PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e Informática
Implementação de uma Agenda de Contatos

Curso : *Engenharia de Software*
Disciplina : *Algoritmos e Estruturas de Dados II*
Professora : *Eveline Alonso Veloso*

Exercício:

1. Agenda de contatos

Você foi contratado para desenvolver uma agenda de contatos para um escritório de advocacia. Um colega sugeriu implementar esta agenda de contatos por meio de uma tabela *hash* que trata colisões por meio de árvores AVL. Você deseja implementar uma solução que seja facilmente estendida para outros domínios. Assim, devem ser criadas as seguintes classes e seus respectivos métodos:

Classe *TabelaHash<K extends Comparable<K>, V>*, que implementa uma *tabela hash* que trata possíveis colisões por meio de árvores AVL.
Nesta classe devem ser implementados os seguintes métodos, que devem operar conforme descrito a seguir, respeitando-se parâmetros e tipos de retorno:

- *public* ***TabelaHash****(int capacidade)*: construtor da tabela *hash* de tamanho *capacidade*.
- *private int* ***funcaoHash****(K chave)*: função aritmética de transformação da tabela *hash*. Esse método deve retornar o valor absoluto do resto da divisão inteira de *hashCode* de *chave* pela capacidade da tabela *hash*.
- *public int* ***inserir****(K chave, V item)*: insere o par chave-valor *chave-item* na tabela *hash*.
- *public V* ***pesquisar****(K chave)*: localiza e retorna o valor associado à *chave*.
- *public void* ***imprimirChavesEmOrdem****()*: método responsável por imprimir todos os itens armazenados na tabela *hash* em ordem crescente. Para isso, esse método deve criar uma lista duplamente encadeada com todos os pares chave-valor presentes na tabela *hash*, ordenar essa lista e imprimir seu conteúdo.

Classe *AVL<K extends Comparable<K>, V>*, que implementa uma árvore AVL de pares chave-valor.
Nesta classe devem ser implementados os seguintes métodos, que devem operar conforme descrito a seguir, respeitando-se parâmetros e tipos de retorno:

- *public* ***AVL****()*: construtor da classe que cria uma árvore AVL vazia.
- *public Boolean* ***vazia****()*: método que verifica e retorna se a árvore AVL está vazia.
- *public V* ***pesquisar****(K procurado)*: pesquisa e retorna o valor associado à chave *procurado*.

- *public void* ***adicionar****(K chave, V item)*: insere o par chave-valor *chave-item* na árvore AVL.
- *public ListaDuplamenteEncadeada<K, V>* ***obterTodosElementos****()*: retorna uma lista duplamente encadeada com todos os pares chave-valor presentes na árvore AVL.

Devem ser implementados também todos os métodos privados comuns às árvores AVL, inclusive os necessários para a implementação dos métodos acima e para o balanceamento da árvore. Adicionalmente, a **classe *No<K extends Comparable<K>, V>*** também deve ser implementada.

Classe *ListaDuplamenteEncadeada<K extends Comparable<K>, V>*, que implementa uma lista duplamente encadeada de pares chave-valor.
Nesta classe devem ser implementados os seguintes métodos, que devem operar conforme descrito a seguir, respeitando-se parâmetros e tipos de retorno:

- *public* ***ListaDuplamenteEncadeada****()*: construtor da classe que cria uma lista duplamente encadeada vazia. Considere nessa implementação que a lista duplamente encadeada apresenta a célula sentinela.
- *public Boolean* ***vazia****()*: método que verifica e retorna se a lista duplamente encadeada está vazia.
- *public void* ***inserirFinal****(K chave, V novo)*: insere o par chave-valor *chave-novo* no final da lista duplamente encadeada.
- *public void* ***ordenar****()*: ordena os registros armazenados na lista duplamente encadeada em ordem crescente. Deve ser implementado o método de ordenação *quicksort*.
- *public void* ***concatenar****(ListaDuplamenteEncadeada<K, V> outraLista)*: método responsável por concatenar, ao final da lista original, a *outraLista* passada como parâmetro para esse método.
- *public void* ***imprimir****()*: método responsável por imprimir todos os itens presentes na lista duplamente encadeada.

Devem ser implementados também todos os métodos privados necessários para a implementação do método ***ordenar***. Adicionalmente, a **classe *Celula<K extends Comparable<K>, V>*** também deve ser implementada.

Todas as classes indicadas acima devem ser implementadas de **forma genérica**, de modo que possam ser empregadas em outras aplicações e domínios. Além disso, o princípio de **encapsulamento** deve ser observado mantendo-se todos os seus atributos privados.

Adicionalmente, implemente uma **classe *Contato***, que deve armazenar os dados de um contato. Seus atributos devem ser mantidos privados, sendo acessados apenas por meio de métodos públicos.

Seu programa deve ler, da entrada padrão, uma agenda de contatos, realizar algumas pesquisas e, por fim, imprimir todo o conteúdo ordenado da agenda.

A **entrada padrão** é dividida em **duas partes**. A **primeira** contém, em cada linha, uma *string* indicando os **dados de um contato** separados por **vírgula**. Esses contatos devem ser inseridos na tabela *hash*, na ordem em que são apresentados. Após a palavra FIM, inicia-se a segunda parte da entrada padrão.
A primeira linha dessa **segunda parte da entrada padrão** apresenta um número inteiro *n* indicando a **quantidade de pesquisas** que serão em seguida realizadas. Nas próximas *n* linhas, tem-se *n* **nomes** que devem ser pesquisados neste exercício.

A **saída padrão** deve apresentar uma linha para **cada contato pesquisado**.
Em seguida, teremos, ainda na saída padrão, os dados relativos aos **contatos armazenados na tabela *hash* em ordem crescente** (observe o formato de cada linha da saída esperada).