Pontifícia Universidade Católica de Minas Gerais

Praça da Liberdade

Engenharia de Software

Manhã/Noite

Wesley Dias Maciel

Laboratório de Computação II

Distribuição de Pontos

Manhã:

Avaliação	Data	Pontos
Prova 01	26/09/2017	30
Trabalho Prático	14/11/2017	15
Práticas de Laboratório	21/11/2017	25
Prova 02	28/11/2017	30
Total		100

Noite:

Avaliação	Data	Pontos
Prova 01	30/09/2017	30
Trabalho Prático	11/11/2017	15
Práticas de Laboratório	18/11/2017	25
Prova 02	25/11/2017	30
Total		100

Observações:

Trabalhos Práticos:

1) O trabalho prático poderá ser realizado em equipes de até 03 alunos por equipe.
2) Cada equipe deverá apresentar seu trabalho para o professor na data especificada acima. Durante a apresentação, o professor fará perguntas sobre o trabalho, para cada aluno da equipe. Cada aluno da equipe deverá responder às perguntas individualmente. A nota de cada integrante da equipe será independente das notas dos demais. Assim, cada integrante da equipe poderá ter uma nota diferente.
3) Cada equipe deve também postar o trabalho no Edmodo até as 23:45hs da data apresentada acima. O Edmodo não aceitará trabalho postado fora desse prazo. O trabalho não poderá ser enviado por email ou outros meios. Apenas um arquivo compactado com todas as respostas do trabalho deve ser postado no Edmodo. O arquivo deve conter os nomes dos integrantes da equipe.

Práticas de Laboratório:

1) As práticas de laboratório poderão ser realizadas em equipes de até 03 alunos.
2) Cada equipe deverá apresentar suas práticas para o professor na data especificada acima. Durante a apresentação, o professor fará perguntas sobre as práticas, para cada aluno da equipe. Cada aluno da equipe deverá responder às perguntas individualmente. A nota de cada integrante da equipe será independente das notas dos demais. Assim, cada integrante da equipe poderá ter uma nota diferente.

Prática 01

Revisão sobre a linguagem de programação C.

1) Em linguagem de programação C, escreva um algoritmo que leia dois números inteiros a partir do teclado e que apresente como resposta a soma dos dois números informados.

2) Em linguagem de programação C, escreva um algoritmo que leia dois números positivos e maiores que zero a partir do teclado e que apresente como resposta o logaritmo do primeiro número na base representada pelo segundo número.
   OBS:
$$\log_b(x) = \frac{\log_a(x)}{\log_a(b)}$$

3) Em linguagem de programação C, escreva um algoritmo que leia dois números reais a partir do teclado e que apresente como resposta o maior número informado.

4) Em linguagem de programação C, escreva um algoritmo que leia o código de uma operação a partir o teclado e que realize a operação associada a esse código. O código é um valor inteiro. Os códigos e operações são listados abaixo:

Código	Operação
1	Área de um retângulo. Os valores da base e da altura do retângulo devem ser informados a partir do teclado. OBS: $a_{retângulo}$ = base * altura.
2	Área de um triângulo. Os valores da base e da altura do triângulo devem ser informados a partir do teclado. OBS: $a_{triângulo}$ = (base * altura) / 2.
3	Área de um círculo. O valor do raio do círculo deve ser informado a partir do teclado. OBS: $a_{círculo}$ = 3,14 * (raio)^2.
4	Área de um trapézio. Os valores da base maior, da base menor e da altura do trapézio devem ser informados a partir do teclado. OBS: $a_{trapézio}$ = [(base maior + base menor) * altura] / 2.

5) Altere o algoritmo do exercício anterior, para que ele seja executado indefinidamente.

6) Altere o algoritmo do exercício anterior, para que ele seja executado enquanto o usuário informar o caractere ‘s’ ou ‘S’ a partir do teclado.

7) Em linguagem de programação C, escreva um algoritmo que leia um limite inferior e um limite superior a partir do teclado e que apresente os números pares entre esses dois limites. Os limites devem ser números inteiros.

8) Em linguagem de programação C, escreva um algoritmo que leia 5 números reais a partir do teclado e os armazene em um vetor de 5 posições. O algoritmo deve apresentar na tela o valor de cada posição do vetor com um desconto de 10%.

9) Em linguagem de programação C, escreva um algoritmo que leia os valores de duas matrizes 4 x 4 a partir do teclado e que calcule a soma das duas matrizes. Apresente a matriz resultante na tela.

10) Em linguagem de programação C, escreva um algoritmo que possua um procedimento para calcular o volume de uma esfera. O valor do raio da esfera deve ser lido a partir do teclado. O volume da esfera deve ser apresentado na tela.

    OBS: $volume_{esfera} = (4 * 3{,}14 * raio^3) / 3$.

11) Em linguagem de programação C, escreva um algoritmo que possua um procedimento para converter uma temperatura fornecida em graus Fahrenheit para a temperatura correspondente em graus Celsius. A temperatura em graus Fahrenheit deve ser informada como parâmetro. A temperatura em graus Celsius deve ser apresentada na tela.
    OBS: $C = (5 / 9) * (F – 32)$.

12) Em linguagem de programação C, escreva um algoritmo que possua uma função que receba três valores reais como parâmetro e que retorne a média aritmética desses três valores.

13) Em linguagem de programação C, escreva um algoritmo que leia um número real a partir do teclado. Crie um ponteiro para esse número. Apresente na tela:
    a. O endereço da variável que armazena o número real.
    b. O conteúdo da variável que armazena o número real.
    c. O endereço do ponteiro.
    d. O conteúdo do ponteiro.
    e. O conteúdo da área para onde o ponteiro aponta.

14) Em linguagem de programação C, escreva um algoritmo que possua uma estrutura aluno. A estrutura deve conter os campos nome, dada de nascimento, curso e nota. O algoritmo deve ler os dados de três alunos a partir do teclado. O algoritmo deve apresentar na tela os dados do aluno que tiver a maior nota.

15) Altere o algoritmo do exercício anterior. Escreva:
    a. Um procedimento que preencha a estrutura com os dados de um aluno. O procedimento deve receber o endereço da estrutura como parâmetro.
    b. Um procedimento para apresentar na tela os dados do aluno. O procedimento deve receber o endereço da estrutura como parâmetro.

16) Altere o algoritmo anterior. Ele deve alocar dinamicamente a área de memória para a estrutura aluno. A área de memória deve ser desalocada ao final do algoritmo.

17) Altere o algoritmo anterior. Ele deve possuir um procedimento que receba um vetor de estruturas aluno como parâmetro e que apresente na tela os dados do aluno que tiver a menor nota.

Prática 02

Introdução à linguagem de programação Java.

1) Traduza os algoritmos da prática 01 para a linguagem de programação Java.

Prática 03

Princípios de Programação Orientada a Objeto (POO) em linguagem de programação Java: abstração e herança.

Abstração: utilizada para a definir entidades do mundo real com seus atributos (características, propriedades) e métodos (ações, fucnionalidades).

1) Em linguagem de programação Java, escreva um algoritmo que:
    a. Possua um pacote com nome “pessoa”. O pacote “pessoa” deve possuir uma classe com nome “Pessoa”. A classe “Pessoa” deve possuir:
        i. Atributos: nome, endereço e telefone.
        ii. Construtores: construtores da classe (vazio e com os parâmetros de cada atributo).
        iii. Métodos:
            1. Métodos de acesso para cada atributo (get/set).
            2. Método “getIdentificacao ()”, conforme apresentado abaixo:

```
public String getIdentificacao () {
  return this.getNome () + ", " +
         this.getTelefone () + ", " +
         this.getEndereco () + ".";
}
```

    b. Possua um pacote com nome “inicio”. O pacote “inicio” deve possuir uma classe com nome “Principal”. A classe “Principal” deve:
        i. Instanciar um objeto da classe “Pessoa”.
        ii. Pedir ao usuário que informe os atributos da classe “Pessoa” a partir do teclado.
        iii. Imprimir na tela os atributos informados pelo usuário.
        iv. Imprimir na tela a identificação gerada pela classe “Pessoa”.

Herança: utilizada para que uma classe herde atributos e métodos de outra classe. Uma vantagem de se usar herança é a reutilização de código. A reutilização de código ocorre quando se identifica atributos e métodos comuns entre classes distintas. Para efetuar uma herança entre duas classes, emprega-se a palavra reservada “extends”. A classe herdada é chamada de superclasse. A classe que herda é chamada de subclasse. Para verificar se a herança foi empregada corretamente, realiza-se o teste “É UM”. Esse teste ajuda perceber se a subclasse pode herdar a superclasse.

2) Altere o exercício 1 de forma que:
    a. O pacote pessoa possua as classes “PessoaFisica” e “PessoaJuridica” que estendem/herdam da classe “Pessoa”. Exemplo de teste: uma “PessoaFisica” É UMA “Pessoa”.
    b. A classe “PessoaFisica” deve possuir:
        i. Atributos: carteira de identidade e CPF.
        ii. Construtores: chamada do construtor da superclasse e construtores da classe (vazio e com os parâmetros de cada atributo).
        iii. Métodos:
            1. Métodos de acesso para cada atributo (get/set).
    c. A classe “PessoaJurídica” deve possuir:
        i. Atributo: CNPJ.

ii. Construtores: chamada do construtor da superclasse e construtores da classe (vazio e com o parâmetro do atributo CNPJ).
iii. Métodos:
1. Métodos de acesso para o atributo CNPJ (get/set).

d. A classe “Principal” deve:
i. Instanciar um objeto da classe “PessoaFísica” e um objeto da classe “PessoaJurídica”.
ii. Pedir ao usuário que informe a partir do teclado os atributos de cada um dos objetos instanciados.
iii. Imprimir na tela os atributos informados pelo usuário, para cada um dos objetos instanciados.
iv. Imprimir na tela a identificação gerada pela classe “Pessoa”, para cada um dos objetos instanciados.

