FACULDADE · DE · CIÊNCIAS UNIVERSIDADE · DE · LISBOA

INTRODUÇÃO À PROGRAMAÇÃO
Exercícios 2008/2009

1 Fundamentos de Programação Imperativa

1.1 Tipos, Operadores, Variáveis e Expressões

1. Para cada uma das seguintes expressões, indique a ordem de avaliação dos vários operadores, escrevendo um número debaixo de cada um.

   (a) `a - b - c - d`

   (b) `a - b + c - d`

   (c) `a + b / c / d`

   (d) `a / b * c * d`

   (e) `a % b / c * d`

   (f) `(a - (b - c)) - d`

   (g) `a % (b % c) * d * e`

   (h) `(a + b) * c + d * e`

   (i) `(a + b) * (c - d) % e`

2. Considere a seguinte lista de declaração de variáveis.

   ```
   int a = 3;
   double d = 2.19;
   ```

   Determine o tipo e o valor de cada uma das seguintes expressões:

   (a) `a + 3 * a`

   (b) `(a + 3.0) * a`

   (c) `45 - a + 23`

   (d) `3.24 + a * 3`

   (e) `2 * 5.0 / a + 3`

   (f) `2 * 5 / a + 3`

   (g) `4 - d + a / 2`

   (h) `(d + 2) / a`

3. Assuma que `i` e `j` são duas variáveis de tipo numérico e que `b` é uma variável do tipo `boolean`. Elimine os parênteses desnecessários de cada uma das seguintes expressões.

   (a) `((3 * i) + 4) / 2`

   (b) `((3 * j) / (7 - i)) * (i + (-23 * j))`

   (c) `((((i + j) + 3) + j) * (((i - 4) / j) + -323))`

   (d) `(3 >= (j - 3)) == ((323 - (j * -7)) != (43))`

   (e) `((3 >= 5) == (!b || b))`

   (f) `(b || (!(b && (3 == (i * 2)))))`

   (g) `(!(!b) | (b && ((4 >= i+j) || (false))))`

4. Para cada um dos seguintes excertos de código, determine o valor de cada variável no final da execução.

   (a)
   ```
   int j = 2, i = 1;
   j = 3 + i * 2;
   i = j / 2 * i + 3;
   i = i + 1;
   ```

(b)
```
int i = 3;
double d = 3.0;
d = d - 2.3;
i = (int) d;
```

(c)
```
double b = 3.1, c = 0.0;
c = c + 2.0;
b = b * (c + 3.0);
int i = (int) (c + b);
i = i - 1;
```

(d)
```
int x;
int y = 4;
x = y + y;
```

(e)
```
int x = 5;
int y = x;
x = x + y;
```

(f)
```
int x;
int y = 4, z = 3;
x = y / z;
```

5. Suponha que vamos precisar de tratar os seguintes dados:

   - uma idade;
   - um peso;
   - um número de lotaria;
   - um salário mensal;
   - o género (masculino ou feminino) de uma pessoa;
   - o estado civil de uma pessoa (solteira, casada, divorciada, viúva);
   - uma distância entre corpos celestes medida em anos-luz;
   - uma distância terrena medida em metros.

   Discuta os nomes a dar a variáveis para armazenarem estes tipos. Para cada uma, escolha o tipo adequado e discuta se faz sentido inicializá-la aquando da declaração.

1.2 Programação sobre Números

1. Escreva um método `void imprimeMultiplos()` que imprima no monitor os múltiplos de 7 menores que 500.
2. Escreva um método `void imprimeMultiplos(int n)` que imprima no monitor os múltiplos de 7 menores que `n`.
3. Escreva um método `void imprimeMultiplos(int k, int n)` que imprima no monitor os múltiplos de `k` menores que `n`.
4. Escreva um método `void asteriscos(int n)` que imprima no monitor uma linha com `n` asteriscos.
5. Escreva um método `void linhasAsteriscos(int n)` que imprima no monitor linhas com 1, 2, ..., `n` asteriscos. Por exemplo, para `n`=5 o resultado deve ser o seguinte.

   ```
   *
   **
   ***
   ****
   *****
   ```

6. Escreva um método `int soma(int n)` para calcular a soma dos números naturais menores que `n`.
7. Escreva um método `int somaExcede(int k)` para encontrar o menor `n` tal que a soma dos números naturais menores que `n` excede `k`.
8. Escreva um método `int somaEntre(int m, int n)` para calcular a soma dos números naturais maiores que `m` e menores que `n`.

