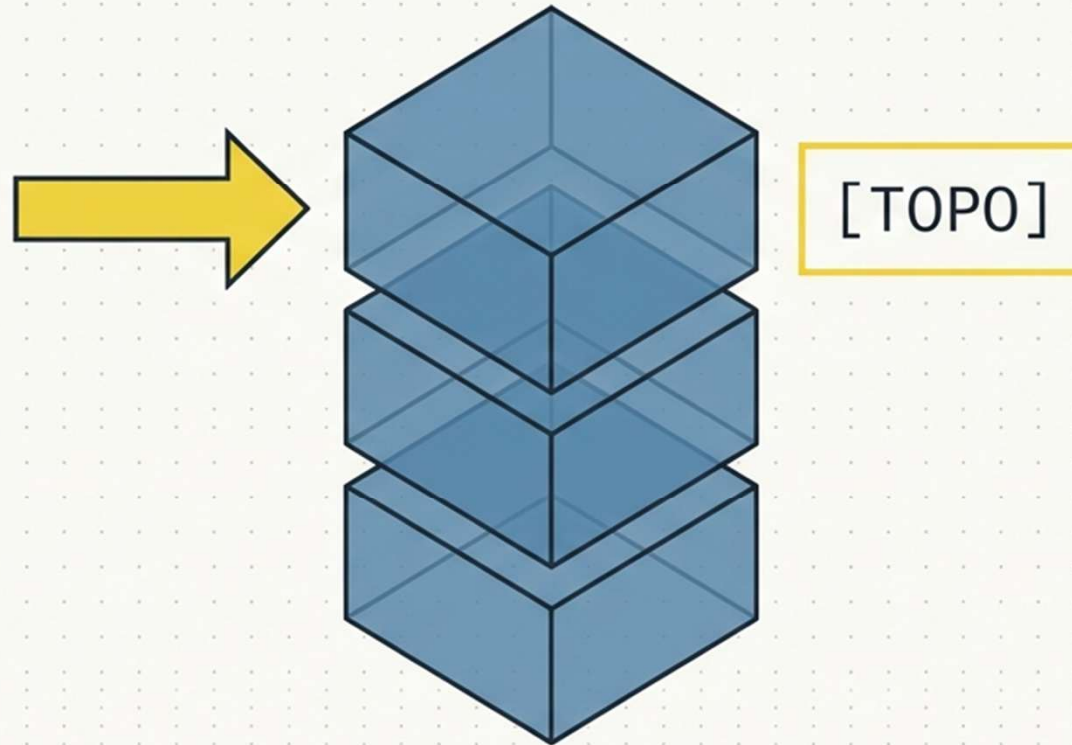
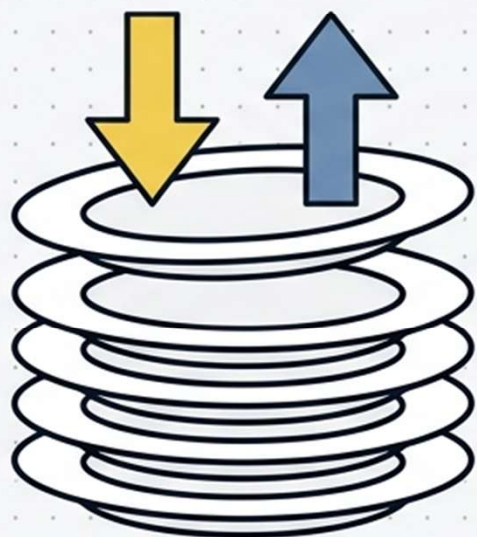




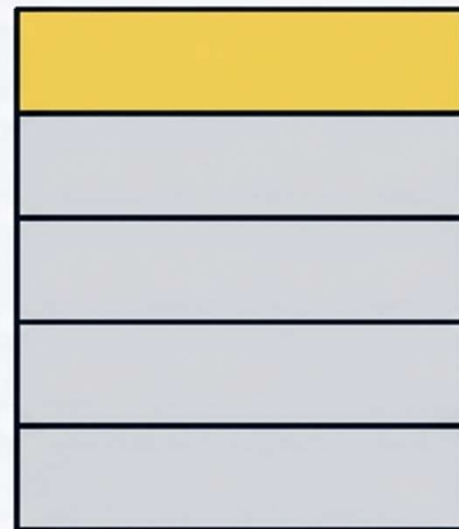
A Arquitetura da Estrutura de Dados Pilha

Do conceito visual LIFO à implementação estática a nível de memória em Java.

