Usando I/O

Lendo e gravando dados Binários

- Até agora, lemos e gravamos apenas bytes contendo caracteres ASCII, mas é possível – na verdade, comum – ler e gravar outros tipos de dados.
- Por exemplo, poderíamos querer criar um arquivo contendo **int**s, **double**s ou **short**s. Para ler e gravar valores binários de tipos primitivos Java, usaremos **DataInputStream** e **DataOutputStream**.
- O exemplo da pg 343 demonstra **DataInputStream** e **DataOutputStream**. Ele grava e depois lê vários tipos de dados em um arquivo.

Arquivos de acesso aleatório

- Java também nos permite acessar o conteúdo de um arquivo em ordem aleatória.

- Para fazer isso, usaremos **RandomAccessFile**, que encapsula um arquivo de acesso aleatório.

- Para fazer isso, usaremos **RandomAccessFile** não é derivada de **InputStream** ou **OutputStream**.

Arquivos de acesso aleatório

- Em vez disso, ela implementa as interfaces **DataInput** e **DataOutput**, que definem os métodos básicos de I/O.
- **RandomAccessFile** implementa os métodos **read()** e **write()**.
- Também implementa as interfaces **DataInput** e **DataOutput**, ou seja, métodos de leitura e gravação ds tipos primitivos, como **readIn()** e **writeDouble()**, estão disponíveis.
- O exemplo da pg. 347 demonstra arquivos de acesso aleatório.

Entrada de fluxo pelo console

- Para códigos que são internacionalizados, obter entradas de console com o uso de fluxos Java baseados em caracteres, é uma maneira melhor e mais conveniente de ler caracteres no teclado do que usar os fluxos de bytes.
- No entanto, já que **System.in** é um fluxo de bytes, você terá que encapsulá-los em algum tipo de **Reader**. A melhor classe para a leitura de entradas de console é **BufferedReader**, que dá suporte a um fluxo de entrada armazenada em bufffer.

Entrada de fluxo pelo console

- Contudo, você não pode construir um **BufferedReader** diretamente a partir de **System.in**.
- Em vez disso, primeiro deve convertê-lo em um fluxo de caracteres.
- Para fazê-lo usar **InputStreamReader**, que deve converter bytes em caracteres.
- Para obter um objeto InputStreamReader, vinculado a System.in, use o construtor:

InputStreamReader(InputStream *fluxoEntrada*)

Lendo Strings

- Para ler um string no teclado, use a versão de **readLine()** que é membro da classe **BufferReader**. Sua forma geral é:

```
String readLine() throws IOException
```

- Ela retorna um objeto String contendo caracteres lidos.
- Quando é feita uma tentativa de leitura no fim do fluxo, retorna nulo.
- Exemplo da pg. 352.

Fluxo de Caracteres

- Embora ainda seja permitido usar **System.out** em Java para gravações no console, seu uso mais recomendado é para fins de depuração ou para exemplos como os que vimos.
- Para programas do mundo real, o melhor método de gravação no console quando se usa Java é com um fluxo **PrintWriter.**
- **PrintWriter** é uma das classes baseadas em caracteres.
- Exemplo da pg. 353.

I/O de arquivo

- A vantagem dos fluxos de caracteres é que eles operam diretamente sobre os caracteres Unicode.
- Logo, se quisermos armazenar texto Unicode, certamente os fluxos de caracteres serão a melhor opção.
- Em geral, para executar I/O de arquivo baseada em caracteres, usamos as classes **FileReader** e **FileWriter**.

I/O de arquivo

- **FileWriter** cria um objeto Writer que podemos usar para fazer gravações em um arquivo.

- Exemplo da pg. 354.

- A classe **FileReader** cria um objeto **Reader** que pode ser usado na leitura do conteúdo de um arquivo.

- Exemplo da pg. 355.

Usando os encapsuladores

- Antes de finalizarmos I/O, examinaremos uma técnica útil na leitura de strings numéricos.
- Os encapsuladores de tipos Java são classes que encapsulam, ou empacotam, os tipos primitivos.
- Eles são necessários porque os tipos primitivos não são objetos. Isso limita seu uso.
- Exemplo da pg. 357.
- Exercício da pg. 359