9. Escreva um método `int somaPares(int n)` que devolva o somatório de todos os números pares menores que `n`.

10. Escreva um método `int factorial(int n)` que devolva o factorial de `n`.

11. Escreva um método `int factorialDuplo(int n)` que devolva `n!!` ($n!! = 1 \times 3 \times 5 \times \cdots \times n$, se `n` for ímpar, e $n!! = 2 \times 4 \times 6 \times \cdots \times n$ se n for par).

12. Escreva um método `int logaritmo(int n)` que devolva o logaritmo inteiro de `n` na base 2.

13. Escreva um método `int contaDivisores(int n)` que devolva o número de divisores de `n`.

14. Um número perfeito é um número que é igual à soma dos seus divisores próprios. Por exemplo, 6 é um número perfeito: os seus divisores próprios são $\{1, 2, 3\}$ e $1 + 2 + 3 = 6$.

    Escreva um método `boolean ePerfeito(int n)` que verifique se o número `n` é perfeito.

15. Escreva um método `int contaPerfeitos(int n)` que devolva o número de números perfeitos menores que `n`.

16. Escreva um método `boolean ePrimo(int n)` que indique se `n` é primo ou não.

17. Escreva um método `int contaPrimos(int n)` que devolva o número de números primos menores que `n`.

18. Escreva um método `int maiorDiferenca(int n)` que devolva a maior diferença entre dois números primos consecutivos menores que `n`.

19. Escreva um método `int mdc(int m, int n)` que calcule o máximo divisor comum de `m` e `n` usando o algoritmo de Euclides:

$$\begin{cases} \text{mdc}(m, m) = m \\ \text{mdc}(m, n) = \text{mdc}(m, n - m) & m < n \\ \text{mdc}(m, n) = \text{mdc}(m - n, m) & m > n \end{cases}$$

20. Escreva um método `int mmc(int m, int n)` que devolva o mínimo múltiplo comum de `m` e `n`.

21. Escreva um método `int primeiroAlgarismo(int n)` que devolva o primeiro algarismo da representação decimal de `n`.

22. Escreva um método `int eCapicua(int n)` que decide se `n` é ou não uma capicua.

23. Escreva um método `int primeiroAlgarismoBase(int n, int k)` que devolva o primeiro algarismo da representação de `n` na base `k`.

24. Escreva um método `int potencia(int k)` que devolva o menor natural `n` tal que $2^{\mathtt{n}}$ começa por `k`. O que tem de assumir sobre `k`?

25. Considere a sequência de números que se obtém a partir dum número natural dado da seguinte forma: um número par é dividido por 2; um número ímpar é multiplicado por 3 e soma-se-lhe uma unidade. Escreva um método `int ateUm(int n)` que descubra quantas operações é necessário realizar a partir de `n` para chegar a 1.

1.3 Programação com Vectores

1. Escreva um método `double soma(double[] v)` que calcule a soma dos valores de `v`.

2. Escreva um método `int zeros(int[] v)` que devolva o número de zeros em `v`.

3. Escreva um método `int conta(int[] v, int n)` que conte quantas vezes `n` ocorre em `v`.

4. Escreva um método `int menores(int[] v, int n)` que devolva o número de elementos de `v` menores que `n`.

5. Escreva um método `boolean pertence(int[] v, int n)` que verifica se `n` ocorre em `v`.

6. Escreva um método `boolean doisZeros(int[] v)` que verifique se existem dois zeros consecutivos em `v`.

7. Escreva um método `int[] quadrados(int n)` que devolva um vector com os quadrados perfeitos dos números naturais entre 1 e `n`.

8. Escreva um método `int[] quadradosDecrescente(int n)` que devolva um vector com os quadrados perfeitos dos números naturais entre `n` e 1.

9. Escreva um método `int[] divisores(int n)` que devolva um vector contendo os divisores de `n`.

10. Escreva um método `double maximo(double[] v)` que devolva o maior elemento em `v`.

11. Escreva um método `int primeiraPosicaoMaximo(int[] v)` que devolva o índice da primeira posição onde ocorre o maior elemento em `v`.

12. Escreva um método `int ultimaPosicaoMaximo(int[] v)` que devolva o índice da última posição onde ocorre o maior elemento em `v`.

13. Escreva um método `int somaPosicoesMaximo(int[] v)` que devolva a soma das posições onde ocorre o maior elemento em `v`.

14. Escreva um método `int[] posicoesMaximo(int[] v)` que devolva um vector contendo as posições onde ocorre o maior elemento em `v`.

15. Escreva um método `void aoQuadrado(int[] v)` que substitui cada elemento em `v` pelo seu quadrado.

16. Escreva um método `void inverte(int[] v)` que inverta os valores de `v` (i.e., troca o primeiro elemento com último, o segundo com o penúltimo, ...).

