NOVA SCHOOL OF SCIENCE & TECHNOLOGY
NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

ALGORITMOS E ESTRUTURAS DE DADOS 2023/2024
TIPOS ABSTRATOS DE DADOS

Armanda Rodrigues

13 de setembro 2023

Tipos Abstratos de Dados (TAD)

TIPOS DE DADOS

ABSTRAÇÃO

Tipo de Dados

- Conceito aplicado originalmente aos tipos de dados primitivos disponíveis numa linguagem de programação
  - E.g. Em Java, `int` e `double`;
- Quando se fala num tipo primitivo referimo-nos a:
  - Um conjunto de Itens de dados com determinadas características (Domínio);
  - Um conjunto de operações que podem ser efetuadas sobre esses itens.
- As operações permitidas num tipo de dados são inseparáveis da sua identidade
- Para compreendermos o tipo precisamos de compreender que operações podem ser executadas sobre o mesmo

Exemplo:
O tipo `int` em Java está associado aos números inteiros entre –2,147,483,648 and +2,147,483,647
e aos operadores +, –, *, /,…

Tipos de Dados

- Quando usamos Programação Orientada a Objetos, os tipos de dados que pretendemos adicionar podem ser criados através de Classes.
- Os tipos de dados podem representar quantidades numéricas que são usadas de forma muito similar aos tipos primitivos
  - E.g. Uma classe representativa de fracções (com campos de numerador e denominador)
- Estas classes envolvem operações parecidas com as dos tipos primitivos mas que têm de ser aplicadas utilizando uma notação funcional
  - Por exemplo `add()` e `sub()` em vez de + e -
- O termo TIPO de DADOS aplica-se de forma natural a estas classes

Tipos de Dados

- Existem classes que não incluem este aspeto quantitativo
- Qualquer classe representa uma implementação de um tipo de dados, tendo:
  - uma componente de dados (que pode ser realizada com um conjunto de atributos);
  - Um conjunto de operações permitidas sobre os dados (métodos)
- Assim, quando pretendemos representar um conceito do dia a dia, como uma Conta Bancária, ou uma Pilha de Livros, estes conceitos também podem ser tratados como Tipos de Dados

Abstração

- *"Separação mental de um ou mais elementos concretos de uma entidade complexa desprezando outros que lhe são inerentes"* - Dicionário Porto Editora da Língua Portuguesa
- Uma abstração é a essência, as características importantes de algo
- Por exemplo, o Presidente da República Portuguesa é uma abstração, considerada à parte do indivíduo que ocupa o lugar num determinado momento.
  - Os poderes e as responsabilidades do cargo mantêm-se enquanto que os indivíduos vão e vêm

Tipo Abstrato de Dados em OO

- Em Programação OO um TAD é composto de:
  - Descrição dos dados (atributos)
  - Lista de operações (métodos) que podem ser aplicadas aos atributos
  - Instruções sobre como utilizar as mesmas operações

- São propositadamente excluídos os detalhes relativos à maneira como os métodos executam as suas tarefas

- À especificação de um TAD chamamos um *interface*.

Tipo Abstrato de Dados em OO

- Um Tipo Abstrato de Dados pode ser associado a várias implementações, instanciadas em classes
- Cada uma destas classes poderá encapsular a utilização de uma determinada estrutura de dados, para implementar o TAD, com diferentes performances:
  - Em termos do tempo necessário para executar as operações sobre a Estrutura de Dados
  - Em termos do espaço (em memória) necessário para guardar a informação associada ao Tipo de Dados

TADs em AED

- Os exemplos e exercícios que vamos resolver em AED vão implicar a especificação de dois tipos de TADs
  - TADs diretamente relacionados com o domínio do problema que pretendemos resolver
    - Por exemplo, Conta Bancária ou Supermercado
  - TADs genéricos usados em Programação para resolver problemas com determinadas características
    - Por exemplo, Pilha, Fila ou Dicionário
- As implementações de TADs genéricos serão auxiliadas pela escolha de estruturas de dados adaptadas às características subjacentes aos tipos
  - E.g.Uma Pilha pode ser implementada em Vetor ou em Lista Ligada
- As classes resultantes de implementações de TADs genéricos podem contribuir para as implementações de TADs associados ao domínio de certos problemas
  - Uma fila de espera num Supermercado pode ser instanciada através de uma implementação do TAD Fila

