Aula 9
Resolução do trabalho autónomo

Iniciativa Conjunta:

Com o apoio de:

Exercícios sobre vetores

1. Construir um vetor de números naturais até um dado número *n*. Exemplo: naturals(5)->{1,2,3,4,5}
2. Construir um vetor de dígitos aleatórios (números de 0 a 9), dado o comprimento. Exemplo: randomDigits(5)->{8,2,9,1,2}
3. Construir um vetor capaz de armazenar 50 números inteiros. Em seguida faça o seu preenchimento automático com os números 101 a 150, ou seja na posição número 0 fica 101, na posição número 1 fica 102 e por aí adiante.
4. Copiar (replicar) um vetor de inteiros, tendo o novo vetor o mesmo tamanho do argumento. Exemplo: copy({1,2,3},6)->{1,2,3,0,0,0} copy({1,2,3,4,5,6},3)->{1,2,3}
5. Verificar se existe um determinado número num vetor. Exemplo: exists(5,{1,3,4,5})->true exists(1,{2,3})->false
6. Contar o número de ocorrências de um determinado caractere. Exemplo: count(a,{a,b,c,a})->2
7. Construir um sub-vetor de outro vetor, dados os índices do primeiro elemento e último a incluir. Exemplo subarry(2,4,{a,d,r,a,c,r,w})}->{r,a,c}

Exercícios sobre vetores

1. Obter a primeira metade um vetor *v*, incluindo um parâmetro booleano para permitir se o elemento do meio é para incluir (caso o comprimento do vetor seja ímpar). Se o comprimento for par, este parâmetro não terá efeito. Exemplo firstHalf({b,a,s,w,q}, true)->{b,a,s}
2. Construir um vetor juntando outros dois vetores (parte esquerda e parte direita). Exemplo: merge({1,2},{9,10})
3. Construir um vetor invertido com base noutro. Ou seja, o novo vetor será composto pelos elementos do vetor dado pela ordem inversa. Exemplo: invert({t,q,a)}->{a,q,t}
4. Construir um vetor com base noutro, de modo o dobro do tamanho e cada elemento duplicado. Exemplo: duplicateEveryElement({a,s,d})->{a,a,s,s,d,d}
5. Construir um vetor com base noutro, sendo a primeira metade uma cópia e a segunda metade os mesmos elementos para ordem inversa. Exemplo: duplicateInverted({3,2,1})->(3,2,1,1,2,3)
6. Construir um vetor com base noutro, representado um cópia sem o elemento do meio (caso o tamanho seja ímpar) Exemplo: copyWithoutMiddleElement({1,2,3,4,5})->{1,2,4,5}
7. Construir um vetor com n números da sequência Fibonacci. Exemplo: fibonacciSequence(7)-> {0,1,1,2,3,5,8}

Trabalho autónomo da aula 6

1. Crie um sistema de gestão de alunos numa sala. Para tal, deverá criar as classes Room e Student. Cada sala tem a sua capacidade, o nome do bloco (ex: A, B, C, D) e o número da sala. Cada aluno terá o número de aluno, o nome e o curso. Deve ser possível realizar as seguintes operações:
    a. adicionar alunos a uma sala (até ao limite da sua capacidade)
    b. remover um aluno específico, com base no seu número
    c. listar todos os alunos que estão na sala

2. Pretende-se desenvolver um programa que simula uma playlist de músicas. A cada música está associado o título e a duração da mesma. A playlist guarda uma lista de músicas e deverá permitir as seguintes operações:
    a. mostrar a lista de músicas atualmente na playlist
    b. acrescentar ou retirar músicas à playlist
    c. calcular a duração total de todas as músicas contidas na playlist

Trabalho autónomo da aula 6

3. Implemente a classe Calculator. O objeto calculadora deve ser capaz de realizar os cálculos matemáticos comuns de uma calculadora normal:
    a. Soma
    b. Subtração
    c. Multiplicação
    d. Divisão
    e. Potência
    f. Resto da divisão
    g. Fórmula resolvente
    h. A calculadora possui ainda uma história das operações realizadas, armazenadas em formato String num vetor. Desenvolva as funções necessárias para:
        i. guardar o histórico no vetor
        ii. obter o histórico completo
        iii. obter as últimas operações realizadas pela calculadora (histórico parcial)

Trabalho autónomo da aula 6

4. Implemente a classe Car. Um carro possui algumas propriedades e caraterísticas que variam de modelo para modelo, tais como a marca, o modelo, o número de lugares, a matrícula, o mês e ano de registo, o consumo em 100km, etc... Certos atributos devem ser indicados aquando a criação do carro. Deve criar os testes que achar necessário para testar as funções desenvolvidas.
    a. Desenvolva os getters e os setters dos atributos que lhe pareçam adequados.
    b. O carro possui um tanque de combustível, com uma capacidade atual e uma capacidade máxima. Implemente a função encherDeposito() que simula o abastecimento.
    c. A função run() deve simular o percurso feito pelo automóvel em 1 km. O combustível associado a este percurso deve ser descontado na capacidade atual do tanque.
    d. Deve ser possível determinar se o carro se encontra em funcionamento. Implemente a função isLigado() e outras que considera necessárias.
    e. O automóvel possui um proprietário, que é representado pela classe Person (desenvolvida na aula). Crie as funções e atributos necessários que permitem o
    registo do proprietário à viatura.
    f. Implemente a função toString() que deve devolver a informação que achar adequada.