17. Escreva um método `int[] compara(int[] v, int n)` que devolva um vector contendo na primeira posição o número de elementos de `v` maiores que `n`, na segunda posição o número de elementos de `v` iguais a `n` e na terceira posição o número de elementos de `v` menores que `n`.

18. Escreva um método `int paresAposSete(int[] v)` que devolva o número de elementos pares que ocorrem em `v` depois do primeiro 7.

19. Escreva um método `int paresAposUltimoSete(int[] v)` que devolva o número de elementos pares que ocorrem em `v` depois do último 7.

20. Escreva um método `int[] junta(int[] v, int[] w)` que devolva um vector contendo os elementos de `v` seguidos dos elementos de `w`.

21. Escreva um método `int[] juntaOrdenado(int[] v, int[] w)` que receba dois vectores ordenados e devolva um vector ordenado cujos elementos são todos os elementos que ocorrem em `v` ou em `w`.

22. Escreva um método `int maiorSequenciaCrescente(int[] v)` que devolva o comprimento da maior sequência crescente de elementos consecutivos de `v`.

23. Escreva um método `int[] histograma(int[] v)` que receba um vector `v` de notas (inteiros compreendidos entre 0 e 20) e devolva um vector cuja posição `i` contém o número de notas iguais a `i` contidas em `v`.

24. O *crivo de Eratóstenes* é um dos algoritmos mais antigos para encontrar os números primos até um natural n dado. Escrevem-se todos os números de 1 a n e elimina-se o 1; de seguida escolhe-se o próximo número k da tabela que não foi eliminado e eliminam-se todos os múltiplos de k. Quando toda a tabela tiver sido percorrida, os números que não foram eliminados são precisamente os números primos menores que n.

    Escreva um método `int[] eratostenes(int n)` que implemente este algoritmo. Use uma representação eficiente para o vector auxiliar.

2 Problemas

2.1 Resolução de Equações (I)

Escreva uma classe `ResolveEquacao` para resolver equações de segundo grau. Os coeficientes do polinómio deverão ser variáveis do método `main`. O programa deverá imprimir no monitor as soluções, caso existam, ou um aviso, caso contrário.

2.2 Resolução de Equações (II)

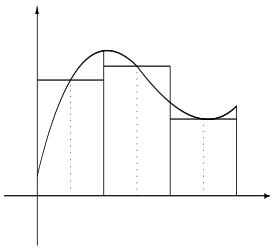
Considere o seguinte método (*método da bissecção*) para encontrar zeros de uma função contínua f. Dados $a < b$ com $f(a) \times f(b) < 0$, calcula-se o ponto médio $c = \frac{a+b}{2}$ e o valor de $f(c)$; se $f(a) \times f(c) < 0$, repete-se o processo com $b = c$; caso contrário, repete-se o processo com $a = c$. O processo termina quando a distância entre a e b for menor que um valor pré-estabelecido, sendo a uma aproximação para o zero de f.

Implemente este método numa classe `Bisseccao` que imprima no monitor o valor encontrado. Os valores iniciais de `a` e `b`, bem como do erro, devem ser variáveis do método `main`. A função f é definida através dum método privado; por exemplo, para $f(x) = x^2$, a sua classe deve incluir o seguinte método.

```
private static double f(double x){
  return x*x;
}
```

2.3 Cálculo de Áreas

Seja f uma função contínua e positiva num intervalo $[a, b]$. A área entre o eixo das abcissas e o gráfico de f no intervalo $[a, b]$ (o *integral* de f em $[a, b]$) pode ser calculada com um erro tão pequeno quanto se queira através do seguinte método: divide-se o intervalo $[a, b]$ em n subintervalos de igual comprimento e aproxima-se o integral de f em cada subintervalo pela área do rectângulo com base no eixo das abcissas e altura igual ao valor da função no ponto médio do intervalo (veja a figura).



Implemente este método numa classe `Integral` que apresente no monitor o valor aproximado para o integral de f no intervalo $[a, b]$. A função f deve ser representada por um método privado como no exercício anterior. Os valores iniciais de `a` e `b` e o número de subintervalos a considerar devem ser variáveis do método `main`.

3 Classes

3.1 Gestão do Tempo

Para o desenvolvimento duma aplicação é necessário representar instantes de tempo.

1. Defina uma classe `Instante` cujos objectos sejam instantes de tempo representados por valores de horas, minutos e segundos. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:

   (a) construtores com 0, 1, 2 ou 3 argumentos, correspondendo (por ordem) às horas, minutos e segundos do instante a criar (assuma o valor 0 para argumentos ausentes);

   (b) um método `boolean legal(int horas, int minutos, int segundos)` que verifique se os argumentos fornecidos podem ser passados ao construtor;

   (c) métodos `void avancaSegundo()`, `void avancaMinuto()` e `void avancaHora()` permitindo avançar respectivamente um segundo, um minuto ou uma hora sobre o instante (considere que às $23 : 59 : 59$ se seguem as $0 : 00 : 00$);

   (d) um método `void avanca(Instante instante)` que avança a hora a quantidade de tempo descrita por `instante`;

   (e) um método `boolean igual(Instante instante)` que determina se o instante actual é o mesmo que `instante`;

   (f) um método `Instante copia()` que devolva uma cópia do instante actual;

   (g) um método `String toString()` que devolva uma representação textual do instante.

