Ficha de Exercícios

Exercícios de introdução

1. Escreva uma aplicação que gere um inteiro aleatório entre 0 e 100, inclusive, e que permita ao utilizador adivinhar o número gerado, através de indicação sucessiva de valores.

2. Escreva uma aplicação que peça ao utilizador para pensar num número entre 1 e 100, inclusive. Através de várias perguntas ao utilizador, o qual responderá 'ACERTOU', 'PENSEI NUM NÚMERO MAIOR' ou 'PENSEI NUM NÚMERO MENOR', a aplicação tentará adivinhar o número. No final deve ser apresentado o número de tentativas que foram necessárias para adivinhar.

3. Escreva uma aplicação de consola que crie um *array* de inteiros, perguntando ao utilizador a dimensão do *array* e quais os valores dos seus elementos. Com base no *array* criado desenvolva funções *static* para...

a) Imprimir o *array*.
b) Calcular e imprimir o maior e o menor elemento presentes no *array*.
c) Somar todos os elementos do *array*, retornando a soma.
d) Calcular e imprimir a média dos valores do *array*.
e) Inverter todos os valores presentes no *array*, $[i_0, i_1, ..., i_{n-1}] => [i_{n-1}, ..., i_1, i_0]$

4. Defina uma classe que contenha, como membro, um *array* de inteiros com 20 elementos. Os valores a incluir neste *array* são inteiros aleatórios entre 0 e 100, inclusive. Um inteiro gerado aleatoriamente apenas deve ser colocado no *array* caso esse valor ainda não esteja armazenado, isto é, o *array* não deve conter valores repetidos. Usar uma função membro booleana (que devolve um valor booleano) para verificar se um determinado valor existe no *array*. Para cada valor gerado, deverá ser chamada a função para verificar se o mesmo já existe no *array*. Se ele existir, o valor deve ser descartado e deve ser gerado um novo valor.
A classe deve ter funções para:

a) Listar os 20 valores do *array*.
b) Mostrar a quantidade de valores que foram gerados em duplicado, ou seja, a diferença entre o total de valores gerados e o número de valores aproveitados (no caso concreto serão 20).

5. Defina uma classe, `Aposta`, que represente uma aposta no Totoloto. Uma aposta é definida por cinco números inteiros compreendidos entre 1 e 49, todos diferentes entre si e não necessariamente introduzidos de forma crescente, e um "número da sorte" compreendido entre 1 e 13 (pode ser igual a um dos 5 números indicados anteriormente). A classe quando é construída não possui qualquer número "preenchido", devendo ser disponibilizados métodos para:

a) Preencher um número de cada vez com vista à construção da aposta. O método deve possuir dois parâmetros: o primeiro é o número a associar à aposta, garantindo que não há valores repetidos, o segundo será um valor booleano para indicar se o número pertence à aposta principal ou se corresponde ao "número da sorte".

b) Verificar se a aposta está completa (cinco números e "número da sorte").
c) Preencher automaticamente uma aposta completa.
d) Comparar a aposta corrente com uma chave ganhadora. A chave sorteada é passada através de dois parâmetros: array de 5 valores inteiros e um valor adicional correspondente ao "número da sorte".
e) Fazer *reset* da aposta, eliminando todos os números eventualmente introduzidos até então.

Crie uma aplicação que utilize esta classe, permitindo a criação de instâncias da classe `Aposta` e a utilização de todas funcionalidades referidas nas alíneas anteriores.

6. Escreva uma aplicação que calcule e imprima a transposta de uma matriz. A matriz deverá ser representada através de classe adequada e deve possuir métodos adequados para as funcionalidades referidas.

7. Escreva uma aplicação que some matrizes retangulares. Uma matriz deverá ser representada através de uma classe adequada. A soma deverá ser realizada através de duas formas distintas:
a) Função membro que acumulará aos valores de uma matriz os valores de outra matriz.
b) Função estática que recebe duas matrizes e retornará uma nova matriz com o resultado da soma, ou seja, não se pretende que sejam alteradas as matrizes originais.

8. Defina uma classe que contenha, como um membro, uma matriz de *m x n* elementos. Esta classe deve ter funções para alterar os elementos da matriz, calcular a soma de cada linha, calcular a soma de cada coluna, calcular a soma de todos os seus elementos e imprimir toda a informação.
Exemplo de output para uma matriz definida programaticamente:

```
Matriz:
1    0    2    -1   3
4    3    2    1    0
1    -2   3    4    5
8    5    1    3    2

Soma das linhas: 5, 10, 11, 19
Soma das colunas: 14, 6, 8, 7, 10
Soma total: 45
```

9. Escreva uma aplicação que calcule e imprima o triângulo de Pascal. O triângulo de Pascal deverá ser representado através de uma classe própria, que deverá incluir funções para: gerar triângulo com uma determinada profundidade, gerar uma *String* representativa do triângulo (`toString()`) e mostrar o triângulo na consola. Quando o objeto for criado poderá ser indicado através do *construtor* a profundidade pretendida.