O Último Elemento a Entrar é Sempre o Primeiro a Sair



LIFO

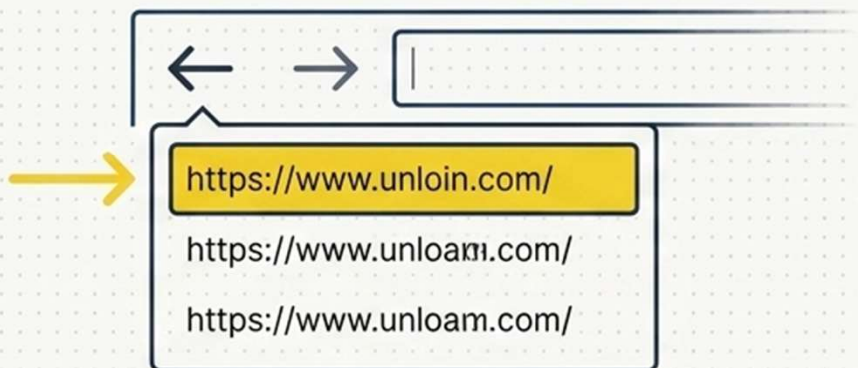


- **LIFO (Last In, First Out):** A regra de ouro. Toda inclusão e remoção ocorre estritamente na mesma extremidade.
- **O Gargalo do Topo:** Elementos subjacentes estão inacessíveis. Eles precisam aguardar a remoção de todos os itens acima deles para serem acessados.

O Motor Invisível por Trás das Interfaces Modernas

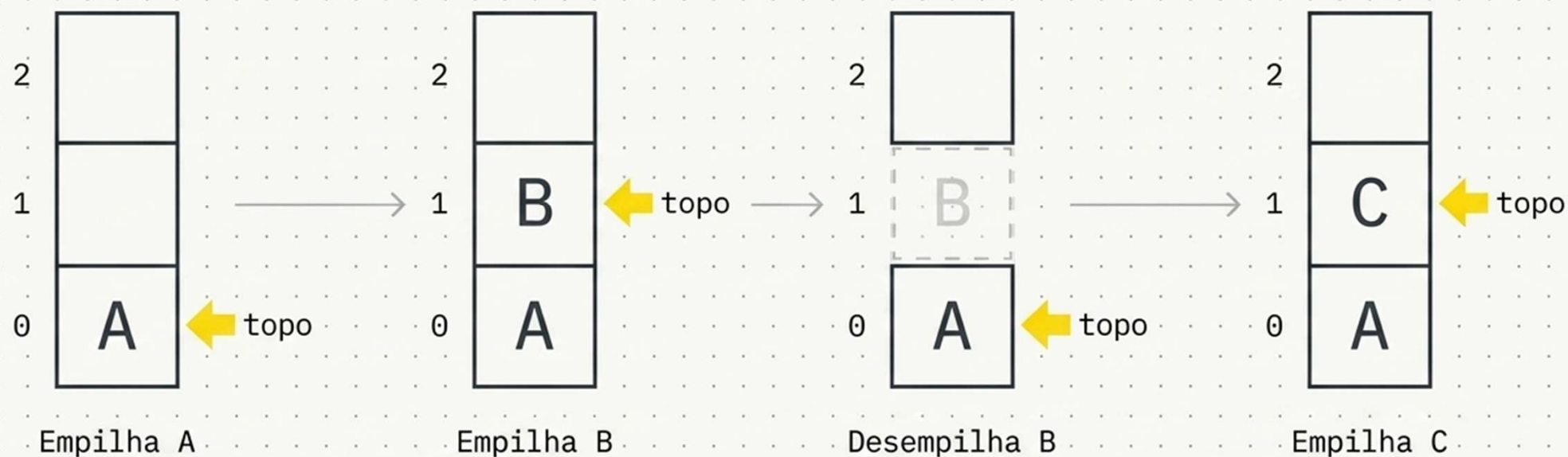


Desfazer (Ctrl+Z): Cada palavra digitada é empilhada. O comando Desfazer desempilha a última ação registrada.