   Escreva um cliente para testar esta classe.

2. Ao continuar a desenvolver a aplicação do exercício anterior, chegou-se à conclusão de que nalguns casos seria necessário enriquecer a representação dos instantes com a indicação da data, representado através do ano, mês e dia.

   Defina uma classe `Data` que represente um instante duma data particular, contendo informação sobre o ano, mês, dia e instante. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Tire o máximo proveito da existência da classe `Instante`. Deve disponibilizar os mesmos métodos que para a classe `Instante` (excepto os construtores) e ainda os seguintes:

   (a) um construtor com três argumentos que crie um objecto correspondente à meia-noite da data indicada;

(b) um construtor com quatro argumentos que crie um objecto correspondente ao instante fornecido da data indicada;

(c) um método `boolean legal(int ano, int mes, int dia)` que verifique se os argumentos fornecidos podem ser passados ao construtor;

(d) métodos `void avancaDia()`, `void avancaMes()` e `void avancaAno()` permitindo avançar respectivamente um dia, um mês ou um ano sobre a data;

(e) métodos `void avancaInstante(Instante instante)` e `void avancaData(Data data)` permitindo avançar o intervalo de tempo indicado sobre a data;

(f) um método `int maiorAno()` que devolva o ano do instante mais no futuro que já foi criado ou referenciado;

(g) um método `boolean igual(Data data)` que determina se o instante e data actuais é o mesmo que `data`;

(h) um método `Instante copia()` que devolva uma cópia do instante e data actuais;

(i) um método `String toString()` que devolva uma representação textual do instante e data actuais.

Escreva um cliente para testar esta classe.

3.2 O Jogo do Galo

O jogo do galo joga-se num tabuleiro de 3×3. Dois jogadores jogam alternadamente escolhendo uma casa livre e marcando-a com **X** (primeiro jogador) ou **O** (segundo jogador). O primeiro jogador que conseguir ocupar as três casas duma linha, coluna ou diagonal vence o jogo.

Desenvolva uma classe `Galo` que permita manipular o tabuleiro do jogo do galo. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:

1. um construtor que crie um tabuleiro vazio no estado inicial (em que o jogador que tem a vez é o que joga com **X**);
2. um método `void jogada(int i,int j)` que simule a jogada do próximo jogador na casa (`i`,`j`), caso esta esteja livre;
3. um método `char seguinte()` que devolva a letra do próximo jogador a jogar;
4. um método `boolean livre(int i,int j)` que verifique se a casa (`i`,`j`) está livre;
5. um método `char valor(int i,int j)` que devolva a letra do jogador que ocupou a casa (`i`,`j`) caso esta esteja ocupada e – caso contrário;
6. um método `boolean ganha(char c)` que verifique se o jogador com a letra `c` venceu o jogo;
7. um método `String toString()` que devolva uma representação textual do estado do jogo.

Escreva um cliente para esta classe que permita que dois jogadores joguem interactivamente ao jogo do galo. O seu cliente deve verificar que todas as invocações dos métodos da classe `Galo` são correctamente aplicadas.

Enriqueça o seu cliente com a possibilidade de jogar contra um jogador automático. Programe o seu jogador automático por forma a garantir sempre um empate ou uma vitória.

3.3 Cofres

Pretende-se desenvolver uma classe para representar cofres. Cada cofre tem capacidade para armazenar um objecto de tipo `String`. Para limitar o acesso a este objecto, o cofre tem ainda uma fechadura, que é um vector de inteiros. Para um utilizador aceder ao conteúdo do cofre tem de introduzir um a um os números da fechadura; caso se engane o valor é ignorado.

Defina uma classe `Cofre` cujos objectos representem cofres. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:

1. um construtor com um argumento, correspondendo à combinação da fechadura;
2. um método `boolean fechado()` que indique se o cofre está fechado;
3. um método `void introduzir(String frase)` que permita armazenar `frase` no cofre, caso este esteja aberto;
4. um método `String conteudo()` que permita obter o conteúdo do cofre, caso este esteja aberto;
5. um método `void insere(int n)` que insere o valor `n` na fechadura;
6. um método `void reiniciar()` que esquece os números da combinação já inseridos.

Escreva um cliente para testar esta classe.

