

DOCUMENTAÇÃO: PROJETO INDIVIDUAL – LISTA INVERTIDA COM REVISÃO POR PARES

Autor: Rafael Mourão Cerqueira Figueiredo, 05/06/2020

MENSAGEM AOS FORASTEIROS:

Este documento é para a orientação do corretor do projeto de Lista Invertida com Revisão por Pares. O projeto será corrigido, além do professor, por outros 3 colegas. Portanto, se você é um forasteiro (rs), recomendo que você leia o documento “ENUNCIADO - Projeto individual com revisão por pares - Lista Invertida.pdf”, que encontra-se disponível neste repositório.

ESTRUTURA DE ARQUIVOS-REGISTRO:

Como explicado no enunciado, a Lista Invertida é dividida em duas estruturas e, consequentemente, dois arquivos:

1. Dicionário: registra os **pares** de [termo, endereço] (onde o endereço é referente ao 2º arquivo). O cabeçalho do arquivo é um inteiro **N**: um meta-dado representando o **tamanho** do dicionário (isto é, quantos pares [termo, endereço] estão registrados ali).
A partir daí, temos **N** registros (um para cada par), registros estes que estão organizados da seguinte forma: [int tamanho_do_registro][String termo][long endereço].
2. Lista Invertida (própriamente dita): registra listas encadeadas de ids. Estes registros serão baseados em **blocos** (afim de ganhar mais eficiência, evitando a movimentação mecânica dos cabeçotes do disco rígido), onde estes blocos possuem uma ordem **O** (representa o máximo de registros que podem ser feitos por bloco) definida pelo usuário e seguem a seguinte estrutura:

N	IDs (possui O registros)	Próximo
N → quantidade de ids registrados no bloco		
IDs → o registro dos IDs propriamente dito		
Próximo → endereço do bloco com a continuação do registro dos IDs.		

Note que você vai se deparar com outro arquivo, chamado de “pessoas.db”. Este arquivo armazenará registros de pessoas, apenas para testar o funcionamento da Lista Invertida.

ESTRUTURA DE ARQUIVOS-FONTE:

Antes de explicar os arquivos, quero que você tenha ciência de uma coisa: existem 3 arquivos de registro, conforme explicado acima. Dentro das classes **Pessoa**, **Dicionario** e **ListaInvertida**, existe um método chamado “*construirTabela*” e alguns atributos de controle, usados neste método. O razão para isso será explicada na próxima sessão, a sessão de **execução**. Por enquanto, quero apenas ressaltar à você, corretor, que não precisa se preocupar com estes métodos e atributos, pois são irrelevantes no funcionamento da Lista Invertida. Agora, à explicação dos arquivos fonte:

O projeto está composto por 5 arquivos de código fonte. Destes, os 3 primeiros foram criados APENAS para testar o funcionamento da Lista Invertida:

1. **CRUD.java:** é um “mini-crud” genérico (digo “mini” pois implementei apenas os métodos *create* e *read*), similar ao CRUD do projeto Amigo Oculto que fizemos na disciplina.

Vale lembrar que o registro dos termos na lista invertida está no método *create* deste arquivo: `this.listaInv.create(objeto.getId(), objeto.chaveSecundaria());` (*linha 55*)

2. **Registro.java:** é a interface de Registros (também similar à interface criada no projeto Amigo Oculto).

As implementações desta interface possuem 6 métodos:

- 2.1. `toByteArray() : byte[]`
- 2.2. `fromByteArray(byte[] dados) : void`
- 2.3. `getId() : int`
- 2.4. `setId(int id) : void`
- 2.5. `chaveSecundaria() : String`
- 2.6. `toString() : String`

A única classe que implementa esta interface é a classe **Pessoa**, e os objetos genérico da classe **CRUD** extendem esta interface. Isto é, só se pode criar CRUDs de objetos de classes que implementem esta interface, e a Lista Invertida é aplicada sobre a **chave secundária** destes objetos (no caso da classe *Pessoa*, a chave secundária corresponde ao atributo *nome*).

Note que existem, além dos 6 métodos de implementação da interface, outros três métodos estáticos, com a finalidade de **alinhar** os dados dentro de um dado espaço, da seguinte forma: *centralizar* (alinhamento ao centro), *preencher* (alinhamento à esquerda) e *preencherEsq* (alinhamento à direita).

Estes métodos são usados nos métodos *construirTabela* (mencionados no primeiro parágrafo desta sessão), sendo assim, você corretor, também não precisa se preocupar com eles.

3. **Pessoa.java:** é a classe criada para o registro de pessoas. Uma pessoa possui dois atributos: *Idade* e *Nome*. Como explicado acima, a Lista Invertida é aplicada à *chave secundária* da implementação, no caso, o atributo *nome*. Logo, ressalto que o atributo *idade* é irrelevante para este projeto. Foi criado apenas para “enfeitar”.

Vale lembrar, também, que o CRUD (de pessoas) e a Lista Invertida estão dentro da classe *Pessoa*, como atributos estáticos. Eles são públicos, então pode-se acessá-los de qualquer lugar, usando “*Pessoa.crud*” e “*Pessoa.listaInv*”, respectivamente.