Exemplo da Biblioteca

- Vamos pensar quais seriam os TADs necessários para a implementação de um Sistema de uma Biblioteca

- Este sistema contém vários subsistemas
- Iremos debruçar-nos sobre eles de acordo com as oportunidades que nos aparecem relativamente aos TADs a especificar e às estruturas de dados disponíveis

Biblioteca - TADs do Domínio

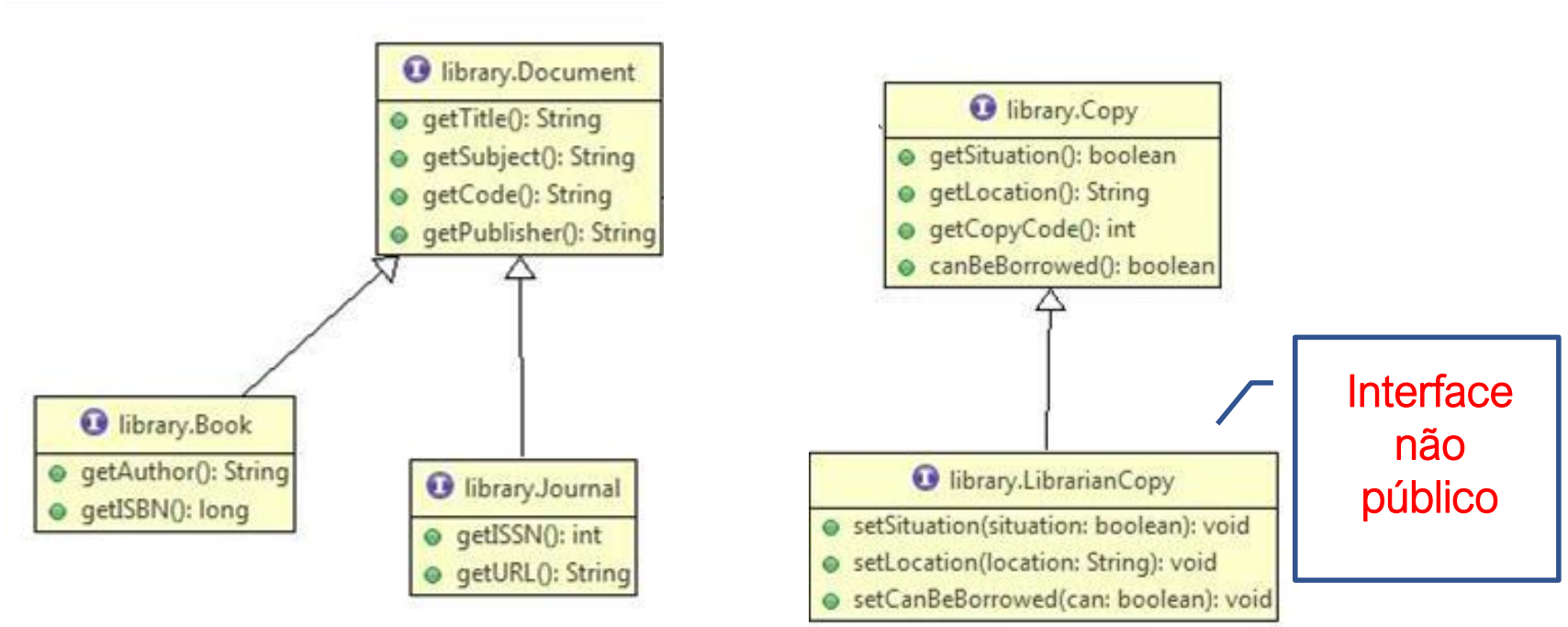
- O Sistema da Biblioteca deverá envolver o armazenamento e tratamento de informação relativa a Documentos existentes na mesma
- Existem vários tipos de **Documentos**: Livros, CDs, DVDs, Revistas, etc…
  - Para simplificar vamos cingir-nos a Livros e a Revistas
- Estes documentos podem existir na biblioteca em várias cópias, ou exemplares
- Como informação de base para um **Documento** existe o Título, o Assunto, a Cota e a Editora
- Um **Livro** terá informação adicional que irá incluir o seu Autor e o ISBN
- Quanto à **Revista**, esta irá incluir ISSN e URL, uma vez que estará disponível online
- Uma Revista está sempre disponível num único exemplar que não pode ser emprestado
- No caso do Livro, podem existir vários exemplares que podem ser emprestados se estiverem disponíveis, com Código, Situação (emprestado ou livre) e Localização na biblioteca

Biblioteca - TADs do Domínio

- Como organizariam os TADs deste exemplo ?

