CAPÍTULO 3

Armazenamento Sequencial

"A morte do homem começa no instante em que ele desiste de aprender"
— Albino Teixeira

3.1 - MOTIVAÇÃO

Todo mundo já experimentou sopa de letrinhas quando criança. Aliás, quando as crianças tomam sopa de letrinhas, elas ficam muito mais interessadas em formar as palavras do que em tomar a sopa.

O que chama mais a atenção é que nesta brincadeira de criança aparece um conceito bem interessante de estrutura de dados. Quando a sopa é servida, as letras estão todas espalhadas sem ordem alguma e sem nenhum significado. Quando você escolhe um grupo de letras e as coloca em seqüência formando uma palavra, este grupo de letras ganha um valor semântico, ou seja, ganha um significado.

O grupo de letrinhas que você escolhe para formar uma palavra pode conter letras repetidas sem problema nenhum. A única regra é que as letras postas em seqüência devem formar uma palavra existente em alguma língua.

Figura 3.1: Sopa de Letrinhas

As músicas também são exemplos em que a definição de uma seqüência dos elementos agrega valor semântico. Uma música é composta de notas musicais. Quando estas notas estão "espalhadas", elas não significam muita coisa. Já quando colocadas em uma seqüência adequada, formam uma música que as pessoas podem apreciar. Além disso, uma mesma nota musical pode aparecer mais de uma vez em uma única música.

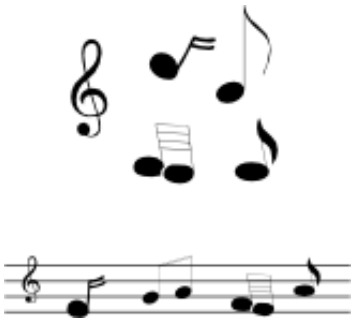


Figura 3.2: Música

Outro exemplo em que a seqüência dos elementos é importante são as rotas de avião. Imagine o nome de várias cidades sem nenhuma ordem. Desta maneira, estes nomes não significam muita coisa. Mas, se eles forem colocados em alguma seqüência então poderiam formar a rota de uma aeronave.

Figura 3.3: Rota de avião

3.2 – O PROBLEMA DA LISTAGEM DE ALUNOS

Uma certa instituição de ensino está incentivando os seus alunos a participarem de eventos acadêmicos em troca de créditos para obter desconto nas mensalidades.

Para participar, o aluno deve comparecer em algum dos eventos cadastrados na

instituição e depois escrever um relatório sobre o conteúdo apresentado no evento. Este relatório será avaliado por um professor e receberá uma pontuação de `0` a `100`.

A instituição quer manter uma listagem dos alunos que entregaram relatórios. Cada relatório entregue por um aluno corresponde a uma entrada na lista. Então, os alunos que entregarem mais de um relatório irão aparecer mais de uma vez na listagem.

Por exemplo, suponha que entregaram relatórios os alunos Rafael, Paulo e Ana. Rafael entregou apenas um relatório; Paulo entregou dois relatórios; e Ana entregou três relatórios. Uma possível listagem neste caso seria assim:

1. Rafael
2. Paulo
3. Ana
4. Paulo
5. Ana
6. Ana

A listagem também deve manter a ordem de pontuação obtidas pelos relatórios dos alunos.

Por exemplo, suponha que o Rafael teve pontuação máxima(100) no seu relatório; O Paulo teve pontuação 70 em um relatório e 50 no outro; Ana teve pontuação 60, 40 e 40 nos seus relatórios. Então, a listagem ficaria assim:

1. Rafael (100)
2. Paulo (70)
3. Ana (60)
4. Paulo (50)
5. Ana (40)
6. Ana (40)

Conforme os alunos forem entregando os relatórios, novas entradas serão adicionadas na listagem. Uma nova entrada pode ser inserida em qualquer

posição. A posição é definida de acordo com a pontuação do relatório do aluno.

Por exemplo, suponha que o Rafael entregou mais um relatório com pontuação 65. A listagem deveria ser atualizada para:

1. Rafael (100)
2. Paulo (70)
3. Rafael (65)
4. Ana (60)
5. Paulo (50)
6. Ana (40)
7. Ana (40)

Para gastar os créditos obtidos com os relatórios, o aluno deve pedir para retirar uma das suas entradas na listagem. Os créditos são proporcionais a colocação da entrada do aluno na listagem.

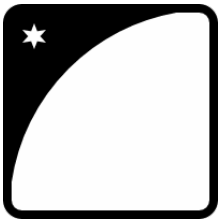
Por exemplo, se o Paulo quiser gastar uma das suas duas entradas na lista ele deve escolher entre a de 70 pontos a 50 pontos. A de 70 é a segunda da lista e a de 50 é a quinta. Suponha que ele escolha a de 50 pontos então a nova listagem ficaria assim:

1. Rafael (100)
2. Paulo (70)
3. Rafael (65)
4. Ana (60)
5. Ana (40)
6. Ana (40)

Quando o aluno quiser usar seus créditos ele deve verificar antes se ele tem entradas na listagem. Para isso, ele deve ir na secretaria da instituição.

A instituição pode querer saber qual é o aluno que está na primeira posição da listagem ou o que está na última. Na verdade, seria interessante para a instituição poder facilmente saber qual é o aluno que está em qualquer posição que ela queira.