3) Em linguagem de programação Java, escreva o algoritmo de controle das contas bancárias de um banco. O algoritmo deve gerenciar as contas correntes e de poupança dos clientes. As contas correntes e de poupança possuem o atributo saldo e os métodos para depósito, saque e verificação de saldo. As contas correntes possuem um método para cobrança da taxa de serviço e administração do banco. A taxa de serviço e administração diminui o valor do saldo em 5%. As contas de poupança possuem um método para rendimento. O rendimento das contas de poupança aumentam o saldo em 3%. O algoritmo deve permitir:
a. A criação de 3 contas bancárias: 1 conta corrente e 2 contas de poupança.
b. O depósito, saque e verificação de saldo das contas.
c. A cobrança da taxa de serviço e administração da conta corrente.
d. A premiação das contas de poupança com o devido rendimento.

4) Uma companhia aérea trabalha com voos nacionais e internacionais. Cada voo possui um identificador, preço, cidade de origem, cidade de destino, data e hora de partida, data e hora de chegada. A companhia cobra uma taxa de embarque e um valor de seguro em todos os voos internacionais. Em linguagem de programação Java, escreva o algoritmo para gerenciamento dos voos dessa companhia. O algoritmo deve permitir:
a. A criação de um vetor com 5 voos: 2 voos nacionais e 3 voos internacionais.
b. O cadastro dos dados de cada voo.
c. A verificação dos dados de cada voo.
d. A determinação do voo mais caro.

5) Uma imobiliária trabalha com aluguel e venda de imóveis. Cada imóvel possui um endereço e um preço. Os imóveis novos recebem um adicional no preço. Os imóveis usados recebem um desconto no preço. Em linguagem de programação Java, escreva o algoritmo para gerenciamento dos imóveis dessa imobiliária. O algoritmo deve permitir:
a. A criação de um vetor com 5 imóveis para alugar: 3 usados e 2 novos.
b. A criação de um vetor com 4 imóveis para venda: 1 usado e 3 novos.
c. A visualização dos dados do imóvel com menor valor de aluguel.
d. A visualização dos dados do imóvel com maior preço de venda.
e. A gerência do vetor de imóveis para alugar quando é firmado o contrato de aluguel de um imóvel.
f. A gerência do vetor de imóveis para venda quando é firmado o contrato de venda de um imóvel.

Prática 04

Princípios de Programação Orientada a Objeto (POO) em linguagem de programação Java: passagem, troca, de mensagem, sobrecarga, sobrescrita e polimorfismo.

Passagem de mensagem: em linguagens orientadas a objetos, os objetos se comunicam a partir de passagem, ou troca, de mensagem. Uma mensagem é um sinal enviado de um objeto a outro, requisitando um serviço, uma operação programada no objeto chamado. A invocação de um método é a passagem de mensagem para o objeto. A passagem de mensagem somente ocorre entre objetos que possuem uma associação. Quando uma mensagem é recebida, uma operação é invocada/executada no objeto chamado.

Exemplo:

```
Pessoa p = new Pessoa ("Ana");
p.identificacao ();
```

Sobrecarga: a sobrecarga de método, ou ***overload***, ocorre quando dois ou mais métodos possuem o mesmo nome, mas suas listas de argumentos são diferentes. Assim, as assinaturas dos métodos são diferentes por possuírem argumentos diferentes. A sobrecarga de métodos pode ocorrer em uma mesma classe ou entre classes que participam de um relacionamento de herança, entre classes ancestrais e descendentes.

Exemplo:

```
public class Pessoa {
  public String nome;

  public Pessoa () {
  }

  public Pessoa (String nome) {
      this.nome = nome;
  }

  public String identificacao () {
      return "Nome: " + this.nome + ".";
  }

  public String identificacao (String CPF) {
      return "Nome: " + this.nome + ", CPF: " + CPF;
  }
}
```

O exemplo apresenta sobrecarga de construtor (`public Pessoa ()` e `public Pessoa (String nome)`) e sobrecarga de método (`public String identificacao ()` e `public String identificacao (String CPF)`).

Sobrescrita: a sobrescrita de método, ou ***override***, ocorre quando dois ou mais métodos possuem a mesma assinatura, mas comportamentos, funcionalidades, distintos. Assim, as assinaturas dos métodos são iguais, mas as operações nos corpos dos métodos são diferentes. A sobrescrita de métodos ocorre entre classes que participam de um relacionamento de herança, entre classes ancestrais e descendentes. Métodos sobrescritos recebem a anotação **@Override**.

Exemplo:

```
public class Pessoa {
  public String nome;

  public Pessoa (String nome) {
      this.nome = nome;
  }

  public String identificacao () {
      return "Nome: " + this.nome + ".";
  }
}

public class PessoaFisica extends Pessoa {
  public String CPF;

  public PessoaFisica (String nome, String CPF) {
    super (nome);
    this.CPF = CPF;
  }

  @Override //Anotação de sobrescrita/reescrita de método.
  public String identificacao () {
    return "Nome: " + this.nome + ", CPF: " + this.CPF + ".";
  }
}
```

No exemplo, o método `public String identificacao ()` da classe “PessoaFisica” sobrescreve o método `public String identificacao ()` da classe “Pessoa”. As classes “Pessoa” e “PessoaFisica” participam de um relacionamento de herança, “PessoaFisica” É UMA “Pessoa”.

OBS: a sobrescrita de métodos ocorre quando há métodos com a mesma identificação, assinatura, mas que possuem comportamentos distintos, especializados para cada classe.

Polimorfismo: o polimorfismo refere-se à capacidade de se processar objetos de formas diferentes, dependendo do tipo, classe, dos objetos. Assim, o polimorfismo significa processar objetos de forma diferente com base em seu tipo de dados. Ocorre quando se fornece uma interface única para entidades de tipos diferentes. Mais especificamente, é a capacidade de redefinir métodos para classes derivadas. Um tipo polimórfico é aquele cujas operações podem ser aplicadas a valores de outros tipos: um método com múltiplas implementações para uma certa ação. A implementação a ser usada é decidida no tempo de execução. Dessa

forma, a funcionalidade é selecionada no decorrer da execução do programa. Logo, um objeto pode se comportar de maneiras diferentes ao receber uma mensagem, dependendo do seu tipo de criação. O comportamento de um método depende do contexto em que o método é invocado. Nesse caso, o contexto é dado pelo tipo, classe, do objeto.

Exemplo:

```
package pessoa;
public class Pessoa {
  public String nome;

  public Pessoa (String nome) {
      this.nome = nome;
  }

  public String identificacao () {
      return "Nome: " + this.nome + ".";
  }
}

package pessoa;
public class PessoaFisica extends Pessoa {
  public String CPF;

  public PessoaFisica (String nome, String CPF) {
    super (nome);
    this.CPF = CPF;
  }

  @Override //Anotação de sobrescrita/reescrita de método.
  public String identificacao () {
    return "Nome: " + this.nome + ", CPF: " + this.CPF + ".";
  }
}

package inicio;
import pessoa.*;
public class Principal {

  public static void main (String[] args) {
    Pessoa p = new PessoaFisica ("Ana", "999.999.999-99");

    System.out.println ("---------- Pessoa Informada: ----------");
    System.out.println (p.identificacao());
  }
}
```

No exemplo, “p” é do tipo “Pessoa”, mas recebe uma referência para um objeto do tipo “PessoaFisica”. Ao passar a mensagem “identificação ()” para “p”, o método sobrescrito `public String identificacao ()` da classe “PessoaFisica” é executado.

1) Em linguagem de programação Java, escreva um algoritmo que possua um pacote com o nome “animal”. Nesse pacote, defina as classes “Animal”, “Elefante”, “Gorila”, “Leao”, “Urso” e “Zebra”, conforme apresentado abaixo:

```java
package animal;
public class Animal {
  public String nome;

  public Animal (String nome) {
    this.nome = nome;
  }

  public String toString () {
    return getClass ().getSimpleName ();
  }

  public void emitirSom () {
    System.out.println ("Som de animal.");
  }
}

package animal;
public class Elefante extends Animal {
  public Elefante () {
    super ("[elefante]");
  }

  @Override //Anotação de sobrescrita/reescrita de método.
  public void emitirSom () {
    System.out.println ("Som de elefante.");
  }
}

package animal;
public class Gorila extends Animal {
  public Gorila () {
    super ("[gorila]");
  }

  @Override //Anotação de sobrescrita/reescrita de método.
  public void emitirSom () {
    System.out.println ("Som de gorila.");
  }
}

package animal;
public class Leao extends Animal {
  public Leao () {
    super ("[leao]");
  }

  @Override //Anotação de sobrescrita/reescrita de método.
  public void emitirSom () {
    System.out.println ("Som de leao.");
  }
}
```

```java
package animal;
public class Urso extends Animal {
  public Urso () {
    super ("[urso]");
  }

  @Override //Anotação de sobrescrita/reescrita de método.
  public void emitirSom () {
    super.emitirSom();
  }
}
```

```java
package animal;
public class Zebra extends Animal {
  public Zebra () {
    super ("[zebra]");
  }
}
```

O algoritmo deve possuir também um pacote com o nome "início". Nesse pacote defina uma classe com o nome "Principal". Na classe "Principal", crie o método `public static void main (String[] args)`. Nesse método, crie um vetor de 10 animais,

```java
Animal[] zoologico = new Animal[10];
```

Cada posição do vetor deve receber a referência para um objeto de uma classe do pacote "animal". Percorra o vetor passando para os objetos a mensagem para execução do método sobrescrito nas respectivas classes.

