

Classes e Objetos

Exercícios: Resolução 2

Vinicius Takeo Friedrich Kuwaki

Universidade do Estado de Santa Catarina

Exercício 1

Resolução Exercício 1

Exercício 2

Resolução Exercício 2

1. Uma pessoa possui nome, idade (em anos), altura (em metros) e massa (em kilogramas). Implemente em Java duas classes, uma que representa a Pessoa, com seus devidos atributos e outra que representa um Grupo de pessoas. A classe Grupo possui um número fixo de pessoas (fica a seu cargo escolher). Além do mais, a classe Pessoa deve possuir um método para calcular o IMC (índice de massa corporal), que é calculado pela seguinte fórmula:

$$IMC = \frac{massa}{altura^2} \quad (1)$$

A classe Grupo também deve possuir um método que exibe as pessoas em ordem decrescente de IMC.

Exercício 1

Resolução Exercício 1

Exercício 2

Resolução Exercício 2

Resolução Exercício 1

- Primeiro vamos definir a classe Pessoa;
- Esta terá quatro atributos como descrito no enunciado:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    private float altura;  
    private float massa;
```

- Utilizaremos um construtor vazio:

```
public Pessoa() {  
  
}
```

- Todos os atributos devem possuir seus métodos getters e setters (boas práticas de programação orientada a objetos):

Resolução Exercício 1

```
public String getNome() {  
    return this.nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}  
  
public int getIdade() {  
    return this.idade;  
}  
  
public void setIdade(int idade) {  
    this.idade = idade;  
}  
  
public float getAltura() {  
    return this.altura;  
}  
  
public void setAltura(float altura) {  
    this.altura = altura;  
}
```

Resolução Exercício 1

```
public float getMassa() {  
    return this.massa;  
}  
  
public void setMassa(float massa) {  
    this.massa = massa;  
}
```

- E também é necessário implementar o método para calcular o IMC, no qual retorna um resultado de ponto flutuante, dado pela divisão entre a massa e o quadrado da altura da pessoa:

```
public float calculaImc() {  
    return this.massa / (this.altura * this.altura);  
}
```

- Agora iremos definir a classe Grupo;
- Esta irá possuir um número fixo de pessoas;
- Eu escolhi o número 5;

Resolução Exercício 1

- Logo, teremos um atributo que será um vetor de objetos do tipo Pessoa. De tamanho 5.
- Haverá também um atributo do tipo inteiro para controlar a quantidade de pessoas no array.
- A classe Grupo também possuirá um construtor vazio:

```
public class Grupo {  
    private Pessoa[] pessoas = new Pessoa[5];  
    private int numeroPessoas = 0;  
  
    public Grupo() {  
  
    }  
}
```

- Como o atributo pessoas é um array, seu set recebe um objeto a ser adicionado e a cada inserção incrementa em um a variável que controla o número de objetos do array.

Resolução Exercício 1

```
public void setPessoa(Pessoa p) {  
    if (this.numeroPessoas < 5) {  
        pessoas[this.numeroPessoas] = p;  
        this.numeroPessoas++;  
    }  
}
```

- Também foi implementado um método ordena, no qual realiza um bubble sort utilizando do método **calculaImc()** da classe Pessoa:

Resolução Exercício 1

```
public void ordena() {  
    for (int i = 0; i < 5; i++) {  
        for (int j = i + 1; j < 5; j++) {  
            if (this.pessoas[j].calculaImc() > this.pessoas[i].calculaImc()) {  
                Pessoa aux = this.pessoas[j];  
                this.pessoas[j] = this.pessoas[i];  
                this.pessoas[i] = aux;  
            }  
        }  
    }  
}
```

- Observe que o método não retorna nada, ele apenas altera as posições do vetor.
- Agora para testar, criaremos uma classe Main, e instanciaremos um objeto do tipo Grupo:

```
public class Main {  
  
    public static void main(String[] args) {  
  
        Grupo g = new Grupo();  
  
    }  
}
```

Resolução Exercício 1

- Criaremos cinco instâncias da classe Pessoa, para adicionar no vetor da Grupo:

```
Pessoa p1 = new Pessoa();  
p1.setNome("Joao");  
p1.setAltura(1.70 f);  
p1.setIdade(19);  
p1.setMassa(70.0 f);
```

```
Pessoa p2 = new Pessoa();  
p2.setNome("Julia");  
p2.setAltura(1.65 f);  
p2.setIdade(19);  
p2.setMassa(62.5 f);
```

```
Pessoa p3 = new Pessoa();  
p3.setNome("Marcos");  
p3.setAltura(1.79 f);  
p3.setIdade(20);  
p3.setMassa(75);
```

Resolução Exercício 1

```
Pessoa p4 = new Pessoa();  
p4.setNome("Luiza");  
p4.setAltura(1.68f);  
p4.setIdade(20);  
p4.setMassa(65);
```

```
Pessoa p5 = new Pessoa();  
p5.setNome("Leticia");  
p5.setAltura(1.66f);  
p5.setIdade(20);  
p5.setMassa(69);
```

- Para definir valores do tipo float é necessário incluir o "f" após o número!
- Agora é necessário adicionar todas as pessoas a Grupo "t":

```
g.setPessoa(p1);  
g.setPessoa(p2);  
g.setPessoa(p3);  
g.setPessoa(p4);  
g.setPessoa(p5);
```

Resolução Exercício 1

- Realizar a ordenação delas com o método **ordena()** da classe Grupo:

```
g.ordena();
```

- E por fim, exibiremos no console os atributos e valores após a ordenação:

```
for (int i = 0; i < 5; i++) {  
    System.out.println("Nome: " + g.getPessoas()[i].getNome());  
    System.out.println("Altura: " + g.getPessoas()[i].getAltura());  
    System.out.println("Idade: " + g.getPessoas()[i].getIdade());  
    System.out.println("Massa: " + g.getPessoas()[i].getMassa());  
    System.out.println("IMC: " + g.getPessoas()[i].calculaImc() + "\n");  
}  
}
```

Exercício 1

Resolução Exercício 1

Exercício 2

Resolução Exercício 2

Exercicio 2

2. Faça um programa em Java, orientado a objetos que gerencie as informações de uma pet-shop. Esse programa deve ser capaz de listar os animais atendidos pelos veterinários com todas as informações relacionadas a eles, além de permitir o cadastro de novos animais e donos. O aluno deve desenvolver esse programa como uma aplicação em duas camadas, contendo uma camada com as classes de dados e outra camada contendo a interface com o usuário. Todos os dados referentes ao pet-shop deverão ser fornecidos por meio da interface com o usuário. Segue abaixo a lista de classes que o programa deve conter, com seus respectivos atributos e métodos:

Exercicio 2

Classe Endereço	
Atributos	Métodos
rua numero bairro cidade estado cep	gets()/sets()

Tabela 1: Classe Endereço

Exercicio 2

Classe Veterinário	
Atributos	Métodos
nome salario Endereço Animais[] quantidadeAnimais	gets()/sets()

Tabela 2: Classe Veterinário: Essa classe possui uma lista (array) de Animais, que representam os animais atendidos pelo veterinario, a quantidade de itens do array é controlada pelo atributo *quantidadeAnimais*. Além de possuir um objeto do tipo Endereço.

Exercicio 2

Classe Dono	
Atributos	Métodos
nome Endereço cpf	gets()/sets()

Tabela 3: Classe Dono: essa classe representa o dono de um animal. Todo dono possui um objeto do tipo Endereço.

Classe Animal	
Atributos	Métodos
nome Dono especie descricao	gets()/sets()

Exercicio 2

Tabela 4: Classe Animal: todo animal possui um dono, uma especie (String) e uma descrição sobre o animal (também uma String)

Classe SistemaPetShop	
Atributos	Métodos
Veterinarios[50]	cadastrar Veterinario() mostrar Veterinarios() cadastrar Endereço do Veterinario()
quantidadeVeterinarios	cadastrar Animal() mostrar Animais() cadastrarDono() cadastrar Endereço do Dono()

Tabela 5: Classe SistemaPetShop: abaixo está a descrição completa dos métodos:

Exercicio 2

- **void cadastrarVeterinario():** esse método requisita ao usuário as informações relacionadas ao veterinario, sendo estas: nome e salário. O sistema então adiciona ao array de veterinarios e incrementa o atributo quantidadeVeterinarios
- **mostrarVeterinarios():** esse método deve exibir todos os veterinarios cadastrados no sistema, cada um contendo um número que o identifique (a posição no vetor).
- **void cadastrarEnderecoVeterinario():** esse método deve exibir os veterinarios já cadastrados e o usuário deve escolher qual veterinario ele quer cadastrar o endereço. Após escolhido o veterinário, é requisitado ao usuário as informações referentes ao endereço: rua, numero, bairro, cidade, estado e cep. Após o usuário digitar essas informações, o endereço é cadastrado ao veterinário escolhido.

Exercicio 2

- **cadastrarAnimal():** esse método deve exibir os veterinarios ja cadastrados e o usuário deve escolher qual veterinario ele quer cadastrar um novo animal. Após escolhido o veterinário, é requisitado ao usuário as informações referentes ao animal: nome, especie e descrição. Após o usuário digita essas informações, o animal é cadastrado ao funcionário escolhido e o atributo quantidadeAnimais é incrementado.
- **mostrarAnimais():** esse método deve exibir os veterinarios ja cadastrados e o usuário deve escolher qual veterinario ele deseja visualizar os animais atendidos por ele, cada um contendo um número que o identifique (a posição no vetor).
- **cadastrarDono():** esse método deve exibir os animais já cadastrados e o usuário deve escolher qual animal ele deseja cadastrar um dono. Após escolhido o animal, é requisitado ao usuário as informações referentes ao dono: nome e cpf. Após digitadas essas informações, o dono é cadastrado ao animal escolhido.

Exercicio 2

- **cadastrarEnderecoDono()**: esse método deve exibir os animais cadastrados e o usuário deve escolher qual animal ele deseja cadastrar o endereço do dono. Após escolhido o animal, é exibido as informações referentes ao dono: nome e cpf, e requisitado as informações referentes ao endereço: rua, numero, bairro, cidade, estado e cep. Após o usuário digitar essas informações, o endereço é cadastrado ao dono do animal escolhido.

Seções

Exercício 1

Resolução Exercício 1

Exercício 2

Resolução Exercício 2

Resolução Exercício 2

- Como iremos modelar um sistema em duas camadas, uma irá conter as classes básicas: Endereço, Veterinário, Dono e Animal;
- A outra irá conter a classe que faz a interface das funcionalidades do sistema com o usuário;
- Cada camada normalmente é separada em um package (pacote) da IDE;
- Para esse exercício teremos dois packages: dados e apresentação;
- Vamos primeiro implementar as classes do pacote de dados.
- A classe Endereço, possui a declaração do package a qual ela pertence:

