Implementação de índices primários e secundários

1 Objetivo

O objetivo deste trabalho é praticar os conceitos de índices no contexto de uma aplicação que gerencia registros de uma biblioteca.

2 O ambiente de trabalho

O código fonte Java, disponibilizado juntamente com este enunciado, possui os pacotes mostrados na tabela 1:

Pacote	**Pacote**
data	Pasta destinada a conter os arquivos de dados e os arquivos de índices.
filemanager	Contém as interfaces para as classes de índices primário e secundário além da classe abstrata gerenciadora de arquivos `FileManager`.
queryParser	Contém pacotes e classes relacionados ao parser de consultas.
registro	Contém a classe livro e o gerador de "livros aleatórios"
solution	Esse pacote contém a classe BibMananeger que implementa o gerenciador de arquivos.
ui	Contém as classes para interface gráfica do sistema
uitl	Contém uma classe utilitária que pode fazer a conversão entre dados e bytes

Tabela 1: Pacotes do sistema

Para realizar este trabalho, deve-se criar classes que satisfaçam as interfaces `PrimaryIndex` e `SecondaryIndex` do pacote `filemanager`. Ambas as interfaces compartilham grande parte do código, mas várias funções diferem nos tipos de parâmetros e de retorno. A seguir, discute-se brevemente o que é esperado de cada função em cada uma das interfaces, começando por `PrimaryIndex`, mostrada no Código 1 a seguir.

Código 1: Interface para índice primário

```
1  package filemanager;
2  import java.io.IOException;
3  import java.util.LinkedList;
4
5  /**
6   * Define a interface de um índice primário. O índice deve mapear
7   * uma chave primária (uma instância de comparable) para um endereço
8   * físico do arquivo (um long).
9   */
10 public interface PrimaryIndex {
11     /**
12      * Cuida de toda inicialização do índice secundário.
13      */
14     public void prepare() throws IOException ;
15
16     /**
```

```
17       * Carrega todo o índice na memória principal.
18       */
19      public void loadIndex() throws IOException ;
20
21      /**
22       * Salva todos os registros na memória secundária.
23       */
24      public void saveIndex() throws IOException ;
25
26      /**
27       * Indexa o arquivo de ados pelo campo chave.
28       */
29      public void buildIndex(String dataFilePath) throws IOException;
30
31      /**
32       * Retorna o byteoffset do registro associado a key, em O(lg n).
33       * Caso key não esteja presente no índice, deve retornar -1.
34       */
35      public Long lookForKey(Integer key);
36
37      /**
38       * Adiciona um par (chave, endereço de registro) ao índice.
39       * Substitui o valor associado a chave caso ela já esteja
40       * presente no índice.
41       */
42      public void addKey(Integer key, Long address);
43
44      /**
45       * Remove uma chave do índice.
46       * Não faz nada se a chave não está presente no índice.
47       */
48      public void removeKey(Integer key);
49
50      /**
51       * Retorna um array list com todas as chaves desse índice.
52       */
53      public LinkedList<Integer> getAllKeys();
54  }
55
```

A função `prepare` deve cuidar da inicialização do índice, executando, pelo menos, os seguintes passos:

1. Verificar se o arquivo de índice existe. (Dica: use a classe `File` do pacote `java.io`);
2. Se o arquivo de índice existir, carregar o índice para memória principal.
3. Se o arquivo de índice não existir, lançar a exceção `FileNotFoundException`

A função `loadIndex` deve carregar o arquivo de índice para a memória principal. A função `saveIndex` deve descarregar o conteúdo do índice para o arquivo.

A função `buildIndex` está presente na interface de índice primário e secundário. O funcionamento do método é muito semelhante em ambos os casos, diferindo apenas nas informações coletadas para construção do índice. A função `buildIndex` deve ler cada registro do arquivo, extrair as informações necessárias de acordo com cada índice e, em seguida, armazenar essas informações na memória principal. Ao término do processo, a função `buildIndex` deve salvar o índice construído. No caso do índice primário, por exemplo, `buildIndex` deve coletar o *byte*

offset do início de cada registro e o respectivo ISBN do registro. No caso do índice secundário de editora, por exemplo, `buildIndex` deve coletar o ISBN e a editora de cada registro.

A função `getAllKeys` deve retornar uma lista encadeada de todos os ISBNs presentes no índice, em ordem crescente. Caso não haja quaisquer chaves presentes no índice, a função `getAllKeys` deve retorna uma lista vazia. A função `lookForKey` deve aceitar como parâmetro uma chave (ISBN) e retonar o *byte offset* do registro associado a chave no arquivo de dados. Caso a chave não esteja presente no índice, a função `lookForKey` deve retonar -1.

As funções `addKey` e `removeKey` são auto-explicativas.

A interface para índice secundário é mostrada no Código 2

Código 2: Interface SecondaryIndex

```
1  package filemanager;
2
3  import java.util.LinkedList;
4  import java.io.IOException;
5
6  public interface SecondaryIndex {
7      public void prepare()throws IOException;
8      public void loadIndex() throws IOException;
9      public void saveIndex() throws IOException;
10     public void buildIndex(String dataFilePath) throws IOException;
11     public LinkedList<Integer> lookForKey(String skey);
12     public void addKey(String key, Integer primaryKey);
13     public void removeKey(String key);
14     public LinkedList<String> getAllKeys();
15     public LinkedList<Integer> getAllValues();
16     public void removeAllISBNS(Integer isbn);
17 }
18
```

As funções `prepare`, `loadIndex`, `saveIndex`, `buildIndex` e `getAllKeys` são como nos índices primários, levando-se em conta as devidas mudanças nos tipos de parâmetros e retorno. A função `lookForKey` deve retornar uma lista de ISBNs, em ordem crescente, associados à chave secundária passada como parâmetro. Caso a chave não esteja presente no índice, `lookForKey` deve retornar uma lista vazia.

A função `getAllvalues` deve retornar uma lista de todos os ISBN presentes no índice atual, sem repetições e em ordem crescente. Caso não haja nenhum ISBN presente no índice, `getAllvalues` deve retornar uma lista vazia. A função `removeAllISBNS` deve aceitar como parâmetro um ISBN e remover todas as ocorrências do ISBN dado de cada uma as entradas do índice secundário. Se alguma entrada do índice contiver apenas o ISBN dado, então essa chave secundária devera ser removida.

Para ilustrar melhor esse último caso, suponha um índice secundário para autor no qual, em algum momento, haja uma chave para o autor `"Bender"` a qual possua apenas o ISBN 12345678 associado a este autor. Se o ISBN 12345678 for removido, a entrada `"Bender"` não terá nenhum ISBN associado a ele e, portanto, esta entrada deverá ser removida do índice.

3 Integrando a solução ao ambiente de trabalho

ATENÇÃO:
O código fonte do ambiente de trabalho contém classes e pacotes que o aluno `não` precisa compreender. A saber os pacotes que devem ser vistos como caixas pretas são `ui` e `queryParser`. No pacote `registro` o aluno só precisa compreender a API pública da classe livro, explicada na seção 6 deste documento. No pacote `fileManager` as interfaces já foram devidamente explicadas. No pacote `util` a pode ser conveniente conhecer a classe `ListUtils` que define operações de união e interseção de listas encadeadas de inteiros. Conhecer o ambiente de trabalho enquanto sua solução é desenvolvida é uma estratégia melhor do que tentar aprender todo o ambiente de uma só vez.

Antes de implementar a solução, certifique-se de que o ambiente de trabalho está funcionado. Para compilar o ambiente de trabalho no linux ou em um ambiente compatível com linux, como Cygwin no Windows, utilize o comando `make` dentro da pasta Biblioteca ou digite o comando `javac -cp . Main.java` dentro da pasta. Ambos os comandos devem terminar sem exibir mensagens de erros ou warnings. Feito isso, execute o programa principal com o comando `java -cp . Main.java`. O sistema deve exibir uma interface gráfica, o que significa que o programa está funcionado corretamente. Caso alguma coisa não funcione como o que foi relatado, procure o professor da disciplina.

Aos que desejarem usar alguma IDE, atentem-se para as configurações da IDE, especialmente com relação ao **diretório de trabalho**. A IDE Eclipse, por exemplo, passa como diretório de trabalho para a aplicação o endereço da pasta do projeto no Workapace. Isso fará com que o programa não encontre os arquivos de dados. Para resolver esse problema na IDE Eclipse, vá ao menu `Run -> Run Configurations`, selecione a configuração de execução do "Main"do projeto que deseja configurar e, em seguida, selecione a aba `Arguments`. Na seção `Working Directory`, selecione a opção `other` e insira `${workspace_loc:bin}` no campo de texto.

A solução deve ser implementada dentro do pacote `solution`. Note que cada um dos índices deve definir o nome e o local de seus respectivos arquivos de dados. Para o correto funcionamento do programa, todos os arquivos de dados devem ser salvos no pacote `data`.

Vale observar que o programa salva os índices sempre que alguma alteração é feita, i.e. sempre que algum livro é inserido ou removido. Desse modo recomenda-se que os índices não mantenham *streams* de dados abertos o tempo todo. Os índices deve abrir *streams* de dados apenas no momento da leitura/escrita e fechá-los imediatamente após o termino da operação.

ATENÇÃO:
Para fins de implementação você pode utilizar quaisquer classes disponíveis nas bibliotecas nativas de Java. Todos os arquivos de dados salvos devem estar no formato binário e devem conter apenas dados. Não é permitida a utilização da interface *Serializable* de Java nem da classe *ObjectStream*.

Para testar a solução implementada, abra a classe `BibManager` no pacote `solution`. Utilize o método set apropriado para definir o índice implementado no método `initilize` dessa classe.

Por exemplo, suponha que temos a implementação do índice secundário para autor e do índice primário e suas classes se chamam AutorIndex e ISBNIndex, respectivamente. Essas soluções podem ser integradas como apresentado no Código 3, a seguir:

Código 3: Classe BibManager

```
1 public  class BibManager extends FileManager {
2    public BibManager() throws IOException {
3        super("data"+File.separator + "livros.dat");
4    }
5    public void initialize() throws IOException{
6        setPrimaryIndex(new ISBNIndex());
7        setAutorSecondaryIdx(new AutorIndex());
8    }
9 }
```

Para cada índice existe um *getter* e um *setter* na classe `FileManager`. A saber os *getters* são:

- `public PrimaryIndex getPrimaryIndex()`
- `public SecondaryIndex getAutorIndex()`
- `public SecondaryIndex getEditoraIndex()`
- `public SecondaryIndex getTituloIndex()`

E os *seters* são:

- `public void setPrimaryIndex`
- `public void setAutorSecondaryIdx`
- `public void setTituloSecondaryIdx`
- `public void setEditoraSecondaryIdx`

4 Testando a solução

Compile o programa e execute-o. O programa deve exibir a interface gráfica da figura 1.

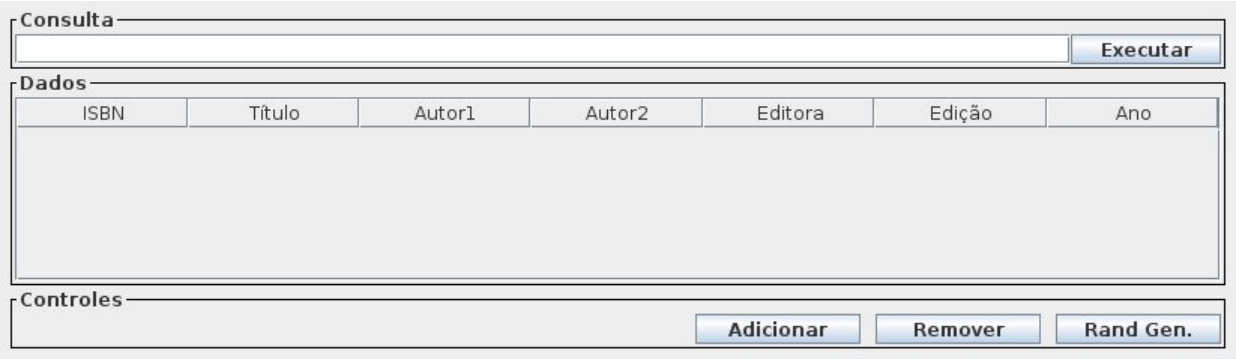

Figura 1: Tela inicial da interface gráfica

Da primeira vez que o programa for utilizado, será necessário criar um arquivo de dados de teste. Isso pode ser feito clicando-se no botão "Rand Gen.". Uma janela será exibida requerendo o número de registros a serem gerados. Sugere-se que, inicialmente, sejam usados valores

pequenos, como 10, 20 ou 50 registros. Após informar o valor, o sistema irá gerar o arquivo chamado "livros.dat"dentro da pasta data.

Você pode adicionar manualmente um registro clicando no botão "adicionar". Uma janela como a mostrada na Figura 2 será exibida. Note que você pode optar por preencher automaticamente os campos clicando no botão "Preencher".

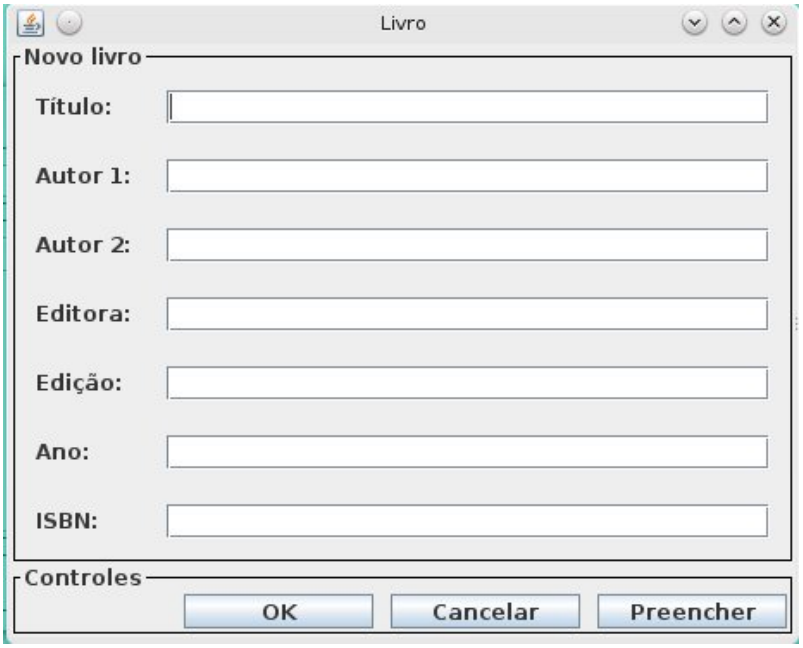

Figura 2: Janela de cadastro de um novo livro.

Consultas podem ser realizadas digitando-se expressões de busca no campo consulta da janela da Figura **??**. As expressões de busca podem ser simples ou compostas. As consultas simples têm a forma `Campo` ou `Campo` $\oplus$ `Valor` ou ainda `Valor` $\oplus$ `Campo`. Vale observar que o sistema é sensível à caixa. Logo diferenças entre letras maiúsculas e minúsculas são levadas em consideração ao produzir o resultado.

- Um campo pode ser:
  - isbn
  - autor
  - titulo
  - editor
- Um valor pode ser:
  - Uma string qualquer entre aspas duplas (`"`), se o campo não for ISBN.
  - Um valor inteiro positivo, se o campo for ISBN.

O operador $\oplus$ pode ser qualquer um dos operadores `<,> ou =`. A seguir apresenta-se exemplos de consultas simples :

- `isbn` : Lista todos os livros do arquivo;

- `autor = "Enzo Garcia"` : Lista todos os livros cujo o campo primeiro autor seja igual a Enzo Garcia.
- `editora > "B"` : Lista todos os livros cuja a string do campo `editora` seja menor que a string `"B"`. Essa comparação entre strings ocorre como definido no método `campareTo` da classe `String` de java.
- `"O ano da fúria"= titulo` : Retorna a lista de todos os livros cujo titulo é igual a O ano da fúria

Consultas compostas têm uma das seguintes formas: *ConsutlaSimples* & *ConsultaComposta* ou *ConsutlaSimples* | *ConsultaComposta*. No primeiro caso é feita a interseção das listas de ISBN resultantes de cada consulta e no segundo caso é feita a união. Toda consulta simples também pode ser vista como um consulta composta. A seguir apresenta-se exemplos de consultas compostas:

- `isbn > 100000 & isbn < 200000` : Lista todos os livros do arquivo cujos os ISBNs estejam acima de 100000 e abaixo de 200000.
- `autor = "Enzo Garcia"| editora = "Twilli` : Lista todos os livros cujo o campo `autor1` seja igual a Enzo Garcia, ou cujo campo editora seja igual a Twilli.
- `editora > "B"| 10 > isbn` : Lista todos os livros cujo a string do campo `editora` seja menor que a string `"B"`, ou o ISBN seja menor que 10.

Qualquer consulta que não esteja de acordo com essas especificações é considerada como erro pelo programa. Após digitar a consulta, clique no botão "Executar"para que o programa interprete a consulta. Vale ressaltar que cada um das campos de consulta (autor, editora, titulo e isbn) estão relacionadas ao seus respectivos índices. A Figura 3 mostra o resultado de uma consulta realizada no sistema.

Figura 3: Resultado de uma consulta realizada na interface

5 Entrega

O trabalho deve ser entregue `até o dia 22/02/2016` no e-mail `eltonm.cardoso@gmail.com`. O assunto da mensagem deve ser: `NOME-DO-ALUNO-TP2-AEDSII`. Deve ser enviada apenas a pasta solution comprimida com zip ou rar. Após a data de entrega, será aplicada uma penalidade à nota de acordo com a seguinte fórmula, na qual n é o número de dias de atraso:

$$f(n) = \frac{2^{n-1}}{60}$$

6 Apêndice: API da classe livro

Todas os métodos apresentados a seguir são parte da API pública de livro e podem ser usados.

Construtores

- `Livro()` : Construtor padrão para a classe livro. Todos os campos numéricos são inicializados com zero e todos os campos string são inicializados com a string vazia ;
- `Livro(String titulo, String autor1, String autor2, String editora, int ano, int isbn, int edicao)` Construtor parametrizado pelo valor de cada campo.

Métodos:

- `int recordSize()` : Retorna o tamanho, em bytes do registro.
- `byte[] pack()` : Converte o livro em um vetor de bytes;
- `void unpack(byte[] b)`: Dado um vetor de bytes, que contém um registro serializado, converte cada campo do vetor b para o atributo apropriado do livro.
- `void writeToStream(OutputStream s) throws IOException`: Escreve este livro para o OutputStream dado.
- `boolean write(RandomAccessFile f) throws IOException`: Escreve este livro para o RandomAcessFile dado. Retorna `true` se o livro foi completamente escrito.
- `boolean readFromStream(InputStream s)throws IOException` Lê o livro do InputStream dado. Retorna `true` se o livro foi completamente lido.
- boolean read(RandomAccessFile f) throws IOException Lê o livro do RandomAcessFile dado. Retorna `true` se o livro foi completamente lido.
- `public String getTitulo()` : Retorna o valor do campo título do livro.
- `public String getAutor1()` : Retorna o valor do campo primeiro autor do livro.
- `public String getAutor2()` : Retorna o valor do campo segundo autor do livro.
- `public String getEditora()` : Retorna o valor do campo editora do livro.
- `public int getIsbn()` : Retorna o valor do campo ISBN do livro.
- `public int getAno()` : Retorna o valor do campo ano de publicação do livro.
- `public int getEdicao()` : Retorna o valor do campo edição do livro.