ALGORITMOS E LÓGICA DE PROGRAMAÇÃO II

Prof. Wilson Lourenço

wilson.slourenco@sp.senac.br

PILHAS

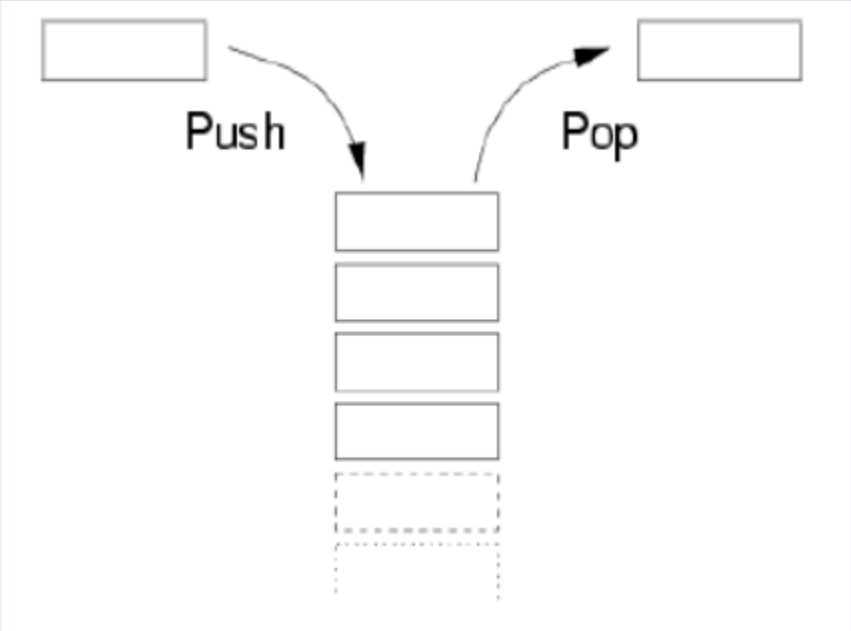
PILHAS

- Lista LIFO (Last In, First Out);

- Os elementos são colocados na estrutura (pilha) e retirados em ordem inversa a de sua chegada

    1. Inserção apenas no topo da pilha
    2. Remoção apenas no topo da pilha

PILHAS



PILHAS

- Operações possíveis:

    1. Esvaziar a pilha;
    2. Verificar se a pilha está vazia;
    3. Colocar um dado no topo da pilha (empilhar);
    4. Retirar o dado do topo da pilha (desempilhar);
    5. Etc.

PILHAS

- Implementação de pilhas
  - Usando vetores
    1. Pode-se implementar uma pilha de tamanho fixo usando vetores.
    2. Este tamanho determinará o número máximo de elementos que poderão estar na pilha ao mesmo tempo
    3. É necessário um inteiro para armazenar o valor da posição do vetor onde se encontra o topo da pilha

PILHAS

- Usando uma lista ligada

    1. A implementação de uma pilha que usa como estrutura básica uma lista ligada é mais simples, pois a lista não é uma estrutura de tamanho fixo;
    2. Basicamente, os dados devem ser colocados (empilhados) em alguma das extremidades da lista e retirados (desempilhados) a partir desta mesma extremidade.

PILHAS

- É ideal para processamento de estruturas aninhadas de profundidade imprevisível:

- Uma pilha contém uma sequência de obrigações adiadas.

- A ordem de remoção garante que os dados mais internos serão processados depois dos mais externos

Ponteiros

- Os ponteiros são variáveis especiais que apontam para uma área de memória.
- Os ponteiros não possuem as informações em si, mas dizem onde a informação está.
- Podemos, em um nível mais alto, fazer um comparativo com páginas web.

- Quando criamos uma página, adicionamos um conteúdo nela, informações, imagens, etc. Contudo, não podemos colocar tudo numa página só e nem fica bom para se ler, então podemos separar nosso conteúdo por tópicos e para cada um fazemos uma página diferente.

- Para navegar de uma página a outra, criamos os links.

- Esses links possuem o endereço web da outra página que queremos acessar e naquela outra página é que está o conteúdo.

- Podemos ver os ponteiros, mais ou menos como esses links, ou seja, os ponteiros não possuem a informação, mas o endereço de onde está a informação.

