

Aula 02: 17/03/2017

Leitura: [Pilhas, PF](#)

Pilhas

Resumo:

- pilha e sua API
- implementações em vetor
- pilhas genéricas
- redimensionamento
- implementação em lista ligada

Pré-requisitos:

- vetores
- listas ligadas
- ADT, API, cliente, interface, implementação

API

```
public class Stack<Item>

Stack() construtor // cria uma pilha de Items vazia
void    push(Item item) // insere item nesta pilha
Item    pop() // remove o Item mais recente desta pilha
boolean isEmpty() // esta pilha está vazia?
int     size() // número de Items nesta pilha
```

Pilhas em vetores

Pilha de strings

```
public class StackOfString {
    private String[] a = null;
    private int n = 0;

    public StackOfString(int cap) {
        a = new String[cap];
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
```

```

        return n;
    }

    public void push(String item) {
        a[n++] = item;
    }

    public String pop() {
        return a[--n];
    }

    \\ unit test
    public static void main(String[] args) {

    }
}

```

Cliente

```

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class ClientStackString {

    public static void main(String[] args) {
        StackOfString pilha;
        pilha = new StackOfString(20);

        while (!StdIn.isEmpty()) {
            String str = StdIn.readString();
            if (!str.equals("-"))
                pilha.push(str);
            else if (!pilha.isEmpty())
                StdOut.println(pilha.pop() + " ");
        }
        StdOut.println("(" + pilha.size() + " left on stack)");
    }
}

```

Pilha de inteiros

```

public class StackOfInteger {
    private int[] a = null;
    private int n = 0;

    public StackOfInteger(int cap) {
        a = new int[cap];
    }
}

```

```

public boolean isEmpty() {
    return n == 0;
}

public int size() {
    return n;
}

public void push(int item) {
    a[n++] = item;
}

public String pop() {
    return a[--n];
}

\\ unit test
public static void main(String[] args) {

}
}

```

Pilhas genéricas

Item é uma tipo genérico, ou parâmetro de tipo, que deve ser substituído por um tipo concreto quando uma instância da pilha é criada.

Note o *casting* `a = (Item[]) new Object[1];` e `a = (Item[]) new Object[cap];`

```

public class Stack<Item> {

    private Item[] a = null;
    private int n = 0;

    public Stack(int cap) {
        a = (Item[]) new Object[cap];
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public int size() {
        return n;
    }

    public void push(Item item) {
        a[n++] = item;
    }
}

```

```

    public Item pop() {
        return a[--n];
    }
}

```

Cliente

```

import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.StdOut;

public class ClientStackString {

    public static void main(String[] args) {
        Stack<String> pilha;
        pilha = new Stack<String>(20);

        while (!StdIn.isEmpty()) {
            String str = StdIn.readString();
            if (!str.equals("-"))
                pilha.push(str);
            else if (!pilha.isEmpty())
                StdOut.println(pilha.pop() + " ");
        }
        StdOut.println("(" + pilha.size() + " left on stack");
    }
}

```

Pilhas com redimensionamento

Pilha implementada em vetor com redimensionamento (resizing array).

`resize()`: método privado que faz um redimensionamento: aumenta ou diminui o vetor que abriga a pilha.

Depois do redimensionamento, não é necessário liberar o espaço ocupado pelo antigo vetor pois o mecanismo de coleta de lixo do Java cuida disso automaticamente.

```

public class Stack<Item> {
    private Item[] a = null;
    private int n = 0;

    public Stack() {
        a = (Item[]) new Object[1];
        n = 0;
    }

    public boolean isEmpty() {
        return n == 0;
    }
}

```

```

public int size() {
    return n;
}

public void push(Item item) {
    if (n == a.length) resize(2*a.length);
    a[n++] = item;
}

public Item pop() {
    Item item = a[--n];
    a[n] = null; // Avoid loitering
    if (n > 0 && n == a.length/4) resize(a.length/2);
    return item;
}

private void resize(int max) {
    Item[] tmp = (Item[]) new Object[max];
    for (int i = 0; i < n; i++) {
        tmp[i] = a[n-i-1];
    }
    a = tmp;
}
}

```

Loitering object: objeto ocioso.

Para evitar a presença de objetos ociosos, devemos atribuir null para avisar o coletor de lixo que a memória não é mais necessária.

Pilha em lista ligada

```

public class Stack<Item> {

    private Node first;

    private class Node {
        Item item;
        Node next;
    }

    public boolean isEmpty() {
        return first == null;
    }

    public void push(Item item) {
        Node oldfirst = first;
        first = new Node();
        first.item = item;
        first.next = oldfirst;
    }
}

```

```
}

public Item pop() {
    Item item = first.item;
    first = first.next;
    return item;
}

public static void main(String[] args) {
    StackL<String> s = new StackL<String>();
    while (!StdIn.isEmpty()) {
        String item = StdIn.readString();
        if (!item.equals("-")) s.push(item);
        else if (!s.isEmpty()) StdOut.print(s.pop() + " ");
    }
    StdOut.println("(" + s.size() + " left on stack)");
}
}
```