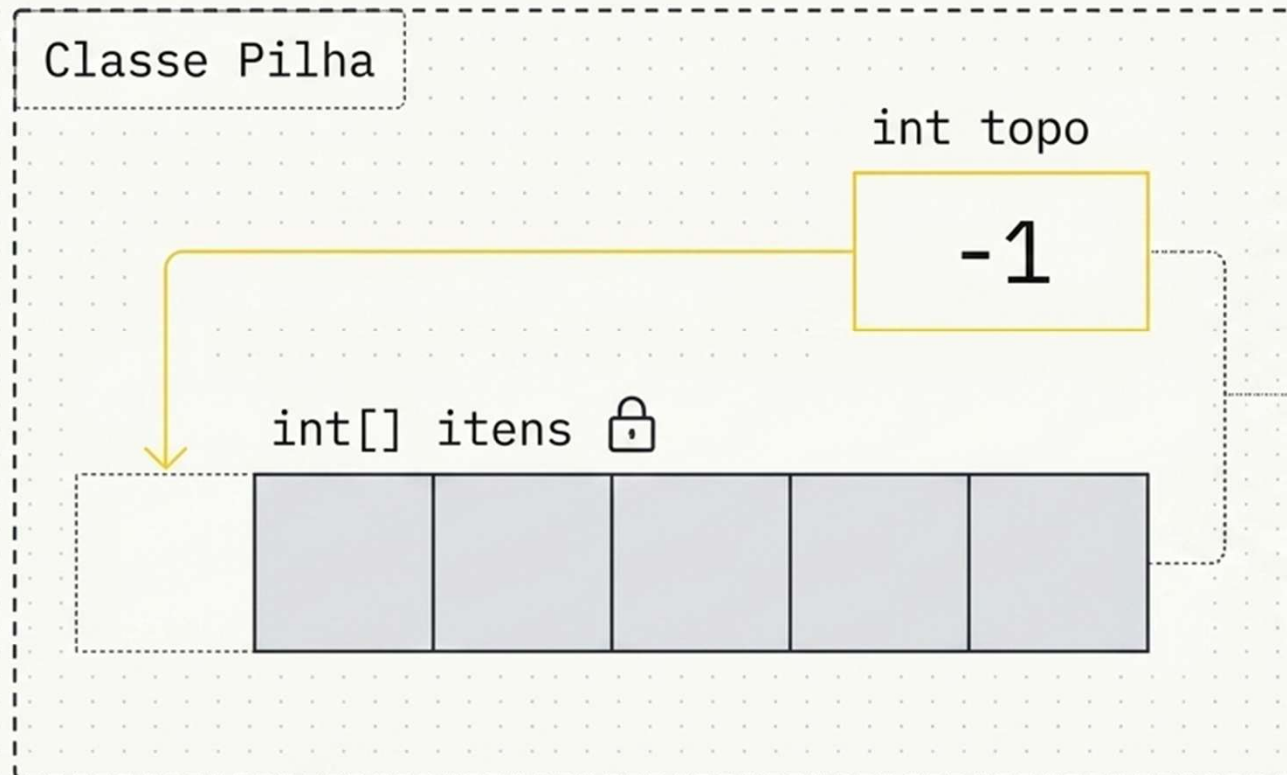
Histórico de Navegação: Páginas visitadas formam uma pilha. Voltar uma página significa retirar o topo atual e revelar o endereço anterior.

A Dinâmica Constante de Vai e Vem do Ponteiro de Topo



O fluxo de dados é ditado puramente pelo movimento do ponteiro.
O espaço é reutilizado instantaneamente.

