Algoritmos e Estruturas de Dados

Algoritmos de Ordenação

Autores:
Carlos Urbano
Catarina Reis
José Magno
Marco Ferreira

Algoritmos de Ordenação

- SelectionSort
- BubbleSort
- BubbleSort Otimizado
- QuickSort

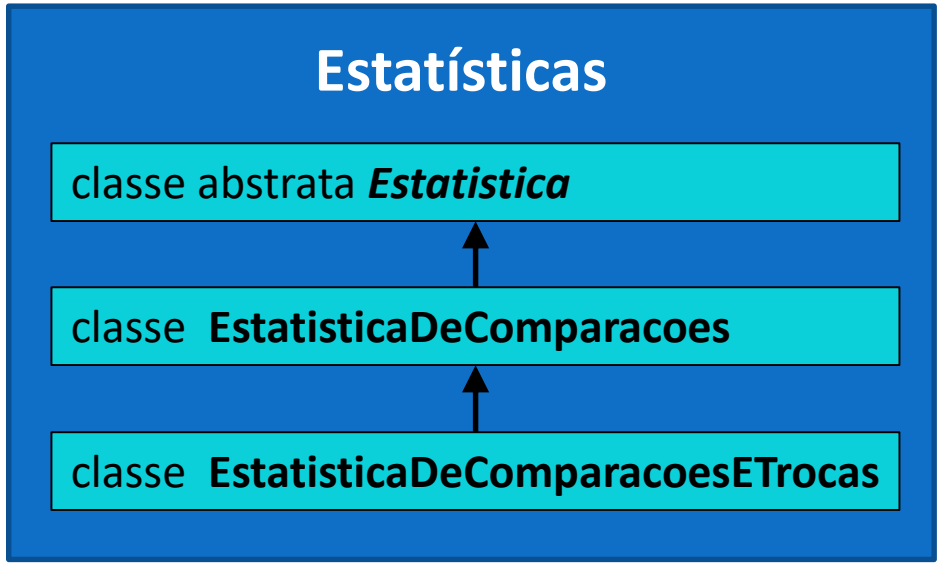
Algoritmos de Ordenação

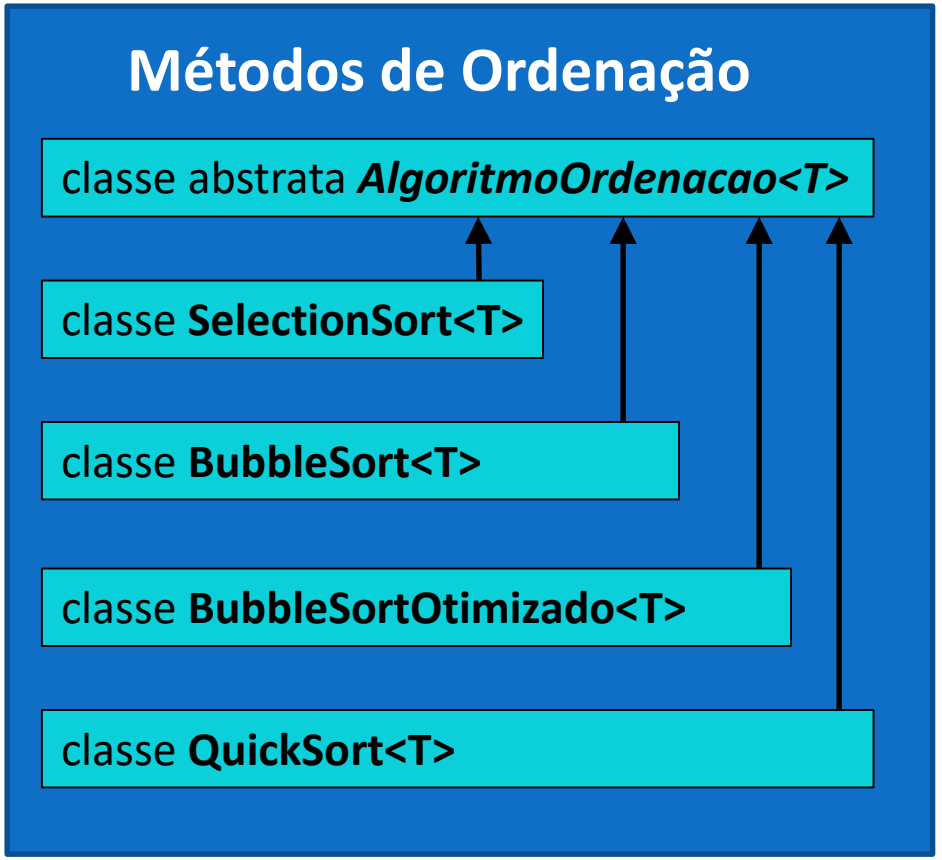
- Um algoritmo de ordenação distribui uma sequência de elementos de acordo com uma determinada ordem (critério de comparação dos elementos a ordenar)

