Amigo Oculto 2 - CRUD

As operações em um arquivo

A próxima implementação no projeto é a criação de um **CRUD** de usuários. CRUD é a sigla de *Create*, *Read*, *Update* e *Delete*. A implementação de um CRUD, portanto, é a implementação de recursos que nos permitam fazer as operações básicas de usuários em algum arquivo de dados, especialmente as operações de incluir, alterar, excluir e consultar.

CRUD

A classe do CRUD

Precisamos criar, agora, um as rotinas para fazermos inclusão, alteração, exclusão e busca de usuários. No entanto, não implementaremos essas operações diretamente, mas os métodos básicos do CRUD (*create*, *read*, *update* e *delete*):

- O primeiro método é o `create()`, que deve ter a seguinte assinatura:
  `int create(dadosDoUsuário)`
  Esse método receberá os dados do novo usuário a ser incluído em um arquivo de dados e retornará o identificador desse novo usuário (`idUsuário`). Esse identificador será criado automaticamente pelo arquivo. Geralmente, é apenas o último identificador usado adicionado de uma unidade. Se o usuário não puder ser incluído no arquivo por qualquer razão, o método deve retornar *null* ou um valor negativo.
- O segundo método é o `read()` que é usado em quase todas as operações. Precisamos fazer a leitura de um usuário nas operações de consulta, alteração e exclusão. Esse método deve ter a seguinte assinatura:
  `Usuario read(idUsuário)`
  Esse método receberá o identificador do usuário para localizá-lo em um arquivo de dados e retornará um objeto contendo os dados desse usuário. Se nenhum usuário com esse identificador for encontrado no arquivo, então o método deve retornar um valor como *null*. Veremos como obter o identificador do usuário mais tarde.
- O próximo método é o `update()`. Esse método deve receber os novos dados do usuário como parâmetro, além do próprio identificador. Isso pode ser passado por meio de um objeto que contenha todas essas informações do usuário. O método retornará um valor lógico que informará se a operação foi bem sucedida ou não. Sua assinatura será, portanto, assim:
  `boolean update(usuário)`
- O quarto método é o `delete()` que será parecido com o método *read()*. Esse método efetivará a exclusão do usuário do arquivo de dados. O método também retornará um valor lógico que informará se a operação foi bem sucedida ou não. Sua assinatura será essa:
  `boolean delete(idUsuário)`

Arquivo de Usuários

Você deve criar uma classe específica para as operações do CRUD de usuários. Isso significa que os métodos do CRUD não serão incluídos na própria classe do Usuário, mas nessa outra classe que será responsável pelas operações do CRUD no arquivo de usuários. Em breve, discutiremos como transformar essa classe em uma que gerencie arquivos de qualquer tipo de objeto e não apenas de usuários.

Assim, é importante que essa classe tenha um manipulador de arquivos. Você pode usar um objeto da classe `RandomAccessFile`. Esse arquivo só será manipulado pelos métodos do CRUD e nunca deverá ser acessado por fora desta classe.

Esse arquivo deve ser aberto pelo construtor do CRUD e mantido aberto durante toda a existência do objeto da classe do CRUD. Na abertura, é importante considerar que, se o arquivo ainda não existir e precisar ser criado pelo construtor, ele deverá ser criado e o cabeçalho inicial do arquivo deverá ser escrito nele. Nesse cabeçalho, deve constar, pelo menos, o valor do último ID usado no arquivo (inicializado com zero).

Índices

O seu arquivo de usuários também deverá ter pelo menos dois índices. O primeiro índice será um índice direto baseado no ID dos usuários, que permitirá as operações de leitura, atualização e exclusão. Assim, esse índice conterá pares de IDs e endereços. O segundo índice é um índice indireto baseado no email do usuário, pois precisaremos validar o acesso dos usuários por meio desse atributo. Esse índice, portanto, será de pares de emails e IDs. Lembre-se que esse valor de email será retornado pelo método `chaveSecundaria()` .

Todos os índices devem ser abertos junto com o arquivo de dados dos usuários e mantidos abertos durante as operações do CRUD.

Como ponto de partida, você pode usar as minhas classes de índices:

- **indice.direto.zip (https://pucminas.instructure.com/courses/9019/files/654603/download?wrap=1)**
- **indice.indireto.zip (https://pucminas.instructure.com/courses/9019/files/654599/download?wrap=1)**

Create

O método `create()` deve receber o conjunto de dados do Usuário: nome, email e senha. Se você tiver adicionado outros dados à classe do Usuário, eles também deverão ser passados a este método.

Esse método deve seguir a seguinte sequência de passos:

1. Ler o último ID usado, que deve estar armazenado no cabeçalho do arquivo;
2. Incrementar esse ID em uma unidade e atualizá-lo no cabeçalho do arquivo;
3. Criar um novo objeto Usuário com o novo ID e com os dados recebidos como parâmetros;
4. Mover o ponteiro do arquivo para o fim do arquivo;
5. Identificar o endereço em que o registro será escrito;
6. Escrever o registro do usuário (lápide, indicador de tamanho e vetor de bytes que representa o objeto Usuário) nesse endereço (fim do arquivo);
7. Incluir o par (ID, endereço) no índice direto (baseado em IDs);
8. Incluir o par (chave secundária, ID) no índice indireto;
9. Retornar o novo ID.