```
package dados ;
```

- Possui também 6 atributos como descritos no enunciado:

Resolução Exercício 2

```
public class Endereco {  
    private String rua;  
    private int numero;  
    private String bairro;  
    private String cidade;  
    private String estado;  
    private int cep;
```

- E para cada atributo, possui um método get (retorna o valor do atributo) e um método set (define o valor do atributo):

```
    public String getRua() {  
        return this.rua;  
    }  
  
    public void setRua(String rua) {  
        this.rua = rua;  
    }  
  
    public int getNumero() {  
        return this.numero;
```

Resolução Exercício 2

```
}  
  
public void setNumero(int numero) {  
    this.numero = numero;  
}  
  
public String getBairro() {  
    return this.bairro;  
}  
  
public void setBairro(String bairro) {  
    this.bairro = bairro;  
}  
  
public String getCidade() {  
    return this.cidade;  
}  
  
public void setCidade(String cidade) {  
    this.cidade = cidade;  
}  
  
public String getEstado() {  
    return this.estado;  
}
```

Resolução Exercício 2

```
}  
  
public void setEstado(String estado) {  
    this.estado = estado;  
}  
  
public int getCep() {  
    return this.cep;  
}  
  
public void setCep(int cep) {  
    this.cep = cep;  
}
```

- A maioria das IDE's possui geradores de getters e setters, logo, não serão mais apresentados aqui os métodos getters e setters das classes seguintes.
- A classe Endereço também possui um método **toString()** que retorna todos os atributos concatenados em uma String só:

Resolução Exercício 2

