Ficha Prática 10

António Nestor Ribeiro, António Fernandes, F. Mário Martins
{anr,af,fmm}@di.uminho.pt
PPIV (LESI)

2003/04

Objectivos

1. Input e Output em Java.
2. Streams Java para sequências de caracteres e de bytes.
3. As classes `java.io.Writer`, `java.io.Reader`, `java.io.OutputStream` e `java.io.InputStream`.
4. ObjectStreams como mecanismos de manipulação de objectos.
5. Exercícios.

Input e Output em Java

Em Java existe um conceito - *stream* - que abstrai as diferentes formas que podem assumir a leitura e escrita de dados.

Uma stream é uma abstracção para uma qualquer forma de entrada de dados ou escrita de dados. Permite que o programador se isole das particularidades de estar a escrever ou ler, de um ficheiro, de uma porta numa outra máquina, entre outras situações.

As streams podem ser catalogadas em dois grandes grupos:

- streams de caracteres (streams de texto)
- streams de bytes (streams binárias)

Do primeiro grupo interessa ao programador conhecer as sub-hierarquias do package `java.io` que se encontram delimitadas pelas classes `java.io.Reader` (para streams de leitura) e `java.io.Writer` (para streams de escrita).

Streams de texto

Para escrever e ler de streams de texto existem várias possibilidades. No entanto para simplificar a escolha e ter um bom desempenho sugere-se a utilização das classes:

- `FileWriter` - classe de conveniência para escrever ficheiros de texto. Oferece métodos normais como `write` para escrever sequências de caracteres, `flush` e `close` para gerir a disponibilidade do output.
- `FileReader` - comportamento idêntico ao da classe acima, mas para leitura.

A utilização destas classes para escrita e leitura de informação resultaria em código de baixo nível, uma vez que a gestão do interface com as fontes de informação teria de ser feita pelo utilizador.

Por isso utilizam-se classes com interface mais optimizado como as existentes em:

- `BufferedWriter` - permite a escrita de texto, faz a gestão do buffer (logo optimizando os tempos de acesso) e disponibiliza métodos para a escrita de caracteres, arrays e strings.
- `BufferedReader` - permite a leitura de texto de uma stream de texto, efectuando a gestão do buffer de caracteres. Oferece métodos para ler eficientemente caracteres, arrays e linhas de texto.

Streams binárias

Para escrever e ler sequências de bytes, guardando os valores em formato binário existem várias possibilidades. As sub-hierarquias de classes do package `java.io` que começam em `OutputStream` e `InputStream` disponibilizam várias classes para esse efeito.

Para eficientemente guardarmos objectos em formato binários precisamos apenas de utilizar as seguintes classes:

- `FileOutputStream` - permite criar uma stream de escrita, para escrever sequências de bytes.
- `FileInputStream` - uma instância desta classe serve para obter bytes de um File existente. A sua escolha deve ser feita sempre que se pretender ler bytes e não caracteres (caso em que se deveria utilizar uma instância de `FileReader`).

Tal como nas streams de texto, é necessário agora escolher classes que possibilitem um maior nível de abstracção na manipulação da escrita e leitura de bytes. Para tal utilizam-se as classes:

- `ObjectOutputStream` - escreve para uma stream binária objectos com toda a informação que permite posteriormente reconstruí-los. Para que seja possível a um objecto ser escrito a classe respectiva deve implementar a interface `Serializable`.

  Exemplo de utilização:

  ```
  FileOutputStream fos = new FileOutputStream("t.tmp");
  ObjectOutputStream oos = new ObjectOutputStream(fos);

  oos.writeInt(12345);
  oos.writeObject("Today");
  oos.writeObject(new Date());

  oos.close();
  ```

  O método mais importante desta classe é: `public final void writeObject(Object obj) throws IOException` , pois permite gravar um objecto de uma só vez.

- `ObjectInputStream` - uma instância desta classe lê dados e objectos gravados utilizando uma `ObjectOutputStream`. Só é possível ler objectos cujas classes implementem a interface `Serializable`.

  Exemplo de utilização:

  ```
  FileInputStream fis = new FileInputStream("t.tmp");
  ObjectInputStream ois = new ObjectInputStream(fis);

  int i = ois.readInt();
  String today = (String) ois.readObject();
  Date date = (Date) ois.readObject();

  ois.close();
  ```

  O método mais importante que esta classe oferece é: `public final Object readObject() throws IOException, ClassNotFoundException` , pois permite ler um objecto de uma só vez.

Exercícios

1. Crie na classe `AccaoBolsa` um método que permita gravar a informação de uma acção num fincheiro de texto. O método deve aceitar o nome do ficheiro como parâmetro.

2. Como faria agora para a partir de um ficheiro de texto com a informação de uma `AccaoBolsa` reconstruir o objecto?

3. Crie na classe `CarteiraAccoes` um método que permita guardar em ficheiro de texto toda a informação da carteira.

4. Crie na classe `CarteiraAccoes` os métodos que permitem

   (a) gravar em ficheiro de objectos a informação da carteira

   (b) ler de ficheiros de objectos uma carteira de accções anteriormente gravada.

5. Faça os métodos que permitam gravar e ler de/para ficheiros de objectos uma instância de `Corretora`.