Read (ID)

O método `read()` terá duas implementações, pois, em diferentes rotinas, precisaremos localizar usuários por meio de seus IDs e de seus emails.

Assim, teremos uma primeira implementação que localizará usuários por meio dos seus IDs. Sua função será, basicamente, retornar o objeto do usuário a partir do ID passado como parâmetro. Para isso, deve executar os seguintes passos:

1. Buscar o endereço do registro do usuário no índice direto usando o seu ID;
2. Mover o ponteiro no arquivo de usuários para o endereço recuperado;
3. Ler o registro do usuário e criar um novo objeto Usuário a partir desse registro;
4. Retornar esse objeto de usuário.

Read (chave secundária)

A segunda implementação do método `read()` usará a chave secundária (email) como referência para recuperar o objeto do usuário. Ela deve conter os seguintes passos:

1. Buscar o ID do usuário no índice indireto usando o valor retornado pelo método `chaveSecundaria()`;
2. Usando esse ID, invocar o método `read()` anterior (baseado em IDs) para recuperar o objeto do usuário;
3. Retornar esse objeto do usuário.

Como você pode perceber, essa segunda implementação do método `read()` depende da primeira para funcionar. Ela apenas faz a consulta ao índice indireto, para que possa ser feita a busca baseada em IDs.

Update

O método `update()` será responsável pela atualização dos dados do usuário. Devemos, porém, tomar cuidado com o tamanho do registro, medido a partir da quantidade de *bytes* que ele usa. Esse tamanho, após a alteração, pode ter aumentado, diminuído ou mantido sem alterações. Se o tamanho não tiver sido alterado, então o registro pode permanecer na posição que estava. Se o tamanho tiver aumentado, então o registro deve ser movido para outro espaço (por exemplo, no fim do arquivo). E se o tamanho tiver diminuído, então podemos adotar duas estratégias: a primeira é também mover o registro para um novo espaço e a segunda é gerenciar o espaço excedente (usando as estratégias discutidas em aula).

O método `update()` deve receber o novo objeto do usuário, que substituirá o objeto atual. Mas atenção: **o ID de um usuário nunca deve ser alterado!** O método deve conter a seguinte sequência de passos:

1. Buscar o endereço do registro do usuário no índice direto usando o seu ID;
2. Mover o ponteiro no arquivo de usuários para o endereço recuperado;
3. Ler o registro do usuário e identificar se houve variação no tamanho do registro;
4. Se o tamanho não tiver sido alterado,
    1. Retornar o ponteiro do arquivo para o endereço recuperado no índice;
    2. Escrever o novo registro do usuário, sobrescrevendo o atual;
5. Se o tamanho tiver sido alterado,
    1. Retornar o ponteiro do arquivo para o endereço recuperado no índice;
    2. Marcar o campo lápide;

3. Mover o ponteiro do arquivo para o fim do arquivo;
4. Identificar o endereço em que o registro será escrito;
5. Escrever o novo registro do usuário;
6. Atualizar o par (ID, endereço) no índice direto.
6. Se o valor da chave secundária (email) tiver sido alterado, atualizar o par (chaveSecundária, ID) no índice indireto.

Observe que, no passo 5, estamos movendo o registro para o fim do arquivo tanto quanto o registro aumenta de tamanho quanto quando ele diminui. Podemos fazer isso de uma forma mais eficiente, usando alguma estratégia de gerenciamento de espaços excedentes nos registros e usando alguma estratégia para aproveitamento dos espaços vazios deixados por registros movimentados (ou excluídos). Essas duas estratégias ficam como desafios complementares para você. O primeiro deles é apresentado abaixo.

DESAFIO 1: Crie uma estratégia para gerenciar o espaço excedente em cada registro que tiver diminuído de tamanho, evitando que ele precise ser movimentado. Estabeleça um limite percentual máximo para esse espaço (em relação ao tamanho do registro), que, se estourado, deve provocar a movimentação do registro.

Delete

O método `delete()` excluirá o usuário do arquivo. Ele é bastante parecido com o método `update()`, mas sem a preocupação com a escrita de novos dados. A única consideração que devemos fazer é como aproveitar os espaços vazios deixados pelos registros excluídos. Esse é o seu segundo desafio e é apresentado mais abaixo.

O método `delete()` deve receber apenas o ID do objeto que será excluído. Para isso, deve executar os seguintes passos:

1. Buscar o endereço do registro do usuário no índice direto usando o seu ID;
2. Mover o ponteiro no arquivo de usuários para o endereço recuperado;
3. Ler o registro do usuário, para obter a sua chave secundária;
4. Retornar o ponteiro do arquivo para o endereço recuperado no índice;
5. Marcar o campo lápide;
6. Excluir a entrada no índice indireto, usando o valor da chave secundária desse usuário;
7. Excluir a entrada no índice direto, usando o ID desse usuário.

E agora, o seu segundo desafio:

DESAFIO 2: Crie um mecanismo para aproveitar os espaços vazios deixados pelos registros movidos (ou excluídos). Uma forma de se

fazer isso é por meio de uma lista encadeada, ordenada por tamanho, desses espaços, cujo ponteiro para o primeiro espaço deve estar no cabeçalho do arquivo.