Lista – Exercícios sobre genéricos no Java

Prof. Tiago Moraes

Projeto ArrayUtil: criando **métodos genéricos**. Crie a classe ArrayUtil com os seguintes métodos estáticos que operam em cima de um vetor genérico.

1) count(vetor, elemento): conta o número de ocorrências de elemento em vetor e retorna (int). Perceba que a forma genérica de comparar se dois objetos são iguais é o "equals" e não o "==" e como o vetor é genérico, poderá ser de Integer mas poderá ser por exemplo de "String". Faça os devidos testes para Integers e Strings.
2) max(vetor): retorna o maior elemento de um vetor. Perceba que nesse caso, para que o algoritmo seja genérico, deverá utilizar o método compareTo da interface Comparable. Caso o vetor seja nulo ou vazio lance uma IllegalArgumentException com a mensagem: "Array vazio ou null". Faça os devidos testes para Integers e Strings.
3) min(vetor): equivalente ao exercício 2 mas retorna o mínimo valor.

Projeto Lists: criando **classes genéricas**. Complemente a classe ListaEncadeada criada em aula e crie a ListaVetor genéricas.

4) Classe ListaEncadeada:
   i. Crie o método add(valor, int pos): adiciona um valor (tipo genérico) na posição, considere as posições de 0 a tamanho-1. Caso se passe um valor fora do intervalo lance uma IllegalArgumentException com uma mensagem. Bônus: faça com que o método add(valor) que adiciona no final, chame esse método para evitar lógicas duplicadas.
   ii. Crie o método remove(int pos): remove e retorna um elemento na posição passada por parâmetro (não esqueça de decrementar o tamanho). Caso o parâmetro seja fora do intervalo de posições lance uma IllegalArgumentException.
   iii. Crie o método remove(valor): que remove o valor (tipo genérico) passado por parâmetro. Retorne um boolean que informa se removeu algum elemento ou não da lista.
5) Classe ListaVetor: lista implementada com um vetor internamente. Na definição do vetor você pode escolher uma tamanho grande o suficiente para os seus testes, ou implementá-la variando o tamanho do vetor na medida que a lista cresca (opção mais difícil, elaborada e próxima do que é implementado no ArrayList do Java). Para a criação do vetor, você vai se deparar com uma limitação dos genéricos, a criação de Objetos genéricos (não se pode instanciar um tipo genérico, por exemplo: " `T varGen = new T();` "). Para isso você vai precisar criar um vetor de Object e usar o casting, por exemplo:

```
T[] this.itens = (T[]) new Object[10];
```

Esse tipo de instrução, necessária no mínimo no seu construtor vai gerar um Alert: "Type safety: Unchecked cast", isso significa que o o compilador não pode garantir o "Type Safety" pois ele não sabe dizer se o Cast irá lançar um *exception* ou não. Porém você, o programador pode dizer... Se o elemento (this.itens) estiver encapsulado, e o programador puder garantir que aquele casting sempre funcionará (o que é o caso), podemos informar isso com uma *annotation*: `@SuppressWarnings("unchecked")`, na linha de cima do *casting*.

Crie os seguintes métodos, equivalentes aos implementados na ListaEncadeada:

i. add(valor, int pos): adiciona elemento na posição
ii. add(valor): adiciona no final da lista
iii. get(int pos): retorna elemento na posição
iv. remove(int): remove e retorna elemento na posição passada por parâmetro
v. remove(valor): remove o elemento passado por parâmetro
vi. size() retorna o tamanho da lista

Criando interfaces genéricas:

6) Perceba que a interface pública das classes (seus métodos) dos exercícios 4 e 5 são basicamente as mesmas, com isso podemos extrair uma interface. Tente fazer isso e crie a interface ***Lista***. E faça com que as classes ListaEncadeada e ListaVetor implementem essa interface.

Usando coringas (wildcards):

7) Crie uma classe ListaUtil com métodos max, min e count assim como feito para nos exercícios 1, 2 e 3 só que para as listas (dos exercícios 4, 5 e 6) ao invés de arrays. Perceba que é necessário passar para os métodos estáticos como parâmetro uma lista o mais genérica possível:
   i. Pense em utilizar a interface ao invés das classes para poder usar para os dois tipos de listas.
   ii. Pense que a lista pode ser parametrizada por qualquer tipo: Integer, String, etc. Para isso, pense na utilização dos wildcards.
   iii. Pense que para os métodos min() e max() é interessante limitar o quão genérico o wildcard é para utilizar o compareTo(). Para isso, pense na utilização de wildcards limitados.