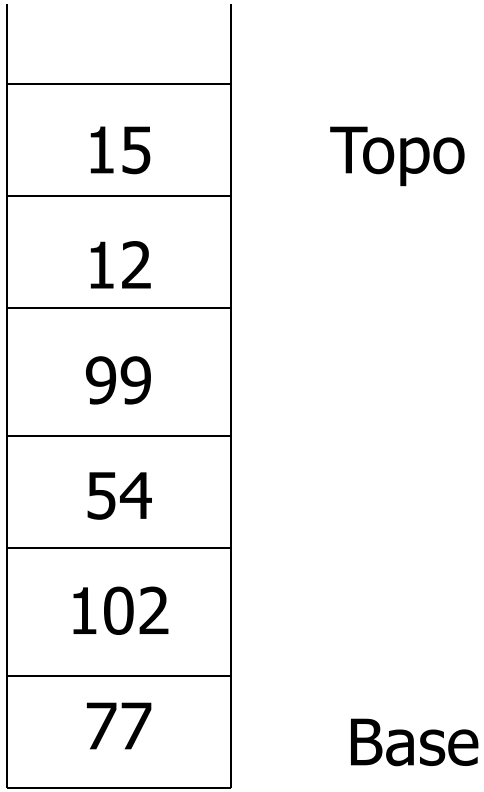
Pilhas Dinâmicas

Prof. Leandro Colevati

Introdução

- Definição
  - É uma estrutura de dados de tamanho variável, sendo que elementos são incluídos (empilhados) e/ou removidos (desempilhados) apenas pela extremidade topo.

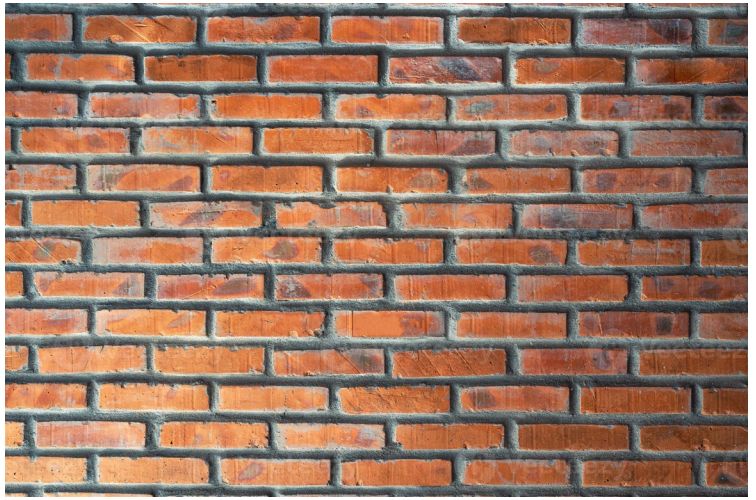


Introdução

- Note que uma pilha é uma estrutura de dados do tipo LIFO (Last In First Out).

- Isto porquê o primeiro elemento empilhado é sempre o último a ser desempilhado.

Introdução



Introdução

- Operações Básicas:
  - Teste de pilha vazia
  - Criação da pilha
  - Empilhamento
  - Desempilhamento
  - Acesso aos elementos da pilha
    - Topo
    - Tamanho

Simular operações

Push(1)
Push(2)
Push(10)
Pop()
Top()
Push(5)
Push(8)
Pop()
Pop()
Top()
Size()

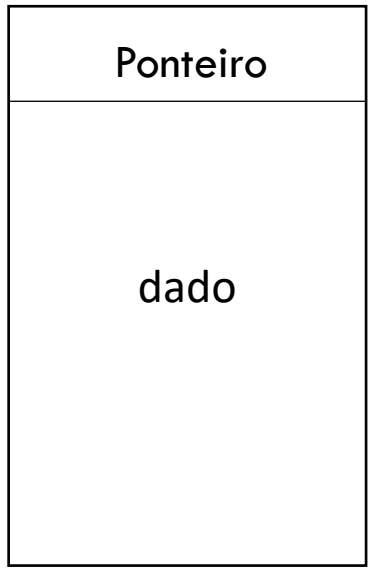
Introdução

- Do ponto de vista da alocação de memória para esse tipo de estrutura de dados, podem ser implementadas usando:
  - Alocação Estática: Em geral através de arranjo ou vetor;
  - Alocação Dinâmica: Utilizando ponteiro (Implícito ou Explícito).

