Universidade Federal do Ceará - Campus de Quixadá

Programação para Design

Métodos de Comparação

Prof. Anderson Lemos

Introdução

- Ordenar coisas é uma tarefa comum no dia-a-dia.
- Ordenação segue algum critério (livro, muitas opções de critério).
  - Título
  - Ano
  - Tamanho
- Critérios de desempate.

Algoritmos de Ordenação

- Elementares
  - BubbleSort
  - InsertionSort
  - SelectionSort
- Eficientes
  - QuickSort
  - MergeSort
  - HeapSort

Ordenação

- Muitas linguagens oferecem algoritmos de ordenação prontos.
  - C
    - qsort
  - JAVA
    - Arrays.sort
    - Collections.sort
  - Python
    - sort
    - sorted
  - TypeScript
    - sort

Ordenação de números

- Ordenação de números é simples de imaginar (ordem crescente ou decrescente).
- BubbleSort

```
public bubbleSort(vetor : Array<number>) : void {
    let t : number = vetor.length;
    while(t > 1){
        for(let j=0;j<t-1;j++){
            if(vetor[j] > vetor[j+1]){
                let aux : number = vetor[j];
                vetor[j] = vetor[j+1];
                vetor[j+1] = aux;
            }
        }
        t--;
    }
}
```

Ordenação de números

- Ordenação de números é simples de imaginar (ordem crescente ou decrescente).
- BubbleSort: ordem crescente. O ***if*** verifica isso.

```
public bubbleSort(vetor : Array<number>) : void {
    let t : number = vetor.length;
    while(t > 1){
        for(let j=0;j<t-1;j++){
            if(vetor[j] > vetor[j+1]){
                let aux : number = vetor[j];
                vetor[j] = vetor[j+1];
                vetor[j+1] = aux;
            }
        }
        t--;
    }
}
```

Ordenação

- Mas e se não fossem números?
  - Se fosse um objeto como ***Livro***, por exemplo?
- Os operadores **>** e **<** não funcionariam.

```
class Livro{
    private titulo : string;
    private ano : number;
    private tamanho : number;

    //assuma construtores, gets e sets feitos

}
```

Ordenação

- Possível solução: fazer ***if*** para algum atributo.
- Mas e se houver critérios de desempate?
  - Fazer mais ***if's*** para mais atributos.

```
public bubbleSort(vetor : Array<Livro>) : void {
    let t : number = vetor.length;
    while(t > 1){
        for(let j=0;j<t-1;j++){
            if(vetor[j].getTitulo() > vetor[j+1].getTitulo() ||
                (vetor[j].getTitulo() == vetor[j+1].getTitulo() &&
                vetor[j].getAno() > vetor[j+1].getAno())){
                let aux : Livro = vetor[j];
                vetor[j] = vetor[j+1];
                vetor[j+1] = aux;
            }
        }
        t--;
    }
}
```

Ordenação

- Ok, o método de ordenação para Livro funciona.
- Mas se quisermos ordenar outro tipo de Objeto?
  - Problemas:
    - O método recebe Array de Livros.
    - Dentro do método a comparação é feita com atributos de Livro.
- Possível solução: fazer outro método para o outro Objeto.
  - Copiando e colando o método de Livro e mudando o que for necessário.
- Problema: e se quisermos, no nosso sistema, ordenar um milhão de tipos de objetos diferentes?
  - Inviável copiar e colar o método tantas vezes. Retrabalho.
- E as linguagens que já tem métodos padrões, como eles agem sem saber quais objetos estão ordenando?

Interface de Comparação

- Solução: criar uma interface que serve como “intermediário” entre o método de ordenação e o objeto.
- Assim, os métodos de ordenação prontos, exigem que os objetos a serem ordenados implementem essa interface.
- O método de ordenação usa os métodos obrigatórios da interface para saber a ordem dos objetos.

