Universidade Federal de Viçosa
Campus Rio Paranaíba
Sistemas de Informação
Prof. Alan Diêgo Aurélio Carneiro

SIN141
Computação Orientada a Objetos
Roteiro Aula Prática 3
25/03/2024

Introdução à Orientação a Objetos (Classes)

Professor

Exercício 1

Na nossa primeira aula prática de OO, vamos iniciar a implementação de uma classe **String**, a qual irá cada vez mais ficando melhor com o passar das aulas práticas, isto é, com mais funcionalidades a medida que o conteúdo for evoluindo.

Defina uma classe **String** seguindo o diagrama UML abaixo. **Dica:** Quando necessário, use a função correta da implementação para manipulação de strings da linguagem C.

String
- texto : string
+ digitaString() : void + imprimeString() : string + comprimento() : int

Execute este código **main** para verificar se sua classe ficou correta e confira com a saída esperada:

```
S = String()
s.digitaString();
print("String:", s.imprimeString() )
print(s.imprimeString(), “possui”,  s.comprimento(), “caracteres”)
```

Saída Esperada:

```
Digite sua string:
SIN141-Computacao Orientada a Objetos (INFORMACAO DIGITADA)
String: SIN141-Computacao Orientada a Objetos
SIN141-Computacao Orientada a Objetos possui 37 caracteres.
```

Obs: A sua classe String poderá ser usada por você futuramente, logo mantenha-a em dia nas entregas do Moodle. Além disso inclua este cabeçalho no código da sua classe:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# ----------------------------------------------------------------------------
# Criado por  : Prof Alan Carneiro
# Data de criação: 25/03/2024
```

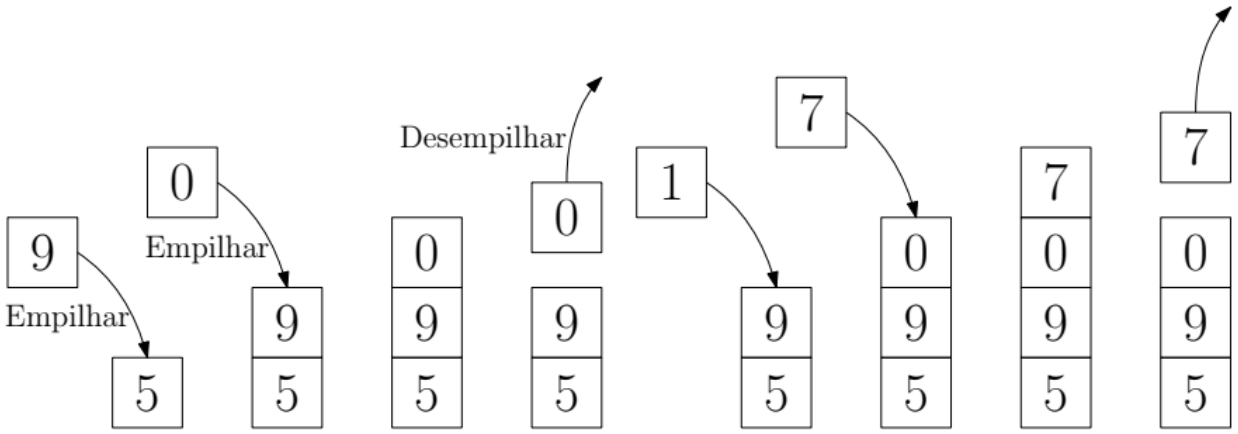
```
# versão ='1.0'
# ---------------------------------------------------------------------------
# ---------------------------------------------------------------------------
""" Classe String contendo alguns métodos de manipuação de
textos"""
# ---------------------------------------------------------------------------
```

Exercício 2

Neste exercício, você deverá tentar separar todo o seu código da classe **String** de sua aplicação (**Main**) para usar o makefile, crie o makefile conforme visto na aula e teste-o.

Roteiro
Exercício 1

Uma pilha é um tipo de estrutura de dados utilizado em computação que funciona como uma pilha de objetos do nosso dia a dia: o último elemento a ser inserido é o primeiro a ser retirado. Por causa da forma como os dados são guardados e acessados, essa estrutura também é chamada de LIFO (last-in first-out, o último a entrar é o primeiro a sair). Exemplo de uma pilha de inteiros:



a.

A implementação de uma estrutura de dados do tipo pilha pode ser feita atraves de uma classe. Crie a classe Pilha e de na para ela os seguintes metodos:

a. Atributos: elementos com valor default igual a uma lista vazia. Outros atributos podem ser criados caso ache necessário, Para os demais exercícios, considere que o elemento da posição 0 do atributo criado no construtor é aquele da base da pilha e o elemento da posição final é aquele do topo da pilha.
b. Metodo empilhar: Recebe um elemento passado por argumento e acrescenta o mesmo ao topo da pilha.
c. Metodo desempilhar: Retira e retorna o elemento que esta no topo da pilha. Caso a pilha esteja vazia, o metodo deve imprimir uma mensagem de erro.
d. Metodo getPilha: Retorna uma copia do atributo com os elementos da pilha.
e. Metodo lenPilha: Retorna o numero de elementos da pilha.

Teste a classe criada:

b. Crie duas instâncias de pilha, p1 e p2, onde p1 deve ser inicializado com os elemento [1,7,9] e p2 deve ser inicializado sem passar nenhum argumento.

c. Veri que o que acontece se tentamos criar uma terceira instância passando um inteiro como argu-mento. Foi criada a instância uma instância de Pilha?

d. Utilizando o metodo empilhar da instância p2, adicione os elementos 4 e 10. Utilize em seguida getPilha para veri car se os elementos foram adicionados da forma certa.

e. Utilize o metodo desempilhar da instância p2 ate que a mensagem informando que a pilha esta vazia apareca. Veri que se o ultimo elemento empilhado do item anterior foi o primeiro a ser desempilhado.

f. Utilizando o metodo empilhar da instância p2, adicione o elemento 5.

g. Crie a instância p3 atraves da soma de p2 e p1 de forma que os elementos de p2 quem na base da pilha. Utilize getPilha para veri car se os elementos foram adicionados em p3 da forma certa. Agora faca o contrario: crie a instância p4 da soma de p2 e p1 de forma que os elementos de p1 quem na base da pilha. Utilize getPilha para veri car se os elementos foram adicionados em p4 da forma certa.

h. Utilize o metodo lenPilha de p3 e veri que se o resultado esta correto.

Exercício 2

Vamos organizar todas as informações e métodos de uma **Classe Pessoa** seguindo o diagrama UML abaixo:

Pessoa
+ Nome: char [100] (*note que este atributo é público*) - Idade: int - Peso: double - Altura: double
+ setIdade (int) : void + setPeso (double) : void + setAltura (double) : void + getIdade () : int + getPeso () : double + getAltura () : double + imprimirDados() : void + retorna_IMC() : double;

Crie uma função main que permita o usuário digitar os dados de duas pessoas diferentes agora

pensando em **Objetos**:

a) Chame o método imprimirDados para cada uma das duas pessoas.

b) Na main, imprima os dados da pessoa que for mais velha;

c) Mostre o IMC de cada pessoa.

OBS: IMC = peso/(altura*altura);

Entrada
As informações de duas pessoas distintas, nome, idade, peso, altura;

Saída
As informações solicitadas nos itens a, b, e c;