Alocação Dinâmica

- Considere a definição do tipo Pilha abaixo:

```
class No {
   tipo    dado;
   No      próximo; //Ponteiro
}
```



Alocação Dinâmica

- Teste de pilha vazia:

```
No topo;
booleano pilhaVazia() {
    se (topo == nulo) {
        retorne verdadeiro;
    } senão {
        retorne falso;
    }
}
```

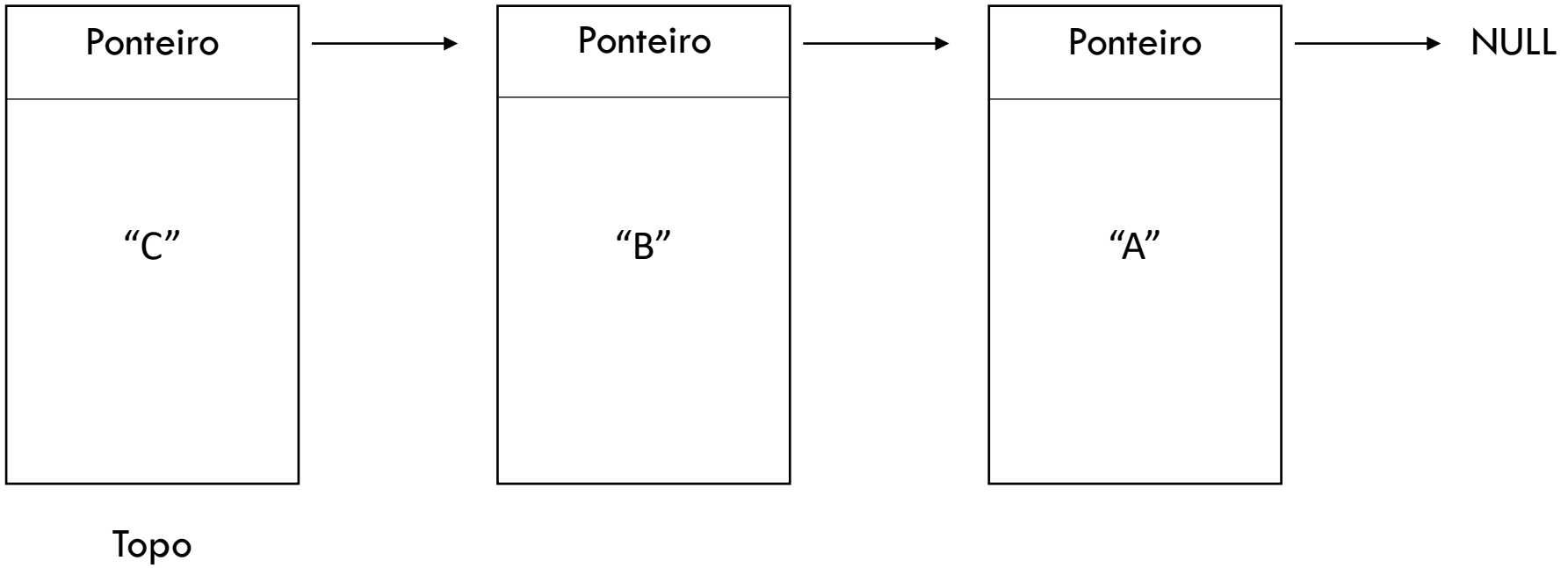
Alocação Dinâmica

- Empilhando um elemento (Push):

```
No topo;
void push (tipo e) {
    No elemento = new No();
    elemento.dado = e;
    se (pilhaVazia() == verdadeiro) {
        topo = elemento;
    } senão {
        elemento.proximo = topo;
        topo = elemento;
    }
}
```

Alocação Dinâmica

- Empilhando um elemento (Push):