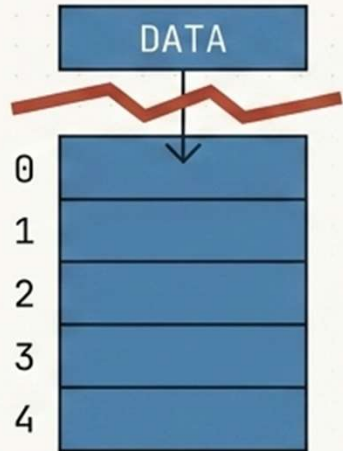
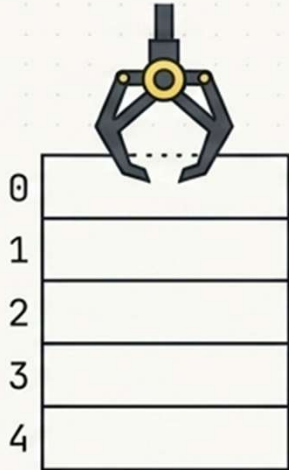
A Anatomia de uma Pilha Estática em Memória



Vetor Estático: O tamanho é definido na criação e permanece inalterado do início ao fim. Não utilizamos `ArrayList` para garantir o controle absoluto em nível hard.

O Maestro: A variável `topo` é o único guia; ela diz exatamente onde a ação acontece.

Fronteiras Críticas e Diagnóstico de Falhas de Memória

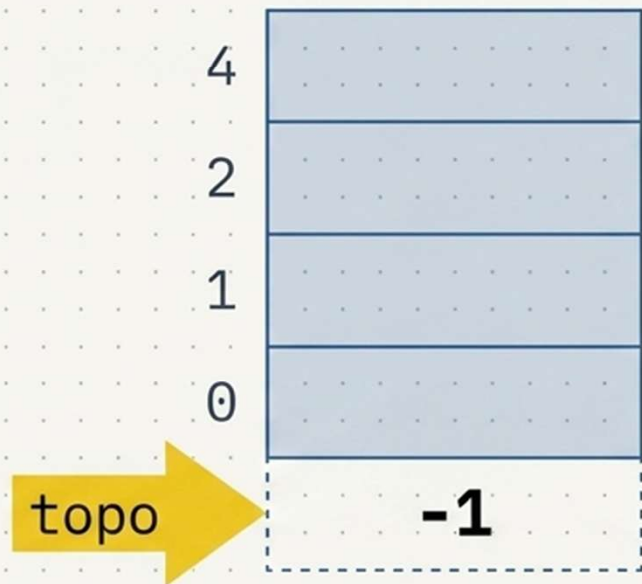
OVERFLOW	UNDERFLOW
	
Cenário Tentativa de inclusão em pilha cheia.	Cenário Tentativa de remoção em pilha vazia.
Solução Algorítmica Validar o limite superior do vetor antes de empilhar.	Solução Algorítmica Validar a ausência de dados antes de desempilhar.

O Contrato de Operações da Estrutura

✓	<code>empilhar(int valor)</code>	Insere no topo.
✓	<code>desempilhar()</code>	Remove do topo.
✓	<code>verificarPilhaVazia()</code>	Retorna boolean.
✓	<code>verificarPilhaCheia()</code>	Retorna boolean.
✓	<code>imprimirPilha()</code>	Visualiza o estado.

A implementação exigirá construir estes cinco pilares fundamentais.

O Construtor: O Ponto de Partida da Estrutura



```
public Pilha(int tamanho) {  
    this.topo = -1;  
    this.itens = new int[tamanho];  
}
```

Por que -1?: O índice 0 já representa uma posição válida e ocupada no vetor. Iniciar em -1 garante que a pilha nasça logicamente vazia.

Tamanho Flexível: O tamanho é passado por parâmetro na criação, permitindo criar pilhas de 4 ou 100 elementos sem fixar um limite universal.

