Aula 6 - Listas e Genéricos

Docupedia Export

Author:Silio Leonardo (SO/OPM-TS21-BR)
Date:06-May-2024 16:37

Table of Contents

- Listas
- Genéricos
- Exercícios Propostos

1 Listas

Talvez você tenha percebido no desafio 6 a dificuldade de trabalhar com uma quantidade arbitrária de dados. Usamos um vetor de tamanho fixo e gerenciamos o seu uso. Com a ajuda da OO isso pode ficar mais fácil, pois podemos implementar estruturas que façam isso para a gente e usar seus objetos. Dito isso, vamos implementar agora uma matriz dinâmica que é, basicamente, um vetor que cresce todas as vezes que você precisa de um vetor maior. Diferente da lista encadeada vista do Exemplo 1, está lista é mais rápida ao se acessar um valor qualquer (pois estamos falando de um vetor, afinal de contas), mas para muitas adições pode ter desempenho comprometido. Observe a implementação com cuidado:

```
1  class ListInt
2  {
3      private int pos = 0;
4      private int[] vetor = new int[10];
5
6      void add(int value)
7      {
8          int len = vetor.length;
9          // Vetor estouraria, vamos aumentá-lo
10         if (pos == len)
11         {
12             // Criamos um vetor maior
13             int[] newVetor = new int[2 * len];
14
15             // Copiamos
16             for (int i = 0; i < pos; i++)
17                 newVetor[i] = vetor[i];
18
19             // Jogamos a referência antiga fora e agora o ponteiro 'vetor' irá apontar para um novo vetor
20             vetor = newVetor;
21         }
22
23         vetor[pos] = value;
24         pos++;
25     }
26
27     int size() {
28         return pos;
29     }
30
31     void set(int i, int value) {
```

```
32          this.vetor[i] = value;
33      }
34
35      int get(int i) {
36          return this.vetor[i];
37      }
38  }
```

2 Genéricos

Um problema do código acima é claro: Só funciona para int. Isso é um grande problema, pois para cada lista que quisermos fazer precisamos de uma nova implementação. Mas como é de se esperar, o C# tem uma solução a este imenso problema: Os genéricos. Bascimente, podemos criar um parâmetro que recebe um tipo e varia suas funcionalidade a partir disso. Observe o exemplo simplificado:

```
1  public class Main
2  {
3      public static void main(String[] args)
4      {
5          Caixa<String> caixa = new Caixa<String>("xispita");
6          System.out.println(caixa.abrir());
7      }
8  }
9
10 // Caixa.java
11 class Caixa<T> // Parâmetro Genérico
12 {
13     Caixa(T valor) {
14         this.conteudo = valor;
15     }
16
17     T conteudo;
18     T abrir() {
19         T valor = this.conteudo;
20         this.conteudo = null;
21         return valor;
22     }
23 }
24
25 // Podemos aplicar o mesmo principio a nossa lista:
26
27 // List.java
28 class List<T>
29 {
30     private int pos = 0;
31
32     @SuppressWarnings("unchecked")
33     private T[] vetor = (T[])new Object[10];
```

```
34
35      void add(T value)
36      {
37          int len = vetor.length;
38          // Vetor estouraria, vamos aumentá-lo
39          if (pos == len)
40          {
41              // Criamos um vetor maior
42              @SuppressWarnings("unchecked")
43              T[] newVetor = (T[]) new Object[2 * len];
44
45              // Copiamos
46              for (int i = 0; i < pos; i++)
47                  newVetor[i] = vetor[i];
48
49              // Jogamos a referência antiga fora e agora o ponteiro 'vetor' irá apontar para um novo vetor
50              vetor = newVetor;
51          }
52
53          vetor[pos] = value;
54          pos++;
55      }
56
57      int size() {
58          return pos;
59      }
60
61      void set(int i, T value) {
62          this.vetor[i] = value;
63      }
64
65      T get(int i) {
66          return this.vetor[i];
67      }
68  }
```

3 Exercícios Propostos

1. Agora é sua vez. Use a nossa classe List genérica como base para implementar a classe Stack (sim, que nem a estrutura do Java). Stack é uma pilha e ela funciona desta maneira, como uma pilha de roupas. Você coloca e tira coisas apenas do topo, nunca debaixo. Ela possui os seguintes componentes:
    a. push: Adiciona um valor ao topo da pilha
    b. pop: Remove e retorna o valor no topo da pilha
    c. peek: Retorna, sem remover, o valor no topo da pilha
    d. size: Tamanho da pilha
2. Depois disso implementaremos a queue - uma fila. Na fila, entram coisas de um lado e se retiram do outro:
    a. enqueue: Adiciona um valor no início da fila
    b. dequeue: Remove e retorna o valor no final da fila
    c. peek: Retorna, sem remover, o valor no final da fila
    d. size: Tamanho da fila
3. Importante: Para implementa ruma stack nós podemos usar a mesma implementação do array dinâmico, mas para queue é melhor uma lista ligada.