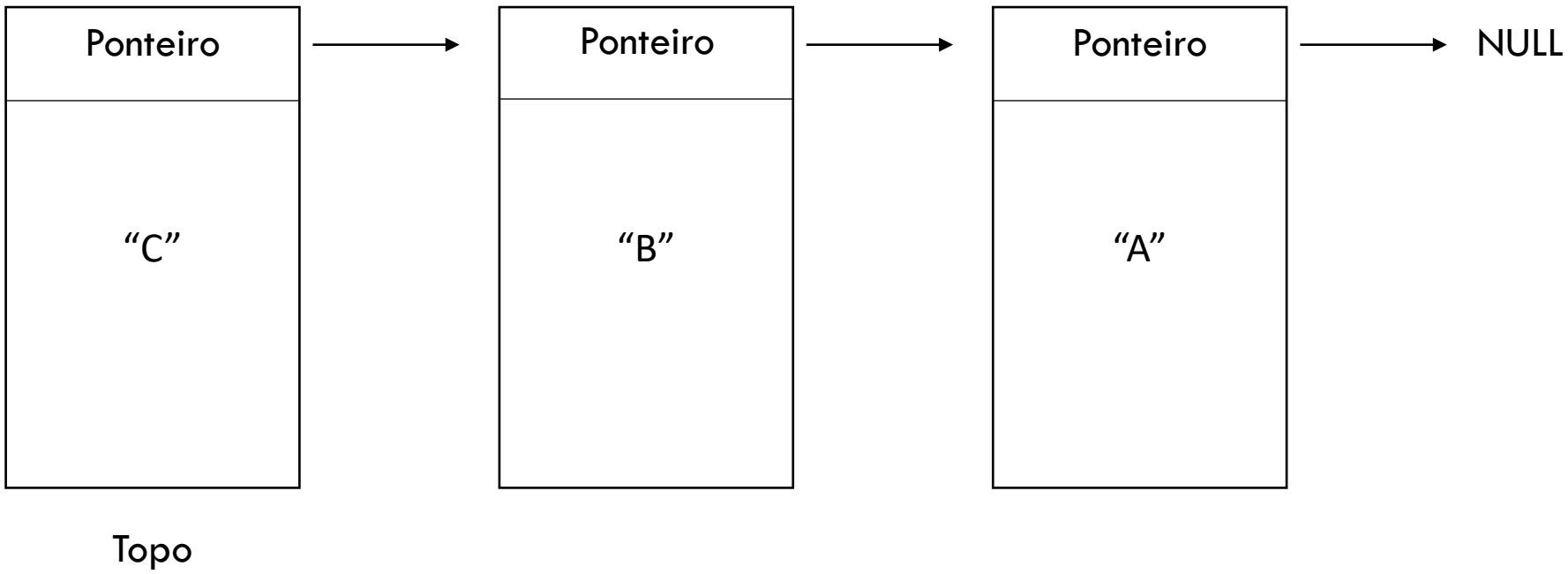
Alocação Dinâmica

- Desempilhando um elemento (Pop):

```
No topo;
tipo pop() {
    se (pilhaVazia() == verdadeiro) {
            exceção("Não há elementos para desempilhar");
    }
    tipo valor = topo.dado;
    topo = topo.proximo;
    retorne valor;
}
```

Alocação Dinâmica

- Desempilhando um elemento (Pop):



Alocação Dinâmica

- Acessando elementos da pilha
  - Como estamos usando uma lista simplesmente encadeada podemos acessar todos os elementos da pilha, a partir do topo, sem ter a necessidade de desempilhá-los.

Alocação Dinâmica

- Verificar o topo da pilha:

```
No topo;
tipo topo() {
    se (pilhaVazia() == verdadeiro) {
        exceção("Não há elementos na pilha");
    }
    tipo valor = topo.dado;
    retorne valor;
}
```

Alocação Dinâmica

- Verificar o tamanho da pilha:

```
No topo;
tipo tamanho() {
    int cont = 0;
    se (pilhaVazia() == falso) {
        No auxiliar = topo;
        cont = 1;
        enquanto (auxiliar.proximo != null) {
            cont = cont + 1;
            auxiliar = auxiliar.proximo;
        }
    }
    retorne cont;
}
```

Alocação Dinâmica

- Exemplo(pilha de inteiros):

```
class exemplo {
        void main(String[] args) {
            Pilha p = new Pilha();
            p.push(5);
            p.push(4);
            p.push(3);
            p.push(2);
            p.push(1);
            inteiro topo = p.topo();
            escreva("Topo:"+topo);
            inteiro tamanho = p.tamanho();
            escreva("Tamanho da Pilha:"+tamanho);
            ...
        }
}
```