3.4 Fracções

Embora o `java` disponibilize números de vírgula flutuante, há situações em que é útil trabalhar com fracções de forma exacta, ou seja, sem erros de arredondamento. Uma forma de o fazer é representar cada fracção como um par de inteiros (numerador e denominador), com a restrição de o denominador ser positivo.

Defina uma classe `Fraccao` cujos objectos representem fracções. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:

1. construtores com zero, um ou dois argumentos permitindo construir a fracção `1`, uma fracção correspondendo a um número inteiro ou uma fracção arbitrária;
2. métodos `void soma(Fraccao f)`, `void multiplica(Fraccao f)` e `void divide(Fraccao f)` permitindo, respectivamente, somar, multiplicar ou dividir esta fracção com/pela fracção `f`;
3. um método `void simplifica()` que transforma a fracção numa fracção irredutível (ou seja, cujo numerador e denominador não têm divisores comuns);
4. um método `double valor()` que devolve uma aproximação do valor representado pela fracção;
5. um método `boolean igual(Fraccao fraccao)` que verifique se a fracção é igual a `fraccao`;
6. um método `Fraccao copia()` que devolva uma fracção igual;
7. um método `String toString()` que devolva uma representação textual da fracção.

Escreva um cliente para testar esta classe.

3.5 Geometria no plano

Um ponto no plano é definido por duas coordenadas, as suas componentes horizontal e vertical.

1. Defina uma classe `Ponto2D` cujos objectos representem pontos no plano. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:
   (a) um construtor que crie um ponto a partir das suas coordenadas;
   (b) um método `boolean eOrigem()` que decida se o ponto é a origem;
   (c) um método `void translacao(double deltaX, double deltaY)` que efectue uma translação do ponto segundo o vector (`deltaX`,`deltaY`);
   (d) um método `double distanciaAOrigem()` que calcule a distância do ponto à origem;
   (e) um método `int quantasOrigens()` que indique quantos objectos existem correspondendo à origem;
   (f) um método `boolean igual(Ponto2D ponto)` que verifique se o ponto é igual a `ponto`;
   (g) um método `Ponto2D copia()` que devolva um ponto igual;
   (h) um método `String toString()` que devolva uma representação textual do ponto.

   Considere o seguinte cliente para a classe `Ponto2D`. Descreva os efeitos da execução deste código.

```
int x = 6, n = -1;
Ponto2D p = new Ponto2D(4, x);
if ( !p.eOrigem() )
  p.translacao(n, n*2);
System.out.println(p.distanciaAOrigem());
```

   Considere agora um segundo cliente para esta classe. Represente graficamente o estado da memória depois de cada instrução.

```
Ponto2D p1 = new Ponto2D(0, 0);
Ponto2D p2 = p1;
Ponto2D p3;

p1.translacao(1, 1);
System.out.println(p2.toString());

p2.translacao(3, 3);
System.out.println(p1.toString());
System.out.println(p3.toString());
```

2. Um polígono é uma região do plano delimitada por segmentos de recta.

   Defina uma classe `Poligono` cujos objectos representem polígonos. Um polígono deve ser representado como uma sequência de pontos (os seus vértices) tal que cada dois pontos consecutivos são unidos por uma aresta, bem como o primeiro e o último. Tire o máximo proveito da classe `Ponto2D`. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Cada polígono deve ter um identificador que o distinga de todos os outros. Deve ainda disponibilizar os seguintes métodos:

   (a) um construtor que crie um polígono a partir dum vector de `Ponto2D` com os seus vértices;

   (b) um método `double perimetro()` que retorne o perímetro do polígono;

   (c) um método `Ponto2D maisProximo()` que devolva o vértice mais próximo da origem;

   (d) um método `double maiorAresta()` que retorne o comprimento da aresta mais comprida;

   (e) um método `void translacao(double deltaX, double deltaY)` que efectue uma translação do polígono segundo o vector (`deltaX`,`deltaY`);

   (f) um método `int verticesNoQuadrante(int n)` que devolva o número de vértices do polígono no quadrante `n`;

   (g) um método `boolean eTriangulo()` que determina se o polígono é um triângulo;

   (h) um método `boolean eRectangulo()` que determina se o polígono é um rectângulo;

   (i) um método `int id()` que devolva o identificador do polígono;

   (j) um método `Poligono trianguloMaisRecente()` que devolva o triângulo criado mais recentemente;

   (k) um método `boolean igual(Poligono poligono)` que verifique se o polígono é igual ao polígono `poligono` (observe que os vértices do polígono não têm de ser dados sempre pela mesma ordem);

   (l) um método `Poligono copia()` que devolva um polígono igual;

   (m) um método `String toString()` que devolva uma representação textual do polígono.

