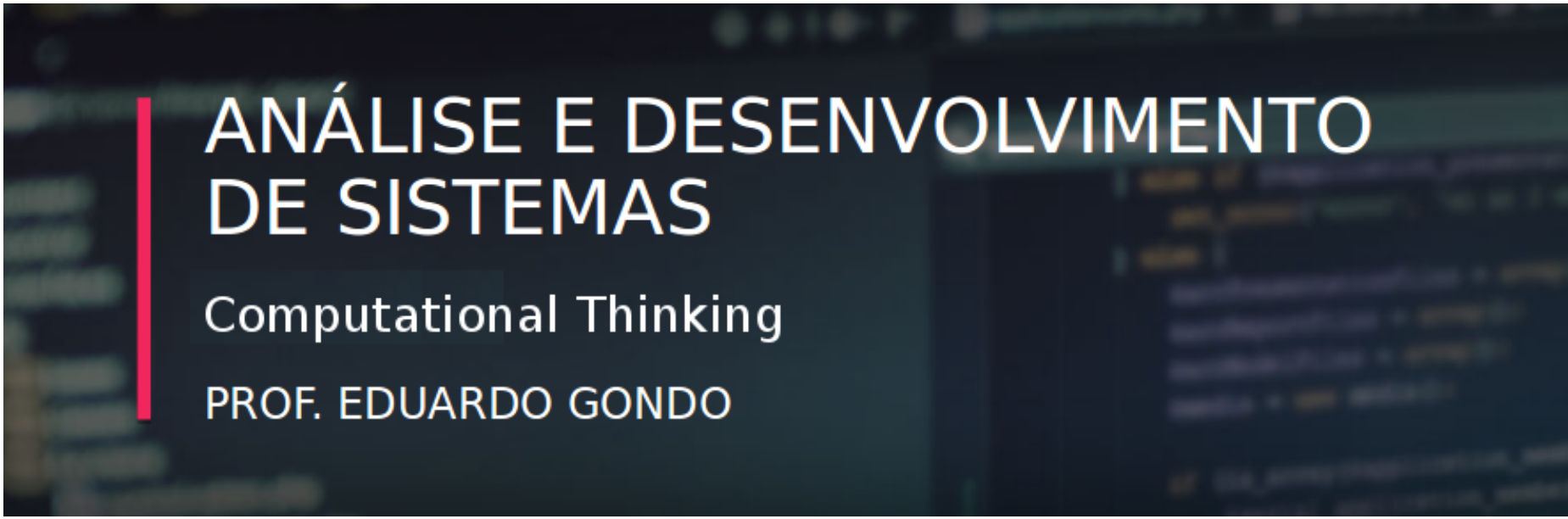
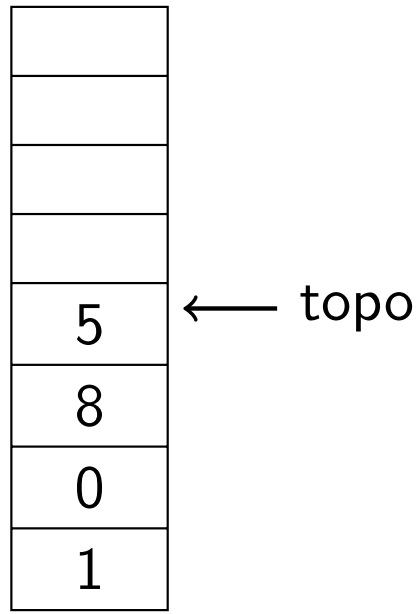

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

Estrutura de Dados

- modo como armazenamos e manipulamos as informações dentro dos nossos algoritmos
- na prática consiste nas variáveis e nas funções que acessam essas variáveis
- ou trazendo para o paradigma orientado a objetos, consiste nos atributos e métodos de uma ou mais classes
- as estruturas de dados que apresentaremos são a **pilha** e a **fila** que são exemplos de **listas lineares**
- basicamente elas diferem no modo como as informações são inseridas e recuperadas dessas listas

