LEIM – Modelação e Programação – 18/19 SV
Trabalho prático 1 – TP1

Deve colocar as várias resoluções relativas a este trabalho no package tps.tp1. Comece por verificar que já existem ou então por criar os packages tps.tp1, tps.tp2, tps.tp3 e tps.tp4.
Cada pedido de *input* deve ser precedido de uma mensagem a indicar o que é solicitado.

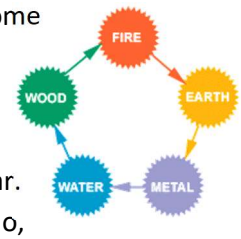
Decisões

Coloque as alíneas deste grupo no package **tps.tp1.pack1Decisoes**.
As limitações de utilização referem-se às instruções de if, switch, for, while e do-while. Os programas não devem fazer *return* no *main* ou utilizar *System.exit()*.

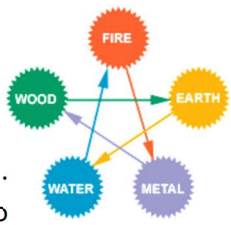
1. Conceba o programa **P01Ifs**, que no seu *main*, peça ao utilizador a introdução de um número entre 0 e 20. Se o número introduzido for negativo ou maior que 20 o programa deverá indicar esse facto na consola e terminar. Caso o número seja válido o programa deverá indicar: **a)** qual o valor do quociente e do resto da divisão inteira por 6; e **b)** se o número é múltiplo de 7 ou não. Cada afirmação a) e b) deverá ser escrita na consola por uma única instrução de System.out.println. Para esta aliena só se deve utilizar a instrução de controlo de fluxo IF.

2. Conceba o programa **P02Ifs** que no seu *main* peça ao utilizador a introdução de três inteiros e que os apresente por ordem crescente. Deve declarar e utilizar as variáveis "menor", "meio" e "maior". Após a introdução dos valores, o programa deverá apresentar o resultado num texto com a seguinte aparência: "O menor valor é 10, o valor do meio é 20 e o maior valor é 30", para os valores introduzidos de 20, 30 e 10. Para esta aliena só deve utilizar a instrução IF e o resultado apenas deverá ser escrito na consola por uma só instrução System.out.println. O programa só deve ter duas instruções System.out.println.

3. Conceba o programa **P03OpTernário** que no seu *main* peça ao utilizador que introduza o nome de um dos elementos da Teoria Taoista dos 5 Elementos (Fire, Earth, Metal, Water, e Wood) e que mostre o seguinte elemento segundo o diagrama da criação, tal como se mostra na figura ao lado. O programa deve começar por validar se o elemento introduzido é válido ou não utilizando um IF para tal. Caso não seja válido deve assinalar esse facto na consola e terminar. Caso seja um elemento válido deve mostrar na consola uma frase em inglês com o pretendido, com : "Water generates Wood", para Water como o elemento introduzido. Para a obtenção do elemento seguinte só se pode utilizar como instrução de controlo de fluxo o *operador ternário*. O elemento introduzido pode estar em maiúscula, minúsculas ou um "mix". O elemento seguinte deve ser colocado numa variável e depois colocado na frase para o output.



4. Conceba o programa **P04Switch** análogo em termos de *input* e *output* à alínea anterior, mas que mostre **também** o próximo elemento do ciclo da destruição, tal como se mostra na figura aqui ao lado. Assim para o elemento Water pretende-se uma frase: "Water generates Wood and Water destroys Fire". A validação do elemento introduzido deve ser efetuada por um IF. Depois deve-se colocar nas variáveis elemGenerated e elemDestroyed o próximo elemento gerado e destruído pelo elemento introduzido. O elemento gerado deve ser determinado pela utilização de um só *switch* e o elemento destruído deve ser determinado da seguinte forma: um *switch* transforma o elemento num número (Fire = 0, Earth = 1, …); esse número é incrementado, **duas vezes**, de forma circular (0, 1, 2, 3, 4, 0, …); o valor resultante é utilizado num segundo *switch* de transformação inversa que coloca na variável elemDestroyed o nome do elemento destruído.