- Assim, cada algoritmo de comparação deve receber o critério de comparação dos elementos a ordenar
  - Para distribuir os elementos por uma determinada ordem é necessário saber como comparar os elementos 2 a 2, isto é, uma função que efetue a comparação de dois elementos

Algoritmos de Ordenação

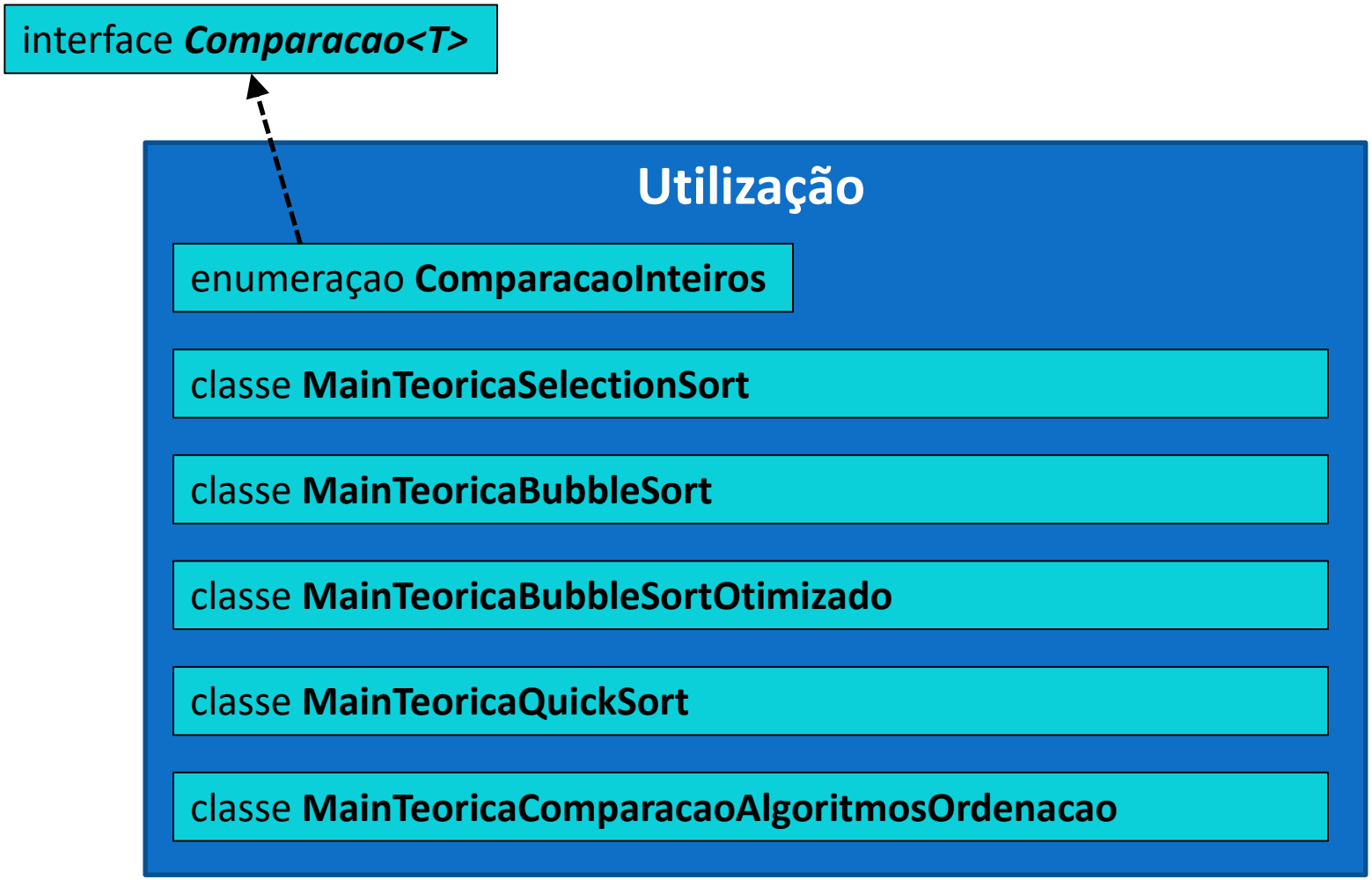
- Classes a construir de seguida





Algoritmos de Ordenação

- Classes a construir de seguida



Algoritmos de Ordenação

- Para definir o critério de comparação dos elementos da sequência, vamos considerar a seguinte interface:

```
public interface Comparacao<T> {
    /**
    * Este método compara dois elementos segundo uma ordem definida por um critério.
    * @param o1 referência para o 1º elemento
    * @param o2 referência para o 2º elemento
    * @return valor negativo se a ordem de o1 for menor do que a ordem de o2
    *         valor positivo se a ordem de o1 for maior do que a ordem de o2
    *         zero se a ordem de o1 for igual à ordem de o2
    */
    int comparar(T o1, T o2);
}
```