Guardiões de Estado Contra Falhas Estruturais

```
public boolean verificarPilhaVazia() {  
    return topo == -1;  
}
```

Código Limpo: Em vez de blocos verbosos de if/else, retornamos o resultado da expressão lógica diretamente, uma prática recomenda para métodos booleanos.

```
public boolean verificarPilhaCheia() {  
    return topo == itens.length - 1;  
}
```

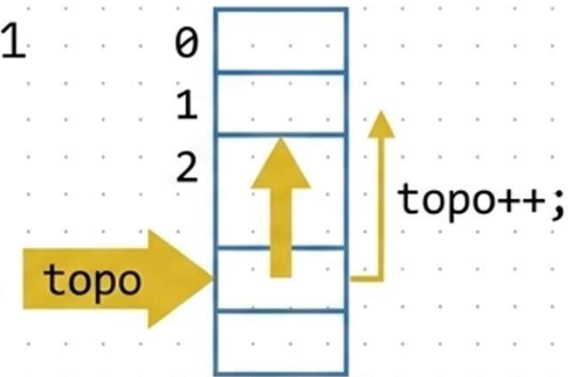


A Fronteira Máxima: A pilha está cheia quando o topo atinge exatamente o tamanho total do vetor menos um (length - 1).

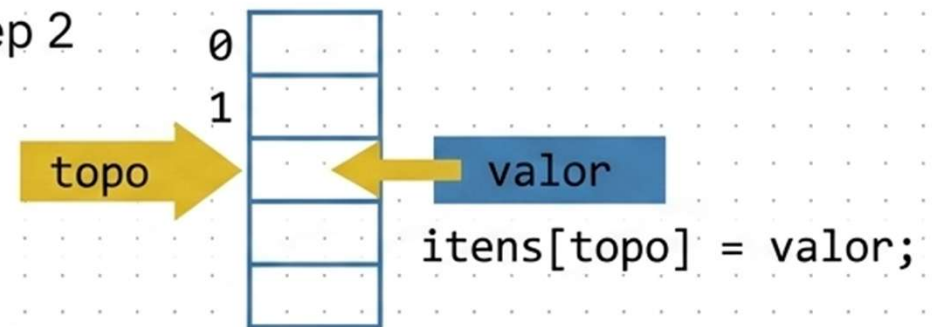
Inserção Exige Preparar o Terreno Primeiro

```
public void empilhar(int valor) {  
    if (!verificarPilhaCheia()) {  
        topo++;  
        itens[topo] = valor;  
    }  
}
```

Step 1



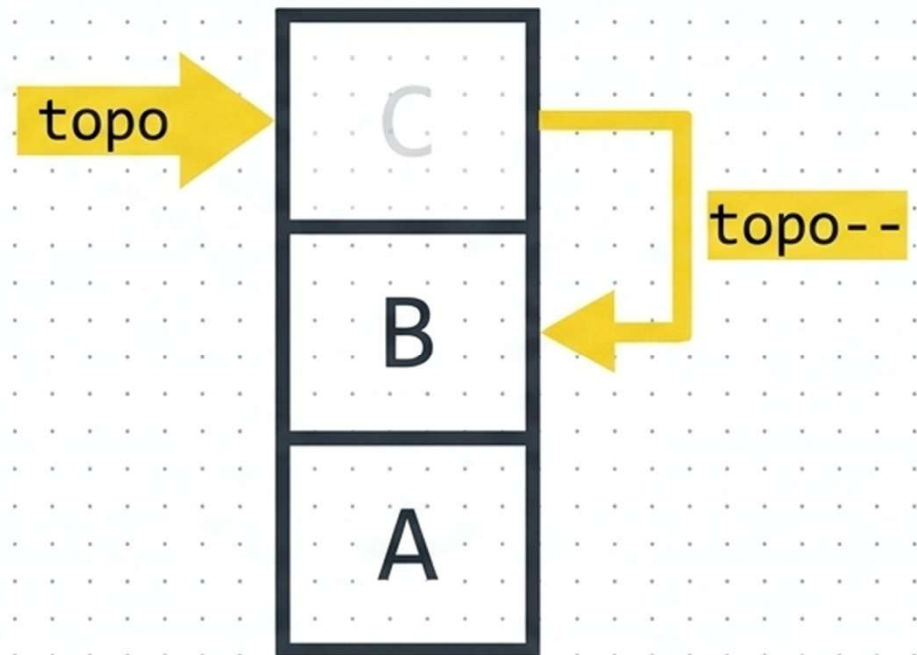
Step 2



A ordem é inegociável. O ponteiro deve ser deslocado antes de injetar o valor para não sobrescrever o elemento atual do topo.