Trabalho autónomo da aula 6

5. Implemente a classe CreditCard cujas instâncias deverão ter um comportamento semelhante ao do conhecido cartão. Cada instância da classe CreditCard deverá possuir, ao ser criada, um titular, um número de 12 dígitos, um mês e ano de validade e um valor máximo de débito autorizado. Deve também guardar o montante despendido até ao momento e um histórico dos movimentos realizados. Por questões de simplicidade, considere que um movimento é uma String com o valor e uma descrição (ex: “30EUR - Bilhete de futebol”). A classe deve implementar os seguintes métodos:
    a. int saldo() - devolver o saldo do cartão (diferença entre o montante gasto e o limite de endividamento)
    b. void pagarCredito(int pag) - efetuar um pagamento, isto é, abater o valor pag ao montante em dívida
    c. void gastar(int quantia, String descr) - registar um movimento e atualizar o montante gasto
    d. String obterTalao() - devolver a String que corresponde ao último movimento realizado
    e. String getMovimentos() - devolver a lista de movimentos efetuados sob a forma de uma String

Trabalho autónomo da aula 8

1. Crie a classe-base peça de xadrez que deve ser inicializada com uma posição (pode codificar a posição como uma coordenada com dois inteiros. Se quiser, no toString pode traduzir para a notação de coordenadas do xadrez).

   Crie as classes derivadas peão e cavalo e um método, abstrato na classe-base, chamado movimentosPossiveis() que devolve uma lista de todos os movimentos válidos para a peça em questão (sem contar com a possível tomada de peças nem com casas ocupadas). Implemente o método para peões e cavalos e teste.

Trabalho autónomo da aula 8

2. Implemente a hierarquia de classes ContaBancaria (superclasse), ContaCorrente (com senha, número, saldo e quantidade de transações realizadas) e ContaPoupanca (com senha, número, saldo e taxa de rendimento).
    a. Quando uma ContaBancaria for criada, informe a senha da conta por parâmetro.
    b. Na classe ContaBancaria, crie os seguintes métodos abstratos:
        i. Levanta(double valor);
        ii. deposita(double valor)
        iii. tiraExtrato()
    c. Nesta mesma classe, crie o método alteraSenha, que recebe uma senha por parâmetro e deve confirmar a senha anterior (via teclado), e somente se a senha anterior estiver correta a senha recebida por parâmetro deve ser atribuída.
    d. Implemente os métodos abstratos nas classes ContaCorrente e ContaPoupanca.
    e. Crie os métodos de acesso (getters) para os atributos de ContaCorrente e ContaPoupanca.

Trabalho autónomo da aula 8

3. Crie uma classe calculadora. Esta classe deve ser abstrata e implementar as operações básicas (soma, subtração, divisão e multiplicação). Utilizando o conceito de herança crie uma classe chamada calculadora científica que implementa os seguintes cálculos:
   a. raiz quadrada;
   b. potência.

   Dica: utilize a classe Math do pacote java.lang.

Trabalho autónomo da aula 8

4. Implemente uma classe abstracta de nome Forma onde são declarados dois métodos abstractos: float calcularArea(); e float calcularPerimetro();
    a. Crie, como subclasse de Forma, uma classe de nome Rectangulo cujas instâncias são caracterizadas pelos atributos lado e altura ambos do tipo float.
    b. Implemente na classe Rectangulo os métodos herdados de Forma e outros que ache necessários.
    c. Crie, como subclasse de Forma, uma classe de nome Círculo cujas instâncias são caracterizadas pelo atributo raio do tipo float.
    d. Implemente na classe Circulo os métodos herdados de Forma e outros que ache necessários. Dica: poderá aceder ao valor de Pi fazendo Math.Pi.
    e. Crie, como subclasse de Rectangulo, uma classe de nome Quadrado cujas instâncias são caracterizadas por terem os atributos lado e altura com o mesmo valor.
    f. Elabore um programa de teste onde é declarado um array, de dimensão 5, do tipo Forma. Nesse array, devem ser guardadas instâncias de Rectangulo, Circulo e Quadrado.
    g. Depois, implemente um ciclo que percorra o array evocando, relativamente a cada um dos objectos guardados, os métodos calcularArea e calcularPerimetro e imprima a informação para o ecrã.