Estruturação de programas e Coleções de dados

10. Escreva uma aplicação que implemente o jogo do enforcado. Na implementação do jogo deverá garantir uma separação entre o código referente a interação com o utilizador e as classes necessárias para realizar a sua gestão. O código deverá ser estruturado nas seguintes classes:

- `JogoEnforcado`, que inclui (apenas) a função `main`;
- `JogoEnforcadoDicionario`, que permite disponibilizar as palavras a usar no jogo. Deverá possuir apenas membros estáticos para armazenar uma tabela com as palavras, e métodos para acesso à quantidade de palavras armazenadas e a uma palavra (através do seu índice);
- `JogoEnforcadoModelo`, que efetuará a gestão de todo o jogo (sem qualquer código de interação com o utilizador). Deverá conter métodos que permitem iniciar jogo (sorteando uma nova palavra), tentar/verificar uma letra, verificar fim de jogo, gerir informação de evolução de jogo: número de tentativas, letras já tentadas, ...;
- `JogoEnforcadoIU`, onde será implementada toda a interação com o utilizador.

Exemplo da classe `JogoEnforcado`:

```
public class JogoEnforcado {
    public static void main(String args[]) {
        JogoEnforcadoModelo jogo = new JogoEnforcadoModelo();
        JogoEnforcadoIU jogoIU = new JogoEnforcadoIU(jogo);
        jogoIU.jogar();
    }
}
```

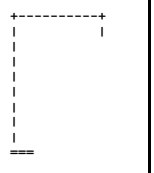
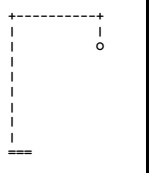
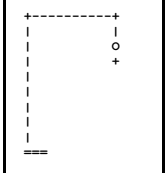
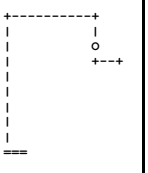
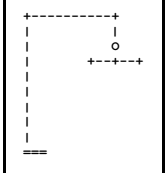
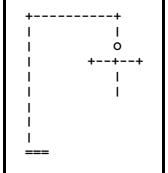
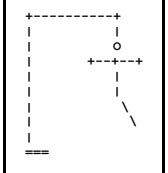
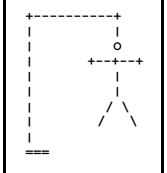
Exemplo de interação do programa:

```
Situação atual: ....
Nr. tentativas: 0
Erros: 0 (máx.: 7)
Caracteres já tentados:

Tentativa: U

Situação atual: ....
Nr. tentativas: 1
Erros: 1 (máx.: 7)
Caracteres já tentados: U

Tentativa: A

Situação atual: ..A.
Nr. tentativas: 2
Erros: 1 (máx.: 7)
Caracteres já tentados: UA

Tentativa: E
```

```
Situação atual: .EA.
Nr. tentativas: 3
Erros: 1 (máx.: 7)
Caracteres já tentados: UAE

Tentativa: O

Situação atual: .EAO
Nr. tentativas: 4
Erros: 1 (máx.: 7)
Caracteres já tentados: UAEO

Tentativa: P

Situação atual: .EAO
Nr. tentativas: 5
Erros: 2 (máx.: 7)
Caracteres já tentados: UAEOP

Tentativa: L
Parabéns! Acertou na palavra LEAO em 6 tentativas
```

Desafio:

Desenho o "enforcado" a ser desenhado de acordo com a evolução mostrada na tabela seguinte:

0 Erros	1 Erro	2 Erros	3 Erros	4 Erros	5 Erros	6 Erros	7 Erros
							

11. Defina uma classe para representar a informação acerca de um relatório. Um relatório (instância da classe `Report`) é definido através dos seguintes campos:

- *Titulo* (`String`)
- *Conjunto de autores* (1ª versão: array de *strings*; 2ª versão: `ArrayList` de *strings*)
- *Texto* (`StringBuilder` ou `StringBuffer`)

A classe deve disponibilizar as seguintes funcionalidades:

- Acrescentar um autor ao relatório garantindo que não existem repetidos.
- Remover um autor, garantido a manutenção da ordem dos restantes.
- Acrescentar texto. O texto será concatenado ao final do texto já existente.
- Substituir por letras maiúsculas as primeiras letras das palavras depois de pontos finais.
- Contar as palavras do texto (as palavras podem estar separadas por mais do que um separador: espaços, tabs, mudanças de linha, vírgulas e pontos). A função deverá retornar o número de palavras.
- Contar as ocorrências de uma dada palavra. A função deverá retornar o número de ocorrências ou 0 (zero) caso a palavra não exista no texto.
- Gerar `String` com toda a informação do relatório (método `toString()`).

Implemente uma aplicação Java que permita testar todas as funcionalidades da classe `Report`.