...continuação

```
class exemplo {
        void main(String[] args) {
            ...
            enquanto (p.pilhaVazia() == false) {
              inteiro dado = p.pop();
              escreva(dado);
              escreva(" ");

              tamanho = p.tamanho();
              escreva("Tamanho da Pilha: "+tamanho);
              topo = p.topo();
              escreva("Topo:"+topo);
            }
        }
}
```

Alocação Dinâmica

- Exemplo:

Console:

```
Topo: 1
Tamanho da pilha: 5
Pop: 1

Tamanho da pilha: 4

Elemento do topo: 2
Pop: 2

Tamanho da pilha: 3

Elemento do topo: 3
```

```
Pop: 3

Tamanho da pilha: 2

Elemento do topo: 4
Pop: 4

Tamanho da pilha: 1

Elemento do topo: 5
Pop: 5

Tamanho da pilha: 0

Exception: Pilha vazia
```

Exercício 1

Simular o comportamento de pilhas dinâmicas para os algoritmos abaixo (A simulação deve deixar evidente a pilha que sobrou na memória):

a)

```
Para (i = 0 ; i < 10 ; i++) {
        Se (i % 2 == 0) {
                Push(i * i);
        } Senão {
                Se (i <= 5) {
                        Push(i);
                } Senão {
                        Pop();
                }
        }
        Top();
}
Size();
```

b)

```
Para (i = 100 ; i < 115 ; i++) {
        Se (isEmpty()) {
                Push(i + 100);
        } Senão {
                Se (Size() <= 4) {
                        Push(i + 50);
                } Senão {
                        Pop();
                }
        }
        Top();
}
Size();
```

Exercício 2

a) Transformar o projeto em uma biblioteca de uma Pilha de Inteiros, gerando o JAR PilhaInt.

b) Adaptar o modelo de Pilha Dinâmica desenvolvido em aula, com os métodos esperados, para uma Pilha de Strings. Transformar o projeto em uma biblioteca, gerando o JAR PilhaStrings.

Exercício 3

Implementar um novo projeto Java com base biblioteca PilhaInt que permita a conversão de decimais para binários, a qual se dá dividindo, sucessivamente, o valor de entrada por 2 e concatenando os restos da divisão do último para o primeiro.

O projeto deve ter uma classe ConverteController no package controller, que inicialize uma pilha de inteiros e com um método decToBin(int decimal): String, que, recebendo um número decimal e realizando as operações, irá inserindo os restos das divisões na pilha. Ao término do empilhamento, deverá ser feita a operação de desempilhar, concatenando cada número desempilhado (Convertendo para String) com o próximo, até a pilha esvaziar.

Deve-se ter também uma classe Principal no package view que permita ao usuário inserir um número decimal limitado a 1000.

Exercício 4

Implementar um novo projeto Java com a biblioteca PilhaInt.

Esse projeto deve implementar uma solução para uma calculadora em Notação Polonesa Reversa (NPR), também conhecida como posfixa. Calculadoras HP, como a 48G ou a 12C utilizam esse formato de cálculo, em detrimento da maneira algébrica (infixa).

A lógica da NPR se dá como a seguir:

Notação Polonesa Reversa:
(O vídeo https://www.youtube.com/watch?v=-b-f9-9_xAI mostra a HP 50G em operações infixa e posfixa)

- Enquanto for digitado número, ele será empilhado.
- Quando for digitada uma operação (+,-,*,/), 2 valores devem ser desempilhados, se faz a operação com eles e o resultado retorna à pilha
- É importante verificar que a pilha deve ter, no mínimo 2 valores para fazer a operação

Exercício 4

O projeto deve ter uma classe de controle (NPRController) que inicializa uma nova Pilha e deve ter duas operações:

- Operação insereValor(Pilha p, int valor):void, faz um push() na pilha

- Operação npr(Pilha p, String op):int. O método deve verificar se a String se trata de uma operação (+,-,*,/), verifica se é possível fazer 2 pop() e, em sendo possível, fazer os 2 pop(), fazer a operação, gravar em uma variável resultado (que é o retorno da operação) e fazer o push() do resultado.
  - Para operações de subtração e divisão (que a ordem importa), fazer o valor do 2º pop() operação valor do 1º pop(), ou seja o valor mais antigo à esquerda da operação
  - Se não houverem 2 valores, deve-se lançar um Exception de pilha com valores insuficientes

A classe view Principal, deve inicializar a pilha e solicitar dados (número ou operação) ao usuário até alguma condição de encerramento, definido por você.