Pilha (Stack) — Introdução

- LIFO - Last In First Out
- todas as inserções e remoções são feitas em uma das extremidades da lista
- no caso da pilha, denominamos essa extremidade de topo
- podemos imaginar que a pilha dentro da programação funciona igual a uma pilha de pratos do restaurante
- usaremos um vetor para representar os elementos da pilha
- ao lado temos a representação gráfica da pilha



Pilha — Operações

Uma pilha possui as seguintes operações (métodos e construtor):

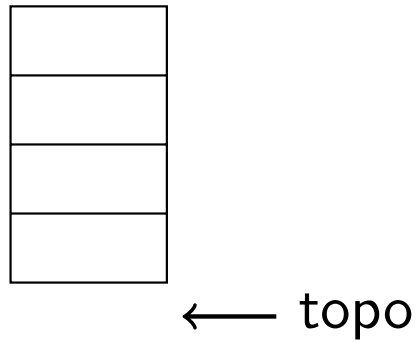
- `Pilha(n)`: instancia uma pilha com capacidade para armazenar *n* informações
- `boolean isCheia()`: retorna true se a pilha está cheia
- `boolean isVazia()`: retorna true se a pilha está vazia
- `void empilha(<info>)`: coloca info na pilha
- `<info> desempilha()`: devolve info da pilha removendo-a da pilha
- `<info> topo()`: devolve a info da pilha sem removê-la

, onde `<info>` representa o **tipo** de dado armazenado na pilha.

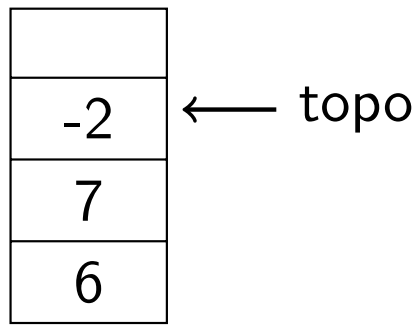
Antes de mostrarmos a implementação de uma Pilha, vamos mostrar como fica o desenho da pilha após a execução de algumas instruções:

Pilha — Teste de Mesa

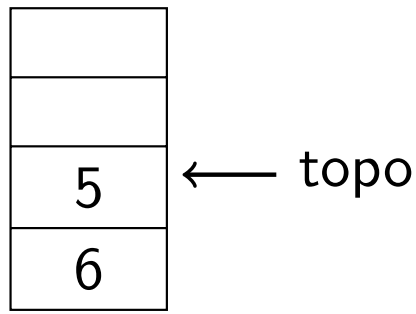
```
1  Pilha p = new Pilha(4);
```




```
1  p.empilha(6);
2  p.empilha(7);
3  p.empilha(-2)
```

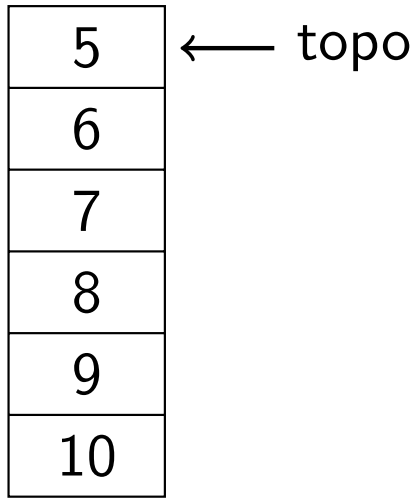



```
1  p.desempilha();
2  p.desempilha();
3  p.empilha(5);
```



Pilha — Teste de Mesa 2

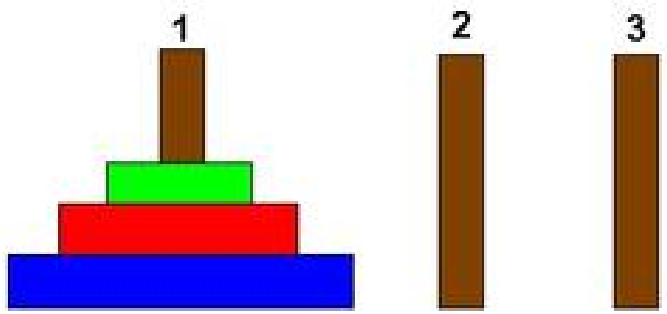
```
1  Pilha p = new Pilha(6);
2  int i = 10;
3  while (!p.isCheia()) {
4     p.empilha(i);
5     i--;
6  }
```



