Aula Prática 10

Resumo:
- Pilhas (*stacks*) e filas (*queues*).

Exercício 10.1

Faça um programa que detecte se uma sequência de letras e dígitos é um palíndromo (ou seja, se a sequência lida do início para o fim é igual à sequência lida do fim para o início). O programa deve ignorar todos os caracteres que não sejam letras ou dígitos, assim como ignorar a diferença entre maiúsculas e minúsculas. Por exemplo as frases: `"somos"` e `"O galo nada no lago"` são palíndromos. Utilize uma *pilha* e uma *fila* para resolver este problema.

Exercício 10.2

Escreva um programa que permita visualizar, passo a passo, a resolução do problema das Torres de Hanói. Para esse fim utilize uma *pilha* (acrescida de um método apropriado de escrita de toda a pilha numa linha, indo da base para o topo) como representação interna de cada torre. A escrita das torres pode ter o seguinte aspecto:

```
After 0 moves:          After 1 moves:          After 2 moves:
TA: [5, 4, 3, 2, 1]     TA: [5, 4, 3, 2]        TA: [5, 4, 3]
TB: []                  TB: [1]                 TB: [1]
TC: []                  TC: []                  TC: [2]
```

Para poder visualizar a resolução do problema passo a passo utilize, por exemplo, uma instrução de leitura de uma linha `in.nextLine()` (ignorando essa entrada).

Sugestão: A resolução do problema fica facilitada se criar um módulo `HanoiTowers` (inicializado com o número de discos n), com três campos (internos) do tipo pilha (um para cada torre), e com um serviço de escrita de todas as torres (para além, é claro, do serviço que resolve o problema, cujo algoritmo é idêntico ao feito na aula de recursividade).

Exercício 10.3

As técnicas de processamento digital de sinais usam frequentemente uma estrutura de armazenamento chamada *linha de atraso* (*digital delay line*), que é essencialmente uma fila

de tamanho fixo que permite acesso direto (de leitura) a qualquer dos elementos. Quando se introduz um novo elemento, o mais antigo é automaticamente descartado. Inicialmente, todos os elementos da linha de atraso têm um valor pré-definido (geralmente zero).

O programa `P103.java` usa uma linha de atraso para mostrar a temperatura média das últimas 24 horas, hora-a-hora ao longo de vários dias.

a. Complete a classe `DelayLine` com os métodos necessários para que o programa funcione corretamente.

b. Uma vez que a linha de atraso requer acesso aleatório e o tamanho é fixo, é muito vantajoso implementá-la com a técnica de array circular. Crie uma classe `DelayArray` com a mesma interface, mas usando essa implementação. Teste o novo módulo num programa `P103b`.

Exercício 10.4

Construa uma calculadora com as quatro operações aritméticas básicas que funcione com a notação pós-fixa (*Reverse Polish Notation*). Nesta notação os operandos são colocados antes do operador. Assim `2 + 3` passa a ser expresso por `2 3 +`. Esta notação dispensa a utilização de parênteses e tem um implementação muito simples assente na utilização de uma pilha de números reais. Sempre que aparece um operando (número) ele é carregado para a pilha. Sempre que aparece um operador (binário), são retirados os dois últimos números da pilha e o resultado da operação é colocado na PILHA. Se a pilha não tiver o número de operandos necessário, então há um erro sintáctico na expressão.

O programa deve ler os operandos e os operadores como palavras (strings separadas por espaços) lidas do *standard input*. Exemplo de utilização:

```
$ echo "1 2 3 * +" | java -ea P104
Stack: 1.0
Stack: 1.0 2.0
Stack: 1.0 2.0 3.0
Stack: 1.0 6.0
Stack: 7.0
```

Exercício 10.5

Um dos problemas que os vectores em Java podem colocar reside no facto de o seu número de elementos ser imutável (uma vez criado o vector). Essa limitação faz com que, na presença de problemas onde a dimensão máxima do vector possa ter de variar em tempo de execução, sejamos obrigados a tirar cópias do vector (para um novo vector de dimensão adequada), ou a utilizar outros módulos mais flexíveis.

Neste exercício pretende-se desenvolver um módulo `BlockArrayInt` com uma interface tipo vector (acesso aleatório aos elementos por intermédio de um índice inteiro), mas em que a representação interna desse vector assenta numa lista ligada de blocos, sendo cada bloco um vector (primitivo) de dimensão fixa (definida no construtor do módulo). Se

o número de elementos por bloco for `blockSize`, então os índices de `[0,blockSize-1]` apontarão para dentro do primeiro bloco (primeiro elemento da lista ligada), os índices `[blockSize,2*blockSize-1]` apontarão para o segundo, etc..

Para servir de teste ao módulo, o programa `P105.java` armazena (num objecto do tipo `BlockArrayInt`) e escreve todos os números primos encontrados até um valor colocado pelo utilizador como argumento do programa. Nesse ficheiro pode também encontrar a interface desejada para o módulo, mas onde a implementação está incompleta (apenas define um bloco).

Comece por experimentar o programa com valores menores do que 233 para perceber o funcionamento do programa. Caso utilize um valor acima de 232 verificará que ocorre um erro em tempo de execução. Implemente o módulo usando a representação interna acima descrita, por forma a que o programa funcione para qualquer valor numérico (quando o módulo estiver a funcionar experimente, por exemplo, utilizar o valor 1000000).