Algoritmos de Ordenação

- Para os diversos algoritmos de ordenação, para além do tempo, vamos analisar o número de comparações e trocas

```
public class EstatisticaDeComparacoes extends Estatistica {
    public EstatisticaDeComparacoes(int tamanho) {
        super(tamanho, "Número de Comparações");
    }

    protected EstatisticaDeComparacoes(int tamanho, String... nomesContadores) {
        super(tamanho, nomesContadores);
    }

    public void incrementarComparacoes() {
        incrementarContador(0);
    }

    public long getComparacoes() {
        return getContador(0);
    }
}
```

Algoritmos de Ordenação

```
public class EstatisticaDeComparacoesETrocas extends EstatisticaDeComparacoes {
    public EstatisticaDeComparacoesETrocas(int tamanho) {
        super(tamanho, "Número de Comparações", "Número de Trocas");
    }

    public void incrementarTrocas() {
        incrementarContador(1);
    }

    public long getTrocas() {
        return getContador(1);
    }
}
```

Algoritmos de Ordenação

```
public class Vetor {
    public static <T> void apresentar(int limite, T... elementos) {
        String simbolosTerminais;
        int tamanho;

        if (elementos.length == 0) {
            System.out.println("[]");
            return;
        }

        if (elementos.length <= limite) {
            tamanho = elementos.length;
            simbolosTerminais = "]";
        } else {
            tamanho = limite;
            simbolosTerminais = "...";
        }

        System.out.print("[");
        for (int i = 0; i < tamanho - 1; i++) {
            System.out.print(elementos[i] + ", ");
        }
        System.out.println(elementos[tamanho - 1] + simbolosTerminais);
    }
}
```

Algoritmos de Ordenação

- O código comum entre os vários algoritmos de ordenação é definido pela classe AlgoritmoOrdenacao

```
public abstract class AlgoritmoOrdenacao<T> {
    protected final Comparacao<T> criterio;

    public AlgoritmoOrdenacao(Comparacao<T> criterio) {
        this.criterio = criterio;
    }

    public abstract void ordenar(EstatisticaDeComparacoesETrocas estatistica,
                                 T... elementos);
```

Algoritmos de Ordenação

```
    public EstatisticaDeComparacoesETrocas getEstatistica(T... elementos) {
        EstatisticaDeComparacoesETrocas estatistica =
                                new EstatisticaDeComparacoesETrocas(elementos.length);
        ordenar(estatistica, elementos);
        estatistica.parar();
        System.out.print("Sequência ordenada por " +
                                                    getClass().getSimpleName() + ": ");
        Vetor.apresentar(10, elementos);
        estatistica.apresentar();
        return estatistica;
    }

    protected void trocar(T[] elementos, int indice1, int indice2) {
        T aux = elementos[indice1];
        elementos[indice1] = elementos[indice2];
        elementos[indice2] = aux;
    }

}
```

Algoritmos de Ordenação
SelectionSort

- Descrição do funcionamento:
  - O algoritmo consiste em dividir uma sequência em duas partes:
    - a subsequência ordenada (parte esquerda, inicialmente vazia)
    - a subsequência por ordenar (parte direita)
  - Assim, em cada iteração, o algoritmo **seleciona** o elemento de menor ordem da subsequência por ordenar trocando-o, de seguida, com o 1º elemento dessa subsequência
  - O algoritmo termina quando a subsequência por ordenar tiver menos de 2 elementos
  - Daí chamar-se ordenação por seleção

Algoritmos de Ordenação
SelectionSort

```
public class SelectionSort<T> extends AlgoritmoDeOrdenacao<T> {
    public SelectionSort(Comparacao<T> criterio) {
        super(criterio);
    }
    public void ordenar(EstatisticaDeComparacoesETrocas estatistica,
                        T... elementos) {
        for (int i = 0; i < elementos.length - 1; i++) {
            int menor = i;
            for (int j = i + 1; j < elementos.length; j++) {
                estatistica.incrementarComparacoes();
                if (criterio.comparar(elementos[j], elementos[menor]) < 0) {
                    menor = j;
                }
            }
            if (menor != i) {
                estatistica.incrementarTrocas();
                trocar(elementos, i, menor);
            }
        }
    }
}
```

