**FLÁVIA PARREIRA DE SAMPAIO**

**MATRÍCULA: 202208817335**

**CAMPUS: WEST SHOPPING, CAMPO GRANDE –**

**ESTÁCIO DE SÁ**

**PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA**

AULA 6   
ATIVIDADE VERIFICADORA DE APRENDIZAGEM - escrever uma classe em Java chamada ContaBancária para representar uma conta bancária comum

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.Texto

O conteúdo gerado por IA pode estar incorreto.

CÓDIGO:   
import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.text.NumberFormat;

import java.util.Locale;

class ContaBancaria {

private double saldo;

private final LocalDate dataAbertura;

public ContaBancaria() {

this.saldo = 0.00;

this.dataAbertura = LocalDate.now();

}

public double getSaldo() {

return saldo;

}

public void setSaldo(double saldo) {

this.saldo = saldo;

}

public LocalDate getDataAbertura() {

return dataAbertura;

}

public String getDataAberturaFormatada() {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

return dataAbertura.format(formatter);

}

public String getSaldoFormatado() {

NumberFormat nf = NumberFormat.getCurrencyInstance(new Locale("pt", "BR"));

return nf.format(saldo);

}

public void depositar(double valor) {

if (valor > 0) {

this.saldo += valor;

}

}

public boolean sacar(double valor) {

if (valor > 0 && valor <= saldo) {

this.saldo -= valor;

return true;

} else {

return false;

}

}

}

public class Main {

public static void main(String[] args) {

ContaBancaria conta = new ContaBancaria();

System.out.println("Data de Abertura: " + conta.getDataAberturaFormatada());

System.out.println("Saldo inicial: " + conta.getSaldoFormatado());

conta.depositar(1000.00);

System.out.println("Após depósito de R$1000,00: " + conta.getSaldoFormatado());

if (conta.sacar(400.00)) {

System.out.println("Após saque de R$400,00: " + conta.getSaldoFormatado());

} else {

System.out.println("Saque de R$400,00 não permitido.");

}

if (conta.sacar(700.00)) {

System.out.println("Após saque de R$700,00: " + conta.getSaldoFormatado());

} else {

System.out.println("Saque de R$700,00 não permitido (saldo insuficiente).");

}

System.out.println("Saldo final: " + conta.getSaldoFormatado());

}

}

ATIVIDADE AUTÔNOMA AURA

1) (COSULPLAN-TRF, 2017) Na orientação a objetos o encapsulamento é um mecanismo utilizado para restringir o acesso ao comportamento interno de um objeto. O objeto requisitante envia uma mensagem e não precisa conhecer a forma pela qual a operação requisitada é realizada, tudo o que importa ao objeto remetente é a realização da operação. Na linguagem Java, os modificadores de acesso que aplicam o princípio de encapsulamento, definindo a visibilidade de um atributo ou método dentro de uma classe, definem que:

I. Apenas membros da classe possuem acesso.

II. Apenas membros da classe e subclasse possuem acesso.

III. Todos possuem acesso.

IV. Somente classe do mesmo pacote possuem acesso.

A respeito das definições anteriores, assinale a

alternativa correta:  
a) I. Protected II. Private III. Public IV. Default.

b) I. Protected II.Private III.Default IV.Public.

c) I. Private II.Default III.Public IV.Protected.

**d) I. Private II. Protected III. Public IV. Default. << RESPOSTA**

e) I. Private II. Public III. Public IV. Default  
  
  
2) (FUNCAB-MDA, 2014) Em Java, para determinar o nível de acesso dos elementos de uma classe, são utilizados qualificadores de acesso. Um nível ALFA é considerado o mais rígido, em que apenas os métodos da própria classe tem acesso a variáveis e outros métodos, enquanto o nível BETA e intermediário de encapsulamento em que as variáveis e métodos podem ser acessados pela própria classe ou subclasses. Os níveis ALFA e BETA caracterizam, respectivamente, os seguintes qualificadores de acesso:  
a) protected e public.

**b) private e protected. . << RESPOSTA**

c) priority e private.  
d) public e package.

e) package e priority.