- Se você clicar em um link e a página do endereço do link não existir, teremos como resultado um erro 404.

- Da mesma forma, teremos um erro se tentarmos acessar o conteúdo de um ponteiro que não foi alocado (nulo).

- Para podermos acessar o conteúdo do ponteiro, precisamos apontar ele para uma área da memória que já existe ou alocar espaço usando new e apontar para essa área recém criada.

Variáveis por Valores

Considere a classe Ponto:

```
public class Ponto {
	double x;
	double y;
}
```

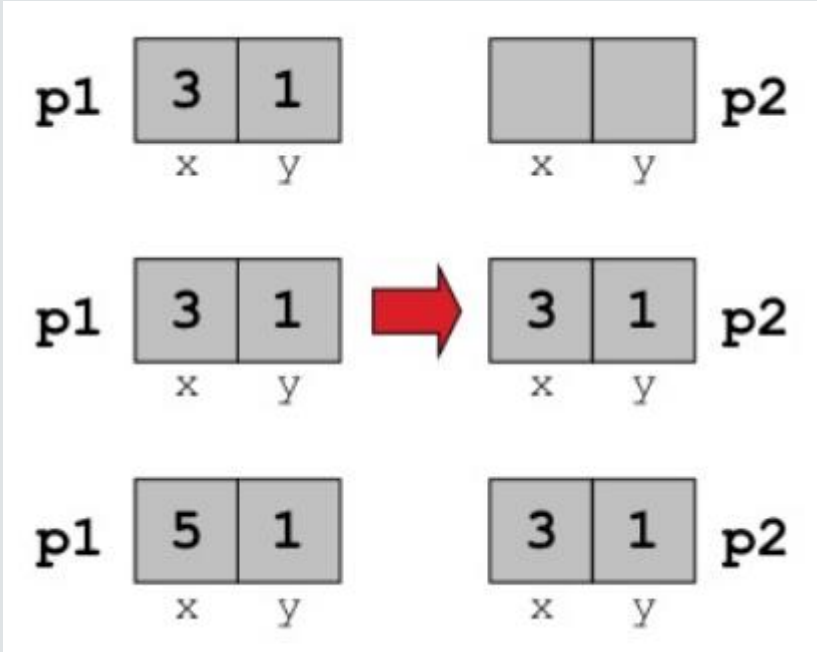
Atribuir uma variável de valor a outra apenas copia seu conteúdo atual. Uma modificação em uma delas não afeta a outra variável.

Ponto p1, p2;

p1.x = 3;
p1.y = 1;

p2.x = p1.x;
p2.y = p1.y;

p1.x = 5;



Variáveis por Referência

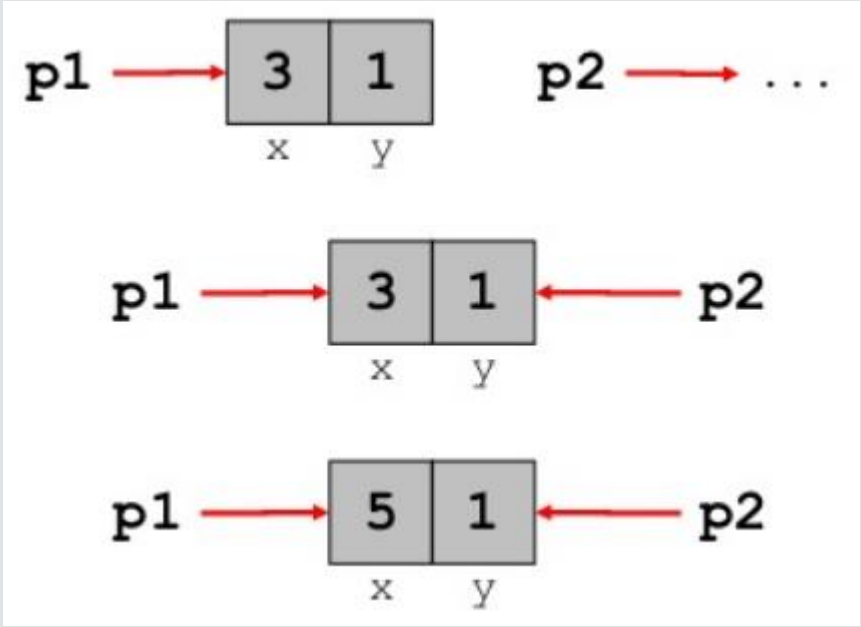
Atribuir uma referência faz com que ambas apontem para o mesmo objeto. Qualquer alteração afeta as duas variáveis.