Interface de Comparação

- Principais métodos das interfaces de comparação:
  - ***compareTo***
  - ***equals***
  - ***hashCode***

Principais métodos de comparação:

- ***compareTo***
  - Recebe um objeto do mesmo tipo e compara o objeto passado como parâmetro com o objeto que chama o método.
  - Retorna um número negativo se o objeto que chamou o método é “menor”, um número positivo se o objeto que chamou o método é “maior” e retorna zero se os dois objetos são “iguais”.

Principais métodos de comparação:

- ***equals***
  - Recebe um objeto do mesmo tipo e compara o objeto passado como parâmetro com o objeto que chama o método.
  - Retorna ***true*** se os objetos são “iguais” e retorna um ***false*** se são “diferentes”.

Principais métodos de comparação:

- ***hashCode***
  - Apenas retorna o código *hash* do objeto que chama o método.
    - Ou seja, é a função *hash* do objeto.
  - Uma função *hash* converte um valor de tamanho variável em um valor de tamanho fixo.
    - Um mesmo valor de entrada sempre gera o mesmo código *hash*.
    - Uma vez transformado em código *hash*, é difícil voltar ao valor original.
    - Utilizado em senhas e garantia de integridade.
  - Algumas estruturas de dados usam códigos *hash* para melhorar sua eficiência.
    - HashMap
    - HashSet

Interface de Comparação

- A linguagem C, não tem interface, assim usa-se ponteiros genéricos e ponteiros de funções.
- A linguagem JAVA, utiliza a interface ***Comparable*** que obriga o objeto a implementar o método ***compareTo***, que compara dois objetos.
  - Todo objeto em JAVA herda da classe ***Object***, que por sua vez, tem os métodos ***equals*** e ***hashCode***, que também são usados em comparações.
- Esses três métodos são os mais usados para comparação.
- TypeScript, por padrão, não tem esses métodos ou essa interface.
  - Assim, faremos a nossa implementação da interface.

Interface de Comparação

- A interface recebe como “parâmetro” o tipo a qual se quer comparar.
  - Geralmente passamos o mesmo tipo do objeto que a implementa.
- Implementação da interface ***Comparable***:

```
interface Comparable<T> {
    compareTo(t : T) : number;
    equals(t : T) : boolean;
    hashCode(t : T) : number;
}
```

Interface de Comparação

- Fazemos Livro implementar a interface.
  - Dizendo que compararemos Livro com outro Livro.

```
class Livro implements Comparable<Livro>{
    private titulo : string;
    private ano : number;
    private tamanho : number;

    //assuma construtores, gets e sets feitos

    compareTo(t : Livro) : number {
        if(this.getTitulo() > t.getTitulo()) return 1;
        else if (this.getTitulo() < t.getTitulo()) return -1;
        else if (this.getAno() > t.getAno()) return 1;
        else if (this.getAno() < t.getAno()) return -1;
        return 0;
    }

    // continuação de livro
    public equals(t : Livro) : boolean {
        return (this.getTitulo()==t.getTitulo()
            && this.getAno()==t.getAno());
    }

    public hashCode(t : Livro) : number{
        return (this.ano**3)%1000;
    }
}
```

Interface de Comparação

- Agora fazemos nosso método de ordenação genérico, que recebe um Array de ***Comparable*** e usa o método ***compareTo***:

```
public bubbleSort<T extends Comparable<T>>(vetor : Array<T>) : void {
    let t : number = vetor.length;
    while(t > 1){
        for(let j=0;j<t-1;j++){
            if(vetor[j].compareTo(vetor[j+1]) > 0){
                let aux : T = vetor[j];
                vetor[j] = vetor[j+1];
                vetor[j+1] = aux;
            }
        }
        t--;
    }
}
```

Interface de Comparação

- Repare que agora o método de comparação não precisa ser alterado, pois serve para qualquer objeto que implemente a interface ***Comparable***.
  - Na classe que se quer ordenar, basta implementar a interface e usar o mesmo método.
  - No método sort do TypeScript, é possível passar uma função de comparação como parâmetro, mas a interface nos dá mais flexibilidade.
- Note também que podemos usar o método ***equals*** para comparar dois objetos mais facilmente.
- Também podemos utilizar nossa própria implementação de ***Map***, utilizando o ***hashCode***.

Perguntas?