Sistemas Operacionais I FATEC ZL
Prof. M.Sc. Leandro Colevati dos Santos

Exercício de Sistemas de Arquivos

Diversas estruturas de dados são utilizadas nas funcionalidades dos sistemas operacionais, como Pilhas, Listas, Árvores, Filas, etc. Maiores informações serão dadas abaixo.
Pensando nisso, a atividade consiste em pegar o arquivo csv em anexo e criar um projeto Java para manipulá-lo. O csv em questão apresenta todas as características de todos os jogadores disponíveis no jogo FIFA ‘19 da Eletronic Arts. O arquivo está com jogadores ordenados por seu Overall, do maior para o menor, portanto, o primeiro jogador da lista é o Leonel Messi, Overall 94 e o último é G. Nugent, com Overall 46.
O arquivo data.csv deve ser colocado na pasta C:\TEMP
O projeto deverá conter :

controller
- IFifaController (Interface)
- FifaController (Classe que implementa a interface IFifaController)

view
- Principal (Classe com main)

A interface deve conter as seguintes operações:

```
public Stack<String> empilhaBrasileiros(String caminho, String nome) throws IOException
public void desmpilhaBonsBrasileiros(Stack<String> pilha) throws IOException
public List<String> listaRevelacoes(String caminho, String nome) throws IOException
public void buscaListaBonsJovens(List<String> lista) throws IOException
```

O método empilhaBrasileiros recebe o caminho e o nome do arquivo, deverá inicializar uma pilha, abrir o arquivo, ler o arquivo, verificar se na coluna referente à nacionalidade existe o valor “Brazil” e empilhar (push) a linha inteira, apenas de jogadores brasileiros. O método deve retornar essa pilha.
O método desempilhaBonsBrasileiros deve receber uma pilha de Strings como parâmetro, percorrer a pilha, desempilhar (pop) e imprimir (somente nome e Overall) apenas de jogadores com Overall acima de 80. Perceba que, como estão ordenados, no arquivo, do melhor para o pior, os jogadores serão empilhados do melhor para o pior, portanto, o primeiro impresso deverá ser o pior dentre os escolhidos e o último deverá ser Neymar Jr, com Overall 92.
O método listaRevelacoes recebe o caminho e o nome do arquivo, deverá inicializar uma lista encadeada, abrir o arquivo, ler o arquivo, verificar se na coluna referente à idade o valor é menor ou igual a 20, e adicionar (add) à lista, a linha inteira, apenas de jogadores jovens. O método deve retornar essa lista.
O método buscaListaBonsJovens deve receber uma lista de Strings como parâmetro, percorrer a lista do último para o primeiro, imprimir (somente nome, idade e Overall) apenas de jogadores com Overall acima de 80 e 20 anos ou menos.

* Considere que, para separar cada elemento de cada linha, deve se usar o método split;
** Considere que a primeira linha do arquivo deve ser ignorada

Sobre Pilha:

A pilha é uma das estruturas de dados e trabalha com o formato LIFO (o último a entrar é o primeiro a sair, “Last In, First Out”, em inglês). Lembre-se da pilha como uma pilha de livros, em que o primeiro livro que foi inserido na pilha, normalmente é o último que sai dela, enquanto o último adicionado é o primeiro a ser retirado.
A estrutura da pilha, segundo Farias “são estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último”.
Apesar de não ser a melhor abordagem pois é muito generalista, o Java já tem uma pilha genérica implementada com as principais operações (PUSH, POP, SIZE), o exemplo abaixo considera um vetor de Strings, empilha os elementos em uma pilha, e, um a um, enquanto houver elementos na pilha, desempilha os valores.

```
	public void operaPilha(){
		String[] vetString = {"Maçã", "Banana", "Pera", "Uva", "Goiaba"};

		Stack<String> pilha = new Stack<String>();

		//empilha
		for (String s : vetString){
			pilha.push(s);
		}

		//verifica tamanho da pilha
		int tamanhoPilha = pilha.size();

		//desempilha
		for (int i = 0 ; i < tamanhoPilha ; i++){
			String fruta = pilha.pop();
			System.out.println(fruta);
		}
	}
```

A saída é:

```
Goiaba
Uva
Pera
Banana
Maçã
```

Sobre Listas Encadeadas:

Diferente das listas sequenciais, os dados de uma lista encadeada estão espalhados na memória, isto é, os elementos não estão juntos em grandes blocos de memória, mas em lugares diferentes dela. Isso faz com que as operações de inserção e remoção sejam feitas com mais facilidade, tanto no início como no fim da lista. Vejamos o porquê.

Num vetor, os dados ficam armazenados na memória sequencialmente; dessa forma, pra descobrirmos a posição de qualquer elemento, só precisamos saber onde está o primeiro e saber qual o tamanho que cada elemento ocupa. Entretanto, nas LinkedLists cada objeto está espalhado na memória. Como acha-los?

A solução é simples: nas listas encadeadas, cada elemento guarda a posição do próximo, ou seja, se eu quiser achar o quinto elemento, preciso do quarto; para achar o quarto, preciso do terceiro. E assim sucessivamente até percebermos que precisamos percorrer toda a lista se quisermos chegar num elemento qualquer. Com isso, já achamos um ponto negativo das LinkedLists. Mas será tão negativo assim? Isso varia muito da utilização, pois geralmente ocorre o acesso à **lista inteira**, e não a um elemento só.

A **LinkedList** do Java, especificamente, implementa uma lista circular duplamente encadeada: cada elemento guarda a posição do próximo e do anterior, formando um círculo, onde o elemento depois do último é o primeiro, e o anterior a este, é aquele.

O exemplo abaixo apresenta dados de um vetor sendo armazenados em um LinkedList, que no Java é um implementação da interface List(java.util), a lista sendo percorrida do primeiro para o último elemento, a verificação do tamanho da lista e a lista sendo percorrida do último para o primeiro elemento.

```
public void operaLista(){
	String[] vetString = {"Maçã", "Banana", "Pera", "Uva", "Goiaba"};

	List<String> lista = new LinkedList<String>();

	//adiciona ordenamente na lista
	for (String s : vetString){
		lista.add(s);
	}

	//percorre a lista na do primeiro para o último elemento
	Iterator<String> it = lista.iterator();
	while (it.hasNext()){
		String fruta = it.next();
		System.out.println(fruta);
	}

	//verifica tamanho da lista
	int tamanhoLista = lista.size();

	//percorre a lista na do último para o primeiro elemento
	for (int i = tamanhoLista - 1; i >= 0; i--){
		String fruta = lista.get(i);
		System.out.println(fruta);
	}
```

```
    }
```

A saída é:

```
Maçã
Banana
Pera
Uva
Goiaba
Goiaba
Uva
Pera
Banana
Maçã
```