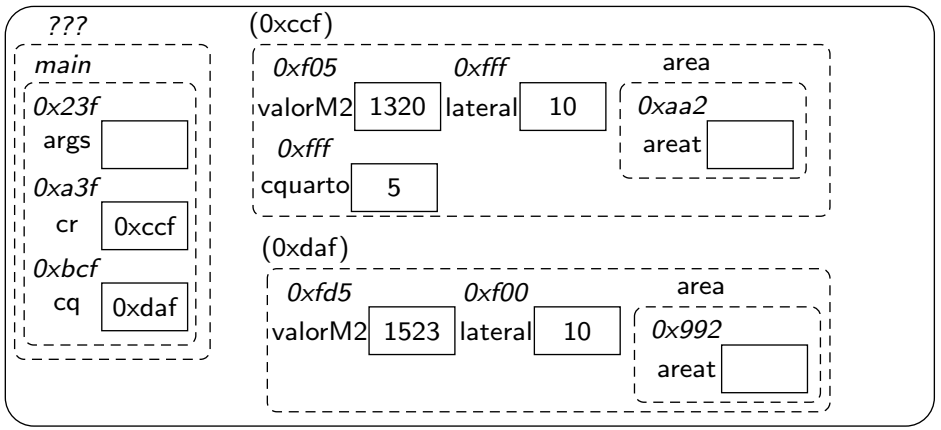
```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Herança e Memória

- Cada objeto tem seus próprios atributos e métodos
- Não há compartilhamento de nada

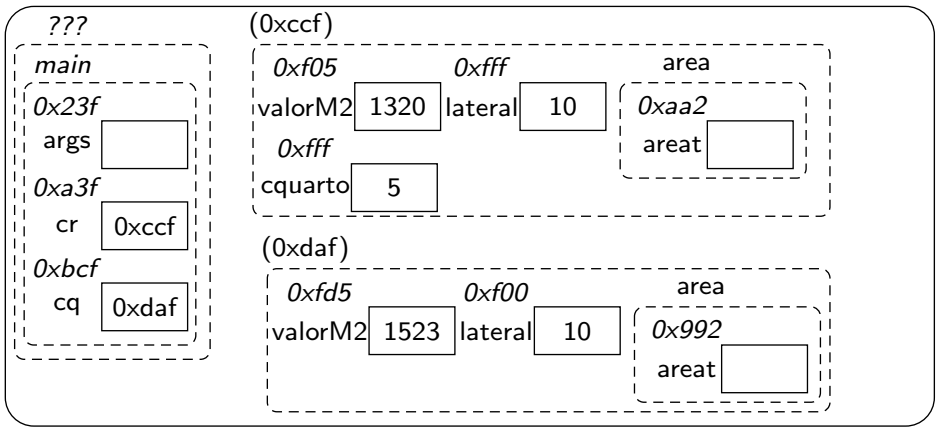
```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Herança e Memória

- Cada objeto tem seus próprios atributos e métodos
- Não há compartilhamento de nada
  - A menos que haja algum atributo ou método `static`

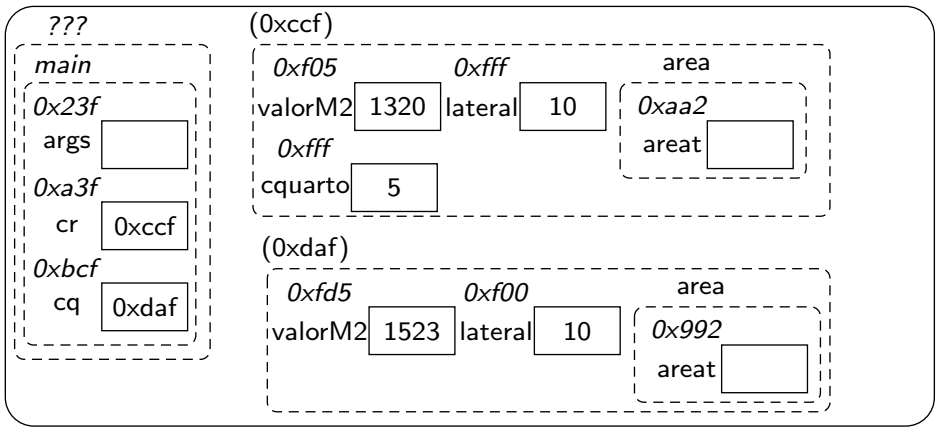
```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Herança e Memória

- Cada objeto tem seus próprios atributos e métodos
- Não há compartilhamento de nada
  - A menos que haja algum atributo ou método `static`
  - Nesse caso, <u>todos</u> os objetos compartilharão do mesmo atributo/método na memória

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);

  System.out.println("Quarto: "+
                                  cr.cquarto);
  System.out.println("Lateral da sala: "+
                                  cr.lateral);
  System.out.println("Área: "+cr.area());
  System.out.println();
  System.out.println("Lateral da casa: "+
                                  cq.lateral);
  System.out.println("Área: "+cq.area());
}
```



Videoaula

https://www.youtube.com/watch?v=_4iTg2K4bqY

e

https://www.youtube.com/watch?v=pnGhdy5B02I
(pequena parte)

(cobrem parcialmente o conteúdo)