Ponto p1, p2;

p1.x = 3;
p1.y = 1;

p2 = p1;

p1.x = 5;



Referências em Java

Em java, todos os tipos primitivos são representados por valores.

```
char c;
int x, y;
c = 'a';
x = y = 3;
```

Os tipos compostos (classes), por outro lado, utilizam referências.

```
String s;
Ponto p;
Integer n;
Vector v;

s = “SENAC";

p = new Ponto(3, 1);
p.x = x;
```

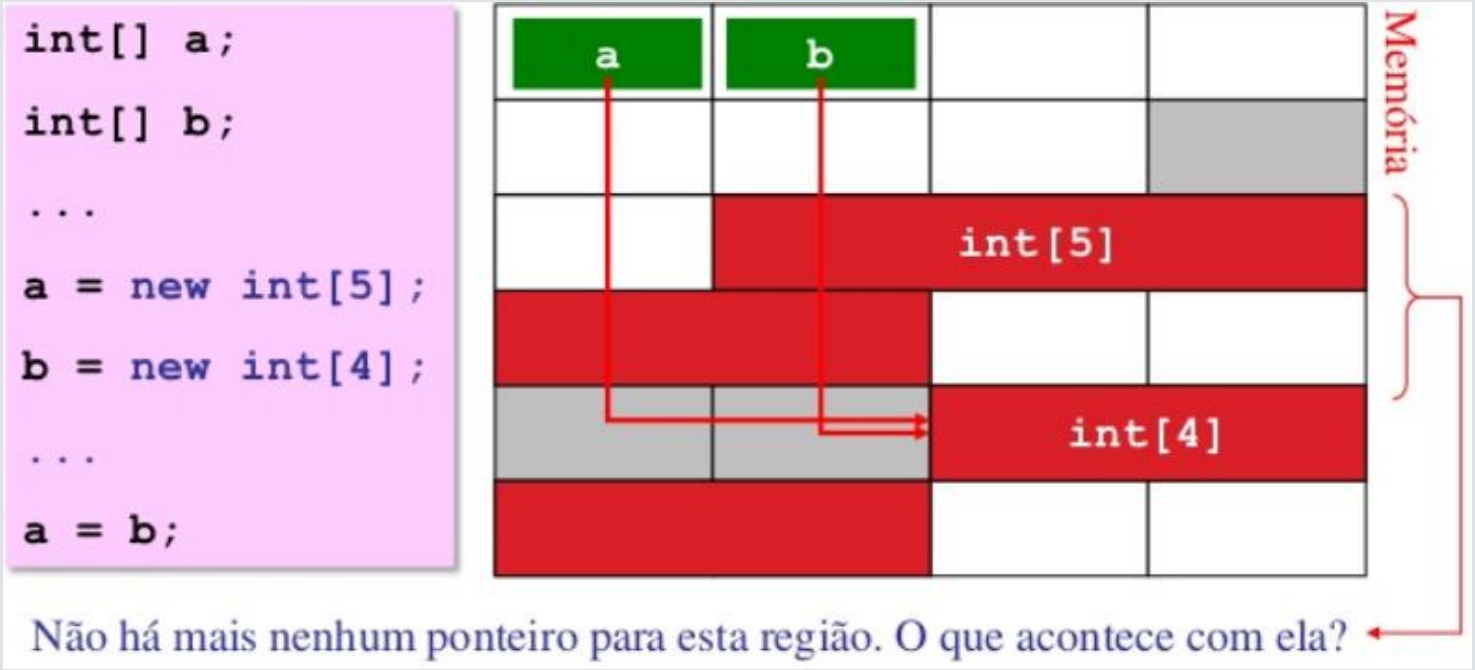
n = new Integer(3);

v = null;

A memória só é requisitada no momento em que o objeto é necessário, evitando o desperdício.