Biblioteca - TADs do Domínio

- Como organizariam os TADs deste exemplo ?



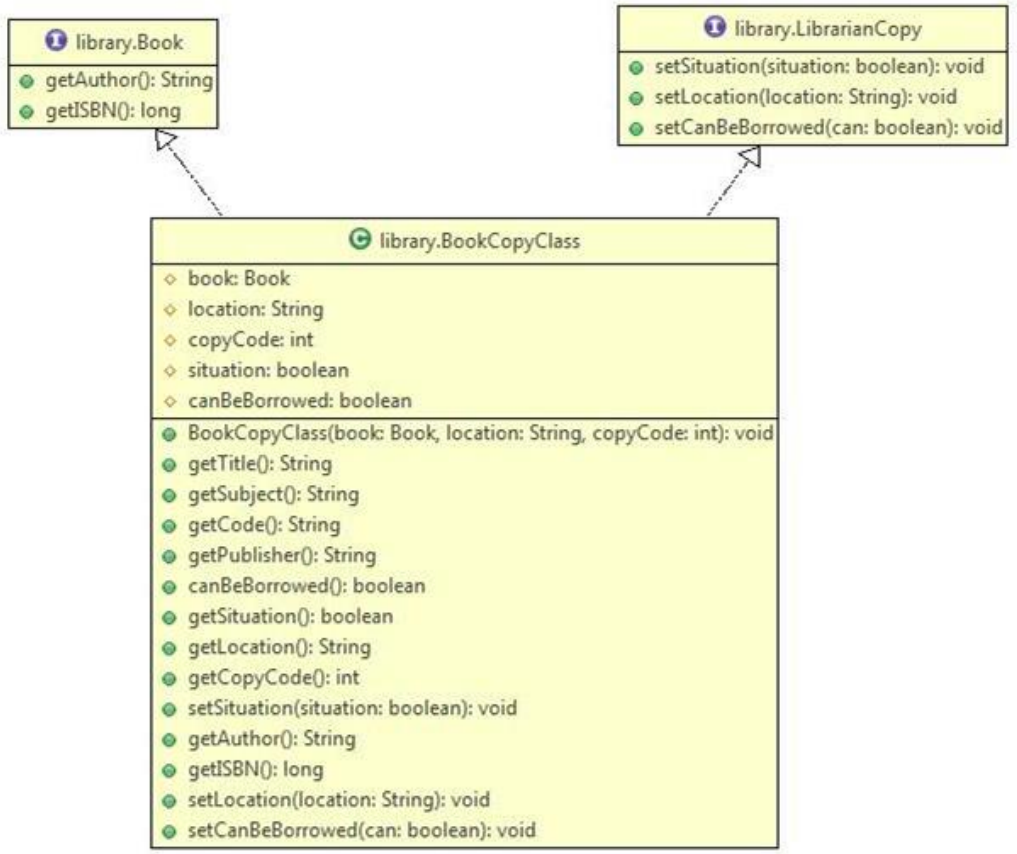
Biblioteca - TADs do Domínio

- Como seria uma classe Cópia de Livro ?

Biblioteca - TADs do Domínio

- Como seria uma classe Cópia de Livro ?

Esta classe poderia ser disponibilizada a aplicações que não alterassem a situação da cópia, através dos interfaces Copy ou Book