Ciclos

Coloque as alíneas deste grupo no package **tps.tp1.pack2Ciclos**.
Agora já pode utilizar as instruções de decisão (if, switch e ?:) e de repetição (while, do-while e for).

1. Conceba o programa **P01Potencia** que no seu *main* peça ao utilizador um número inteiro a replicar (*num*) entre 1 e 20, um número de replicações (*nReps*) entre 1 e 10 e que mostre o valor das multiplicações de *num* *nReps* vezes tal como se mostra no exemplo que se segue. No caso de introdução de valores incorretos, dever-se-á apresentar uma mensagem de erro e voltar a pedir o número, até o valor estar correto. Nesta alínea não pode utilizar a instrução "for".

Introduza um inteiro entre 1 e 20 -> 7
Introduza o número de repetições (1..10) - > 5
Potência de base 7 e expoente 5 = 7 * 7 * 7 * 7 * 7 = 16807

2. Conceba o programa **P02Triangulo** que no seu *main* peça ao utilizador um número entre 1 e 21 e que: primeiro mostre se o nº é válido ou não e que repita o pedido e a introdução até este ser válido; e que depois mostre um triângulo desenhado com '#'s tal como se mostra em seguida, com tantas linhas quantas o nº introduzido, começando com uma linha com 1 '#', ir aumentando 1 '#' por cada linha, começando alinhado à direita, mas de forma a terminar com o triangulo encostado ao início da linha. Exemplo para um valor de 3:

```
  #
 ##
###
```

3. Implemente o interior do método `drawColors(Graphics g, int dimX, int dimY, Color outsideColor, Color insideColor)` do programa **P03ColorFrame**, cujo código base se disponibiliza, tal que o referido método preencha a zona de desenho recebida em g, que tem as dimensões de dimX e dimY, com retas com uma variação de cor do centro paras as extremidades, começando com insideColor e terminando com outsideColor, em que a primeira será a "Start Color" da interface gráfica e a segunda e a "End Color". A cor deve ir variando de forma progressiva em red, em green e em blue, separadamente. De forma a evitar os arredondamentos dos inteiros, sugere-se que se trabalhe sempre no domínio dos *double* e só se passe para *int* quando for para desenhar as retas. Fazer isso com a cor e com a coordenadas. Apenas se deve fazer o interior do referido método, tudo o resto é disponibilizado e não é necessária qualquer compreensão do mesmo. A visualização deve-se adaptar as dimensões da janela onde está.

Arrays

Coloque as alíneas deste grupo no package tps.tp1.**pack3Arrays**.
Recomenda-se a estruturação dos programas em métodos estáticos. Assim o código de um método pode ser utilizado mais do que uma vez no programa. Não é permitido a utilização de variáveis globais estáticas, assim o que for necessário para a execução dos métodos deverá ser passado como argumento dos mesmos.

1. Conceba o programa **P01ArrayIntsInit** que no seu *main*: declare um array de 10 inteiros inicializados aquando da sua declaração com valores escolhidos ao acaso pelo programador (exº: new int[]{5, 2, 90, …}); mostre na consola o conteúdo do array na forma "[5, 2, 90, …]"; mostre na consola qual o maior número ímpar; altere os números que são pares adicionando-lhes a cada um o próximo número ímpar (depois do número em questão) se existir; e volte a mostrar todos os seus elementos. Declare e utilize o método "void printArray(int[] array)" que imprime um array na consola e na forma indicada.