Ponto p = new Ponto(20, 34);

A memória é alocada para o novo objeto e sua posição armazenada em uma referência (ponteiro):



*Em Java não é necessário, pois o garbage collector libera a memória automaticamente.

EXEMPLO DE PILHA 1 (Ex_Pilha_01.java)

```
public class Ex_Pilha_01 {

    static int pilha[] = new int[10];
    static int pos=0;  // topo da pilha

    public static void main(String args[]) {

        System.out.println("\nColoca dados na pilha!");

        push(4);
        System.out.println("\nValor adicionado na pilha: " + 4);

        push(9);
        System.out.println("\nValor adicionado na pilha: " + 9);
```

```
    push(3);
    System.out.println("\nValor adicionado na pilha: " + 3);

    System.out.println("\n\nTamanho da pilha " + size());

    System.out.println("\nPegando dado da pilha: " + pop());
    System.out.println("\nPegando dado da pilha: " + pop());
    System.out.println("\nPegando dado da pilha: " + pop());

    System.out.println("\n\nTamanho da pilha " + size());
}

static void push(int valor) {
  pilha[pos]=valor;
  /* Empilha um novo elemento no topo da pilha.
    Repare que não é verificada a capacidade máxima da pilha.*/
  pos++;  // Atualiza o topo da pilha
}
```

```
    static int pop() {
        /* Retorna o elemento do topo da ilha.
        Não é verificado o final da pilha. */
       return (pilha[--pos]);
    }
    static int size() {
      return pos; /* retorna o topo da pilha */
    }
    static int stacktop() /* retorna o elemento do topo da pilha
                          sem desempilhar */
    {
           return pilha[pos];
    }
}
```

EXEMPLO DE PILHA 2 (Ex_Pilha_02.java)

```
public class Ex_Pilha_02 {

    public Object[] pilha;
    public int posicaoPilha;

    public Ex_Pilha_02() {

        // indica que esta nula, vazia, pois posição um indica
        //que contém informação
        this.posicaoPilha = -1;

        // criando uma pilha com 10 posições
        this.pilha = new Object[10];
    }
```

```
public static void main(String args[]) {

    Ex_Pilha_02 p = new Ex_Pilha_02();

    p.empilhar("Portuguesa ");
    p.empilhar("Frango com catupiry ");
    p.empilhar("Calabresa ");
    p.empilhar("Quatro queijos ");
    p.empilhar(10);

    while (p.pilhaVazia() == false) {
        System.out.println(p.desempilhar());
    }
}
```

```
public boolean pilhaVazia() {
    //isEmpty
    if (this.posicaoPilha == -1) {
        return true;
    }
    return false;
}

public int tamanho() {
    //is
    if (this.pilhaVazia()) {
        return 0;
    }
    return this.posicaoPilha + 1;
}
```

```
public int tamanho() {
    //is
    if (this.pilhaVazia()) {
        return 0;
    }
    return this.posicaoPilha + 1;
}

public Object exibeUltimoValor() {
    //top
    if (this.pilhaVazia()) {
        return null;
    }
    return this.pilha[this.posicaoPilha];
}
```

```
    public Object desempilhar() {
        //pop
        if (pilhaVazia()) {
            return null;
        }
        return this.pilha[this.posicaoPilha--];
    }

    public void empilhar(Object valor) {
        // push
        if (this.posicaoPilha < this.pilha.length - 1) {
            this.pilha[++posicaoPilha] = valor;
        }
    }
}
```

EXEMPLO DE PILHA 3 (Ex_Pilha_03.java)

Primeira classe!!!!

```
import java.util.Scanner;

public class Ex_Pilha_03 {

   public static void main(String args[]) {

    Scanner entrada = new Scanner(System.in);

    System.out.println("Entre com um número:");
    int valor = entrada.nextInt();
```

```
    Pilha novaPilha = new Pilha();

    novaPilha.push(valor);
    novaPilha.push(70);
    novaPilha.push(88);

    System.out.println("Item de valor " + novaPilha.pop() + " retirado da pilha");
    System.out.println("Item restante: " + novaPilha.peek());
    System.out.println("Item de valor " + novaPilha.pop() + " retirado da pilha");
    System.out.println("Item restante: " + novaPilha.peek());
  }
}
```