2) Em linguagem de programação Java, escreva um algoritmo que possua uma classe "Figura". A classe "Figura" deve possuir o método `public double getArea ()` que retorna a área de uma figura. As classes "Círculo", "Retangulo", "Triângulo" e "Trapezio" devem ser subclasses de "Figura". Cada subclasse deve sobrescrever o método `public double getArea ()` com sua correspondente implementação para cálculo de área. Crie uma matriz 3 x 3 de figuras. Percorra a matriz passando para os objetos a mensagem para execução do método sobrescrito nas respectivas classes.

   OBS: $a_{círculo} = 3{,}14 * (raio)^2$, $a_{retângulo} = base * altura$, $a_{triângulo} = (base * altura) / 2$, $a_{trapézio} = [(base\ maior + base\ menor) * altura] / 2$.

3) Em linguagem de programação Java, escreva um algoritmo para o controle de estoque de uma empresa. Essa empresa trabalha com as seguintes categorias de produtos: armário, mesa, cadeira, fogão e refrigerador. O algoritmo deve informar:
   a. O número de produto de cada categoria no estoque.
   b. O total de produtos no estoque.
   c. O preço de cada categoria de produto no estoque.
   d. O valor total em mercadorias no estoque.

   O algoritmo deve empregar sobrecarga de construtor/método, sobrescrita de método e polimorfismo.

Prática 05

Princípios de Programação Orientada a Objeto (POO) em linguagem de programação Java: classes abstratas e interfaces.

Classes Abstratas: classes abstratas não podem ser instanciadas e são definidas pela palavra reservada “abstract”. Elas servem como “molde”, ou “modelo”, para as classes que dela herdam. As classes concretas que herdam de uma classe abstrata podem ser instanciadas. Classes abstratas podem conter métodos abstratos. Os métodos abstratos não possuem definição, não possuem corpo. Os métodos abstratos precisam ser definidos nas classes concretas que herdam da classe abstrata. Esses métodos devem ser sobrescritos nas classes concretas.

Exemplo:

```
package conta;

public abstract class Conta {
  private double saldo;

  public void setSaldo (double saldo) {
    this.saldo = saldo;
  }

  public double getSaldo () {
    return saldo;
  }

  public abstract void imprimeExtrato ();
}


package conta;

import java.text.SimpleDateFormat;
import java.util.Date;

public class ContaPoupanca extends Conta {

  @Override
  public void imprimeExtrato () {
    Date date = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat ("dd/MM/yyyy
                                                  HH:mm:ss");

    System.out.println ("---------- Extrato da conta ----------");
    System.out.println ("Data: "+ sdf.format (date));
    System.out.println ("Saldo: " + this.getSaldo ());
    System.out.println ("--------------------------------------");
  }
}

package inicio;
```

```java
import conta.*;

public class Principal {
  public static void main(String[] args) {
    Conta cp = new ContaPoupanca ();
    cp.setSaldo (3458.78);
    cp.imprimeExtrato ();
  }
}
```

Interface: uma interface especifica os métodos que devem ser definidos pelas classes que a implementam. Uma interface funciona como um contrato, uma imposição. O contrato define o conjunto de métodos a ser definido nas classes que implementam a interface. Uma interface é completamente abstrata, seus métodos são definidos como “abstract”. Caso haja variáveis numa interface, elas são sempre constantes (“static final”) por padrão. Uma interface é definida através da palavra reservada “interface”. Uma classe implementa uma interface através da palavra reservada “implements”. A linguagem Java não permite herança múltipla: cada classe herda apenas uma classe. Entretanto, uma classe pode implementar várias interfaces. A classe que implementa uma interface tem que definir todos os métodos da interface ou se tornar uma classe abstrata.

Exemplo:

```java
package conta;

public interface Conta {
  public abstract void depositar (double valor);
  abstract void sacar (double valor);
  double getSaldo ();
}
```

```java
package conta;

public class ContaCorrente implements Conta {
  private double saldo;
  private double taxaOperacao = 0.45;

  @Override
  public void depositar (double valor) {
    this.saldo += valor - taxaOperacao;
  }

  @Override
  public void sacar (double valor) {
    this.saldo -= valor + taxaOperacao;
  }

  @Override
  public double getSaldo () {
    return this.saldo;
  }
```

```
}
```

```
package conta;

public class ContaPoupanca implements Conta {
  private double saldo;

  @Override
  public void depositar (double valor) {
    this.saldo += valor;
  }

  @Override
  public void sacar (double valor) {
    this.saldo -= valor;
  }

  @Override
  public double getSaldo() {
    return this.saldo;
  }
}
```

```
package conta;

import java.text.SimpleDateFormat;
import java.util.Date;

public class Extrato {
  public void getExtrato (Conta conta) {
    Date date = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat ("dd/MM/yyyy
                                                  HH:mm:ss");
    System.out.println ("---------- Extrato da conta ----------");
    System.out.println ("Data: "+ sdf.format (date));
    System.out.println ("Extrato atual: " + conta.getSaldo ());
    System.out.println ("-------------------------------------\n\n");
  }
}
```

```
package inicio;

import conta.*;

public class Principal {
  public static void main (String[] args) {
    ContaCorrente cc = new ContaCorrente ();
    cc.depositar (270.20);
    cc.sacar (60);
```

```
    ContaPoupanca cp = new ContaPoupanca ();
    cp.depositar (90.50);
    cp.sacar (25);

    Extrato e = new Extrato ();
    e.getExtrato (cc);
    e.getExtrato (cp);
  }
}
```

1) Em linguagem de programação Java, escreva um algoritmo para gerenciamento de uma empresa que trabalha com aluguel de veículos. A empresa aluga caminhões, ônibus, utilitários, carros e motos. A empresa cobra uma taxa de seguro de:
    a. 4% pela locação de motos.
    b. 3% pela locação de carro.

   A empresa concede um desconto de:
    a. 3% pela locação de ônibus.
    b. 4% pela locação de caminhões.

   A empresa precisa se manter informada sobre:
    a. O número de veículos alugados de cada categoria.
    b. O número total de veículos alugados.
    c. O número de veículos de cada categoria disponíveis para locação.
    d. O número total de veículos disponíveis.
    e. O valor do aluguel de cada categoria.
    f. O valor a ser recebido pelas locações.

   O algoritmo deve empregar classe abstrata.

2) Em linguagem de programação Java, escreva um algoritmo polimórfico para operações matemáticas. A partir de valores informados pelo usuário, o algoritmo deve realizar uma operação matemática e imprimir o seu resultado. O algoritmo deve empregar interface. Cada classe deve possuir um construtor padrão.

3) Em linguagem de programação Java, escreva um algoritmo polimórfico para gerencia das vendas realizadas por uma empresa. O algoritmo deve possuir uma interface “Venda”. O algoritmo deve percorrer um vetor das vendas realizadas pela empresa e calcular o total das vendas e o número de produtos vendidos. A empresa trabalha com vendas a vista, com pagamento por boleto, com pagamento por cartão de débito e com pagamento por cartão de crédito.

Prática 06

Recursividade.

1) Em linguagem de programação Java, escreva um algoritmo que possua um método recursivo para cálculo de potenciação. Exemplo: $base^5 = base * base^4$, $base^n = base * base^{(n-1)}$.

2) Em linguagem de programação Java, escreva um algoritmo que implemente o método recursivo abaixo.

```
public long fatorial (long n) {
    if (n < 0) return 0; //Finalização por caso não previsto.
    if (n == 0) return 1; //Caso base.
    else return n * fatorial (n - 1); //Etapa recursiva.
}
```

3) Em linguagem de programação Java, escreva um algoritmo que implemente o método recursivo da questão anterior com uma única instrução “return”.

4) Em linguagem de programação Java, escreva um algoritmo recursivo que apresente na tela a série de Fibonacci. O usuário deve informar o número k de termos da série que devem ser apresentados. Na série de Fibonacci, um novo termo é obtido pela soma dos dois termos anteriores. Os dois termos iniciais são 0 e 1. Assim, o terceiro número na série de Fibonacci (fib_3) é obtido pela soma dos dois termos anteriores ($fib_2 + fib_1$): 0, 1, 1. O k-ésimo termo (fib_k) é obtido por $fib_{k-1} + fib_{k-2}$: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

5) Em linguagem de programação Java, escreva uma versão iterativa para o algoritmo de Fibonacci apresentado na questão anterior.