3. A representação discreta duma função f é um vector `grafico` de `Ponto2D` tal que cada ponto (x, y) do vector satisfaz $f(x) = y$. Assume-se ainda que os pontos do vector `grafico` estão ordenados pela primeira coordenada.

   Desenvolva uma classe `Funcao` cujos objectos são representações discretas de funções. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:

   (a) um construtor que receba um vector adequado e crie a função correspondente;

   (b) um método `boolean legal(Ponto2D[] grafico)` que verifique se o argumento fornecido corresponde à representação discreta duma função;

   (c) um método `double maximo()` que calcule o máximo da função, de acordo com a sua representação discreta;

   (d) um método `boolean crescente()` que indique se a função é crescente ou não;

   (e) um método `double[] taxaVariacao()` que retorne um vector com menos uma posição que o vector `grafico` contendo a taxa de variação média da função em cada intervalo;

   (f) um método `boolean igual(Funcao f)` que determina se a função em causa tem a mesma representação discreta que `f`;

   (g) um método `Funcao copia()` que devolva uma cópia da função;

   (h) um método `String toString()` que devolva uma representação textual da função.

3.6 Matrizes

Um vector bidimensional é designado por matriz.

Desenvolva uma classe `Matriz` cujos objectos representem matrizes. Decida quais os atributos que a sua classe deve conter e quais os métodos que deve disponibilizar para consultar/modificar os valores desses atributos. Deve ainda disponibilizar os seguintes métodos:

1. um construtor com um argumento `int n` que crie uma matriz identidade de dimensão `nxn`;

2. um construtor com dois argumentos `int n` e `int m` que crie uma matriz de zeros de dimensão `mxn`;

3. um construtor com um argumento `double[] v` que crie uma matriz com os elementos de `v` na diagonal principal e zero nas restantes posições;

4. um método `void multiplicaEscalar(int k)` que multiplique todos os elementos da matriz por `k`;

5. um método `Matriz transposta()` que devolva a transposta da matriz;
6. um método `double soma()` que devolva a soma de todos os elementos da matriz;
7. um método `void soma(Matriz m)` que some a matriz com a matriz `m`, componente a componente, caso as dimensões das matrizes sejam compatíveis;
8. um método `double media()` que devolva a média dos elementos da matriz;
9. um método `boolean eMenor(Matriz m)` que verifique se cada elemento da matriz é menor que o elemento de `m` na mesma posição;
10. um método `boolean eTriangularSuperior()` que indique se a matriz é triangular superior;
11. um método `boolean eTriangularInferior()` que indique se a matriz é triangular inferior;
12. um método `boolean eDiagonal()` que indique se a matriz é diagonal;
13. um método `Matriz produto(Matriz m)` que devolva o produto da matriz por `m`, caso as dimensões sejam compatíveis;
14. um método `void gauss()` que efectue o método de eliminação de Gauss sobre a matriz;
15. um método `double determinante()` que calcule o determinante da matriz;
16. um método `boolean igual(Matriz m)` que verifique se a matriz é igual à matriz `m` (observe que duas matrizes iguais têm de ter a mesma dimensão e elementos iguais nas mesmas posições);
17. um método `Matriz copia()` que devolva uma matriz igual;
18. um método `String toString()` que devolva uma representação textual da matriz.

4 Projectos

4.1 Transformação de Textos

O objectivo desta aula é ajudar a equipa TUTEDECC ("Tem Um TExto Demasiado Claro? Contacte-nos!") no seu estudo sobre várias técnicas de obscurecimento de mensagens. Vamos ajudá-los a transformar textos, obscurecendo-os, de variadas e diferentes maneiras para posteriormente serem apresentados a um grupo de "cobaias" que os tentarão ler.

As técnicas de obscurecimento de textos que a equipa TUTEDECC pretende implementar são as seguintes.

- Retirar as vogais a todas as palavras do texto exceptuando aquelas palavras totalmente compostas por vogais – essas não serão alteradas.

  Exemplo. O texto `O Pai Natal nao existe` ficaria `O P Ntl n xst`

- Após apagar os espaços entre as palavras no texto original, inserir um espaço a cada n caracteres (o texto obtido nunca deverá terminar com espaço).

  Exemplo. Para $n = 3$, o texto `O Pai Natal nao existe` ficaria `OPa iNa tal nao exi ste`

- Inverter todas as palavras individualmente.

  Exemplo. O texto `O Pai Natal nao existe` ficaria `O iaP lataN oan etsixe`

- Inverter os caracteres do texto de m em m caracteres (incluindo espaços); o último bloco é sempre invertido mesmo que tenha menos que m caracteres.