Pilha — Torres de Hanoi

Exemplo 1: Torre de Hanoi é um jogo que consiste de 3 pinos representando as torres e n discos de diâmetros diferentes colocados em um dos pinos. Veja a figura abaixo representando as torres de Hanoi com 3 discos ($n = 3$). O objetivo do jogo é movimentar todos os discos do pino A para o pino C obedecendo duas regras:

- apenas um disco pode ser movimentado por vez
- um disco de diâmetro menor nunca pode ficar sobre um disco de diâmetro maior



Pilha — Sequência de símbolos

Exemplo 2: A maioria dos compiladores consegue perceber quando uma sequência de símbolos: (), [] e está bem formada ou não
Seguem abaixo alguns exemplos de sequências:

- `[[({}){}[{}]]]` *ok*
- `[()])` *não ok*
- `{{{)}}}` *não ok*
- `[[(])]` *não ok*
- `()()[{}]` *ok*
- `)` *não ok*

O problema consiste em você receber uma sequência de símbolos e decidir se essa sequência é bem formada ou não

Pilha — Sequência de símbolos (continuação)

- Um dos modos de resolver o problema é classificar os símbolos entre aqueles que são de abertura e os que são de fechamento
- Sempre um símbolo de abertura vai para a pilha
- Quando é lido um símbolo de fechamento, desempilha a pilha e verifica se os símbolos "casam"ou "não casam"
- Note que não deve sobrar nenhum símbolo na pilha ao fim da sequência de símbolos

Vamos executar o algoritmo acima no papel desenhando a pilha e colocando os símbolos dentro dela.

Pilha — Implementação

Ambos os exemplos são facilmente resolvidos usando pilhas, no caso das Torres de Hanoi três pilhas para ser mais exato.
Vejamos abaixo um exemplo de implementação de uma pilha de caracteres para resolver o problema da sequência de símbolos.

```
1  public class Pilha {
2
3    private int topo;
4    private char[] lista;
5
6    public Pilha(int tamanho) {
7      lista = new char[tamanho];
8      topo = -1;
9    }
10
11   public char topo() {
12     return lista[topo];
13   }
```

Pilha — Implementação

```
14    public char desempilha() {
15      return lista[topo--];
16    }
17
18    public void empilha(char info) {
19          topo++;
20          lista[topo] = info;
21    }
22
23    public boolean isCheia() {
24       if (topo == lista.length - 1) return true;
25       else return false;
26    }
27
28    public boolean isVazia() {
29      if (topo == -1) return true;
30      else return false;
31    }
32  }
```

Pilha — Genérica

A implementação da classe Pilha anterior somente funciona para armazenar dados do tipo caracter. Podemos usar o recurso de *Generics* do Java para montar uma pilha capaz de armazenar qualquer tipo de objetos.

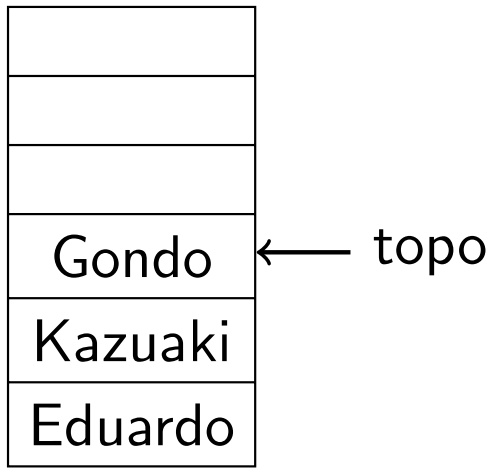
```
1  public class Pilha<T> {
2
3     private int topo;
4     private Object[] lista;
5
6     public Pilha(int tamanho) {
7           lista = new Object[tamanho];
8           topo = -1;
9     }
10
11    public T topo() {
12          return (T)lista[topo];
13    }
```