6) Em linguagem de programação Java, escreva um algoritmo que:
   a. Possua um método recursivo para apresentar os elementos de um vetor na ordem em que foram informados.
   b. Possua um método recursivo para apresentar os elementos de um vetor na ordem inversa daquela em que foram informados.
   c. Possua um método recursivo para inverter os valores inseridos em um vetor. O último elemento deve ser deslocado para a primeira posição; o penúltimo elemento, para a segunda posição e assim por diante.
   d. Possua uma versão iterativa do método do item anterior.

7) Em linguagem de programação Java, escreva um algoritmo que implemente o método recursivo abaixo que soma os elementos inseridos em um vetor.

```
public float soma (float[] A, int N) {
  if(N == 1) return A[0]; /*Caso base: se tamanho igual a um, a
                            solução é o primeiro elemento.*/
  else return (A[N - 1] + soma (A, (N - 1)));
} // fim de soma ().
```

8) Em linguagem de programação Java, escreva um algoritmo que implemente o método recursivo da questão anterior com uma única instrução “return”. A instrução “return” deve finalizar o método.

9) Em linguagem de programação Java, escreva um algoritmo que implemente um método recursivo para calcular e retornar o maior valor presente em um vetor.

10) Em linguagem de programação Java, escreva um algoritmo que implemente um método recursivo para calcular e retornar o maior valor presente em uma matriz.

11) Em linguagem de programação Java, escreva um algoritmo que implemente um método recursivo para verificar se uma string é um palíndromo ou não. Em um palíndromo, é obtido o mesmo valor não importa o sentido da leitura (da direita para a esquerda ou da esquerda para a direita). Exemplo: “arara”, “radar”, “ovo”, “ana”, “anilina”.

12) Em linguagem de programação Java, escreva uma versão iterativa para o algoritmo da questão anterior.

Prática 07

Princípios de Programação Orientada a Objeto (POO) em linguagem de programação Java: encapsulamento.

Encapsulamento: utilizado para esconder, não expor, detalhes internos da classe, tornando partes do sistema independentes. O encapsulamento oculta detalhes de implementação da classe. A interface corresponde ao que é disponibilizado para o usuário, tudo que é visível para o usuário. O encapsulamento é conseguido através dos **modificadores de acesso**: "*public*", "*protected*", *default* (*package private*) e "*private*". O acesso aos membros (atributos e métodos) da classe é realizado através de métodos de acesso "*get*" e "*set*". As classes podem ser escritas com os modificadores de acesso "*public*" ou *default* (*package-private*, sem modificador explícito). Uma classe declarada como "*public"* é visível a todas as classes do programa. Uma classe sem modificador de acesso é visível apenas em seu pacote. Os membros de uma classe (atributos e métodos) podem ser escritos com os modificadores "*public*", "*protected*", "*package-private*" (*default*, sem modificador explícito) ou "*private*". Visisbilidade dos modificadores de acesso para os membros de classe:

1) ***private***: membros declarados com acesso privado são acessíveis apenas na própria classe.
2) ***package-private***: membros declarados sem modificador de acesso são acessíveis apenas às classes dentro do mesmo pacote.
3) ***protected***: membros declarados com acesso protegido são acessíveis às classes do pacote e adicionalmente por suas subclasses.
4) ***public***: membros declarados com acesso público são acessíveis de qualquer lugar do programa.

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	o
no modifier	+	+	+	o	o
private	+	o	o	o	o

+ : accessible
o : not accessible

Use o nível de acesso mais restrito possível para os membros de uma classe. Dessa forma, use o modificador de acesso "*private*" sempre que possível. Evite membros com o modificador de acesso "public", a não ser que seja um atributo constante. Membros públicos aumentam o acoplamento em relação a uma implementação específica e reduz a flexibilidade do sistema a mudanças.

1) Analise o algoritmo abaixo escrito em linguagem de programação Java e execute-o no IDE Eclipse. Há erro(s) no algoritmo? Caso haja, apresente a causa desse(s) erro(s) e corrija-o(s), usando o nível de acesso mais restrito possível?.

```
package teste;
```

```java
public class ClasseZ {
  private   String a;
            String b;
  protected String c;
  public    String d;
}
```

```java
package teste;

public class ClasseY extends ClasseZ {
  protected String e;
  ClasseZ z = new ClasseZ ();

  public ClasseY () {
    a = "oi"; //Atributo "private" de Z.
    b = "oi"; //Atributo "package private", ou sem modificador de
              //acesso, de Z.
    c = "oi"; //Atributo "protected" de Z.
    d = "oi"; //Atributo "publico" de Z.
    e = "oi"; //Atributo "protected" da própria classe Y.
    z.c = "oi"; //Atributo "protected" de Z.
  }
}
```

```java
package inicio;

import teste.ClasseZ;

public class ClasseX extends ClasseZ {
  protected String f;
  ClasseZ z = new ClasseZ ();

  public ClasseX () {
    a = "oi"; //Atributo "private" de Z.
    b = "oi"; //Atributo "package private", ou sem modificador de
acesso, de Z.
    c = "oi"; //Atributo "protected" de Z.
    d = "oi"; //Atributo "publico" de Z.
    f = "oi"; //Atributo "protected" da própria classe Y.
    z.c = "oi"; //Atributo "protected" de Z.
  }
}
```

```java
package inicio;

import teste.*;

public class Principal {

      public static void main (String[] args) {
            ClasseX x = new ClasseX ();
            ClasseY y = new ClasseY ();
```

```
            ClasseZ z = new ClasseZ ();

            x.a = "oi";
            x.b = "oi";
            x.c = "oi";
            x.d = "oi";
            x.f = "oi";

            y.a = "oi";
            y.b = "oi";
            y.c = "oi";
            y.d = "oi";
            y.e = "oi";

            z.a = "oi";
            z.b = "oi";
            z.c = "oi";
            z.d = "oi";
        }

}
```

OBS: textos para leitura:

1) http://high5devs.com/2015/01/nome-dos-arquivos-e-nome-das-classes-na-linguagem-java/
2) https://www.caelum.com.br/apostila-java-orientacao-objetos/modificadores-de-acesso-e-atributos-de-classe/

Prática 08

Arquivo em linguagem e programação Java.