  Exemplo. Para $m = 3$, o texto `O Pai Natal nao existe` ficaria `P O iataN laoanxe tsie`

- Fazer um *shift* à direita de k posições a cada palavra (k pode não ser menor que o tamanho de todas as palavras).

  Exemplo. Para $n = 2$, o texto `O Pai Natal nao existe` ficaria `O aiP talNa ona teexis`

Considera-se que o texto está dividido em linhas e que nenhuma palavra é cortada de uma linha para a outra. Considera-se ainda que o espaço é o único carácter usado para além das letras do alfabeto e que não são usados caracteres acentuados.

Neste trabalho, qualquer das técnicas de obscurecimento referidas é para ser aplicada linha a linha. Como exemplo, a aplicação das quatro técnicas ao texto original seguinte:

```
O Pai Natal nao existe
Mas o coelho da Pascoa existe
E a Fada dos Dentes tambem
```

para $n = 4$ e $m = 3$ teria como resultado os textos:

```
O P Ntl n xst
Ms o clh d Psc xst
E a Fd ds Dnts tmbm
```

```
OPai Nata lnao exis te
Maso coel hoda Pasc oaex iste
EaFa dado sDen test ambem
```

```
O iaP lataN oan etsixe
saM o ohleoc ad aocsaP etsixe
E a adaF sod setneD mebmat
```

```
P O iataN laoanxe tsie
saM o eocohlad aP ocse asixet
a EaF adsodeD etnt sbmame
```

Pode ser útil saber que o alfabeto minúsculo corresponde aos códigos ASCII entre 97 e 122 e o alfabeto maiúsculo aos códigos entre 65 e 90.

A sua tarefa é desenvolver uma classe `Obscurecedor` contendo os atributos e métodos necessários para armazenar o texto a transformar e lhe aplicar as operações descritas acima.

Construa ainda uma classe `RunObscuredor` que seja cliente da classe `Obscurecedor` e que implemente o seguinte algoritmo:

1. fazer a interação com o utilizador perguntando-lhe os nomes dos ficheiros de entrada e saída (o ficheiro de entrada contém o texto a ser obscurecido; o ficheiro de saída irá conter, no fim da execução do programa, os textos resultantes de obscurecer o texto original usando as várias técnicas acima descritas) e os valores de n e m para as segunda e quarta técnicas, respectivamente;
2. ler o texto original do ficheiro de entrada;
3. criar um objecto `Obscurecedor` e invocar sobre ele os métodos necessários para obter os textos obscurecidos;
4. escrever estes quatro textos no ficheiro de saída, separados por uma linha em branco;
5. informar o utilizador sobre o número de linhas escritas no ficheiro de saída (contando com as linhas em branco).

4.2 Classificação de Peixes

Numa fábrica de conservas são confeccionadas conservas de vários tipos de peixe. Os peixes chegam directamente do porto onde foram comprados em lotes, mas as várias espécies vêm misturadas. Assim, a fábrica gostaria de resolver automaticamente o problema da triagem do peixe. Foi comprado um equipamento topo de gama, que fotografa os peixes que passam num tapete rolante e mede automaticamente a largura e o comprimento de cada um.

Aos dados que caracterizam um objecto a classificar chama-se *padrão*. No caso dos peixes, o padrão é constituído pelo seu comprimento e largura. O método sugerido para classificar peixes é o do vizinho mais próximo: um conjunto de peixes é analisado e recolhe-se as dimensões e a espécie de cada peixe. As observações realizadas neste conjunto (que se denomina conjunto de referência) são usadas para classificar automaticamente (sem intervenção humana) os novos peixes. Para cada peixe, o algoritmo consiste nos passos seguintes.

1. Obter as suas dimensões.
2. Encontrar o peixe do conjunto de referência que tem dimensões mais semelhantes (o vizinho mais próximo). Para isso é preciso calcular as distâncias euclideanas entre o padrão a classificar e todos os padrões do conjunto de referência. (A distância euclideana a duas dimensões entre dois pontos (x_1, y_1) e (x_2, y_2) é dada pela expressão $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.)
3. Atribuir a espécie do padrão de referência mais próximo ao padrão a classificar.

O seu programa deve abranger o caso geral onde a dimensão dos dados e o número de classes são arbitrários; o conjunto de referência é sempre dado.

A sua tarefa é criar uma classe `Classificador` que disponibiliza os seguintes métodos.