12. Pretende-se uma aplicação para gerir os produtos de uma fábrica.

a) Os produtos, representados através de uma classe `Produto`, são identificados por um número de série (inteiro) que deve ser único e não deve poder ser modificado a partir do exterior da classe. São ainda caracterizados pela data de fabrico e também por um estado (*string*) que inicialmente possui o valor "`não testado`". Um objecto da classe `Produto` só pode ser criado com a indicação do seu número de série. A data de fabrico corresponde à data do momento de criação do objecto. Esta classe deverá ter um método booleano `testaUnidade()` que simula o controlo de qualidade. Quando esta função é invocada por um objeto, se este estiver no estado "não testado", em 90% dos casos estará OK (o estado passará de "`não testado`" para "`aprovado`"). Caso não esteja OK, o estado passará para "`reprovado`". A função `testaUnidade()` não altera o estado caso este seja "`aprovado`" ou "`reprovado`", retornando `true` se o estado é aprovado e `false` se não é. As variáveis-membro devem ser privadas, podendo ser acedidas através de funções *get* e *set*. Nesta classe deve também implementar (redefinir) as funções:
   i. `toString()` que retorna uma *string* com a descrição do objeto;
   ii. `equals()` que representa o critério de identificação de um produto (dois produtos são o mesmo se tiverem o mesmo número de série);
   iii. `hashCode()` que retorna um *hash code* adequado para o objeto.

b) Defina a classe `Fabrica` que gere um conjunto de produtos. Para além de uma coleção de produtos, a fábrica tem um nome e o número de produtos criados até ao momento (que pode ser diferente do número de elementos da coleção de produtos, uma vez que podem já ter sido alguns eliminados). Ao ser criado um objeto da classe

`Fabrica` deve ser dado o seu nome, ficando, à partida, sem registo de nenhum produto. Esta classe deve ter as seguintes funções:

i. `acrescentaProduto()` que cria o produto e acrescenta-o à fabrica;
ii. `pesquisaProduto()` que recebe o número de série de um produto e retorna uma referência para o produto se o encontrar ou `null` se não encontrar;
iii. `eliminaProduto()` que recebe o número de série do produto, eliminando-o se o encontrar. Retorna o valor lógico do sucesso desta operação;
iv. `eliminaReprovados()` que elimina todos os produtos cujo estado é "`reprovado`";
v. `testaUnidades()` que faz o controlo de qualidade a todos os produtos da fábrica;
vi. `toString()` que retorna uma *string* com a descrição do objeto.

c) Desenvolva uma função `main` que permita testar as funcionalidades desenvolvidas nas alíneas anteriores.

Coleções de dados: List, Set e Map

13. Pretende-se uma aplicação para gerir os livros de uma biblioteca. Os livros são identificados por um código (um número inteiro positivo que representa a ordem de criação do registo dos livros na biblioteca). O registo de um livro, para além do referido código, tem obrigatoriamente informação sobre o título e os autores.

a) Defina a classe `Book` que representa este conceito de registo de um livro nesta biblioteca. Deve ser possível criar objectos da classe `Book`, dando informação sobre o título e autores, sendo, o código gerado automaticamente. As variáveis-membro devem ser privadas, podendo ser acedidas através de funções *get* e *set*. A variável-membro código não deve poder ser modificado a partir do exterior da classe. Nesta classe deve também implementar as funções:
   i. `toString()` que retorna uma *string* com a descrição do objeto;
   ii. `equals()` que representa o critério de identificação de um livro (dois livros são o mesmo se tiverem o mesmo código);
   iii. `hashCode()` que retorna o *hash code* do objeto.
   iv. *`getDummyBook()`* método estático que recebe o código de um livro e retorna uma instância da classe `Book` com o código em causa, mas sem titulo e sem autores (`null`).

b) Defina a classe `Library` que representa uma biblioteca que possui um conjunto de livros, geridos com o auxílio de um objecto `ArrayList`. Para além dos livros, a biblioteca tem um nome. Ao ser criado um objeto da classe `Library` deve ser dado o seu nome, ficando, à partida, sem registo de qualquer livro. Esta classe deve ter as seguintes funções:
   i. `addBook()` que recebe toda a informação que permite criar o registo de um livro, cria o registo e acrescenta-o à biblioteca. Deverá retornar o código do livro adicionado;
   ii. `findBook()` que recebe o código de um livro e retorna uma referência para o livro se o encontrar ou `null` se não encontrar. 1ª versão: pesquisa iterativa do livro. 2ª versão: utilização do método `indexOf()` do `ArrayList`;
   iii. `removeBook()` que recebe o código do livro, eliminando-o se o encontrar. Retorna o valor lógico do sucesso desta operação. 1ª versão: pesquisa iterativa do livro. 2ª versão: utilização do método `indexOf()` do `ArrayList`;
   iv. `toString()` que retorna uma string com a descrição do objeto.

c) Desenvolva uma função `main` que permita testar as funcionalidades desenvolvidas nas alíneas anteriores.

d) Desenvolva uma nova versão da classe `Library` recorrendo a um objeto `HashSet` para gerir o conjunto de livros.

e) Desenvolva uma nova versão da classe `Library` recorrendo a um objeto `HashMap` para gerir o conjunto de livros.