1) Analise o algoritmo abaixo escrito em linguagem de programação Java e execute-o no IDE Eclipse. Observe a saída do algoritmo.

```
package aluno;

public class Aluno {
  private long id;
  private String nome;
  private String matricula;
  private float nota;

  public Aluno () {
  }

  public Aluno (long id, String nome, String matricula, float nota) {
    this.id = id;
    this.nome = nome;
    this.matricula = matricula;
    this.nota = nota;
  }

  public long getId () {
    return id;
  }

  public void setId (long id) {
    this.id = id;
  }

  public String getNome () {
    return nome;
  }

  public void setNome (String nome) {
    this.nome = nome;
  }

  public String getMatricula () {
    return matricula;
  }

  public void setMatricula (String matricula) {
    this.matricula = matricula;
  }

  public float getNota () {
    return nota;
  }

  public void setNota (float nota) {
```

```
    this.nota = nota;
  }
}
```

```
package alunoDAO;

import aluno.Aluno;

public interface AlunoDAO {
  public void gravar (Aluno a);
  public Aluno getAluno (long id);
  public Aluno[] getAlunos (int tamanhoVetor);
}
```

```
package alunoDAO;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import aluno.Aluno;

public class AlunoDAOImpl implements AlunoDAO {
  public AlunoDAOImpl () {
  }

  @Override
  public void gravar (Aluno aluno) {
    File arq = null;
    FileOutputStream saida = null;
    OutputStreamWriter gravador = null;
    BufferedWriter buffer_saida = null;

    try {
      arq = new File ("aluno.txt");
      /* ------------------------------------------
       * FileOutputStream (File file, boolean append):
       * ------------------------------------------
       * 1) Se append for true, preserva o conteúdo do arquivo
       *    gravando os novos dados no final do arquivo.
       * 2) Se append for false, grava os novos dados sobre
       *    o conteúdo prévio, sobrescrevendo-o.
       * 3) Se o atributo append for omitido, funciona da mesma
       *    forma quando append tem valor igual a false.
       */
      saida = new FileOutputStream (arq, true);
      gravador = new OutputStreamWriter (saida);
      buffer_saida = new BufferedWriter (gravador);
```

```
      String separadorDeLinha = System.getProperty ("line.separator");

      buffer_saida.write (aluno.getId () + separadorDeLinha);
      buffer_saida.write (aluno.getNome () + separadorDeLinha);
      buffer_saida.write (aluno.getMatricula () + separadorDeLinha);
      buffer_saida.write (aluno.getNota () + separadorDeLinha);
      buffer_saida.flush();
    } catch (Exception e) {
      System.out.println ("ERRO ao gravar o aluno [" + aluno.getId() +
"] " + aluno.getNome() + "no disco rígido!");
      e.printStackTrace ();
    } finally {
      try {
        if (buffer_saida != null)
          buffer_saida.close ();
        if (gravador != null)
          gravador.close ();
        if (saida != null)
          saida.close ();
      } catch (Exception e) {
           System.out.println ("ERRO ao fechar os manipuladores de
escrita do arquivo aluno.txt");
        e.printStackTrace ();
      }
    }
  }

  @Override
  public Aluno getAluno (long id) {
    Aluno aluno = null;

    File arq = null;
    FileInputStream entrada = null;
    InputStreamReader leitor = null;
    BufferedReader buffer_entrada = null;

    try {
      arq = new File ("aluno.txt");
      entrada = new FileInputStream (arq);
      leitor = new InputStreamReader (entrada);
      buffer_entrada = new BufferedReader (leitor);

      String idSTR;
      while ((idSTR = buffer_entrada.readLine ()) != null) {
        Aluno a = new Aluno ();

        a.setId (Long.parseLong (idSTR));
        a.setNome (buffer_entrada.readLine ());
        a.setMatricula (buffer_entrada.readLine ());
        a.setNota (Float.parseFloat (buffer_entrada.readLine ()));

        if (id == a.getId()) {
          aluno = a;
          break;
```

```
        }
      }
    } catch (Exception e) {
      System.out.println ("ERRO ao ler o aluno [" + aluno.getId() + "]
" + aluno.getNome() + "do disco rígido!");
      e.printStackTrace ();
    } finally {
      try {
        if (buffer_entrada != null)
          buffer_entrada.close ();
        if (leitor != null)
          leitor.close ();
        if (entrada != null)
          entrada.close ();
      } catch (Exception e) {
           System.out.println ("ERRO ao fechar os manipuladores de
leitura do arquivo aluno.txt");
        e.printStackTrace ();
      }
    }

   return aluno;
  }

  @Override
  public Aluno[] getAlunos (int tamanhoVetor) {
    Aluno[] alunos = new Aluno[tamanhoVetor];

    File arq = null;
    FileInputStream entrada = null;
    InputStreamReader leitor = null;
    BufferedReader buffer_entrada = null;

    try {
      arq = new File ("aluno.txt");
      entrada = new FileInputStream (arq);
      leitor = new InputStreamReader (entrada);
      buffer_entrada = new BufferedReader (leitor);

      int i = 0;
      String idSTR;
      while (((idSTR = buffer_entrada.readLine ()) != null) && (i <
tamanhoVetor)) {
        alunos[i] = new Aluno ();

        alunos[i].setId (Long.parseLong (idSTR));
        alunos[i].setNome (buffer_entrada.readLine ());
        alunos[i].setMatricula (buffer_entrada.readLine ());
        alunos[i].setNota (Float.parseFloat (buffer_entrada.readLine
()));

        i++;
      }
    } catch (Exception e) {
```

```
      System.out.println ("ERRO ao ler os alunos do disco rígido!");
      e.printStackTrace ();
    } finally {
      try {
        if (buffer_entrada != null)
          buffer_entrada.close ();
        if (leitor != null)
          leitor.close ();
        if (entrada != null)
          entrada.close ();
      } catch (Exception e) {
        System.out.println ("ERRO ao fechar os manipuladores de
leitura do arquivo aluno.txt");
        e.printStackTrace ();
      }
    }

    return alunos;
  }
}

package inicio;

import java.util.Random;

import aluno.Aluno;
import alunoDAO.*;

public class Principal {

  public static void main (String[] args) {
    Principal p = new Principal ();
    p.gravarAlunos ();

    Random rand = new Random ();
    p.pesquisarAlunos (rand.nextInt (7));

    p.listarAlunos ();
  }

  public void gravarAlunos () {
    int i;
    Aluno aluno = new Aluno ();
    AlunoDAOImpl alunoDAO = new AlunoDAOImpl ();
    Random rand = new Random ();
    int n = rand.nextInt (10) + 1;

    System.out.println ("\n-------------------------");
    System.out.println ("   Cadastrando Alunos:");
    System.out.println ("-------------------------");
    for (i = 0; i < n; i++) {
      aluno.setId (i);
      aluno.setNome ("Nome" + i);
      aluno.setMatricula ("Matricula" + i);
```

```
      aluno.setNota (i);

      alunoDAO.gravar (aluno);

      System.out.println ("Aluno [" + aluno.getId () + "] " +
                          aluno.getNome () + " cadastrado.");
    }
  }


  public void pesquisarAlunos (long id) {
    AlunoDAOImpl alunoDAO = new AlunoDAOImpl ();

    Aluno aluno = alunoDAO.getAluno (id);

    if (aluno != null) {
      System.out.println ("\n\n--------------------------");
      System.out.println ("Dados do Aluno Pesquisado:");
      System.out.println ("--------------------------");
      System.out.println ("ID: " + aluno.getId ());
      System.out.println ("Nome: " + aluno.getNome ());
      System.out.println ("Matrcula: " + aluno.getMatricula ());
      System.out.println ("Nota: " + aluno.getNota ());
    } else {
      System.out.println ("\n\n--------------------------");
      System.out.println ("    Pesquisando Aluno");
      System.out.println ("--------------------------");
      System.out.println ("Aluno [" + id + "] Nome" + id + " não
                           encontrado!");
    }
  }


  public void listarAlunos () {
    int i;
    Aluno[] alunos;
    AlunoDAOImpl alunoDAO = new AlunoDAOImpl ();

    alunos = alunoDAO.getAlunos (30);
    System.out.println ("\n\n--------------------------");
    System.out.println ("   Alunos Cadastrados:");
    System.out.println ("--------------------------");
    for (i = 0; i < alunos.length; i++) {
      if (alunos[i] == null)
        break;

      System.out.println ("ID: " + alunos[i].getId ());
      System.out.println ("Nome: " + alunos[i].getNome ());
      System.out.println ("Matrcula: " + alunos[i].getMatricula ());
      System.out.println ("Nota: " + alunos[i].getNota ());
      System.out.println ();
    }
  }
}
```

2) Altere os algoritmos dos exercícios das práticas abaixo, para que os dados sejam gravados e lidos de arquivos.
    a. Prática 03, exercício 03.
    b. Prática 03, exercício 04.
    c. Prática 03, exercício 05.
    d. Prática 04, exercício 03.
    e. Prática 05, exercício 01.
    f. Prática 05, exercício 03.

Prática 09

Lista.

1) Em linguagem de programação Java, escreva um algoritmo que implemente o pacote para o tipo abstrato de dado lista, conforme apresentado no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

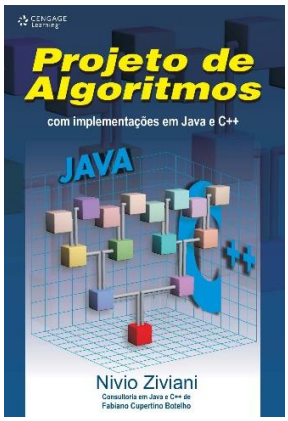


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo.

2) No pacote criado no exercício anterior, acrescente as operações:
    a. Concatenar listas.
    b. Particionar uma lista em duas.
    c. Fazer uma cópia, clone, de uma lista.
    d. Inserção ordenada de itens na lista.
    e. União de listas.
    f. Interseção de listas.
    g. Diferença entre listas.
    h. Intercalação de listas.
    i. Inserção em qualquer posição da lista, após o nodo cabeça.

    Na classe principal, acrescente as instruções para executar as novas operações implementadas.
3) Em linguagem de programação Java, crie um pacote com a implementação de lista sem nodo, célula, cabeça. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.
4) Em linguagem de programação Java, crie um pacote com a implementação de lista circular. Numa lista circular, o último nodo, célula, aponta para o primeiro. A figura 2 apresenta uma lista encadeada circular. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

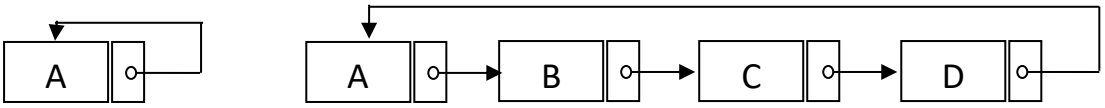


Figura 2: representação de lista encadeada circular.

5) Em linguagem de programação Java, crie um pacote com a implementação de lista duplamente encadeada. Numa lista duplamente encadeada, cada nodo, célula, aponta para seu sucessor e também para seu antecessor. A figura 3 apresenta uma lista duplamente encadeada. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

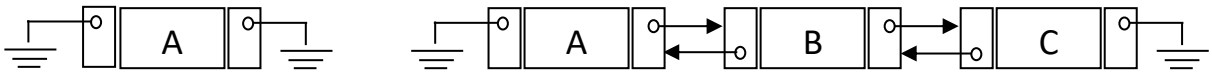


Figura 3: representação de lista duplamente encadeada.

Prática 10

Fila.

1) Em linguagem de programação Java, escreva um algoritmo que implemente o pacote para o tipo abstrato de dado fila, conforme apresentado no livro:
   - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

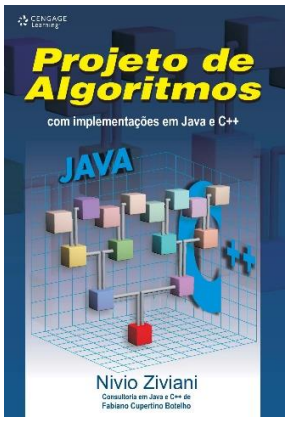


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações, métodos, implementadas.