- Um construtor com dois argumentos:
  1. um vector bidimensional com elementos de tipo `double`, correspondendo ao conjunto de referência; no exemplo descrito anteriormente, o vector teria duas colunas, correspondendo ao comprimento e à largura de cada peixe;
  2. um vector de inteiros correspondendo à espécie de cada peixe do vector anterior; este vector deve ter tantos elementos quantas as linhas do anterior.
- Um construtor com um único argumento, correspondendo a um ficheiro que será lido para inicializar o objecto. O ficheiro deve começar com uma linha contendo dois inteiros: a dimensão do padrão e o número de linhas do ficheiro. As restantes linhas têm um padrão e a sua classificação.
- Um método `int classificar(double[] padrao)` que receba um padrão e retorne a classificação obtida, usando o algoritmo do vizinho mais próximo.
- Um método `int[] classificarConjunto(double[][] conjunto)` que receba um conjunto de padrões (semelhante ao construtor) e retorne o vector das classificações correspondente.
- Um método `double percentagemErro(double[][] conjunto, int[] classificacoes)` que receba um conjunto de padrões e um vector de classificações, classifique cada padrão do conjunto e compare o resultado com o fornecido no segundo argumento, retornando a percentagem de erros de classificação.

Desenvolva ainda uma classe cliente `RunClassificador` para testar o seu programa.

4.3 Números Romanos

O sistema de numeração utilizado por todo o Império Romano recorria a sete letras maiúsculas: `I`, `V`, `X`, `L`, `C`, `D` e `M`, valendo, respectivamente, 1, 5, 10, 50, 100, 500 e 1000. Os números eram compostos por sequências ordenadas destas letras, sujeitas às seguintes restrições:

- as letras aparecem por ordem decrescente de valor;
- não pode haver mais de uma ocorrência de nenhuma das letras `V`, `L` ou `D`, nem mais de quatro ocorrências de nenhuma das letras `I`, `X` ou `C`.

O número representado por uma sequência de caracteres é simplesmente a soma de todas as letras que o compõem. Assim, por exemplo, o número 2341 é escrito como `MMCCCXXXXI`, enquanto `DCCII` corresponde ao número 702. (Observe que este sistema é mais simples que o habitualmente utilizado, em que 2341 seria escrito como `MMCCCXLI`; a substituição de `IIII` por `IV`, `XXXX` por `XL` e `CCCC` por `CD` é posterior ao sistema descrito.)

O objectivo desta aula é criar uma classe `Numeros` para trabalhar com números representados em numeração romana.

A classe `Numeros` deve disponibilizar uma série de métodos que manipulem `Strings` interpretadas como números romanos. Em particular, devem ser desenvolvidos os métodos descritos abaixo.

- `boolean isRomanNum(String s)`: determina se `s` representa um número romano (ou seja, se satisfaz as condições descritas acima).
- `numToRoman(int n)`: para um inteiro `n` entre 1 e 4000, devolve o número romano correspondente.
- `String romanToNum(String num)`: para uma `String` `num` representando um número romano, devolve o número correspondente (de tipo `int`).
- `String add(String num1, String num2)`: para duas `Strings` representando números romanos, devolve a sua soma. Para a coisa ter piada, só pode utilizar métodos sobre `Strings`; ou seja, não é permitido converter os argumentos para tipo `int`, somá-los e reconverter para `Strings`.
- `String diff(String num1, String num2)`: para duas `Strings` representando números romanos, sendo o primeiro (estritamente) maior que o segundo, devolve a sua diferença. Mais uma vez, só pode utilizar métodos sobre `Strings`.

Desenvolva uma classe `TesteNumeros` que seja cliente de `Numeros`. Esta classe deve executar as seguintes operações.

1. Pedir ao utilizador um número inteiro `n` entre 1 e 4000.

2. Imprimir `n` em numeração romana.
3. Pedir ao utilizador um valor `r` em numeração romana.
4. Imprimir o inteiro correspondente a `r`.
5. Imprimir o valor da soma do número romano correspondente a `n` com `r`.
6. Imprimir o valor da diferença entre `MMMMI` e o número correspondente a `n`.

Tenha o cuidado de testar todos os dados fornecidos pelo utilizador antes de os passar aos métodos da classe `Numeros`.

Sugestões de implementação

- Para tornar o código substancialmente mais simples, é útil definir duas constantes privadas na classe `Numeros`, uma contendo os símbolos a utilizar e outra contendo os seus valores.
- No desenvolvimento da função `add`, pode utilizar o seguinte algoritmo em dois passos. No primeiro passo, juntam-se as duas `String`s dadas tendo o cuidado de manter a ordenação das letras, mas sem preocupações com as repetições. No segundo passo, substituem-se as subsequências ilícitas por equivalentes lícitos (por exemplo, `IIIII` seria substituído por `V`). Assim, para somar `MMCCCXXXXI` com `DCCII`, passar-se-ia pela string intermédia `MMDCCCCCXXXXIII`.
- Para subtrair dois números, o mais simples é recorrer à definição: $m - n$ é o único inteiro k tal que $n + k = m$.