Segunda classe!!!

```
public class Pilha {

   static final int MAX = 10;
   int top;
   int a[] = new int[MAX]; // Define tamanho máximo da pilha

   // Construtor
   Pilha() {
      top = -1;
   }

   // Métodos da pilha
   boolean isEmpty() {
     return (top < 0);
   }
```

```
boolean push(int x) {
      if (top >= (MAX-1)) {
         System.out.println("Estouro de Pilha!");
         return false;
      }
      else {
         a[++top] = x;
         return true;
      }
   }
```

```
int pop() {
      if (top < 0) {
         System.out.println("Pilha Vazia!");
         return 0;
      }
      else {
         int x = a[top--];
         return x;
      }
   }
```

```
int peek() {
      if (top < 0) {
         System.out.println("Pilha Vazia!");
         return 0;
      }
      else {
         return a[top];
      }
   }
}
```

EXEMPLO DE PILHA 4 (Ex_Pilha_04.c)

Primeira classe!!!!!!!!!!!!

```
public class Ex_Pilha_04 {

    public static void main(String[] args) {

        String[] names = {
                "Mark", "Berg", "John", "Beni", "Jebb", "June",
                "Mary", "Karl", "Fred", "Hall", "Troy", "Joan"
        };

        filaSimples stack = new filaSimples(10);

        System.out.println(
                "Pilha de " + stack.getCapacity() + " posições criada: " + stack
        );
        System.out.println();
```

```
System.out.println("Preenchendo a pilha:");

for(int i = 0; i < names.length; i++) {

    System.out.print("\tInserindo o nome \"" + names[i] + "\":\t");

    if(stack.push(names[i]) == null)
        System.out.println("PILHA CHEIA!!! impossível inserir...");
    else
        System.out.println(
                stack + ". " + (stack.getCapacity() - stack.getSize()) +
                " posições restantes."
        );
}
System.out.println();

System.out.println("Removendo 5 elementos da pilha:");
```

```
for(int i = 1; i <= 5; i++) {
        System.out.print("\t" + i + "a. remoção: \"" + stack.pop() + "\".");
        System.out.println(" A pilha agora esta assim: " + stack);
}
System.out.println();

System.out.println(
    "O atual nome no topo da pilha é \"" + stack.peek() + "\".\n"
);

System.out.println(stack);

System.out.println();
```

```
        stack.clear();

        System.out.println("Limpando a pilha: " + stack);

        System.out.println();

        System.out.print("Consigo tirar mais algo da pilha? ");
        System.out.println(
                stack.pop() == null ? "Não consigo..." : "Consigo sim!"
        );
    }
}
```

Segunda classe!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```
public class filaSimples {
        /**
         * Vetor de String que armazenará os elementos.
         * Se uma posição estiver nula, esta posição será encarada
         * como uma posição vazia na pilha. Sendo assim, nunca poderá
         * existir uma posição vazia seguida de uma posição
         * não-vazia(diferente de null), pois isso indicaria que, de
         * alguma forma, removemos algum elemento da pilha que não o
         * do topo.
         */
        private String[] elements;
        /**
         * inteiro que indicará quantos elementos NÃO NULOS existem
         * dentro de elements. Note por exemplo que:
         * -Se size for igual a 26, o topo da pilha está na posição
         *  25 de elements
```

```
 * -Se size for igual a 10, o topo da pilha está na posição 9
 *  de elements
 * -Se size for igual a n, o topo da pilha está na posição(n - 1)
 *  de elements
 * Resumindo, podemos assumir QUASE sempre que o topo da pilha é
 * a posição(size - 1) de elements.
 * O "quase" se deve ao caso de quando nossa pilha é vazia.
 * Nesse caso, size é igual a 0. Se fossemos seguir a regra
 * acima indistintamente, acabaríamos assumindo que neste caso
 * o topo da pilha está na posição -1 de elements, o que é um
 * erro... Trata-se então de uma exceção, a qual deveremos estar
 * atentos.
 * Para fins didáticos, criaremos um método privado que retorna
 * a posição exata do topo da pilha, e sempre utilizaremos este
 * método para termos tal informação.
 */
private int size;
```