2) Em linguagem de programação Java, crie um pacote com a implementação de fila sem nodo, célula, cabeça. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

3) Escreva um algoritmo que insira elementos em uma fila F1 e também em uma fila F2. O algoritmo deve trocar os elementos de F1 e F2, armazenando-os na ordem inversa daquela em que foram inseridos. Exemplo:

F1:

Q	T	Y	B	F

Inicio Fim

F2:

D	K	S	Z	P

Início Fim

INVERSÃO:

F1:

P	Z	S	K	D

Início Fim

F2:

F	B	Y	T	Q

Início Fim

Apresente o conteúdo das filas na tela antes e após a inversão. O conteúdo das filas deve ser armazenado em arquivo antes e após a inversão.

4) Implemente um algoritmo que simule a execução de uma jukebox. No algoritmo, as “playlists” são implementadas através de filas. O algoritmo deve implementar a classe Fila com as operações básicas enfileirar, desenfileirar, pesquisar e mostrar. Além disso, novas operações serão requisitadas como avançar, pular para uma música específica, carregar a “playlist” de novo e eliminar uma música da “playlist”. Cada “playlist” possui um título, nome. Os objetos em cada “playlist” são do tipo música. A classe Musica contém a posição da música na “playlist”, o nome da música e o cantor. Todos esses dados serão lidos a partir de arquivo (.txt) informado pelo usuário. O usuário tem total liberdade para criar novas “playlists” com músicas do seu gosto.

Prática 11

Pilha.

1) Em linguagem de programação Java, escreva um algoritmo que implemente o pacote para o tipo abstrato de dado pilha, conforme apresentado no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

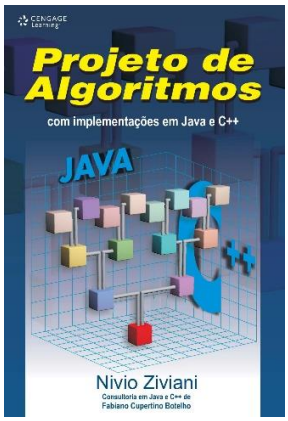


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações, métodos, implementadas.

2) Implemente a funcionalidade de histórico de um navegador Web. O navegador mantém o histórico de URL´s das páginas visitadas em uma pilha. O histórico é armazenado e lido de um arquivo. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

3) Implemente um método recursivo que retorne todos os inteiros pares que estão inseridos em uma pilha. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

4) Implemente um método recursivo que imprima o conteúdo de uma pilha na ordem do topo até a base da pilha. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

5) Dizemos que uma palavra ou frase é um palíndromo se pode ser igualmente lida da esquerda para a direita ou da direita para a esquerda, como por exemplo, ‘subi no onibus’. Codifique um método que use uma pilha para verificar se uma frase é um palíndromo. Dica: ignore espaços em branco. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

6) Escreva um algoritmo que cadastre vários números em uma pilha. Posteriormente, o programa deve gerar duas filas, a primeira com os números pares e a segunda com os números ímpares. A saída do programa deve apresentar a pilha digitada e as filas geradas. Caso alguma das filas seja vazia, o algoritmo deve mostrar uma mensagem. Escreva uma

classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

Prática 12

Lista em Vetores.

1) Em linguagem de programação Java, escreva um algoritmo que implemente o pacote para o tipo abstrato de dado lista usando vetores, conforme apresentado no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

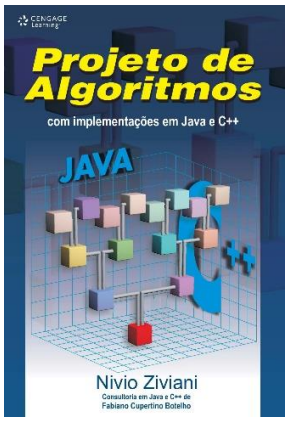


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações, métodos, implementadas.

Prática 13

Fila em Vetores.

1) Em linguagem de programação Java, escreva um algoritmo que implemente o pacote para o tipo abstrato de dado fila usando vetores, conforme apresentado no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

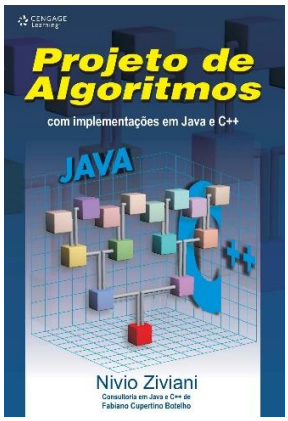


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações, métodos, implementadas.

Prática 14

Pilha em Vetores.

1) Em linguagem de programação Java, escreva um algoritmo que implemente o pacote para o tipo abstrato de dado pilha usando vetores, conforme apresentado no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

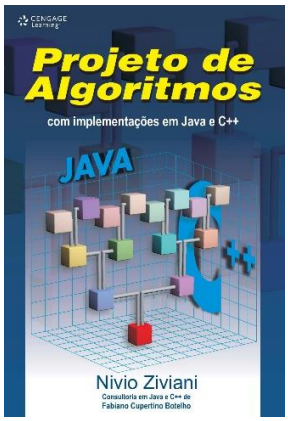


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações, métodos, implementadas.

Prática 15

Deque.

1) Um deque é uma especialização de lista em que as inserções e as remoções de elementos podem ocorrer em qualquer uma das extremidades da lista. Em linguagem de programação Java, crie um pacote com a implementação das operações para o tipo abstrato de dado deque com alocação dinâmica de memória. O pacote pode ser desenvolvido com base na implementação de lista apresentada no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

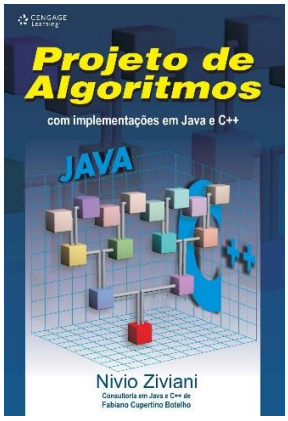


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo.

2) Em linguagem de programação Java, crie um pacote com a implementação de deque circular com alocação dinâmica de memória. Num deque circular, o último nodo, célula, aponta para o primeiro. A figura 2 apresenta uma representação de deque circular. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações, métodos, implementadas.

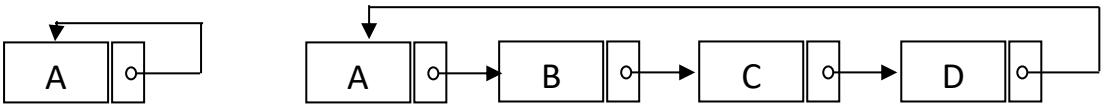


Figura 2: representação de deque circular.

Prática 16

Deque em Vetores.

1) Em linguagem de programação Java, crie um pacote com a implementação das operações para o tipo abstrato de dado deque com alocação estática de memória. O pacote pode ser desenvolvido com base na implementação de lista com alocação estática de memória apresentada no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

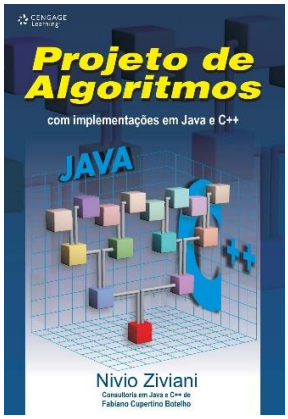


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

2) Em linguagem de programação Java, crie um pacote com a implementação de deque circular com alocação estática de memória. Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

Prática 17

Árvore Binária de Pesquisa.

1) Em linguagem de programação Java, crie um pacote com a implementação das operações para o tipo abstrato de dado árvore binária de pesquisa. O pacote pode ser desenvolvido com base na implementação de árvore binária de pesquisa apresentada no livro:
    - Ziviani, Nivio. Projeto de Algoritmos com Implementação em Java e C++. Thomson, 2007. ISBN-10: 8522108218.

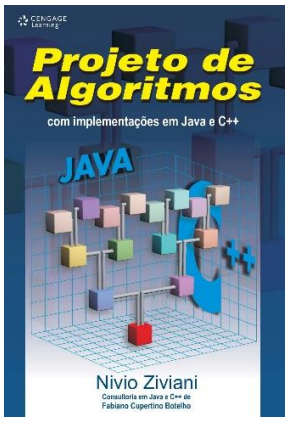


Figura 1: livro Projeto de Algoritmos.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas. O pacote deve incluir as operações:

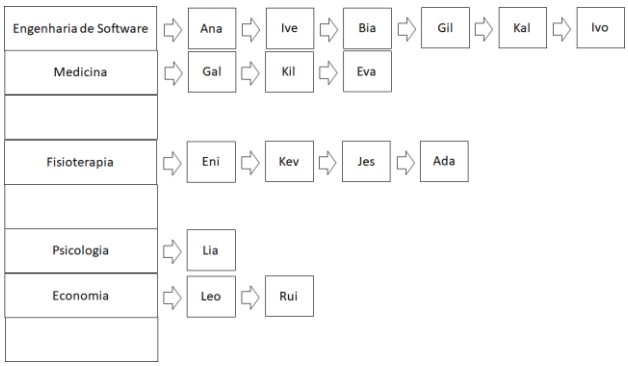
a. Iniciar a árvore como vazia.
b. Inserir um nó na árvore.
c. Verificar se a árvore está vazia.
d. Remover um nó da árvore.
e. Pesquisar a ocorrência de um valor em algum nó da árvore.
f. Informar a altura da árvore.
g. Informar a altura de um nó.
h. Informar a profundidade da árvore.
i. Informar a profundidade de um nó.
j. Informar o nível de um nó.
k. Informar o grau de um nó.
l. Informar o diâmetro da árvore.
m. Percorrer a árvore em pré-ordem.
n. Percorrer a árvore em ordem (ordem simétrica).
o. Percorrer a árvore em pós-ordem.
p. Percorrer a árvore em profundidade.
q. Percorrer a árvore em largura, nível.
r. Salvar o conteúdo da árvore em um arquivo.
s. Ler o conteúdo da árvore a partir de um arquivo.

