Aula Prática 5

Resumo:

- Robustez e Excepções.

Exercício 5.1

O programa `TestInput` utiliza funções para leitura interativa de valores reais. Experimente-o. Se o utilizador introduzir uma linha mal formatada, o programa termina com um `NumberFormatException`.

Altere as funções na classe `util.Input` para as tornar robustas. Mesmo que o utilizador introduza dados mal formatados, a função deverá avisar e voltar a pedir. Pode observar o comportamento pretendido com o programa `TestInput.jar`.

```
Real value X? pi
Invalid input format!
Real value X? 1 2
Invalid input format!
Real value X? 1.2
1.2
Nota? -1
Value must be in range [0.000000, 20.000000]
Nota? vinte
Invalid input format!
```

Acrescente a pré-condição adequada à função `readDouble(String, double, double)`. Descomente a última instrução no programa para testar que a asserção falha.

Exercício 5.2

Altere o programa `PlayGuessGame` da aula anterior por forma a torná-lo totalmente robusto. Em particular, o programa deve garantir a sanidade dos argumentos e dos valores introduzidos pelo utilizador.

Exercício 5.3

Crie um programa `CopyFile` que copie um ficheiro de texto. Os nomes dos dois ficheiros envolvidos devem ser dados como argumentos na linha de comandos.[1] Por exemplo, `java -ea CopyFile Texto1.txt Texto2.txt` deve criar um ficheiro `Texto2.txt` com um

[1] Pretende-se que seja uma versão simplificada do comando `UNIX cp`.

conteúdo igual ao do ficheiro `Texto1.txt`. Se `Texto2.txt` já existir, deve perguntar ao utilizador se deseja reescrever o seu conteúdo.

O programa deve ser robusto. Deve detetar falhas e apresentar mensagens de erro apropriadas. Várias condições têm de estar reunidas para que o programa funcione. O ficheiro original tem de existir, tem de ser um ficheiro normal e o utilizador tem de ter permissão para o ler. Se o ficheiro de destino existir, tem de ter permissão de escrita. Se não existir, então tem de ter permissão de escrita no diretório pai. Mesmo assim, podem ocorrer falhas imprevisíveis, como erros no disco ou de comunicação, que devem ser reportadas.

Exercício 5.4

Crie um programa `ListDir` que faça uma listagem de um diretório com alguma meta-informação dos seus ficheiros. O nome do directório é passado como argumento do programa. Se for chamado sem argumentos, o programa deve listar o directório actual (`.`). Para cada ficheiro, deve ser indicado se é um ficheiro normal ou um directório (`F` ou `D`) e se tem permissão de leitura ou não (`R` ou `-`) e se tem permissão de escrita ou não (`W` ou `-`), como se vê no exemplo abaixo.

```
DRW  ./dir1
DR-  ./dir2
F-W  ./file1
FRW  ./file2
```

Neste programa deve usar os métodos adequados da classe `File`, em particular: `listFiles()`, `getPath()`, `isDirectory()`, `canRead()`, `canWrite()` e outros. Recorra à documentação dessa classe para perceber a sua utilização.

Para testar o programa, pode criar pastas e ficheiros e alterar as suas permissões com os comandos UNIX: `mkdir`, `touch` e `chmod`.

Exercício 5.5

Construa um programa `CutColumn` que, dado um ficheiro de texto, escreva somente a n-ésima palavra de cada linha do texto. Quando a linha não tem n palavras, deve escrever uma linha em branco. Considere que as palavras são separadas por um ou mais espaços ou caracteres de tabulação. (Pode usar `line.split("[ \t]+")` para dividir cada linha.)

Quer o nome do ficheiro quer o número da coluna a extrair devem ser dados como argumentos do programa.

Por exemplo, no caso de termos o seguinte texto de entrada:

```
1 2 3 4 5 6
    boa     tarde
um dois tres quatro cinco seis
```

A saída do programa, se for escolhida a coluna número `4`, deverá ser:

```
4

quatro
```