```
public String toString() {  
    String endereco = "";  
  
    endereco += "Rua: " + this.rua;  
    endereco += ", ";  
    endereco += this.numero;  
    endereco += " - ";  
    endereco += this.bairro;  
    endereco += ", ";  
    endereco += this.cidade;  
    endereco += " - ";  
    endereco += this.estado;  
    endereco += " - CEP: ";  
    endereco += this.cep;  
  
    return endereco;  
}
```

- Agora iremos definir a classe Dono;
- Essa possui um nome, Endereço e cpf;

Resolução Exercício 2

```
package dados;  
  
public class Dono {  
    private String nome;  
    private Endereco endereco;  
    private int cpf;
```

- E também possui seus getters e setters, tal como o do atributo Endereço:

```
    public Endereco getEndereco() {  
        return this.endereco;  
    }  
  
    public void setEndereco(Endereco endereco) {  
        this.endereco = endereco;  
    }
```

- Já a classe Animal, possui um nome, Dono, especie e descrição:

Resolução Exercício 2

```
package dados;  
  
public class Animal {  
  
    private String nome;  
    private Dono dono;  
    private String especie;  
    private String descricao;
```

- Também foi implementado um método **toString()** que será utilizado mais tarde:

```
    public String toString() {  
        String animal = "";  
        animal += "Nome: " + this.nome + "\n";  
        animal += "Especie: " + this.especie + "\n";  
        animal += "Descrição: " + this.descricao + "\n";  
        return animal;  
    }
```

- A classe Veterinario é a mais importante do sistema;

Resolução Exercício 2

- Ela possui os atributos: nome, salario, Endereço, um array de Animais e um atributo para controlar a quantidade de objetos nesse array:

```
package dados;  
  
public class Veterinario {  
    private String nome;  
    private float salario;  
    private Endereco endereco;  
    private Animal[] animais = new Animal[30];  
    private int quantidadeAnimais = 0;
```

- O método **setAnimal()** é um pouco diferente das demais classes;
- Ao adicionar um objeto ao array de animais, o atributo que controla o tamanho é incrementado;

Resolução Exercício 2

```
public void setAnimais(Animal animal) {  
    if (this.quantidadeAnimais < 30) {  
        this.animais[this.quantidadeAnimais] = animal;  
        this.quantidadeAnimais++;  
    }  
}
```

- A classe Veterinario também possui um método **toString()** diferente. Esse método só exibe o endereço do Veterinario, caso este exista:

```
public String toString() {  
    String veterinario = "";  
  
    veterinario += "Nome: " + this.nome + "\n";  
    veterinario += "Salario: " + this.salario + "\n";  
  
    if (this.endereco != null) {  
        veterinario += "Endereço: " + this.endereco.toString() + "\n";  
    }  
}
```


Resolução Exercício 2

```
        return veterinario;  
    }
```

- Agora que as classes do pacote de dados foram definidas, iremos criar a classe Main:
- Como ela pertence ao pacote apresentação, precisamos declará-lo;
- Iremos utilizar a classe Scanner para fazer a leitura de dados via console, por isso é necessário importa-la;
- Também é necessário importar as classes que pertencem a outros pacotes:

```
package apresentacao;  
  
import java.util.Scanner;  
  
import dados.Animal;  
import dados.Dono;  
import dados.Endereco;  
import dados.Veterinario;
```

Resolução Exercício 2

- Como estamos utilizando o método Main, seus atributos serão estáticos:

```
public class SistemaPetShop {  
  
    private static Veterinario[] veterinarios = new Veterinario[50];  
    private static int numeroVeterinarios = 0;  
    private static Scanner s = new Scanner(System.in);
```

- Teremos um array com tamanho máximo de 50, e uma variável para controlar o tamanho desse array.
- Um método do tipo void para imprimir um menu ao usuário:

```
    public static void imprimeMenu() {  
        System.out.println("Escolha uma opção:");  
        System.out.println("0 - Sair");  
        System.out.println("1 - Cadastrar Veterinario");  
        System.out.println("2 - Exibir Veterinarios");  
        System.out.println("3 - Cadastrar Endereço do Veterinario");  
        System.out.println("4 - Cadastrar Animal");  
        System.out.println("5 - Exibir Animais");  
        System.out.println("6 - Cadastrar Dono");  
    }
```

Resolução Exercício 2

- O método **main()** possui uma variável para controlar as escolhas do menu:

```
public static void main(String[] args) {  
    int opcao = -1;
```

- Dentro de um laço de repetição, o menu é exibido e o usuário escolhe uma opção:

```
while (opcao != 0) {  
    imprimeMenu();  
    opcao = s.nextInt();
```

- E um *switch case* determina qual método será chamado:

Resolução Exercício 2

```
switch (opcao) {  
    case 0:  
        break;  
    case 1:  
        cadastrarVeterinario();  
        break;  
    case 2:  
        mostrarVeterinarios();  
        break;  
    case 3:  
        cadastrarEnderecoVeterinario();  
        break;  
    case 4:  
        cadastrarAnimal();  
        break;  
    case 5:  
        mostrarAnimais();  
        break;  
    case 6:  
        cadastrarDono();  
        break;  
    default:  
        System.out.println("Valor incorreto!");  
}
```

Resolução Exercício 2

```
        break;  
    }
```

- O primeiro deles é o método que cadastra novos veterinários;
- Esse método lê do console as informações, instância um novo objeto do tipo veterinário, adiciona-o ao array e incrementa a variável que controla a quantidade de objetos:

```
public static void cadastrarVeterinario() {  
    if (numeroVeterinarios < 50) {  
        Veterinario v = new Veterinario();  
  
        System.out.println("Digite o nome do veterinario:");  
  
        String nome = s.nextLine();  
        nome = s.nextLine();  
        v.setNome(nome);  
  
        System.out.println("Digite o salario do veterinario:");
```

Resolução Exercício 2

```
        float salario = s.nextFloat();
        v.setSalario(salario);

        veterinarios[numeroVeterinarios] = v;
        numeroVeterinarios++;
    }
}
```

- O método seguinte exibe no console os veterinários que existem no array;
- Esse método percorre o array, imprime a posição do array e utiliza o método **toString()** do objeto para exibir as informações no console:

```
public static void mostrarVeterinarios() {
    for (int i = 0; i < numeroVeterinarios; i++) {
        System.out.println("Codigo: " + i);
        System.out.println(veterinarios[i].toString());
    }
}
```

Resolução Exercício 2

- O próximo método é utilizado para o usuário escolher um veterinário. Esse método é utilizado em vários outros métodos do programa. Ele exibe os veterinários cadastrados, o usuário escolhe um, e o método retorna esse objeto:

```
public static Veterinario escolherVeterinario() {  
    System.out.println("Escolha um veterinario para cadastrar um animal:");  
    mostrarVeterinarios();  
  
    int escolha = s.nextInt();  
  
    if (escolha > numeroVeterinarios) {  
        System.out.println("Numero inválido");  
        return null;  
    } else {  
        return veterinarios[escolha];  
    }  
}
```

- Já o próximo método, requisita do usuário as informações referentes ao endereço, seja de um veterinário ou de um dono, e retorna um objeto do tipo endereço com todas as informações:

Resolução Exercício 2

```
public static Endereco cadastrarEndereco() {  
    Endereco e = new Endereco();  
  
    System.out.println("Digite o nome da rua:");  
  
    String rua = s.nextLine();  
    rua = s.nextLine();  
    e.setRua(rua);  
  
    System.out.println("Digite o n mero da casa:");  
    e.setNumero(s.nextInt());  
  
    System.out.println("Digite o nome do bairro:");  
  
    String bairro = s.nextLine();  
    bairro = s.nextLine();  
    e.setBairro(bairro);  
  
    System.out.println("Digite o nome da cidade:");  
  
    String cidade = s.nextLine();  
  
    e.setCidade(cidade);  
}
```


Resolução Exercício 2

```
        System.out.println("Digite o nome do estado:");  
  
        String estado = s.nextLine();  
  
        e.setEstado(estado);  
  