O próximo arquivo é referente à Lista Invertida de fato. Portanto, é com ele que você, corretor, deve se importar:

4. **ListaInvertida.java:** dentro deste arquivo, existem, ao todo, três classes:

1. **Dicionario:** é a classe para a implementação e gerenciamento da primeira estrutura da Lista Invertida (conforme explicado acima). Possui, claro, os métodos *create* e *read* (além dos métodos e atributos mencionados no início desta sessão).
2. **Termo:** esta é uma classe **interna** à classe *Dicionario*, e serve para representar um par [termo, endereço]. Sua existência não é necessária, mas eu a criei na tentativa de deixar o código mais organizado e, consequentemente, mais fácil de entender.
3. **ListaInvertida:** é a classe para a implementação da Lista Invertida propriamente dita. Da mesma forma que eu criei a subclasse *Termo* para o dicionário, eu poderia ter criado, aqui, a subclasse *Bloco*. Não o fiz pois achei que não iria contribuir para a organização e simplificação do código. Analogamente, criei o método *novoBloco(int id)*. Entenda este método como um **construtor** de blocos, que recebe um id (chave que estamos inserindo) e retorna o endereço deste novo bloco no arquivo.

Entenda este 4º arquivo da seguinte forma: cada Lista Invertida possui um dicionário e uma lista invertida propriamente dita, e o **único** lugar que usamos objetos da classe *Dionario* é, justamente, dentro da classe *ListaInvertida*.

Por causa disso, a classe *Dionario* deveria ser uma subclasse da *ListaInvertida* (isto é, deveria estar **dentro** da mesma). Eu não o fiz, pois achei que isso atrapalharia a organização do código, embora, lógicamente falando, acredito que o mais organizado seria fazê-lo.

Quero que você, que estiver lendo este projeto, entenda como se **TODO O CÓDIGO** presente dentro deste 4º arquivo (*ListaInvertida.java*) estivesse **DENTRO** da classe *ListaInvertida*.

5. **TesteListaInvertida.java:** este último arquivo corresponde à classe **principal** e possuí, além da *main*, outras duas funções:

1. *visualizarRegistros()* - esta função basicamente chama e imprime os métodos *construirTabela* (mencionados no primeiro parágrafo desta sessão).
2. *criarRegistros()* - esta função simplesmente cria registros de pessoas, de forma manual.

EXECUÇÃO:

À respeito da execução, existem algumas coisas que eu quero te informar, antes que você precise descobrir sozinho. Vou dividir esta sessão em tópicos, pontuando tudo que eu quero que você saiba à respeito:

- **DA COMPILAÇÃO E EXECUÇÃO:** Nada de anormal nestas tarefas. Faça-as como você faria para qualquer outro projeto em java: **compile** com “javac *TesteListaInvertida.java*” e **execute** com “java *TesteListaInvertida*”.
- **DA VISUALIZAÇÃO DOS DADOS:** No início da sessão “estrutura de arquivos-fonte”, eu mencionei a existência de métodos com o nome de *construirTabela* e atributos de controle usados por eles. Agora, estou te devendo a explicação do **porquê** os criei, visto que eles não influenciam no funcionamento da Lista Invertida de forma alguma:

Basicamente, visualizar os registros de pessoas pelo código fonte é muito fácil, visto que eu os criei de forma bastante explícita e organizada dentro do método *criarRegistros()* (arquivo principal). Entretanto, o mesmo não ocorre com os registros da Lista Invertida. Estes são feitos de forma implícita, pelo próprio algoritmo que criei. Sendo assim, fica muito mais desagradável visualizar o funcionamento da Lista Invertida.

Em outras palavras: os métodos *construirTabela* literalmente **constroem** uma tabela com os dados do respectivo arquivo-registro da classe em questão, e eu os criei simplesmente para facilitar a visualização da Lista Invertida em funcionamento. Quando você executar o código, vai perceber que 3 tabelas serão impressas, uma para cada arquivo-registro.

- **DA ALTERAÇÃO DOS DADOS PARA TESTE:** Como eu já disse, eu mesmo criei um conjunto de registros de pessoas para testar o projeto. Entretanto, sinta-se livre para modificar os registros livremente, basta acessar o método *criarRegistros()* (arquivo principal) e mexer nos construtores da classe Pessoa.

Contudo, tome cuidado com apenas um detalhe: **registros idênticos são permitidos**. Por exemplo, se você colocar dois construtores **exatamente iguais**, você terá dois registros **exatamente iguais**.

Quero que você se sinta livre também para, se quiser, para testar outros tipos de registros, além dos registros de pessoas. Para isso, a classe do novo tipo de registro deve **implementar** a interface *Registro*, e você deve ter em mente que a Lista Invertida será implementada em cima da **chave secundária** desta nova classe.

Para testar uma nova classe, recomendo que você a crie seguindo o mesmo padrão que eu usei para criar a classe *Pessoa*.

Se você quiser colocar esta nova classe para funcionar “ao lado” da classe *Pessoa* (isto é, se você quer ter duas Listas Invertidas, para dois tipos diferentes de registros), você precisará, também, adaptar a *main* para que o programa trabalhe com os dois registros simultaneamente.