AULA 7

Crie uma estrutura hierárquica que contenha as seguintes classes:

* **Veiculo** (classe abstrata)  
  Os métodos da classe Veiculo são todos abstratos e possuem a seguinte assinatura:

-public float acelerar(float velocidade);

-public void parar();

* Estes métodos devem ser implementados nas subclasses **Automóvel** e **Bicicleta** (exibir mensagens sobre a operação).
* Acrescentar na classe **Automóvel** o método informando sobre a troca de óleo:

-public void mudarOleo(float litros);

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

CÓDIGO:  
abstract class Veiculo {

public abstract float acelerar(float velocidade);

public abstract void parar();

}

class Bicicleta extends Veiculo {

@Override

public float acelerar(float velocidade) {

System.out.println("Pedalando a bicicleta até atingir " + velocidade + " km/h.");

return velocidade;

}

@Override

public void parar() {

System.out.println("Freando a bicicleta até parar completamente.");

}

}

class Automovel extends Veiculo {

@Override

public float acelerar(float velocidade) {

System.out.println("Acelerando o automóvel até " + velocidade + " km/h.");

return velocidade;

}

@Override

public void parar() {

System.out.println("Freando o automóvel até parar completamente.");

}

public void mudarOleo(float litros) {

System.out.println("Trocando " + litros + " litros de óleo do automóvel.");

}

}

public class Main {

public static void main(String[] args) {

Veiculo bicicleta = new Bicicleta();

Veiculo automovel = new Automovel();

System.out.println("=== Bicicleta ===");

bicicleta.acelerar(15);

bicicleta.parar();

System.out.println("\n=== Automóvel ===");

automovel.acelerar(80);

automovel.parar();

if (automovel instanceof Automovel) {

((Automovel) automovel).mudarOleo(4.5f);

}

}

}

ATIVIDADE AUTÔNOMA AURA

1. (FAURGS-HCPA, 2016) No que se refere a classes abstratas e polimorfismo, assinale com V (verdadeiro) ou F (falso) as afirmações abaixo:

( V) Classes abstratas não podem ser instanciadas.

( F) Todos os métodos de uma classe abstrata devem estar sobrescritos na classe concreta que a estende.

(F ) Uma classe concreta corresponde a uma generalização de uma classe abstrata.

(V ) Diferentes classes concretas de uma mesma classe abstrata podem sobrescrever o mesmo método.

A sequência correta de preenchimento dos parênteses, de cima para baixo, é:

**a) V - F - F - V. <<<< RESPOSTA**

b) F - V - V - F.

c) V - V - F - F.

d) F - F - V - V.

e) V - V - F - V.

2) (FCC-SANASA Campinas, 2019) Considere que um Analista de TI sabe que uma classe Pessoa Física e uma classe Pessoa Jurídica possuem o atributo nome como uma informação em comum e que o CPF é um atributo específico para a Pessoa Física e o CNPJ é um atributo específico para Pessoa Jurídica. Então o Analista criou uma outra classe com o atributo nome e seu objetivo é que haja herança deste e, eventualmente, outros métodos e atributos, para as classes filhas, Pessoa Física e Pessoa Jurídica, que já existiam. Essa classe criada não é instanciada, apenas fornece um modelo para geração de outras classes, e é denominada:

a) Subclasse

b) Classe construtora

**c) Classe abstrata <<< RESPOSTA**

d) Classe sobescrita

e) Pacote

AULA 8

ATIVIDADE VERIFICADORA DE APRENDIZAGEM - Uma empresa de produtos alimentícios contrata uma equipe de desenvolvimento para reconstruir seu sistema de cadastros de produtos. Uma das exigências é que o sistema seja padronizado para que sua manutenção e desenvolvimento possa ser realizado de forma adequada. Dessa forma, todos os cadastros devem seguir um contrato, no caso, devem implementar uma interface. Essa interface deve ser chamada de CadastroDAO e deve possuir os seguintes métodos:

-salvar (retorno void e argumento Object);

- atualizar (retorno void e argumento Object);

-deletar (retorno void e argumento inteiro);

- getObjetoPorId (retorno Object e argumento inteiro);

-listarObjetos (retorno List de Objects e sem argumentos);  
  
Interface gráfica do usuário, Aplicativo, Word

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Aplicativo, Teams

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

CÓDIGO:

import java.util.\*;