2. Conceba o programa **P03ArraysExtractUniqsAndReps** que no seu *main*: declare dois arrays de 10 inteiros, inicializados com recurso **ao gerador aleatório Math.random()**, com valores compreendidos entre 0 e 20 (os dois inclusive) mas que não tenham repetições dentro do mesmo array (repetindo a geração aleatória se necessário); mostre o conteúdo dos dois arrays na consola; obtenha num array somente os valores que não têm repetições entre os dois array iniciais; obtenha num outro array somente os valores que têm repetições entre os dois array iniciais; os dois array finais devem ter a dimensão exata para conterem os devidos elementos; e mostre o conteúdo dos dois arrays finais na consola. Exemplo da utilização do gerador aleatório: "(int)(Math.random()*10)" devolve um valor aleatório no intervalo [0..9]. Exemplo do pretendido, mas com arrays de 5 elementos: a1 = [2, 6, 8, 4, 9], a2 = [5, 6, 2, 7, 8], array de únicos = [4, 5, 7, 9], array de repetidos = [2, 6, 8]. Os arrays finais, de repetidos e únicos, devem ser ordenados utilizando o método java.util.Arrays.sort(int[] array). Para mostrar um array deve-se utilizar o método java.util.Arrays.toString(int[] array), que devolve uma string com a descrição textual do array.

3. Conceba o programa **P03BaralhadorDeNomes** que: peça ao utilizador para introduzir até 10 nomes completos com até 120 caracteres e que cada um contenha pelo menos dois nomes com pelo menos 4 caracteres; terminando quando se tiver atingido o limite ou quando o utilizador escreva "fim" (case insensitive) como um nome e já existirem 5 nomes válidos. Cada nome inválido deve ser descartado, mas deve-se notificar o utilizador desse facto. No final da entrada dos nomes deve-se mostrar uma listagem com os dados introduzidos; depois deve-se obter o primeiro nome de cada nome, que contenha 4 ou mais caracteres; obter o último nome de cada nome, que contenha 4 ou mais caracteres; gerar tantos novos nomes quantos os que foram introduzidos com um primeiro nome e um nome final, dos já obtidos, escolhidos de forma aleatória mas sem repetições (sem mais repetições do que as que já existiam pela introdução de nomes repetidos); e mostrar os novos nomes na consola. **Para processar cada nome deverá utilizar a classe Scanner, e de String só poderá utilizar o método length().**

4. Conceba o programa **P04Xadrez** que, no contexto de um tabuleiro de xadrez, posicione de forma aleatória **uma** torre, **um** bispo e **um** cavalo, mas de forma a não haver ataques entre as peças. Assim para cada peça deve-se guardar as suas coordenadas x e y e comparar se há ataque com as peças já processadas. A comparação deve ser realizada somente com base nas coordenadas das peças. Em caso de ataque deve gerar novas coordenadas e voltar a testar, até não haver qualquer ataque. No final deve-se mostrar o tabuleiro em que a torre é visualizada com um 'T', o bispo com um 'B', o cavalo com um 'C', cada posição que não sofra qualquer ataque deve conter um 'o', cada posição com um só ataque deve conter um '-', e cada posição com pelo menos dois ataques deve conter um '+'. O programa deverá gerar e mostrar uma nova configuração válida quando, e sempre que, se premir a tecla de Enter. O código deverá definir a dimensão do tabuleiro numa variável (final) e estar preparado para funcionar para qualquer valor (>4) dessa variável. A figura mostra um exemplo com 2 cavalos, uma torre e um bispo.

```
-------------------------
| - | T | - | + | - | + |
| o | - | o | o | - | C |
| o | + | o | + | o | o |
| o | - | + | o | - | o |
| C | + | o | o | o | o |
| B | - | - | o | o | o |
-------------------------
```

Exº de um tabuleiro 6x6

Conceba e utilizar os seguintes métodos **estáticos** para a deteção dos ataques e impressão do tabuleiro:

// verificar ataque, devolver *true* se peça em xtbc,ytbc ataca a peça em xp,yp:

// no caso da torre, se xt = xp há um ataque na vertical e se yt = yp há um ataque na horizontal
boolean torreAtacaPeca(int xt, int yt, int xp, int yp)
boolean bispoAtacaPeca(int xb, int yb, int xp, int yp)
boolean cavaloAtacaPeca(int xc, int yc, int xp, int yp)
boolean existeAtaqueEntreTorreEBispo(int xt, int yt, int xb, int yb) // utilizar os métodos anteriores
boolean existeAtaqueEntreCavaloEBispo(int xc, int yc, int xb, int yb) // …
boolean existeAtaqueEntreCavaloETorre(int xc, int yc, int xt, int yt) // …
// adicionar peça ao tabuleiro e assinalar ataques, incrementar o nível de ataque se necessário
void tabuleiroAddTorre(char[][] tabuleiro, int xt, int yt)
void tabuleiroAddBispo(char[][] tabuleiro, int xb, int yb)
void tabuleiroAddCavalo(char[][] tabuleiro, int xc, int yc)
void printTabuleiro(char[][] tabuleiro) // imprimir o tabuleiro tal como se mostra no exemplo

5. Com base no código existente em **P05BitmapTransform** altere o código do método ***reduceImage*** de modo a fazer reduzir a imagem recebida para metade, tanto em altura como em largura. Não é necessário qualquer conhecimento de programação gráfica exceto que uma imagem é um retângulo de pixéis, que é um objeto BufferedImage e que se pode pedir as suas dimensões com *getWidth*() e *getHeight*(). Cada pixel tem a informação da sua cor que é um *int* (4 bytes), em que nele o byte de menor peso tem a componente azul, no byte seguinte a verde e depois a vermelho (RGB – Red, Green, Blue). Basta então copiar pixels de um lado para outro e manipular o RGB dos pixéis. Existem métodos auxiliares de *getRGB* e *setRGB* para *blue*, *green* and *red* que permitem obter e afetar uma componente RGB de um *int*. A redução da imagem para metade deve ser efetuada tendo em conta que um quadrado de 2 por 2 pixels, na imagem original, ficará reduzido a um pixel, na imagem final, com o respetivo valor médio de *red*, de *green* e de *blue* (dos 4 pixels originais) em separado. Exemplo da redução pretendida:

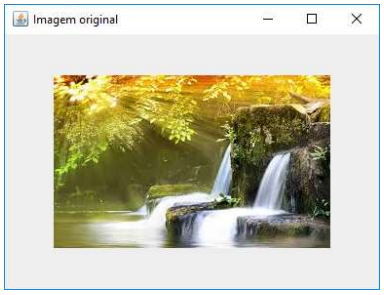


Imagem original

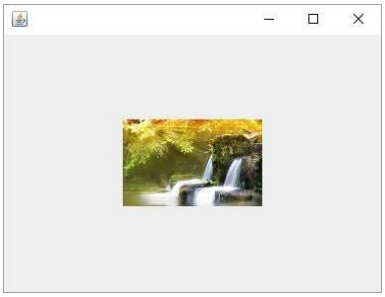


Imagem transformada

Cada aluno tem 7 dias de tolerância para poder entregar todos trabalhos práticos em atraso, ou seja, para além das datas fixadas. Assim um grupo que entregue o TP1 com 3 dias de atraso, os alunos desse grupo, só terão 4 dias, cada um, para poder entregar, depois do prazo todos os outros TPs (no conjunto). Fora desse prazo de tolerância não será aceite qualquer trabalho o que implica a reprovação na UC. Um aluno, que mude de grupo, transporta consigo os seus dias de tolerância e não os poderá ultrapassar (mesmo que os outros membros do grupo possam - ele ficaria reprovado e os outros poderiam prosseguir).

O relatório e o código deste trabalho devem ser entregues até ao dia **24 de março de 2019**.

Cada aluno de um grupo é responsável por todos os trabalhos práticos desse grupo. Os trabalhos são do grupo, não de um aluno. Pelo que, num grupo, pode-se escolher um aluno para responder pela execução dos trabalhos e caso ele não saiba justificar uma qualquer parte dos mesmos, concluiremos que o trabalho foi plagiado e consequentemente todo o grupo será reprovado. **Caso o plágio seja de um trabalho de um grupo, mesmo noutra turma, isso implica a anulação desse trabalho também.**

Bom trabalho, António Teófilo, Jorge Pais e Pedro Fazenda