Você pode também fazer o curso CS-14 dessa apostila na Caelum



Querendo aprender ainda mais sobre estrutura de dados? Esclarecer dúvidas dos exercícios? Ouvir explicações detalhadas com um instrutor?

A Caelum oferece o **curso CS-14** presencial nas cidades de São Paulo, Rio de Janeiro e Brasília, além de turmas incompany.

Consulte as vantagens do curso *Algoritmos e Estruturas de Dados com Java*.

3.3 - Listas

Nesta seção, vamos definir uma estrutura de dados para resolver o problema da listagem de alunos. Chamaremos esta estrutura de **Lista**.

Vimos que uma estrutura de dados deve definir duas coisas:

1. A maneira como a informação será armazenada.
2. A interface de uso com o usuário.

Nos dois próximos capítulos, veremos as principais implementações de Listas. Cada implementação tem uma maneira particular de armazenar os dados, que trará vantagens e desvantagens em determinados casos, em termos de uso de memória e tempo consumido para cada operação. Cabe a você conhecer esses casos e saber fazer a melhor escolha conforme o problema enfrentado.

Neste capítulo, definiremos apenas a interface de uso que a Lista deve fornecer ao usuário.

Com certeza, vamos querer adicionar elementos então precisamos de algumas operações de adição. Talvez duas operações já sejam suficientes, uma que permita adicionar um elemento no fim da Lista e outra que deixe o usuário escolher a posição onde ele quer adicionar um elemento.

Também, precisaremos recuperar elementos então vamos definir uma operação que dado uma posição da Lista ela devolve o elemento que está lá.

Outro tipo de operação necessária é o de remover elementos. Esta operação

deve receber a posição do elemento que deve ser removido.

Seria interessante que a Lista tivesse também uma operação para verificar se um dado elemento está contido na Lista.

Por fim, uma operação que será bastante útil é uma que informe a quantidade de elementos da Lista.

Uma vez definidas todas as operações temos então a interface de uso da nossa estrutura de dados.

Interface da Lista:

1. Adiciona um dado elemento no fim da Lista.
2. Adiciona um dado elemento em um dada posição.
3. Pega o elemento de uma dada posição.
4. Remove o elemento de uma dada posição.
5. Verifica se um dado elemento está contido na Lista.
6. Informa a quantidade de elementos da Lista.

Um fato bem interessante que ocorre quando programamos pensando primeiro na interface, como estamos fazendo aqui, é que após a definição da interface já sabemos como usar a estrutura que ainda nem implementamos.

Se sabemos como usar a estrutura já sabemos como testá-la. Estes testes poderão ser executados durante o desenvolvimento e não somente no fim. Isso é interessante pois possibilita a eliminar erros mais rápido, logo que eles aparecem, e pode evitar erros em cascata (erros que são causados por outros erros).

3.4 - Modelagem

Queremos desenvolver um sistema para resolver o problema da listagem de alunos. Este sistema deve ser orientado a objetos e deve de alguma forma representar os alunos. Em um sistema orientado a OBJETOS, um aluno será representado por um **objeto**.

Para criar objetos, precisamos definir como ele deve ser e o que ele deve fazer, ou seja, devemos criar uma "receita de construir objetos". Em termos técnicos

esta "receita" é uma **Classe**.

```
public class Aluno {

  private String nome;

  public String getNome() {
    return nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }

  public String toString() {
    return this.nome;
  }

  public boolean equals(Object o) {
    Aluno outro = (Aluno)o;
    return this.nome.equals(outro.nome);
  }
}
```

Com a classe **Aluno**, o sistema é capaz de criar objetos para representar os alunos da instituição. Teremos apenas alguns poucos atributos nessa classe, e alguns pares de getters e setters. Vale lembrar que não é boa prática ter classes que apenas carregam dados: seria mais interessante que **Aluno** também tivesse métodos de negócio.

Perceba que rescrevemos os métodos `toString()` e `equals(Object)`. O primeiro será útil para imprimir os alunos na tela. O segundo servirá para comparar dois objetos do tipo `Aluno`, o critério de comparação será os nomes dos alunos.

Devemos tomar cuidado no método `equals(Object)` pois estamos correndo o risco de dois tipos de erro. O primeiro acontece quando a referência recebida no parâmetro não está apontando para um objeto do tipo `Aluno`. O segundo ocorre quando ou a referência do parâmetro é **null** ou o atributo `nome` está **null**.

3.5 – Exercícios: Armazenamento

1. Crie um projeto no eclipse chamado **ed**. Não esqueça de selecionar a opção que separa o código fonte do binário.

2. Crie um pacote no projeto **ed** com o nome **br.com.caelum.ed**.

3. Faça a classe `Aluno` no pacote **br.com.caelum.ed** para poder criar objetos que

representarão os alunos.

Tire suas dúvidas no novo GUJ Respostas

O GUJ é um dos principais fóruns brasileiros de computação e o maior em português sobre Java. A nova versão do GUJ é baseada em uma ferramenta de *perguntas e respostas* (QA) e tem uma comunidade muito forte. São mais de 150 mil usuários pra ajudar você a esclarecer suas dúvidas.

Faça sua pergunta.

CAPÍTULO ANTERIOR:

Introdução

PRÓXIMO CAPÍTULO:

Vetores

Facebook

Newsletter

Casa do Código

Twitter