Prática 18

Tabela Hash.

1) Em linguagem de programação Java, desenvolva um pacote com a implementação das operações para o tipo abstrato de dado Tabela Hash.

2) Com base no pacote desenvolvido para Tabela Hash, crie um algoritmo para cadastro de trabalhadores para uma agência de emprego. Os dados referentes a cada trabalhador são: nome, CPF, endereço, telefone e profissão. O algoritmo deve ler os dados dos trabalhadores a partir de um arquivo. Os dados devem ser armazenados numa Tabela Hash. Cada entrada da Tabela Hash armazena uma profissão e possui uma referência para uma lista encadeada que armazena os dados dos trabalhadores. O algoritmo deve:
   a. Possuir um método para apresentar o número de colisões ocorridas ao cadastrar os trabalhadores na Tabela Hash.
   b. Possuir um método para apresentar o tamanho de cada lista encadeada da Tabela Hash.
   c. Possuir um método para apresentar o tamanho médio das listas encadeadas da Tabela Hash.
   d. Possuir um método que receba uma profissão como parâmetro e que apresente todos os trabalhadores relativos a essa profissão.
   e. Possuir um método que receba o CPF de um trabalhador como parâmetro e que apresente os dados desse trabalhador. Se o CPF não for encontrado na Tabela Hash, o método deve informar que o trabalhador não foi cadastrado. O método deve apresentar o número de comparações realizadas para encontrar o trabalhador na Tabela Hash.
   f. Possuir um método que permita a inclusão de novos trabalhadores no cadastro.
   g. Possuir um método para remover trabalhadores do cadastro.
   h. Possuir um método para atualizar dados dos trabalhadores no cadastro.
   i. Possuir um método que grave o cadastro em arquivo.



   Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

3) Com base no pacote desenvolvido para Tabela Hash, crie um dicionário de sinônimos. O algoritmo deve ler cada termo e seus respectivos sinônimos a partir de um arquivo e armazená-los numa Tabela Hash. Cada entrada da Tabela Hash deve armazenar um termo e apontar para os respectivos sinônimos. O algoritmo deve:

a. Possuir um método que permita a inclusão de novos termos e sinônimos no dicionário.
b. Possuir um método que permita a inclusão de novos sinônimos no dicionário.
c. Possuir um método para remover termos do dicionário.
d. Possuir um método para remover sinônimos do dicionário.
e. Possuir um método para atualizar termos no dicionário.
f. Possuir um método para atualizar sinônimos no dicionário.
g. Possuir um método para pesquisar termos no dicionário.
h. Possuir um método para pesquisar sinônimos no dicionário.
i. Possuir um método que grave o dicionário em arquivo.

Escreva uma classe principal para executar o algoritmo e que exemplifique a funcionalidade das operações implementadas.

Prática 19

Enumeração.

Uma enumeração é um conjunto de constantes, uma lista de valores pré-definidos. Na linguagem de programação Java, um tipo enumeração é definido através da palavra-chave *enum*. O tipo *enum* é um tipo de dado especial que permite que uma variável seja um conjunto de constantes predefinidas. O tipo *enum* implicitamente estende a classe java.lang.Enum. Java não suporta herança múltipla. Logo, uma enum não pode estender outra classe.

OBS:

1) As instâncias do tipo enum são criadas e nomeadas com a declaração da classe, sendo fixas e imutáveis (o valor é fixo).
2) Não é permitido criar novas instâncias com a palavra-chave *new*.
3) O construtor é declarado *private*, embora não precise de modificador *private* explícito.
4) Os nomes declarados na enumeração são escritos com letras maiúsculas, pois, por convenção, objetos constantes e imutáveis (static final) são escritos com letras maiúsculas.
5) As instâncias do tipo *enum* devem obrigatoriamente ter apenas um nome.
6) Opcionalmente, a declaração da classe pode incluir variáveis de instância, construtor, métodos de instância, de classe, etc.

Exemplos:

1) Dias da semana.

Execute o programa abaixo e comente a saída.

```
package enumeracao;

public enum DiasDaSemana {
  DOMINGO, SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO
}
```

```
package inicio;

import enumeracao.DiasDaSemana;

public class Principal {
  public static void main (String[] args) {
    System.out.println ("----------- Dias -----------");
    for (DiasDaSemana d: DiasDaSemana.values ())
      System.out.println ("Nome: " + d.name () +
                          ", Ordem: " + d.ordinal());
    }
}
```

2) Cores.

```
package enumeracao;

public enum Cor {
  VERMELHO (10), AMARELO (11), VERDE (13), AZUL (12);

  private int valor;
  Cor (int valor) {
      this.valor = valor;
  }
  public void setValor (int valor) {
      this.valor = valor;
  }
  public int getValor () {
      return this.valor;
  }
}
```

```
package inicio;

import enumeracao.Cor;
import enumeracao.DiaDaSemana;

public class Principal {

  public static void main (String[] args) {
    System.out.println ("----------- Dias -----------");
    for (DiaDaSemana d: DiaDaSemana.values ())
      System.out.println ("Nome: " + d.name () +
                          ", Ordem: " + d.ordinal());

    System.out.println ("\n----------- Cores -----------");
    for (Cor cor: Cor.values ())
      System.out.println ("Nome: " + cor.name () +
                          ", Ordem: " + cor.ordinal() +
                          ", Valor: " + cor.getValor());
    Cor.AMARELO.setValor(9);
    System.out.println ("Nome: " + Cor.AMARELO.name () +
                        ", Ordem: " + Cor.AMARELO.ordinal() +
                        ", Valor: " + Cor.AMARELO.getValor());

  }
}
```

3) Carros.

```
package enumeracao;

public enum Carro {
  FUSCA (1975, Cor.VERMELHO), GOL (1987, Cor.AMARELO),
  UNO (1992, Cor.VERDE), PALIO (2001, Cor.AZUL);

  private int ano;
  private Cor cor;
```

```
      Carro (int ano, Cor cor) {
            this.ano = ano;
            this.cor = cor;
      }
      public void setAno (int ano) {
            this.ano = ano;
      }
      public int getAno () {
            return this.ano;
      }
      public Cor getCor () {
            return cor;
      }
      public void setCor (Cor cor) {
            this.cor = cor;
      }
}


package inicio;

import enumeracao.Cor;
import enumeracao.DiaDaSemana;
import enumeracao.Carro;

public class Principal {
    public static void main (String[] args) {
    System.out.println ("----------- Dias -----------");
    for (DiaDaSemana d: DiaDaSemana.values ())
      System.out.println ("Nome: " + d.name () +
                          ", Ordem: " + d.ordinal());

    System.out.println ("\n----------- Cores -----------");
    for (Cor cor: Cor.values ())
      System.out.println ("Nome: " + cor.name () +
                          ", Ordem: " + cor.ordinal() +
                          ", Valor: " + cor.getValor());
    Cor.AMARELO.setValor(9);
      System.out.println ("Nome: " + Cor.AMARELO.name () +
                          ", Ordem: " + Cor.AMARELO.ordinal () +
                          ", Valor: " + Cor.AMARELO.getValor ());

    System.out.println ("\n----------- Carros -----------");
    for (Carro carro: Carro.values ())
      System.out.println ("Nome: " + carro.name () +
                          ", Ordem: " + carro.ordinal () +
                          ", Ano: " + carro.getAno () +
                          ", Cor: " + carro.getCor ());

    Carro.FUSCA.setAno (2016);
    Carro.FUSCA.setCor (Cor.AMARELO);
    System.out.println ("Nome: " + Carro.FUSCA.name () +
                        ", Ordem: " + Carro.FUSCA.ordinal () +
                        ", Ano: " + Carro.FUSCA.getAno () +
```

```
                        ", Cor: " + Carro.FUSCA.getCor ());

        Carro c = Carro.UNO;
        System.out.println ("Carro: " + c.name());
    }
}
```

