1º Trabalho Prático

Programação

Realizado por:
<46055> João Martins
<46001> José Santos
<46074> Ricardo Margalhau

Instituto Superior de Engenharia de Lisboa – Inverno 2018 / 2019

Number

1. (Number)
Escrever em milhares, centenas, dezenas e unidades um valor (entre 0 e 9999) introduzido pelo utilizador. São valorizadas as soluções que minimizem o número de variáveis e de chamadas a `print()/println()`.

No programa Number começámos por obter os milhares dividindo por 1000 conservando o valor numa variável. Posteriormente para encontrar o dígito correspondente à casa das centenas dividimos o número original por 100 subtraindo-lhe o valor dos milhares multiplicado por 10, por exemplo, considerando o número 2100 temos que os milhares são (int) (2100 / 1000) = 2 e as centenas (int) (2100 / 100 – 2 * 10) = 1. Repetindo este processo para as seguintes casas do número original podemos obter os milhares, as centenas, as dezenas e as unidades.

Slope

2. (Slope)
Escrever a equação reduzida da reta ($y = \boldsymbol{m}\,x + \boldsymbol{b}$) que passa por dois pontos (x_1,y_1) e (x_2,y_2) introduzidos pelo utilizador. Na equação, ***m*** é o declive (*slope*) da reta obtido pela expressão $\boldsymbol{m} = (y_1-y_2) / (x_1-x_2)$ e ***b*** corresponde à ordenada na origem dada por $\boldsymbol{b} = y_1 - \boldsymbol{m}\,x_1$ ou $\boldsymbol{b} = y_2 - \boldsymbol{m}\,x_2$. Quando o declive é infinito a equação será simplesmente ($x = x_1$).

Para realizar o programa Slope começámos por obter os 4 inputs vindos do utilizador (x1, y1, x2, y2) e verificar se x1 e x2 são iguais e y1 e y2 são iguais. Neste caso o programa iria encontrar uma indeterminação (0 / 0) pelo que enviamos uma mensagem de erro ao utilizador caso isto se venha a verificar.

Depois verificámos se o x1 é igual ao x2 e caso isto venha acontecer a reta não apresenta declive pois irá tratar-se de uma reta do tipo y = b

Caso ambas as opções acima sejam falsas então temos o declive pela fórmula matemática da reta e substituímos um ponto para encontrar a ordenada na origem (b).

Dice

3. (Dice)
Fazer a simulação do lançamento de dois dados num jogo de tabuleiro gerando valores pseudoaleatórios com o método `random()` da classe `Math`. Só é mostrado o segundo dado lançado após o utilizador premir a tecla "enter". No final é mostrada a soma dos dados, que deverá ser dobrada se os dois valores forem iguais.

Em Dice iniciámos o programa calculando o número do primeiro dado através do método *random* da class *Math* (java.lang.Math). Nós conseguimos obter um número random entre 1 e 6 inclusive através da seguinte manipulação matemática:

$$0 \leq \text{random} < 1$$
$$0 * 6 \leq \text{random} * 6 < 1 * 6$$
$$0 + 1 \leq \text{random} * 6 + 1 < 6 + 1$$

$1 \leq$ random * 6 + 1 $<$ 7
$1 \leq$ (int) (random * 6 + 1) $\leq$ 6

Após o utilizador premir a tecla *Enter* no seu teclado (verificámos isto com o método *nextLine* da class Scanner) calculámos o segundo número da mesma forma que o primeiro. Em seguida utilizámos uma operação ternária dentro da instrução *println* para verificar se os dois números gerados são iguais. Caso a comparação venha a ser verdade temos que calcular o total da soma a dobrar, caso a comparação não se verifique apenas apresentamos o total da soma destes dois números.

Count4

4. (Count4)
Ler 4 valores inteiros e indicar quantos são iguais, quantos têm valor par e quantos têm valor ímpar. São valorizadas as soluções que minimizem o número decisões binárias (instruções if).

No programa Count4 para obtermos os números ímpares podemos somar o resto das divisões de cada número inserido por 2, pois cada número ímpar tem resto 1 na divisão por 2. Para o cálculo dos números pares basta subtrair o número total de ímpares, realizado anteriormente, a 4 (número total).

A próxima etapa deste programa foi calcular as igualdades entre números, sendo que para isto começámos por analisar se todos eles são iguais e, caso não sejam, fomos testar a igualdade entre 3 e 2 números sendo que se nenhum for igual temos que o número de números iguais é zero.

Travel

5. (Travel)
Ler a hora de partida e de chegada de uma viagem e apresentar a duração. A partida, a chegada e a duração são indicados em horas (0..23), minutos e segundos (0..59). O programa deve indicar se a partida é depois da chegada apresentando "Partida > Chegada" em vez de apresentar a duração.

No programa *Travel* optámos por converter o tempo da chegada e da partida para segundos, depois fizemos a diferença entre os segundos da chegada e os segundos da partida e, caso esta diferença venha a ser menor que zero então podemos concluir que Partida > Chegada. Caso a diferença for maior ou igual a zero então convertemos a diferença em segundos para horas, minutos e segundos.

Greeting

6. (`Greeting`)
Perguntar a hora e o nome do utilizador e apresentar a mensagem de cumprimento (Bom dia, Boa tarde ou Boa noite) conforme a hora. Considere que a hora introduzida é um valor inteiro entre 0 e 23. Caso seja lida a palavra "auto" (ou outra qualquer) em vez de um valor inteiro, o programa deve utilizar a hora atual, obtida através pela expressão `LocalTime.now().getHour()` usando a classe `LocalTime` do package `java.time`.

No programa Greeting começámos por verificar se o input na consola era do tipo *Int* com o método *hasNextInt* da classe *Scanner* e, caso isto seja verdade então podemos concluir que o utilizador inseriu um *Int* na consola e que a hora irá ser esse mesmo *Int*. Caso contrário vamos obter a hora do sistema operativo do utilizador através do método *getHour* e imprimir essa mesma hora no output.

Após a obtenção da hora fomos obter o nome do utilizador. Nós reparámos que ao obter o nome com o método *nextLine* (utilizámos o *nextLine* pois o utilizador poderia inserir o seu nome completo) tínhamos que utilizar um *nextLine* antes desse de forma a ignorar os inputs anteriores. Após isto feito considerámos que o dia corresponde às horas entre as 7h e as 12h inclusive, a tarde entre as 13h e 17h inclusive e, por fim, a noite às restantes horas.