O Segredo da Remoção Lógica Subjacente

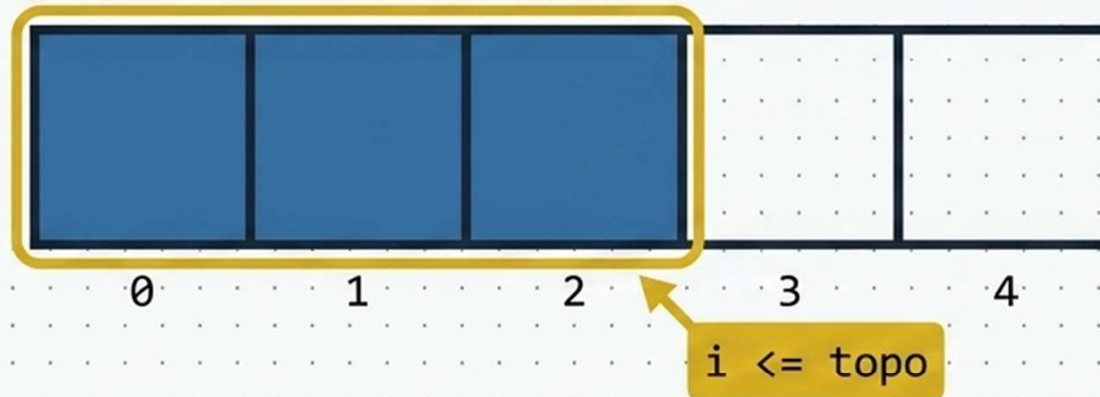
```
public void desempilhar() {  
    if (!verificarPilhaVazia()) {  
        topo--;  
    }  
}
```



Nenhuma Limpeza Necessária: Não precisamos apagar o valor no vetor. Apenas decrementamos o ponteiro. O valor antigo torna-se invisível e será naturalmente sobrescrito na próxima chamada de empilhar().

Leitura Segura Sem Alteração de Estado

```
public void imprimirPilha() {  
    for (int i = 0; i <= topo; i++) {  
        System.out.println(itens[i] + " ");  
    }  
}
```



O Limite Correto: O loop itera de 0 até topo, não até o length do vetor. Isso previne a impressão do lixo de memória (elementos que foram logicamente desempilhados).

O Comportamento da Máquina Sob Estresse

```
Pilha p = new Pilha(2);  
p.empilha(10);  
p.empilha(20);  
p.empilha(30);    // causa erro  
p.empilha(40);    // causa erro  
p.desempilha();  
p.empilha(50);
```

Por que o 40 sumiu?: A pilha de tamanho 2 aceitou apenas 10 e 20. O 30 e 40 causaram overflow e foram rejeitados. O desempilhar removeu o 20. O 50 foi empilhado no lugar dele.

```
Pilha Cheia  
Pilha Cheia  
10 50
```

A Importância do Feedback: Imprimir alertas de erro no console é fundamental para não perder o rastro de inserções falhas.

O Blueprint Definitivo da Pilha Estática

Regra LIFO	Arquitetura	Operações
 Último a entrar é o primeiro a sair.	 Usa vetor estático de tamanho fixo. O topo inicia em -1.	 empilhar (incrementa, depois insere). desempilhar (apenas decrementa, remoção lógica). Guardiões: Nunca opere sem validar Overflow e Underflow.

A simplicidade do ponteiro único é a base das interfaces de navegação e histórico do mundo real.