        System.out.println("Digite o n mero do cep:");  
        e.setCep(s.nextInt());  
  
        return e;  
    }
```

- O método de cadastrar endereço do veterinário, então utiliza esses dois métodos que foram apresentados anteriormente:

Resolução Exercício 2

```
public static void cadastrarEnderecoVeterinario() {  
    Veterinario vet = escolherVeterinario();  
  
    if (vet != null) {  
        vet.setEndereco(cadastrarEndereco());  
    }  
}
```

- Como o método **escolherVeterinario()** pode retornar **null** é necessário fazer a verificação antes.
- O próximo método faz o cadastro de novos animais. Esse método requisita do usuário as informações e instância um novo objeto para então retorná-lo ao método que o chamou:

Resolução Exercício 2

```
public static Animal cadastrarAnimal() {  
    Animal a = new Animal();  
  
    System.out.println("Digite o nome do animal:");  
    String nome = s.nextLine();  
    nome = s.nextLine();  
    a.setNome(nome);  
  
    System.out.println("Digite a especie do animal:");  
    String especie = s.nextLine();  
    a.setEspecie(especie);  
  
    System.out.println("Digite a descriçao do animal:");  
    String descricao = s.nextLine();  
  
    a.setDescricao(descricao);  
  
    return a;  
}
```

Resolução Exercício 2

- Então o método de cadastrar animal de um veterinário utiliza esse método. Primeiro um veterinário é escolhido pelo usuário. Caso o veterinário retornado seja válido, é adicionado um novo animal ao array:

```
public static void cadastrarAnimalVeterinario() {  
    Veterinario vet = escolherVeterinario();  
  
    if (vet != null) {  
        vet.setAnimais(cadastrarAnimal());  
    }  
}
```

- Já o método para cadastrar dono faz o mesmo que o de cadastrar endereço e animal. Faz a entrada de dados e retorna um objeto do tipo Dono com seus dados:

Resolução Exercício 2

```
public static Dono cadastrarDono() {  
  
    Dono d = new Dono();  
  
    System.out.println("Digite o nome do dono:");  
    String nome = s.nextLine();  
    nome = s.nextLine();  
    d.setNome(nome);  
  
    System.out.println("Digite o cpf do dono:");  
    d.setCpf(s.nextInt());  
  
    d.setEndereco(cadastrarEndereco());  
  
    return d;  
  
}
```

- E o método de mostrar animais requisita ao usuário o veterinário que deseja consultar e exibe os animais associados a ele:

Resolução Exercício 2

```
public static void mostrarAnimais() {  
    Veterinario vet = escolherVeterinario();  
  
    if (vet != null) {  
        for (int i = 0; i < vet.getQuantidadeAnimais(); i++) {  
            System.out.println("Animal Atendido " + i);  
            System.out.println(vet.getAnimais()[i].toString());  
        }  
    }  
}
```

- Como o método de mostrar animais não retorna nada, não há como utiliza-lo no método de escolher animais. Entretanto esse método retorna o animal escolhido pelo usuário:


Resolução Exercício 2

```
public static Animal escolherAnimais() {  
    Veterinario vet = escolherVeterinario();  
  
    if (vet != null) {  
        for (int i = 0; i < vet.getQuantidadeAnimais(); i++) {  
            System.out.println("Animal Atendido " + i);  
            System.out.println(vet.getAnimais()[i].toString());  
        }  
  
        int escolha = s.nextInt();  
  
        if (escolha > vet.getQuantidadeAnimais()) {  
            System.out.println("N mero inv lido");  
            return null;  
        } else {  
            return vet.getAnimais()[escolha];  
        }  
    }  
    return null;  
}
```

Resolução Exercício 2

- E por fim o método de cadastrar dono, esse método já faz o cadastro do endereço junto;
- Como todo Dono é associado a um animal, é necessário utilizar o método de escolher um animal, para então cadastrar um dono a ele:

```
public static void cadastrarDonoAnimal() {  
    Animal a = escolherAnimais();  
    if (a != null) {  
        a.setDono(cadastrarDono());  
    }  
}
```


 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:
Vinicius Takeo Friedrich Kuwaki
vinicius.kuwaki@edu.udesc.br
github.com/takeofriedrich



UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA