

SBVESDD: Estruturas de Dados

Aula 02: Estruturas de Dados Lineares - Pilhas

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Pilha

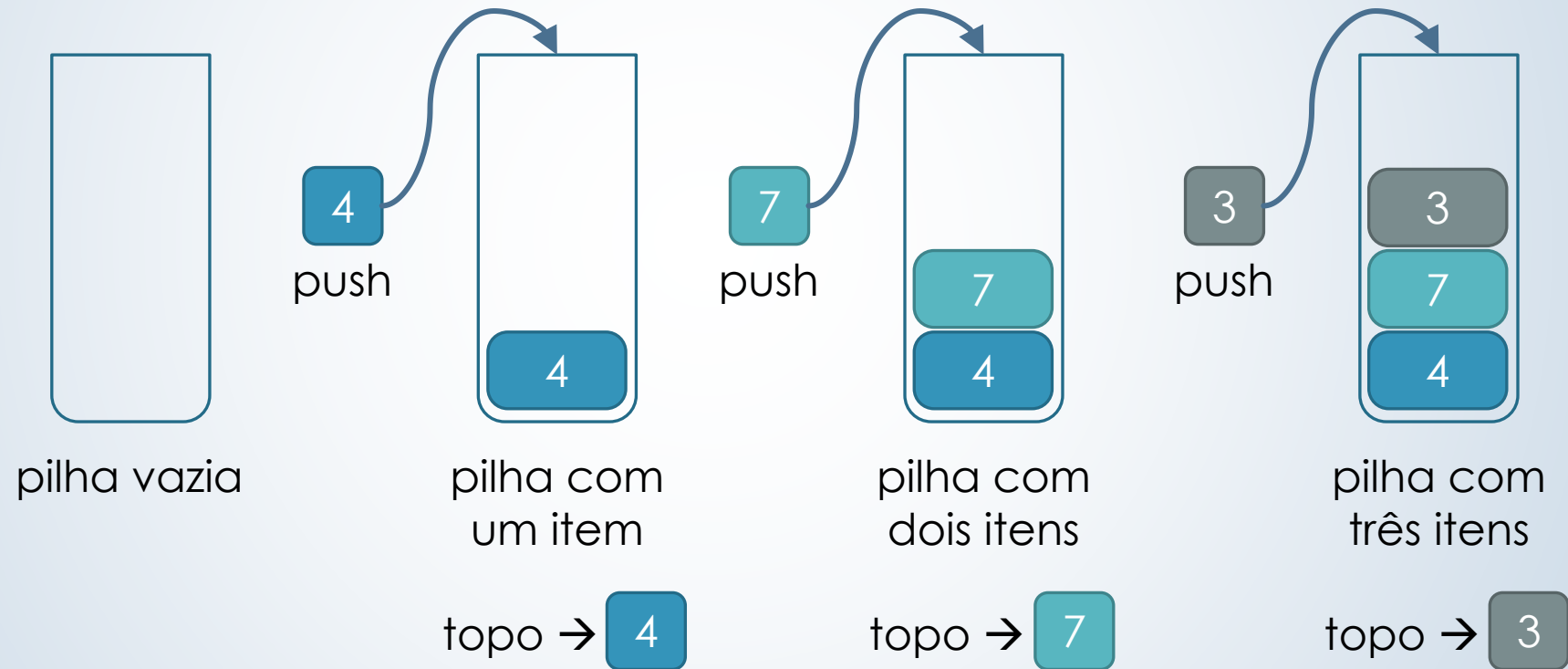
Contextualização

- ▶ A Pilha é talvez uma das estruturas de dados mais importantes na Ciência da Computação, pois é usada na resolução de diversos tipos de problemas, dada a sua característica que possibilita a memorização de ações que foram executadas durante o tempo, ou seja, ela pode armazenar a “memória” do que aconteceu;
- ▶ A Pilha é definida como uma estrutura de dados do tipo LIFO (*Last In First Out*) ou FILO (*First In Last Out*), pois os elementos que são inseridos nela, processo denominado empilhar (**push**), são sempre inseridos no seu final, comumente chamado de **topo** da pilha e o processo inverso do empilhamento é o de desempilhar (**pop**), sendo que o elemento que é desempilhado é sempre aquele que está no **topo** da pilha. A seguir é apresentado um esquema onde é simulada a execução de uma pilha.

Pilha

Operações

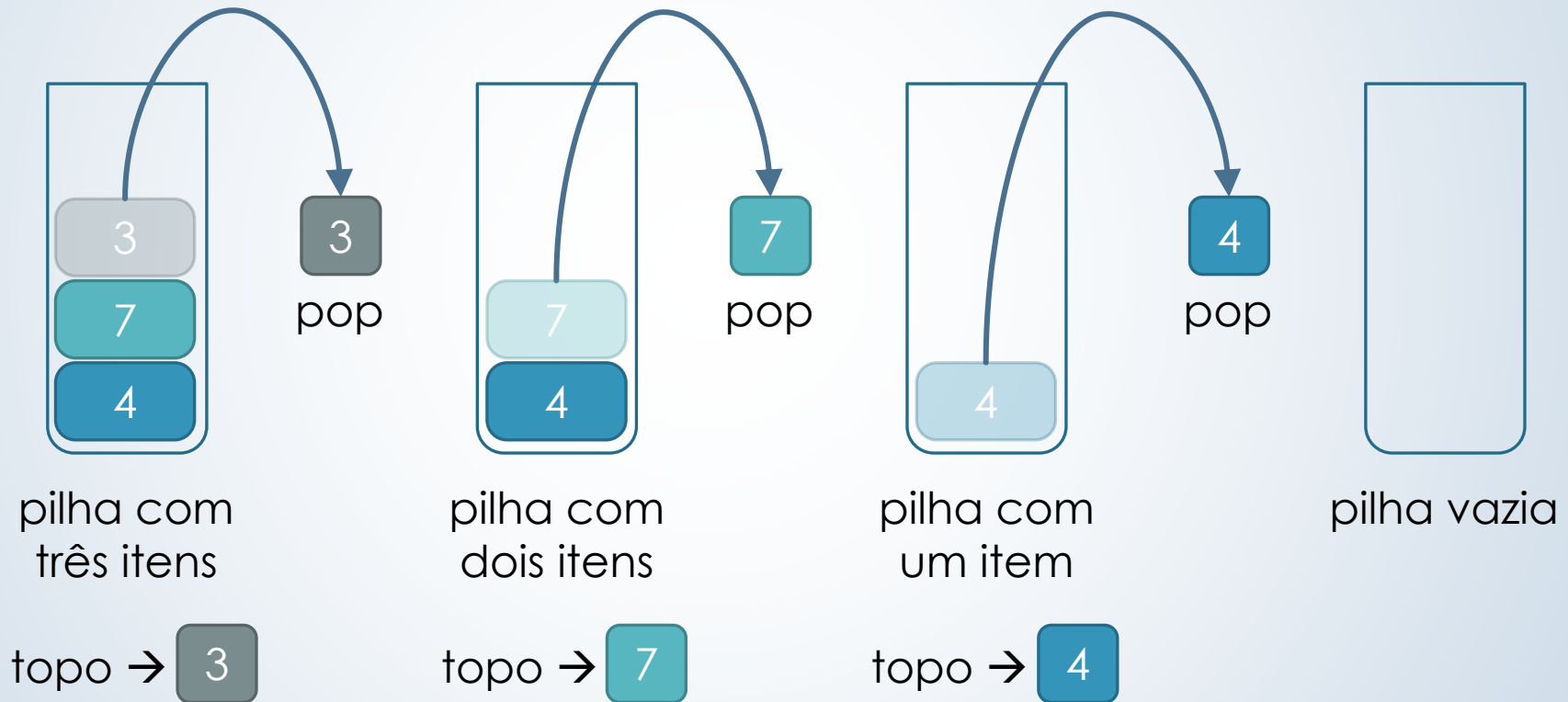
- Operação empilhar (**push**):



Pilha

Operações

► Operação desempilhar (**pop**):



Pilha

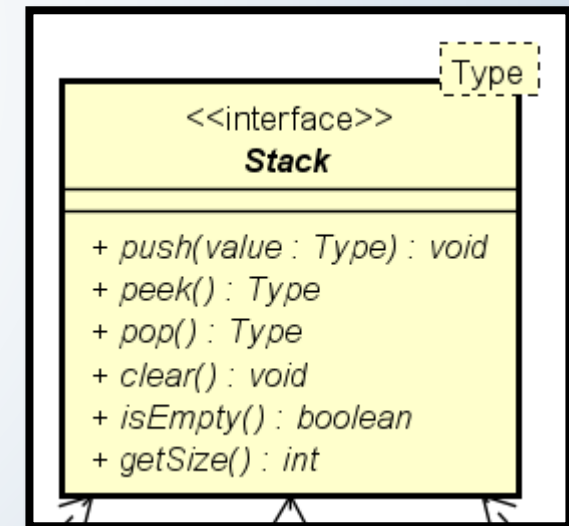
Aplicações

- Dentre as aplicações das pilhas, pode-se citar:
 - Mecanismo de desfazer/refazer (*undo/redo*);
 - Avaliação de expressões (balanceamento de pares de símbolos);
 - Pilhas de invocação de funções/métodos/subprogramas;
 - Avaliação de expressões matemáticas;
 - Conversões entre notações pré-fixada, infixada e pós-fixada;
 - Autômato de/à pilha (reconhecedor de expressões de uma linguagem livre de contexto);
 - Busca em profundidade em grafos.

Pilha

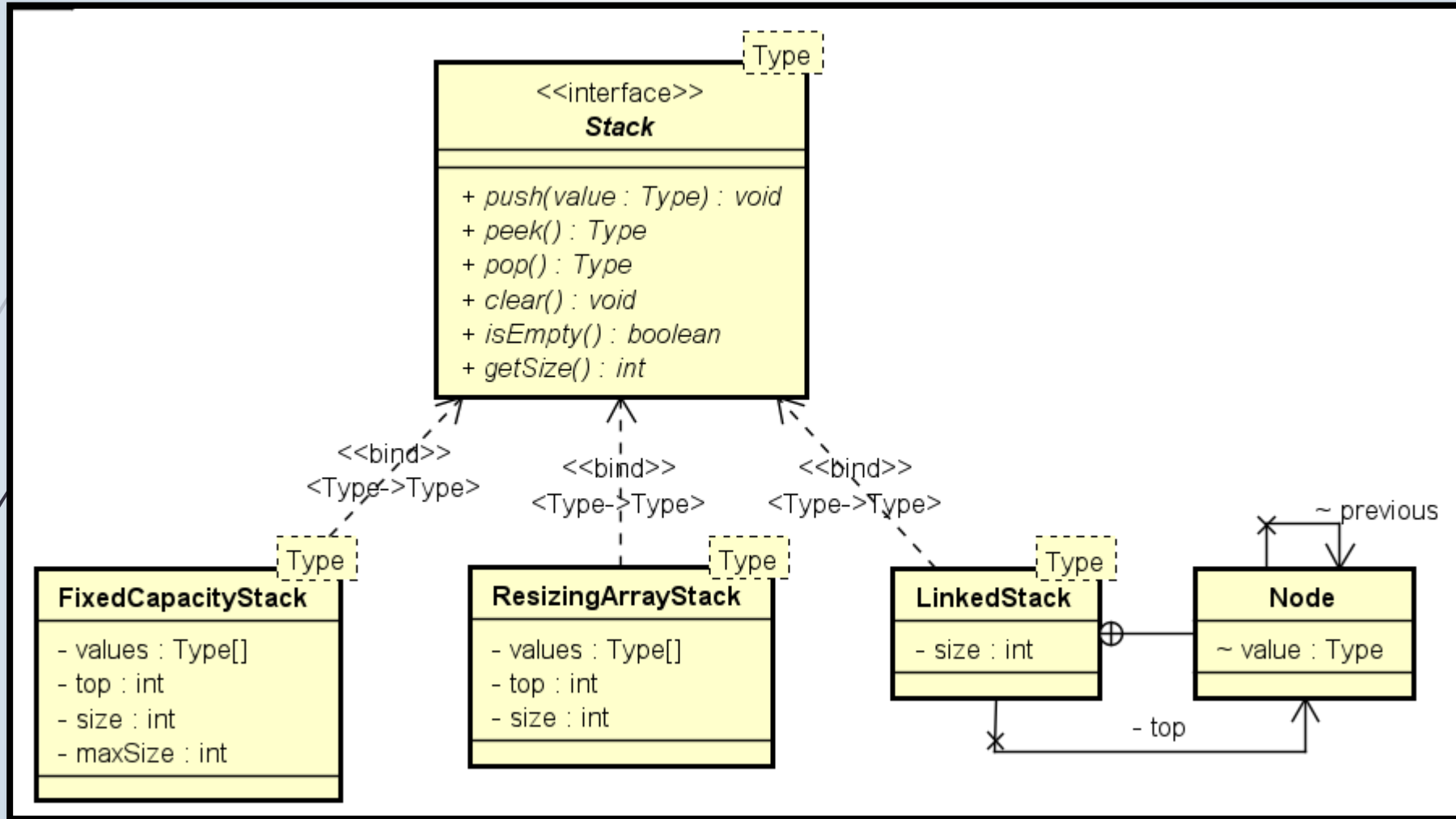
Implementação

- Iremos estudar três formas de implementar Pilhas:
 1. Pilha de capacidade fixa, também chamada de pilha estática;
 2. Pilha redimensionável;
 3. Pilha com estrutura encadeada, também chamada de pilha dinâmica.
- Seguiremos uma API padrão (em inglês):
 - **push**: empilha um item/elemento;
 - **peek**: consulta qual o item/elemento está no topo;
 - **pop**: remove um item/elemento;
 - **clear**: limpa a pilha, removendo todos os elementos;
 - **isEmpty**: verifica se a pilha está vazia;
 - **getSize**: obtém a quantidade de itens/elementos empilhados.



Pilha

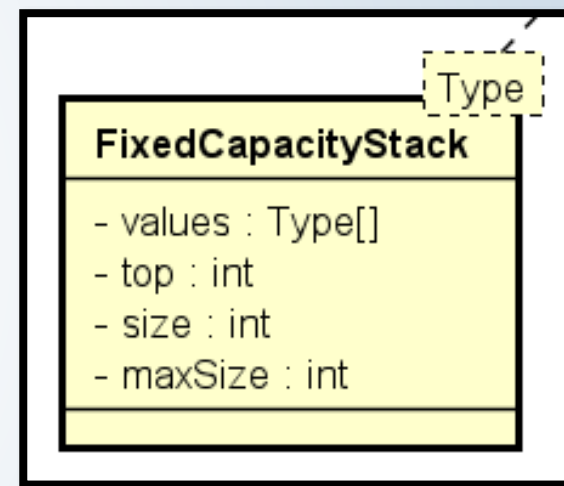
Implementação



Pilha

Implementação: **FixedCapacityStack**

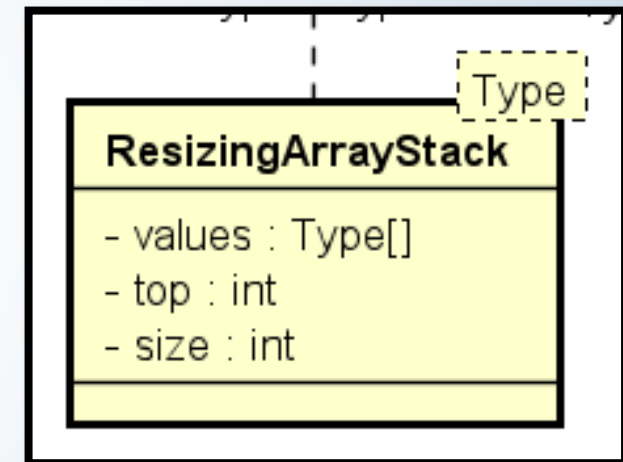
- Implementação usando um array como estrutura de armazenamento;
- O topo da pilha é indicado por um membro inteiro (chamaremos de ponteiro);
- A quantidade de elementos que podem ser armazenados, após a instanciação da pilha, é fixa;
- **Para pensar:** qual a ordem de crescimento, em relação ao tempo de execução, das operações fundamentais da pilha (**push** e **pop**) nessa implementação?



Pilha

Implementação: **ResizingArrayStack**

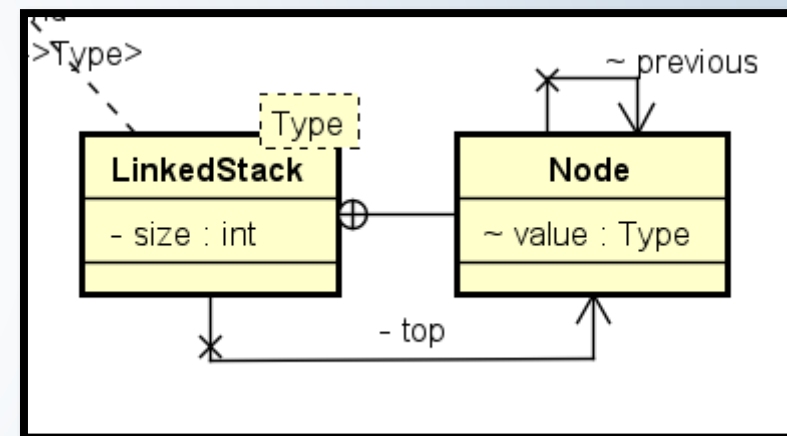
- Implementação usando um array como estrutura de armazenamento;
- O topo da pilha é indicado por um membro inteiro (chamaremos de ponteiro);
- A quantidade de elementos que podem ser armazenados, após a instanciação da pilha, é variável, pois ela crescerá ou diminuirá em função de inserções e remoções;
- **Para pensar:** qual a ordem de crescimento, em relação ao tempo de execução, das operações fundamentais da pilha (**push** e **pop**) nessa implementação?



Pilha

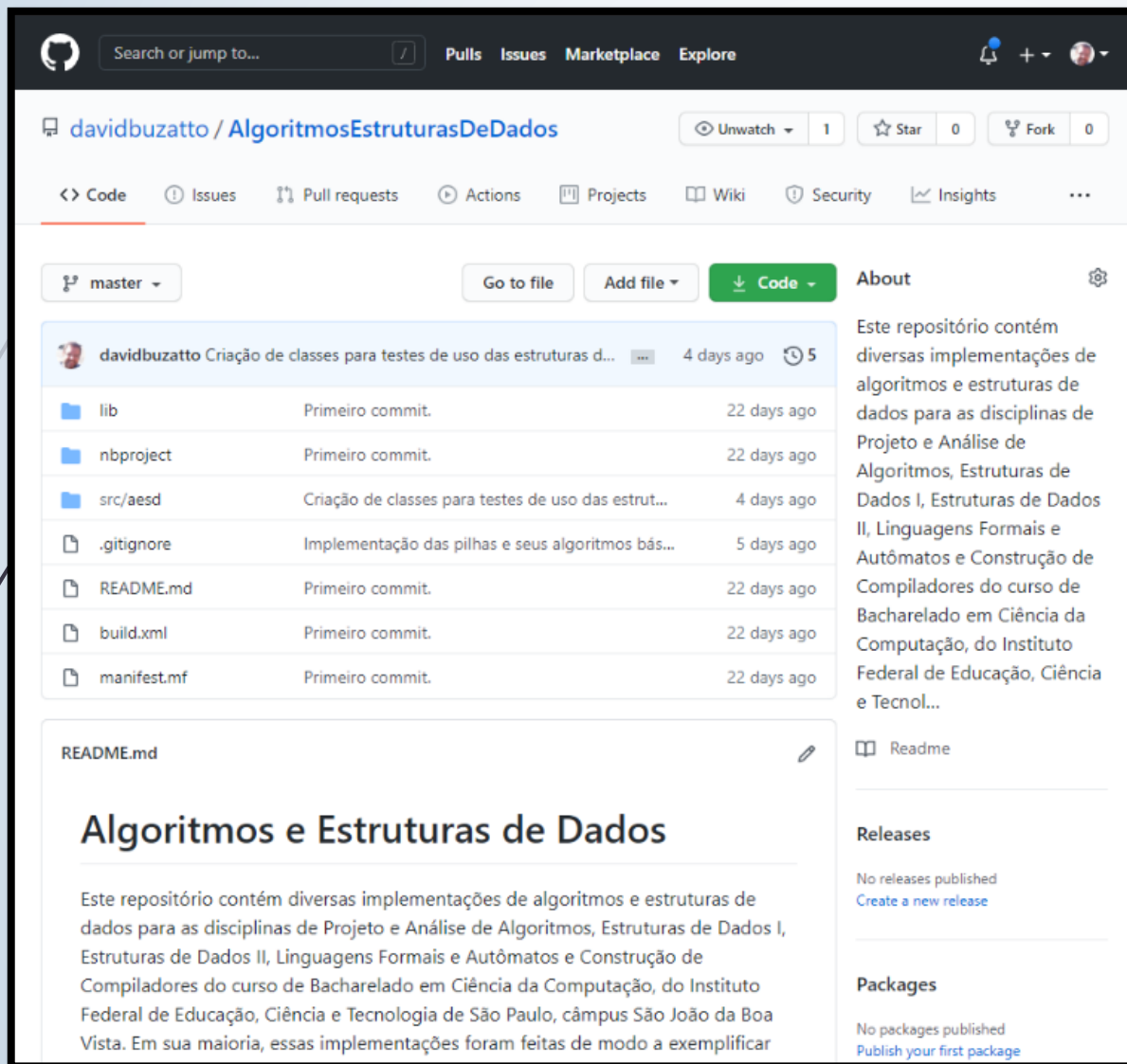
Implementação: **LinkedStack**

- Implementação usando uma estrutura baseada em nós (**nodes**) com encadeamento simples;
- O topo da pilha é indicado por um membro do tipo dos nós;
- A alocação e liberação de espaço para o armazenamento dos elementos tem característica dinâmica;
- **Para pensar:** qual a ordem de crescimento, em relação ao tempo de execução, das operações fundamentais da pilha (**push** e **pop**) nessa implementação?



Pilha

Repositório de Estruturas de Dados e Algoritmos



Search or jump to...

Pulls Issues Marketplace Explore

davidbuzatto / AlgoritmosEstruturasDeDados

Unwatch 1 Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights

master Go to file Add file Code

lib Primeiro commit. 22 days ago

nbproject Primeiro commit. 22 days ago

src/aesd Criação de classes para testes de uso das estru... 4 days ago

.gitignore Implementação das pilhas e seus algoritmos bás... 5 days ago

README.md Primeiro commit. 22 days ago

build.xml Primeiro commit. 22 days ago

manifest.mf Primeiro commit. 22 days ago

README.md

Algoritmos e Estruturas de Dados

Este repositório contém diversas implementações de algoritmos e estruturas de dados para as disciplinas de Projeto e Análise de Algoritmos, Estruturas de Dados I, Estruturas de Dados II, Linguagens Formais e Autômatos e Construção de Compiladores do curso de Bacharelado em Ciência da Computação, do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, câmpus São João da Boa Vista. Em sua maioria, essas implementações foram feitas de modo a exemplificar

Este repositório contém diversas implementações de algoritmos e estruturas de dados para as disciplinas de Projeto e Análise de Algoritmos, Estruturas de Dados I, Linguagens Formais e Autômatos e Construção de Compiladores do curso de Bacharelado em Ciência da Computação, do Instituto Federal de Educação, Ciência e Tecnol...

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

<https://github.com/davidbuzatto/AlgoritmosEstruturasDeDados>

Pilha

Exercícios de Implementação

- **Exercício i2.1:** No projeto **ESDC4Aula02**, implemente, na classe **Exercicioi2p1**, o método:

```
public static boolean isBalanced( String expression, char... pairs ) throws IllegalArgumentException
```

- Esse método deve verificar se uma expressão composta por pares de símbolos arbitrários está balanceada. Uma descrição mais detalhada do método pode ser encontrada no projeto.

Exercícios de Implementação

- **Exercício i2.2:** Expressões aritméticas podem ser escritas em três notações diferentes. Estamos acostumados à notação infixa, em que um operador é cercado por dois operandos. Por exemplo, $7 * 9$. Além da notação infixa, há também as notações pré-fixada (pré-fix) e pós-fixada (pós-fix). Na notação pré-fixada, os operadores vêm antes dos operandos, por exemplo $* 7 9$, enquanto notação pós-fixada, também chamada de Notação Polonesa Reversa (*Reverse Polish Notation* (RPN)), os operadores vêm após os operandos, ou seja, $7 9 *$. Uma vantagem das notações pré e pós-fixadas é que a precedência dos operadores já é codificada na própria expressão, não havendo necessidade de, por exemplo, como na notação infixa, cercar operações por parênteses quando queremos que estas sejam calculadas primeiro. A conversão entre essas três notações pode ser feita usando pilhas! No projeto **ESDC4Aula02**, você encontrará a classe **ExercicioI2p2**, que contém a descrição de todos os algoritmos de conversão. Você deve implementar os 6 métodos a seguir. Perceba também que na mesma classe, existem diversos métodos para lhe auxiliar na implementação.

Pilha

Exercícios de Implementação

➡ **Exercício i2.2:** Implemente os métodos:

a) `public static String prefixToInfix(String prefix) throws IllegalArgumentException`

b) `public static String prefixToPostfix(String prefix) throws IllegalArgumentException`

c) `public static String postfixToInfix(String postfix) throws IllegalArgumentException`

d) `public static String postfixToPrefix(String postfix) throws IllegalArgumentException`

e) `public static String infixToPrefix(String infix) throws IllegalArgumentException`

f) `public static String infixToPostfix(String infix) throws IllegalArgumentException`

Pilha

Exercícios de Implementação

- **Exercício i2.3:** No projeto **ESDC4Aula02**, implemente, na classe **Exercicioi2p3**, o método:

```
public static double evaluate( String expression ) throws IllegalArgumentException
```

- Esse método avalia uma expressão aritmética fornecida em qualquer forma (pré-fixada, infixada ou pós-fixada), gerando o resultado. As operações de adição, subtração, multiplicação e divisão devem ser suportadas. Uma descrição mais detalhada do método pode ser encontrada no projeto. No projeto **AlgoritmosEstruturasDeDados** há uma classe chamada **StackBasicAlgorithms** com a implementação do método **evaluatePostfixExpression** que pode ser usado como base para resolver esse exercício.

Pilha

Exercícios de Implementação

- **Observação:** Os esqueletos de todos os métodos que devem ser implementados, suas descrições detalhadas, bem como todos os testes de unidade, estão disponíveis no projeto **ESDC4Aula02** fornecido no material da disciplina. Esse projeto tem como dependência o projeto **AlgoritmosEstruturasDeDados**, disponível no GitHub. Ele já foi configurado com essa dependência, então você não precisa se preocupar. Os exercícios envolvem o uso de pilhas, sendo assim, você deverá usar a implementação das pilhas que são fornecidas nesse projeto e não as versões disponíveis na sua instalação do Java Development Kit.

Pilha

Desafio

- **Desafio d2.1:** Pesquise sobre o algoritmo de geometria computacional chamado *Graham Scan* (Exame de Graham), que tem como objetivo obter a envoltória convexa (*convex hull*) de um conjunto de pontos. Implemente na classe **Desafiod2p1** do projeto **ESDC4Aula02** o método `public static List<Point2D.Double> getConvexHull(List<Point2D.Double> points)` que processa uma lista de pontos e gera a envoltória convexa desse conjunto. A classe **Desafiod2p1** contém um método **main**, que ao executado (Shift+F6) abrirá uma interface gráfica de teste para você verificar seus resultados.

Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

GOODRICH M. T.; TAMASSIA, R. **Estruturas de Dados & Algoritmos em Java**. Porto Alegre: Bookman, 2013. 700 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.