Entrada/saída baseada em arquivos e Exceções verificadas[1]

Este material complementa o conteúdo de "Exceções". É preciso visitar os conteúdos anteriores para reconhecer os conceitos e exemplos usados neste material.

A Java API inclui o pacote `java.io`, que contém várias classes para suportar operações de entrada/saída. O pacote define a classe de exceção verificada `IOException` como indicador geral de que algo de errado com uma operação de entrada/saída e quase todas as operações devem prever que uma dessas poderia ser lançada. Instâncias de subclasses verificadas de `IOException` podem ser lançadas de vez em quando para fornecer informações de diagnóstico mais detalhadas, como `EOFException` (final de arquivo) e `FileNotFoundException` (arquivo não encontrado).

1 Leitores e escritores

Várias classes do pacote `java.io` dividem-se em uma de duas categorias principais: aquelas que lidam com arquivos de texto e aquelas que lidam com arquivos binários. Os arquivos de texto são aqueles contendo dados semelhantes ao tipo `char`, em geral, informações alfanuméricas, legíveis por humanos, baseadas em texto simples. Arquivos binários são mais variados: arquivos de imagem são um exemplo comum, da mesma maneira que os programas executáveis, como os processadores de texto e media players. Classes relacionadas a arquivos de texto são conhecidas como *leitores* e *escritores* (*readers e writers*), enquanto aquelas relacionadas a arquivos binários são conhecidas como *rotina de tratamento de fluxo* (*stream handlers*). Aqui veremos os leitores e escritores.

1.1 Saída de texto com `FileWriter`

Há três etapas que fazem parte do armazenamento de dados em um arquivo:

1. O arquivo é aberto.
2. Os dados são gravados.
3. O arquivo é fechado.

A natureza da saída de arquivo significa que qualquer uma dessas etapas poderia falhar, por uma série de razões, muitas completamente fora do controle do programador da aplicação. Como consequência, será necessário antecipar exceções que são lançadas em cada etapa.

[1] Textos extraídos do livro "Programação orientada a objetos com Java" de Michael Kolling e David J. Barnes, capítulo 12.

A fim de gravar um arquivo de texto, é normal criar um objeto `FileWriter`, cujo construtor recebe o nome do arquivo a ser gravado. O nome do arquivo pode ser na forma de uma String ou de um objeto `File`. Criar um `FileWriter` tem o efeito de abrir o arquivo externo e prepará-lo para receber alguma saída. Se a tentativa de abrir o arquivo falhar, por qualquer razão, o construtor então lançará uma `IOException`. As razões da falha talvez sejam que as permissões do sistema de arquivo evitam que um usuário grave em certos arquivos ou que o nome de um dado arquivo não corresponda a uma localização válida no sistema de arquivos.

Quando um arquivo é aberto com sucesso, os métodos `write` do escritor podem ser utilizados para armazenar caracteres - frequentemente na forma de strings - no arquivo. Qualquer tentativa de gravar poderia falhar, mesmo se o arquivo tiver sido aberto com sucesso. Essas falhas são raras, mas possíveis.

Uma vez que toda a saída tenha sido gravada, é importante fechar formalmente o arquivo. Isso assegura que todos os dados sejam realmente gravado no sistema de arquivos externo e frequentemente isso tem o efeito de liberar alguns recursos internos ou externos. Mais uma vez, em raras ocasiões, a tentativa de fechar o arquivo pode falhar.

O código a seguir, exemplo usado na última aula, mostra um método que salva em um arquivo o título dos itens de multimídia que estão armazenados na lista `itens`.

```
/**
* Escreve o título de todos os itens armazenados no parâmetro itens
* em um arquivo
* @param nomeArquivo Nome do arquivo que será usado para armazenar os
* títulos dos itens
* @throws IOException Dispara uma exceção se houver uma falha na escrita
* do arquivo
*/
public void escreveDetalhesBasicos(String nomeArquivo) throws IOException{
    FileWriter escritor = new FileWriter(nomeArquivo);
    for (Item item : itens){ //percorre a lista para acessar todos os itens
        escritor.write(item.getTitulo()); //escreve um item no arquivo
        escritor.write('\n'); //adiciona o caractere final de linha
    }
    escritor.close(); //fecha o escritor
}
```

Neste código foi usada a palavra-chave `throws` no cabeçalho do método. O principal problema que surge é como lidar com qualquer exceção lançada durante as três etapas. Lançar uma exceção ao tentar abrir um arquivo é realmente a única coisa a ser feita. Se uma tentativa de gravar no arquivo falhar, é improvável que repetir essa tentativa será algo bem sucedido. De maneira semelhante, uma falha em fechar um arquivo normalmente não merece uma nova tentativa. A consequência provavelmente será um arquivo incompleto.

A chamada a este método deve ser feita dentro de um bloco *try/catch* para que caso haja uma exceção, ela seja capturada e uma mensagem apropriada seja mostrada. No

código a seguir há uma forma de chamar o método. Este exemplo foi usado em aula, usando os objetos criados no projeto multimídia. Foi criado um objeto `estante` da classe `Database`. O nome dado ao arquivo foi "backup".

```
try {
    estante.escreveDetalhesBasicos("backup");
    System.out.println("A seguinte lista foi salva em arquivo:");
    estante.lista();
} catch (IOException e) {
    System.err.println("Falha ao escrever no arquivo.");
}
```

1.2 Entrada de texto com `FileReader`

O complemento à saída de texto com um `FileWriter` é a entrada com um `FileReader`. Enquanto as unidades naturais para gravar texto são caracteres e strings, as unidades naturais para ler texto são caracteres e linhas. Entretanto, embora a classe `FileReader` contenha um método para ler um único caractere, ela não contém um método para ler uma linha. Por essa razão, assim que é criado, o objeto `FileReader` normalmente é empacotado em um objeto `BuffererReader`, porque esta classe define o método `readline()` que lê uma linha completa. O caractere de término de linha sempre é removido da `String` que `readline()` retorna e um valor `null` é utilizado para indicar o final do arquivo.

O método `lerDetalhesBasicos()` do projeto multimídia mostra uma forma de ler um arquivo e mostra na tela cada linha lida.

```
/**
* Ler o arquivo e mostra as informações no console
* @param nomeArquivo Nome do arquivo que será lido
* @throws FileNotFoundException Dispara uma exceção se o arquivo não for
*  encontrado.
* @throws IOException Dispara uma exceção se houver uma falha na leitura
* do arquivo
*/
public void lerDetalhesBasicos(String nomeArquivo)
                            throws FileNotFoundException,IOException{
  BufferedReader leitor = new BufferedReader(new FileReader(nomeArquivo));
  String linha = leitor.readLine(); //ler uma linha completa do arquivo
  System.out.println(linha); //mostra a linha lida
  while (linha != null) {
      linha = leitor.readLine();
      System.out.println(linha); //mostra a linha lida
  }
  leitor.close(); //fecha o leitor
}
```

Neste código são previstos dois tipos de exceções, a `FileNotFoundException` que identifica especificamente uma falha em encontrar o arquivo solicitado, e a `IOException` que prevê qualquer outra falha relacionada à leitura, abertura e fechamento do arquivo. Assim como no exemplo anterior, a palavra-chave `throws` é usada no cabeçalho do método.

A chamada do método deve estar dentro de um bloco *try/catch* para capturar e mostrar uma mensagem de erro adequada. O método tenta ler o arquivo "backup".

```
try {
    estante.lerDetalhesBasicos("backup");

} catch (FileNotFoundException e) {
    System.err.println("Arquivo não encontrado.");
} catch (IOException e) {
    System.err.println("Falha ao ler o arquivo.");
}
```