4) Tempo de Execução de um programa ou instrução.

```
package tempo;

public class Tempo {
  private enum EnumTempo {
    MILISSEGUNDO (1), SEGUNDO (1000, MILISSEGUNDO),
    MINUTO (60, SEGUNDO), HORA (60, MINUTO);

    private int valor;
    EnumTempo (int valor) {
      this.valor = valor;
    }
    EnumTempo (int valor, EnumTempo et) {
      this.valor = valor * et.getValor ();
    }
    public void setValor (int valor) {
      this.valor = valor;
    }
    public int getValor () {
      return this.valor;
    }
  }

  private long anterior;
  private long corrente;

  public Tempo () {
      this.anterior = 0;
      this.corrente = System.currentTimeMillis ();
  }
  public Tempo (long corrente) {
      this.anterior = 0;
      this.corrente = corrente;
  }
  public void setTempoCorrente () {
      this.anterior = this.corrente;
      this.corrente = System.currentTimeMillis ();
  }
  public void setTempoCorrente (long corrente) {
      this.anterior = this.corrente;
      this.corrente = corrente;
  }
  public long getTempoAnterior () {
      return this.anterior;
  }
```

```
  public long getTempoCorrente () {
      return this.corrente;
  }
  public long getTempoDecorrido () {
    return this.corrente - this.anterior;
  }
  public String toString () {
      String tempo = "";
      long decorrido = this.getTempoDecorrido();

      long hora = decorrido / EnumTempo.HORA.getValor ();
      decorrido %= EnumTempo.HORA.getValor ();
      if (hora > 0)
            tempo += hora + " hora(s) ";

      long minuto = decorrido / EnumTempo.MINUTO.getValor ();
      decorrido %= EnumTempo.MINUTO.getValor ();
      if (minuto > 0)
            tempo += minuto + " minuto(s) ";

      long segundo = decorrido / EnumTempo.SEGUNDO.getValor ();
      decorrido %= EnumTempo.SEGUNDO.getValor ();
      if (segundo > 0)
            tempo += segundo + " segundo(s) ";

      long milissegundo = decorrido / EnumTempo.MILISSEGUNDO.getValor
();
      if (milissegundo > 0)
            tempo += milissegundo + " milissegundo(s) ";

      return tempo;
  }
}


package inicio;
import tempo.Tempo;
public class Principal {
  public static void main (String[] args) throws Exception {
    int i;
    Tempo t = new Tempo ();
    for (i = 0; i < 1000000; i++)
      System.out.println ("i = " + i);
    t.setTempoCorrente();
    System.out.println ("Tempo de execução da estrutura de repetição
FOR: " + t.toString ());

    t.setTempoCorrente (0);
    t.setTempoCorrente (35678321);
    System.out.println ("Tempo decorrido: " + t.toString ());

  }

}
```

Prática 20

Ordenação.

1) Escreva um pacote em linguagem Java para ordenação de vetores. O pacote deve conter o algoritmo de ordenação pelo método Bolha, Seleção, Inserção, Quicksort e Heapsort. Em cada algoritmo, contabilize o número de trocas, o número de comparações e o tempo de execução. Teste os algoritmos de ordenação implementados no pacote, usando vetores de tamanho (N):
    a. 10.
    b. 100.
    c. 500.
    d. 1.000
    e. 2.500
    f. 5.000.
    g. 7.500.
    h. 10.000.
    i. 25.000.
    j. 50.000.
    k. 100.000.

    Os elementos dos vetores devem ser gerados aleatoriamente. No Excel, gere 3 gráficos:

    a. Tamanho do vetor x Número de Comparações (gráfico de linha).
    b. Tamanho do vetor x Número de Trocas (gráfico de linha).
    c. Tamanho do vetor x Tempo de Execução (histograma).

    Cada gráfico deve apresentar o desempenho dos algoritmos de ordenação para os vetores de tamanhos N especificados. Apresente também a ordem de complexidade de cada algoritmo implementado (notação O).

Exemplo:

```
package item;
public class Item {
  public int chave;

  public Item (int chave) {this.chave = chave;}

  public int getChave () {return chave;}

  public void setChave (int chave) {
    this.chave = chave;
  }
}
```

```
package ordena;
import item.Item;
import java.util.Random;
public abstract class Ordena {
  Item[] v;

  public Item[] getVetor () {return v;}

  public void setVetor (int n) {
    int i;
    Random rand = new Random ();
    this.v = new Item[n];
    for (i = 0; i < this.v.length; i++)
      this.v[i] = new Item (rand.nextInt (100));
  }

  public void imprime () {
    int i, n = this.v.length;
    for (i = 0; i < n; i++)
      System.out.print (this.v[i].chave + " ");
    System.out.println ();
  }
}
```

```
package ordena;
import item.Item;
public class Bolha extends Ordena {
  public Bolha () {}

  public void bolha () {
    int i, j, n = this.v.length;
    Item aux;

    for (i = 0; i < n - 1; i++)
      for (j = 1; j < n - i; j++)
        if (this.v[j].chave < this.v[j - 1].chave) {
          aux = this.v[j];
          this.v[j] = this.v[j - 1];
          this.v[j - 1] = aux;
        }
  }
public void selecao () {
}
  int i, j, menor, n = this.v.length;
    Item aux;

    for (i = 0; i < n - 1; i++) {
        menor = i;
        for (j = i + 1; j < n; j++) {
            if (this.v[j].chave < this.v[menor].chave) {
                menor = j;
            }
        }
        aux = this.v[i];
        this.v[i] = this.v[menor];
        this.v[menor] = aux;
    }
}
public void insercao () {
    int i, j, n = this.v.length;
    Item aux;
    for (i = 1; i < n; i++) {
        aux = this.v[i];
        for (j = i - 1; (j >= 0) && (aux.chave < this.v[j].chave); j--) {
            this.v[j + 1] = this.v[j];
        }
        this.v[j + 1] = aux;
    }
}
```

```
package principal;
import ordena.Bolha;
import item.Item;
import java.util.Random;

public class Principal {
  public static void main (String[] args) {
    Principal o = new Principal ();
    o.ordenaBolha (5);
  }
  public void ordenaBolha (int n) {
    Bolha b = new Bolha ();
    b.setVetor (n);
    System.out.println ("Método da Bolha");
    System.out.println ("Vetor desordenado: ");
    b.imprime ();
    b.bolha ();
    System.out.println ("Vetor ordenado: ");
    b.imprime ();
  }
}
```

```
package ordena;

import item.Item;

public class Quicksort  extends Ordena {
  public int i, j;

  public Quicksort () {
  }
  public Quicksort (Item[] vetor) {
    this.v = vetor;
  }
  public void quicksort () {
    this.ordena (0, this.v.length - 1);
  }
  public void ordena (int e, int d) {
    this.particao (e, d);
    if (e < this.j)
      this.ordena (e, this.j);
    if (this.i < d)
      this.ordena (this.i, d);
  }
  public void particao (int e, int d) {
    Item x, aux;
    this.i = e;
    this.j = d;
    x = this.v[(this.i + this.j) / 2];
    do {
      while (this.v[this.i].chave < x.chave)
        this.i++;
      while (this.v[this.j].chave > x.chave)
        this.j--;
      if (this.i <= this.j) {
        aux = this.v[this.i];
        this.v[this.i] = this.v[this.j];
        this.v[this.j] = aux;

        this.i++;
        this.j--;
      }
    } while (this.i <= this.j);
  }
}
```

```
heapsort (A) {
    build-max-heap (A)
    for (i = A.comprimento; i > 0; i--)
        troca (A[0],  A[i])
        A.tamanho-do-heap  =  A.tamanho-do-heap  -  1
        max-heapify (A,  0)
}
```

```
build-max-heap (A) {
    A.tamanho-do-heap = A.comprimento
    for (i = piso (A.tamanho-do-heap / 2); i >= 0; i--)
        max-heapify (A, i)
}
```

```
max-heapify (A, i) {
    e = esquerda (i)
    d = direita (i)
    if ((e < A.tamanho-do-heap) e (A[e] > A[i])) maior = e else maior = i
    if ((d < A.tamanho-do-heap) e (A[d] > A[maior])) maior = d
    if (maior != i) {
        troca (A[i], A[maior])
        max-heapify (A, maior)
    }
}
```

Prática 21

Busca.

2) Escreva um pacote em linguagem Java para busca em vetores. O pacote deve conter o algoritmo de busca sequencial, busca sequencial com sentinela e busca binária. Em cada algoritmo, contabilize o número de comparações e o tempo de execução. Teste os algoritmos de busca implementados no pacote, usando vetores de tamanho (N):
    l. 10.
    m. 100.
    n. 500.
    o. 1.000
    p. 2.500
    q. 5.000.
    r. 7.500.
    s. 10.000.
    t. 25.000.
    u. 50.000.
    v. 100.000.

    Os elementos dos vetores devem ser gerados aleatoriamente. No Excel, gere 2 gráficos:

    d. Tamanho do vetor x Número de Comparações (gráfico de linha).
    e. Tamanho do vetor x Tempo de Execução (histograma).

    Cada gráfico deve apresentar o desempenho dos algoritmos de busca para os vetores de tamanhos N especificados. Apresente também a ordem de complexidade de cada algoritmo implementado (notação O).

    Exemplo:

Algoritmo de busca sequencial:

```
int search (int a[], int v, int l, int r) {
   for (int i = l; i <= r; i++)
       if (v == a[i])
           return i;
   return -1;
}
```

OBS:
a: vetor.
v: valor procurado.
l: left (esquerda).
r: right (direira).

Algoritmo de busca sequencial com sentinela:

```
int search (int a[], int v, int l, int r) {
   int i, n = r + 1;
   a[n] = v;
   for (i = l; v != a[i]; i++);
   if (i < n)
      return (i);  /*Chave encontrada!*/
   else
      return (-1); /*Sentinela encontrada.*/
}
```

OBS:
a: vetor.
v: valor procurado.
l: left (esquerda).
r: right (direira).

Algoritmo de busca binária:

```
int search (int a[], int v, int l, int r) {
    while (l <= r) {
        int m = (l + r) / 2;
        if (v == a[m])
            return m;
        if (v < a[m])
            r = m - 1;
        else
            l = m + 1;
    }
    return -1;
}
```

OBS:
a: vetor.
v: valor procurado.
l: left (esquerda).
r: right (direira).