Pilha —Genérica

```
14      public T desempilha() {
15          return (T)lista[topo--];
16      }
17
18      public void empilha(T info) {
19          topo++;
20          lista[topo] = info;
21      }
22
23      public boolean isCheia() {
24          if (topo == lista.length - 1)  return true;
25          else return false;
26      }
27
28      public boolean isVazia() {
29          if (topo == -1) return true;
30          else    return false;
31      }
32  }
```

Pilha — Teste da pilha genérica

```
1  public static void main(String[] args) {
2      Pilha<String> pilha = new Pilha<>(6);
3      pilha.empilha("Eduardo");
4      pilha.empilha("Kazuaki");
5      pilha.empilha("Gondo");
6
7      while (!pilha.isVazia()) {
8          String s = pilha.desempilha();
9          System.out.println(s);
10     }
11 }
```



O segredo para criar um repositório genérico está na declaração da classe colocando a partícula <T> após o nome da classe, isso indica que estamos criando um objeto cujo conteúdo armazenado será definido no momento da instanciação do objeto.

Exercícios

1) Instancie um objeto pilha com capacidade para armazenar 20 elementos e preencha totalmente a pilha com números aleatórios. Para este exercício está proibido a utilização do comando `for`.
2) Escreva um programa que verifica se uma sequência de símbolos estã bem formada. Seu programa recebe uma String contendo uma sequência de símbolos (), {} e []; e imprime na tela se a sequência está bem formada ou não
3) Simule o algoritmo que resolve uma expressão polonesa quando a entrada do algoritmo são as seguintes expressões:
   a) $30\ \ 54 - 6/$
   b) $23\ \ 39 + 45\ \ 55 + 21 - *$
   c) $12\ \ 3 * 56 - 5*$

Exercícios

4) Considere que você recebe na ordem uma sequência de números: 1, 2, 3, 4 e 5. Descreva a sequência de operações de empilhamento e desempilhamento para que os elementos da pilha possam ser impressos nas ordens definidas abaixo:
   a) 3 4 2 5 1
   b) 1 4 3 5 2
   c) 2 1 4 3 5
   d) 4 3 5 2 1
   e) 3 2 5 4 1
5) Implemente um programa para simular o jogo das Torres de Hanói. Note que você deverá criar 3 pilhas para representar as hastes do jogo. Utilize uma boa representação para indicar as mudanças de discos de uma haste para outra. Além disso, imprima mensagens dizendo se o usuário pode ou não pode mover um disco de uma haste para outra.

Exercícios

6) Escreva um algoritmo que recebe um vetor de String representando uma expressão aritmética na notação polonesa e retorna o resultado se existir ou uma mensagem indicando que a expressão é inválida. Note que a expressão possui apenas as quatro operações aritméticas básicas.
7) Dada uma frase você deverá inverter as palavras de dentro da frase mas mantendo sua ordem. Exemplo: "ontem choveu mas hoje fez sol". Deverá ficar como: "metno uevohc sam ejoh zef los". Escreva um método que recebe uma String como parâmetro e devolve uma outra String com a frase invertida conforme exemplificado acima. Utilize, obrigatoriamente, uma pilha como variável auxiliar.
8) Escreva um método que recebe um vetor de inteiros e usando 2 pilhas como variáveis auxiliares ordena crescentemente o vetor.

Referência Bibliográfica

- **Fundamentos da Programação de Computadores: algoritmos, Pascal, C/C++ e Java**, Ascencio e Campos - 2ª ed., Pearson 2007
- **Lógica de Programação e Estrutura de Dados**, Puga e Rissetti - 2ª ed., Pearson Prentice Hall, 2008.
- **Algoritmos em linguagem C**, Feofiloff - Campus/Elsevier, 2009 (`http://www.ime.usp.br/~pf/algoritmos-livro/index.html`)
- **Construção de Algoritmos e Estruturas de Dados**, Forbellone e Eberspacher - Pearson Prentice Hall, 2010.
- **Projeto de Algoritmos com Implementações com Java e C++**, Ziviani - Thompson, 2006
- **Java como Programar**, Deitel e Deitel - 8ª ed., Pearson, 2010
- **Algoritmos**, Cormen, Leiserson, Rivest e Stein - Campus

Copyleft