Pontifícia Universidade Católica de Minas Gerais
ICEI - Curso de Ciência da Computação
Disciplina: Algoritmos e Estruturas de Dados III
Prof. Zenilton Patrocínio Jr.

Trabalho Prático de Algoritmos e Estrutura de Dados III - Entrega 1

Model - Magazine Abakós - ICEI - PUC Minas

Henrique Castro e Silva[1]
Leonardo Caetano Gomide[2]

Resumo

Este trabalho tem como intenção desenvolver um sistema que faça manipulação de arquivos em memória primária e secundária. O sistema em questão, consiste em um cadastro de prontuários para um empresas de planos de saúde, com opções de inserção, alteração, exclusão e impressão. Conforme estipulado pelo cronograma da disciplina, este trabalho será composto por três entregas complementares, cada entrega responsável por entregar diferentes artefatos. Este documento é referente à primeira entrega, que é composta por diferentes artefatos que foram desenvolvidos pelos alunos. Esses artefatos incluem a criação de um arquivo mestre, o desenvolvimento de um sistema que faça todas as operações de CRUD (criar, ler, atualizar e deletar) e produção de um documento de texto. Para o desenvolvimento dos diferentes requisitos técnicos foram desenvolvidas diferentes classes e funções. A formatação do arquivo mestre, a finalidade e as especificações de cada classe e o funcionamento do sistema em geral serão abordadas neste documento.

Palavras-chave: Algoritmos e Estrutura de Dados. CRUD. Manipulação de arquivos.

[1]Programa de Graduação em Ciência da computação da PUC Minas, Brasil– henrique.silva.1281914@pucminas.br
[2]Programa de Graduação em Ciência da computação da PUC Minas, Brasil– lcgomide@sga.pucminas.br

Abstract

This assignment focused on developing a system that may manipulate files in primary and secondary memory. The system consists of a registration of medical records, including insertion, updating, exclusion and printing options. According to the discipline's stipulated schedule, this assignment will be composed of three complementary deliveries, each responsible to deliver different features. This document refers to the first delivery, which is composed by different features developed by the students. These features include the creation of a master file, a system's development, which include all CRUD (create, read, update, delete) operations, and generation of textual document. In order to develop all the technical issues, different classes and functions were developed. The master file's formatting, the purpose and specifications' of each class and the overall systems' running will be discussed in this document.

Keywords: Algorithms and Data Structure. CRUD. File manipulation.

1 INTRODUÇÃO

O trabalho aqui apresentado consiste em um sistema para gerenciamento de prontuários. Para o desenvolvimento desse sistema, foi escolhida a linguagem de programação Java, que permitiu a criação do CRUD completo, a interação com os arquivos de dados e implementação da solução. Nas sessões subsequentes, será abordado a estruturação do projeto e a funcionalidade de seus artefatos.

2 ESTRUTURA DE ARQUIVOS

No intuito de tornar o acesso aos arquivos do projeto mais claro e simples, os diferentes arquivos foram segmentados em diferentes diretórios seguindo a estrutura abaixo.

```
root
├─ /dados
└─ /src
   ├─ /dao
   ├─ /main
   ├─ /manager
   └─ /model
```

2.1 Dados

No diretório "dados"encontram-se os arquivos de dados. Quando o usuário do sistema cria um novo arquivo mestre de dados, ele ficará disponível nesse diretório.

2.2 Source

No diretório "src"encontram-se todos os arquivos de código fonte desenvolvidos e necessários para o funcionamento do sistema. Esse diretório por sua vez possível diferente ramificações, conforme será exposto a seguir.

2.2.1 Dao

O diretório "dao", abreviado do inglês *Data Acess Object*, encontram-se as classes responsáveis por fazer o acesso aos dados por meio das classes *Manager*. Nessas classes encontram-se os métodos de CRUD. Contudo as classes nesse diretório possuem um nível de abstração superior a outras classes, assim sendo, quando executado algum método, elas são responsáveis de interpretar o objeto passado e delegar as respectivas funções necessárias para outras classes.

2.2.2 Main

Nesse diretório encontra-se a classe *App*, que é a primeira classe a ser executada quando o projeto é executado. Por meio dela que ocorre a interação do usuário com o sistema.

2.2.3 Manager

As classes encontradas no diretório "manager"são as classes que possuem métodos de mais baixo nível dentro do sistema. Ou seja, elas que fazem operações com Bytes, seja na conversão de um objeto para um vetor de bytes, ou vice e versa, e também a escrita, a sobrescrita e a leitura de vetores de Byte.

Nelas também ja encontram-se métodos responsáveis por manusear os metadados do arquivo, guardar informações sobre os tamanhos do registro e o caminho para o arquivo mestre.

2.2.4 Model

Por fim, o diretório "model"é responsável por armazenar as classes que definem os objetos do sistem.

3 IMPLEMENTAÇÃO

Uma vez exposta a estruturação do projeto, pode-se detalhar a implementação dos métodos implantados.

3.1 Modelo

As classes de modelo são as estruturas basicas do projeto, definindo os objetos do sistema, como descrito a seguir:

3.1.1 Prontuario

Em nossa implementação da entidade prontuário optamos pelos seguintes atributos:

Tabela 1 – Atributos

Nome	Tipo de Dado	Tamanho (Bytes)
codigo	int	4
nome	String	variável
dataNascimento	java.util.Date	8
sexo	char	2
anotacoes	String	variável

Quando o Prontuário é codificado para ser inserido no arquivo de dados, os atributos são salvos nessa ordem utilizando os métodos da classe *java.io.DataOutputStream*, no caso da *dataNascimento*, é obtido a quantidade de milissegundos desde 1970 e codificado como um *long*, no momento dessa codificação se o registro codificado estourar o tamanho pré-definido de registro do arquivo, uma *IndexOutOfBoundsException* é estourada.

3.2 Arquivos de Dados/Mestre

Para salvar os dados recebidos do usuário, foi utilizado um arquivo de dados com a extensão .db com cabeçalho e os dados em si, como especificado a seguir:

3.2.1 Cabeçalho

O cabeçalho do arquivo contêm diferentes metadados que são utilizados por outras classes do sistema. Os metadados contidos são a variável *registerSize*, especificada pelo usuário, que dita o tamanho fixo do arquivo; a variável *headerSize*, ou tamanho do cabeçalho; o valor do próximo código a ser inserido, denomindado *nextCode*; e por fim o *fileSize*, que armazena o tamanho total do arquivo.

3.2.2 Conteúdo

O conteúdo do arquivo mestre é composto por *n* registros de tamanho *registerSize*. Cada registro é composto pelos campos da classe *Prontuario*, e também é utilizado dois bytes extras como lápide, sendo codificada como *char* com o valor igual a *'\0'* por padrão e igual a *'*'* quando o registro for deletado.

3.3 CRUD

A classe *App* fica responsável de "interagir"com o usuário e servir de menu de opções. Uma vez que o usuário passe valores de input válido, a classe cria um objeto da classe *Prontuario* e passa o objeto para a classe *ProntuarioDAO*, que fica encarregada de terminar a operação.

3.3.1 Create

Para a inserção de um registro, a classe primeiro chama o método *getFirstEmpty* que retorna qual a posição do primeiro registro que foi deletado ou retorna -1, caso o registro tenha que ser inserido no fim do arquivo.

Uma vez com esse *offset*, como foi nomeado dentro do código, a classe DAO converte o objeto Prontuario para um vetor de Bytes, por meio da classe *ProntuarioManager*, e chama a classe *DbManger*, que munida do vetor de bytes e o offset, faz a inserção no arquivo.

3.3.2 Read

A leitura de registro é feita de *n* em *n* registros. Por *default* esse passo é de 100 registros. Para fazer essa leitura é utilizada o método *readFromFileBody*, da classe DbManager, que recebe a quantidade de registros a ser lida e o offset, ou seja, a partir de qual posição ela deverá ler. Esse método retorna um vetor de Bytes, que é convertido em um vetor de Prontuarios. Para registros que possuem um lápide indicando deleção, o registro é convertido em um objeto null.

Uma vez com esse vetor, a classe DAO compara se algum registro possui a ID especificada pelo usuário, caso algum possua ele retorna o prontuário, caso contrário, são lidos novos Prontuários até encontrar o id procurado, ou chegar ao fim do arquivo e retornar null.

3.3.3 Update

O mecanismo de atualização faz uso do método de leitura, descrito acima, para encontrar o prontuário especificado e sua posição no arquivo. Caso encontrado, o objeto prontuário recebe os novos valores passados pelo usuário e em sequência é utilizado o DbManager para sobrescrever o prontuário antigo com as novos dados.

3.3.4 Delete

Similar ao método de atualização, o método de deleção faz uso de outro método para sua operação. Nesse caso, ele utiliza a atualização, que possui uma variável booleana chamada *delete*. Essa variável, é igual a false no método *updateProntuario* onde não se passa esse parâmetro, é responsável por escreve a lápide do registro. Caso o valor do parâmetro seja True, os Bytes da lápide serão escritos de forma a sinalizar a deleção.

Referências