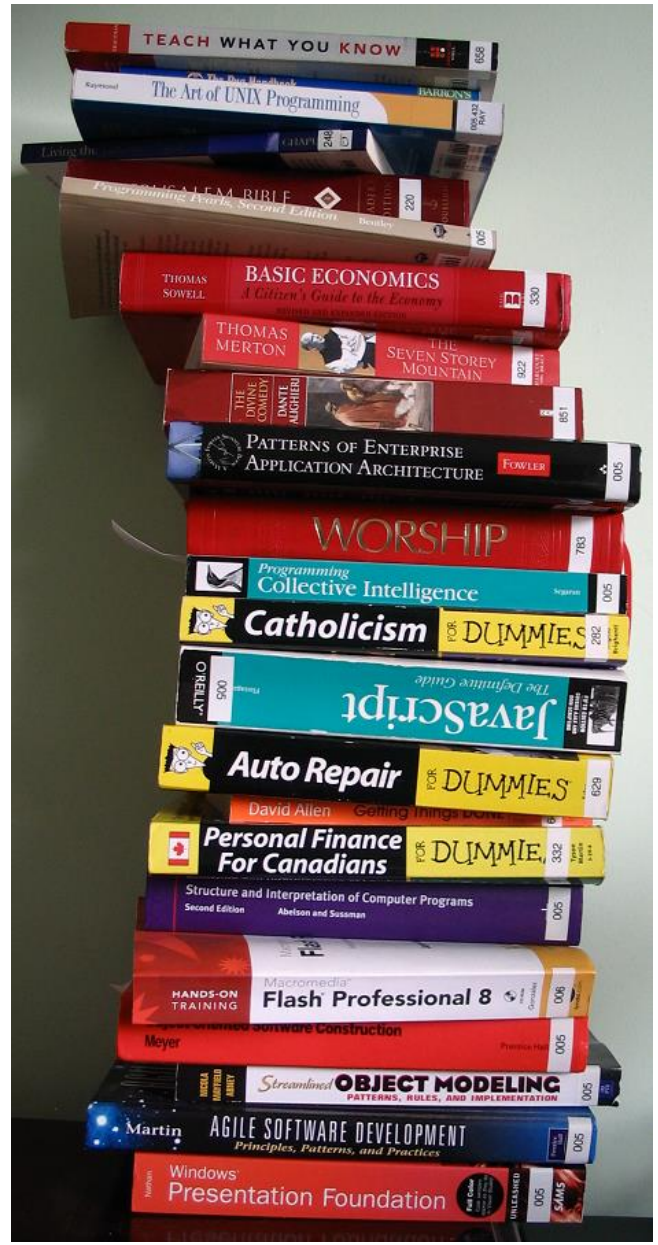


Devoluções de exemplares de livros

- No momento da devolução de um exemplar na Biblioteca, a funcionária poderá não ter tempo para repor o livro juntamente com os exemplares disponíveis
- O processo normal será que todos os exemplares devolvidos fiquem guardados juntos, empilhados ao lado da receção da biblioteca
- Assim, o processo de devolução tem dois passos:
  - **Receber o exemplar**, o que implica inserir o exemplar numa pilha de exemplares a repor. A funcionária poderá receber até **10 devoluções** antes de fazer, obrigatoriamente as reposições dos exemplares.
  - Mais tarde, com tempo, **repor o exemplar no seu lugar**. Neste caso, a funcionária irá, naturalmente, retirar o exemplar que está no topo da pilha de reposição, alterar a informação que lhe está associada (tornando-o disponível para empréstimo) e reinseri-lo no conjunto de cópias disponíveis.

Uma Pilha de Livros

- Para implementar o TAD **DevolucoesExemplar**, vai ser preciso utilizar uma implementação do TAD pilha
- É preciso compreender como o TAD Pilha funciona
- E conhecer pelo menos uma implementação do mesmo TAD



TAD Pilha – Elementos do tipo E

TAD Pilha de Elementos do Tipo E

```
// Retorna true sse a pilha estiver vazia.
boolean vazia( );

// Retorna o elemento do topo da pilha.
// Pré-condição: a pilha não está vazia.
E topo( );

// Coloca o elemento especificado no topo da pilha.
void empilha( E elemento );

// Remove e retorna o elemento do topo da pilha.
// Pré-condição: a pilha não está vazia.
E desempilha( );
```

Interface Pilha de Elementos do Tipo E

Vamos usar Requires para assinalar pré-condições em métodos

```
package dataStructures;
public interface Stack<E>{

   // Returns true iff the stack contains no elements.
   boolean isEmpty( );

   // Returns the number of elements in the stack.
   int size( );

   // Returns the element at the top of the stack.
   // Requires: size() > 0
   E top( ) throws EmptyStackException;

   // Inserts the specified element onto the top of the stack.
   void push( E element );

   // Removes and returns the element at the top of the stack.
   // Requires: size() > 0
   E pop( ) throws EmptyStackException;
}
```