```
/**
 * Construtor que recebe um int como parâmetro, indicando qual
 * será a capacidade da pilha recém-instanciada
 */
public filaSimples(int capacity) {
/*
* Utilizamos o método abs da classe Math só pra evitar que se
    tente criar uma pilha "devedora"...
*/
        elements = new String[Math.abs(capacity)];
        size = 0;
}

/**
 * Método utilizado para adicionar elementos à nossa pilha.
 * Este elemento sempre será inserido no topo da pilha.
 * -Se o elemento for null, vamos lançar uma exceção, informando
```

```
         * que nossa pilha não aceita null como elemento válido.
         * -Se a pilha estiver cheia, retornamos null para indicar que
         * não foi possível inserir elemento.
         * -Se o elemento for inserido com sucesso, retornamos o próprio
         * elemento, indicando assim o sucesso da operação
         */
        public String push(String element) {
            if(element == null)
                throw new IllegalArgumentException("O elemento não pode ser
nulo!");
            if(size == elements.length)
                    return null;
            size++;
            elements[getTopPosition()] = element;
            return element;
        }
```

```
/**
 * Método utilizado para se obter o elemento que está no topo
 * desta pilha, porém, sem removê-lo da mesma.
 * -Se a pilha estiver vazia, retornamos null para indicar que
 * a pilha está vazia.
 * -Se houver ao menos um elemento na pilha, o elemento que está
 * no topo será retornado, indicando o sucesso da operação
 */
public String peek() {
    if(isEmpty())
            return null;
    return elements[getTopPosition()];
}
```

```
/**
 * Método utilizado para retirar("destacar") um elemento da
 * pilha.
 * Este elemento sempre será aquele que se encontra no topo da
 * pilha.
 * -Se a pilha estiver vazia, retornamos null para indicar que a
 * pilha está vazia.
 * -Se houver ao menos um elemento na pilha, o elemento que está
 * no topo será retornado, indicando o sucesso da operação
 */
public String pop() {
    String result = peek();
    /*Se havia um elemento no topo da pilha...*/
    if(result != null) {
        elements[getTopPosition()] = null;
        size--;
    }
    return result;
}
```

```
	/**
	 * Método utilizado para limpar todo o conteúdo da pilha.
	 */
	public void clear() {
		for(int i = 0; i < size; i++)
			elements[i] = null;
		size = 0;
	}

	/**
	 * Método utilizado para se obter o tamanho (número de elementos) da pilha
	 */
	public int getSize() {
		return size;
	}
```

```
/**
 * Método utilizado para se obter a capacidade da pilha
 */
public int getCapacity() {
        return elements.length;
}

/**
 * Método utilizado para verificar se a pilha está vazia.
 * Se for o caso, será retornado true, caso contrário, será
 * retornado false.
 */
public boolean isEmpty() {
    return size <= 0;
}
```

```
/**
 * Este método tem uma finalidade estritamente didática, visando
 * facilitar o entendimento do código desta classe.
 * Este método retorna um inteiro que representa a posição de
 * elements onde se encontra o último elemento inserido nesta
 * pilha (O topo da pilha)
 */
private int getTopPosition() {
    if (isEmpty())
        return 0;
    return size - 1;
}
```

```
	/**
	 * Este método serve para representar textualmente esta pilha
	 * @see java.lang.Object#toString()
	 */
	@Override
	public String toString() {
		StringBuilder sb = new StringBuilder("[");
		for(int i = 0; i < size; i++) {
			sb.append(elements[i]);
			if(i < size - 1)
			  sb.append(" | ");
		}
		sb.append("]");

		return sb.toString();
	}
}
```

Vídeo:

https://www.youtube.com/watch?v=2V91Re1czwA

Dicas para Estudo

Seja “CURIOSO”:

- Procure revisar o que foi estudado.
- Pesquise as referências bibliográficas.

Seja “ANTENADO”:

- Leia a próxima aula.

Seja “COLABORATIVO”:

- Traga assuntos relevantes para a sala de aula.
- Participe da aula.
- Proponha discussões relevantes sobre o conteúdo.

Prof. Me. Wilson Lourenço


Dúvidas?
Não mais..