Algoritmo de ordem $O(N^2)$

Algoritmos de Ordenação

- Para ordenar uma sequência de inteiros por ordem crescente é necessário definir a seguinte enumeração:

```
public enum ComparacaoInteiros implements Comparacao<Integer> {
    CRITERIO;

    public int comparar(Integer o1, Integer o2) {
        return o1.compareTo(o2);
    }
}
```

Algoritmos de Ordenação
SelectionSort

```
public class MainTeoricaSelectionSort {

    public MainTeoricaSelectionSort() {
        SelectionSort<Integer> selectionSort =
                                new SelectionSort<>(ComparacaoInteiros.CRITERIO);
        selectionSort.getEstatistica(3, 7, 2, 5, 4, 1, 6, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaSelectionSort();
    }
}
```

Algoritmos de Ordenação
SelectionSort

jGRASP

Algoritmos de Ordenação
BubbleSort

- Descrição do funcionamento:
  - O algoritmo consiste em dividir uma sequência em duas partes:
    - a subsequência por ordenar (parte esquerda)
    - a subsequência ordenada (parte direita, inicialmente vazia)
  - Assim, em cada iteração, o algoritmo efetua trocas sucessivas de elementos adjacentes fora de ordem da subsequência por ordenar
  - Garantindo, assim, que, no final de cada iteração, o elemento de maior ordem fica mais à direita da subsequência por ordenar, ou seja, na posição correta
  - O algoritmo termina quando a subsequência por ordenar tiver menos de 2 elementos

Algoritmos de Ordenação
BubbleSort

```
public class BubbleSort<T> extends AlgoritmoDeOrdenacao<T> {

    public BubbleSort(Comparacao<T> criterio) {
        super(criterio);
    }

    public void ordenar(EstatisticaDeComparacoesETrocas estatistica,
                        T... elementos) {
        for (int indiceFim = elementos.length - 1; indiceFim > 0; indiceFim--) {
            for (int i = 1; i <= indiceFim; i++) {
                estatistica.incrementarComparacoes();
                if (criterio.comparar(elementos[i], elementos[i - 1]) < 0) {
                    trocar(elementos, i, i - 1);
                    estatistica.incrementarTrocas();
                }
            }
        }
    }
}
```

Algoritmo de ordem $O(N^2)$

Algoritmos de Ordenação
BubbleSort

```
public class MainTeoricaBubbleSort {

    public MainTeoricaBubbleSort() {
        BubbleSort<Integer> bubbleSort =
                                        new BubbleSort<>(ComparacaoInteiros.CRITERIO);
        bubbleSort.getEstatistica(3, 7, 2, 5, 4, 1, 6, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaBubbleSort();
    }
}
```

Algoritmos de Ordenação
BubbleSort

jGRASP

Algoritmos de Ordenação
BubbleSort Otimizado

- Descrição do funcionamento:
  - Analisando o algoritmo anterior, verifica-se que no final de cada iteração a sequência já se encontra ordenada do índice da última troca até ao fim
  - Desta forma, consegue-se otimizar o algoritmo anterior

Algoritmos de Ordenação
BubbleSort Otimizado

```
public class BubbleSortOtimizado<T> extends AlgoritmoDeOrdenacao<T> {

    public BubbleSortOtimizado(Comparacao<T> criterio) {
        super(criterio);
    }

    public void ordenar(EstatisticaDeComparacoesETrocas estatistica, T... elementos) {
        int indiceUltimaTroca = elementos.length;

        do {
            int indiceTroca = 0;
            for (int i = 1; i < indiceUltimaTroca; i++) {
                estatistica.incrementarComparacoes();
                if (criterio.comparar(elementos[i], elementos[i - 1]) < 0) {
                    trocar(elementos, i, i - 1);
                    estatistica.incrementarTrocas();
                    indiceTroca = i;
                }
            }
            indiceUltimaTroca = indiceTroca;
        } while (indiceUltimaTroca > 1);
    }
}
```

Algoritmo de ordem $O(N^2)$

Algoritmos de Ordenação
BubbleSort Otimizado

```
public class MainTeoricaBubbleSortOtimizado {

    public MainTeoricaBubbleSortOtimizado() {
        BubbleSortOtimizado<Integer> bubbleSortOtimizado =
                                new BubbleSortOtimizado<>(ComparacaoInteiros.CRITERIO);
        bubbleSortOtimizado.getEstatistica(3, 7, 2, 5, 4, 1, 6, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaBubbleSortOtimizado();
    }
}
```

Algoritmos de Ordenação
BubbleSort Otimizado

Algoritmos de Ordenação
QuickSort

- Descrição do funcionamento:
  - O algoritmo consiste em dividir, recursivamente, uma sequência ao meio garantindo em, cada chamada recursiva, que o elemento do meio (pivot) fica na posição correta (elemento pivot é de ordem maior ou igual do que todos os elementos à sua esquerda e de ordem menor ou igual do que todos os elementos à sua direita)
  - Assim, em cada chamada recursiva, sempre que encontre:
    - um elemento de ordem maior ou igual à do pivot na subsequência esquerda

      e
    - um elemento de ordem menor ou igual à do pivot na subsequência direita, efetua a troca desses dois elementos
  - O algoritmo termina quando a subsequência a ordenar tiver menos de 2 elementos

Algoritmos de Ordenação
QuickSort

```
public class QuickSort<T> extends AlgoritmoDeOrdenacao<T> {

    public QuickSort(Comparacao<T> criterio) {
        super(criterio);
    }

    public void ordenar(EstatisticaDeComparacoesETrocas estatistica,
                        T... elementos) {
        ordenarRecursivo(estatistica, 0, elementos.length - 1, elementos);
    }

    private void ordenarRecursivo(EstatisticaDeComparacoesETrocas estatistica,
                                  int esq, int dir, T... elementos) {
        int i = esq;
        int j = dir;
        int meio = (i + j) / 2;
        T pivot = elementos[meio];
```

Algoritmo de ordem
$O(N \log_2 N)$

Algoritmos de Ordenação
QuickSort

```
        do {
            estatistica.incrementarComparacoes();
            while (criterio.comparar(elementos[i], pivot) < 0) {
                estatistica.incrementarComparacoes();
                i++;
            }
            estatistica.incrementarComparacoes();
            while (criterio.comparar(elementos[j], pivot) > 0) {
                estatistica.incrementarComparacoes();
                j--;
            }
            if (i <= j) {
                estatistica.incrementarTrocas();
                trocar(elementos, i, j);
                i++;
                j--;
            }
        } while (i <= j);
```

Algoritmos de Ordenação
QuickSort

```
        if (esq < j) {
            ordenarRecursivo(estatistica, esq, j, elementos);
        }
        if (i < dir) {
            ordenarRecursivo(estatistica, i, dir, elementos);
        }
    }
}
```

Algoritmos de Ordenação
QuickSort

```
public class MainTeoricaQuickSort {

    public MainTeoricaQuickSort() {
        QuickSort<Integer> quickSort =
                                        new QuickSort<>(ComparacaoInteiros.CRITERIO);
        quickSort.getEstatistica(3, 7, 2, 5, 4, 1, 6, 8, 9);
    }

    public static void main(String[] args) {
        new MainTeoricaQuickSort();
    }
}
```

Algoritmos de Ordenação
QuickSort

Algoritmos de Ordenação

- Análise comparativa dos algoritmos anteriores

```
public class MainTeoricaComparacaoAlgoritmosOrdenacao {
    private static final int TAMANHO = 50;
    private static final int NUMERO_EXECUCOES = 20;

    public MainTeoricaComparacaoAlgoritmosOrdenacao() {
        VisualizadorEstatisticas v = new VisualizadorEstatisticas();
        v.adicionarEstatisticas("SelectionSort", getEstatisticas(
                                new SelectionSort<>(ComparacaoInteiros.CRITERIO)));
        v.adicionarEstatisticas("BubbleSortOtimizado", getEstatisticas(
                            new BubbleSortOtimizado<>(ComparacaoInteiros.CRITERIO)));
        v.adicionarEstatisticas("QuickSort", getEstatisticas(
                                    new QuickSort<>(ComparacaoInteiros.CRITERIO)));
        v.visualizar();
    }
```

Algoritmos de Ordenação

```
    public static void main(String[] args) {
        new MainTeoricaComparacaoAlgoritmosOrdenacao();
    }

    private List<Estatistica> getEstatisticas(
                                            AlgoritmoOrdenacao<Integer> algoritmo) {
        List<Estatistica> estatisticas = new ArrayList<>();
        for (int i = 1; i <= NUMERO_EXECUCOES; i++) {
            EstatisticaDeComparacoesETrocas estatistica =
                    algoritmo.getEstatistica(VetorDeInteiros.criarAleatorioInteger(
                                TAMANHO * i, -TAMANHO * 10 , TAMANHO * 10, false));
            estatisticas.add(estatistica);
        }
        return estatisticas;
    }
}
```

Algoritmos de Ordenação