// Interface

interface CadastroDAO {

void salvar(Object obj);

void atualizar(Object obj);

void deletar(int id);

Object getObjetoPorId(int id);

List<Object> listarObjetos();

}

// Classe Produto

class Produto {

private int id;

private String nome;

public Produto(int id, String nome) {

this.id = id;

this.nome = nome;

}

public int getId() {

return id;

}

public String getNome() {

return nome;

}

public void setNome(String nome) {

this.nome = nome;

}

@Override

public String toString() {

return "Produto{id=" + id + ", nome='" + nome + "'}";

}

}

// Implementação da interface

class ProdutoDAO implements CadastroDAO {

private List<Produto> produtos = new ArrayList<>();

@Override

public void salvar(Object obj) {

produtos.add((Produto) obj);

}

@Override

public void atualizar(Object obj) {

Produto novoProduto = (Produto) obj;

for (int i = 0; i < produtos.size(); i++) {

if (produtos.get(i).getId() == novoProduto.getId()) {

produtos.set(i, novoProduto);

return;

}

}

}

@Override

public void deletar(int id) {

produtos.removeIf(p -> p.getId() == id);

}

@Override

public Object getObjetoPorId(int id) {

for (Produto p : produtos) {

if (p.getId() == id) {

return p;

}

}

return null;

}

@Override

public List<Object> listarObjetos() {

return new ArrayList<>(produtos);

}

}

// Classe principal

public class Main {

public static void main(String[] args) {

ProdutoDAO dao = new ProdutoDAO();

Produto p1 = new Produto(1, "Arroz");

Produto p2 = new Produto(2, "Feijão");

dao.salvar(p1);

dao.salvar(p2);

System.out.println("Todos os produtos:");

for (Object obj : dao.listarObjetos()) {

System.out.println(obj);

}

System.out.println("\nProduto com ID 1: " + dao.getObjetoPorId(1));

p1.setNome("Arroz Integral");

dao.atualizar(p1);

System.out.println("\nApós atualização:");

System.out.println(dao.getObjetoPorId(1));

dao.deletar(2);

System.out.println("\nApós exclusão:");

for (Object obj : dao.listarObjetos()) {

System.out.println(obj);

}

}

}

ATIVIDADE AUTÔNOMA AURA

1) Um programador está desenvolvendo um sistema na linguagem de programação Java no qual uma das classes pode aproveitar métodos de outras duas classes diferentes, sendo que uma delas, é uma interface. No entanto, esse programador sabe que a linguagem Java não possui suporte a herança múltipla. Ele então pensou uma solução distinta que poderia ajudá-lo a implementar sua ideia. Qual seria essa solução?

a) O programador, mesmo sabendo que a linguagem Java não implementa a herança múltipla, poderia forçar tal comportamento por conta e risco, nesse caso, o interpretador apenas emitiria um alerta.

b) O programador poderia herdar os métodos da interface e implementar os métodos da classe.

**c) O programador poderia herdar métodos da classe e implementar os métodos da interface. >>>> resposta**

d) O programador poderia sobrescrever os métodos da classe e da interface.

e) Não existe nenhuma solução para esse problema.

1. O conceito e orientação a objetos permite a herança múltipla que é quando uma classe herda propriedades e comportamentos de mais de uma superclasse. Diferentes linguagens de programação orientadas a objetos variam quanto ao tratamento de herança múltipla, algumas linguagens de programação permitem herança de múltiplas superclasses, outras não. Java não permite herança múltipla de classes, mas fornece uma outra construção que utiliza interfaces. Assinale a alternativa que apresenta um trecho de código Java correto para a declaração de uma classe Java que simula a herança múltipla.

a) public class Ornitorrinco extends Mamifero, Ave { /\* corpo da classe \*/ } b) public class Ornitorrinco inherits Mamifero implements Ave, Animal {/\* corpo da classe \*/ }

c) public class Ornitorrinco extends Mamifero, Ave implements Animal { /\* corpo da classe \*/ }

d) public class Ornitorrinco inherits Mamifero, Ave, Animal {/\*\* corpo da classe \*/ }

**e) public class Ornitorrinco extends Mamifero implements Ave, Animal { /\*\* corpo da classe \*/ } >>> resposta**