Interface Pilha de Elementos do Tipo E - Javadoc

Todo o código que vos for fornecido será comentado para Javadoc

```
/**
 * Stack Abstract Data Type
 * Includes description of general methods for the
Stack with the LIFO discipline.
 * @author AED  Team
 * @version 1.0
 * @param <E> Generic Element
 *
 */
public interface Stack<E>
{

 /**
  *  Returns true iff the stack contains no
  *  elements.
  * @return true iff the stack contains no
  *         elements, false otherwise
  */
  boolean isEmpty( );

 /**
  *  Returns the number of elements in the stack.
  * @return number of elements in the stack
  */
  int size( );
```

Todo o código submetido pelos alunos deve ser comentado para Javadoc

```
/**
 *  Returns the element at the top of the stack.
 *  Requires
 * @return element at top of stack
 * @throws EmptyStackException when size = 0
 */
E top( ) throws EmptyStackException;

/**
 *  Inserts the specified <code>element</code> onto
 *  the top of the stack.
 * @param element element to be inserted onto the stack
 */
void push( E element );

/**
 *  Removes and returns the element at the top of the
 *  stack.
 * @return element removed from top of stack
 * @throws EmptyStackException when size = 0
 */
 E pop( ) throws EmptyStackException;
```

```
}
```

Regras para comentários em AED

- Todo o código que vos for fornecido será comentado para Javadoc
- Devido às necessidades de leitura dos materiais de disciplina, os slides não contêm estes comentários

- **Todo o código submetido pelos alunos deve ser comentado para Javadoc**
  - Comentar os métodos dos TADs (interfaces)
  - Comentar todos os métodos de classes que não façam parte dos TADs
  - Nas classes, só comentar métodos que correspondem ao interface público em casos de implementações específicas que necessitem de esclarecimento (em caso contrário usar a tag @see ou @Override)

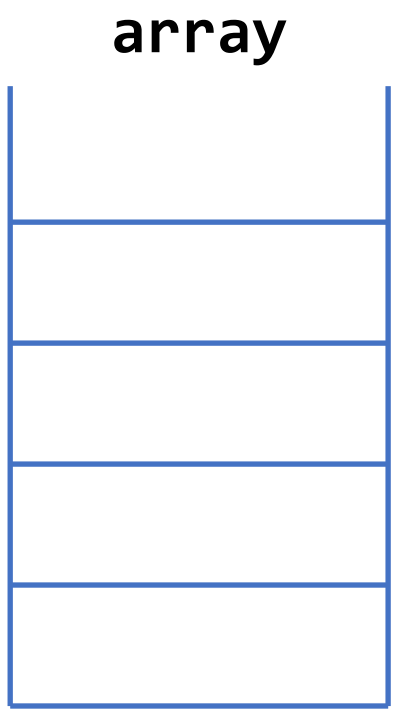
```
/*
 * @see dataStructures.Stack#top()
 */
public E top( ) throws EmptyStackException
{
 if ( this.isEmpty() )
   throw new EmptyStackException("Stack is empty.");
 return array[top];
}
```

Comentar sempre que os comentários façam falta!!!

Implementação do Interface Pilha em Vetor

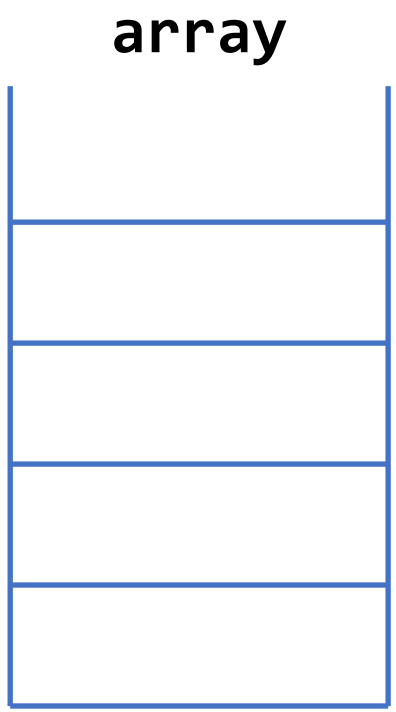
Pilha em vetor

```
capacity = 4
top = -1
```



Pilha em vetor

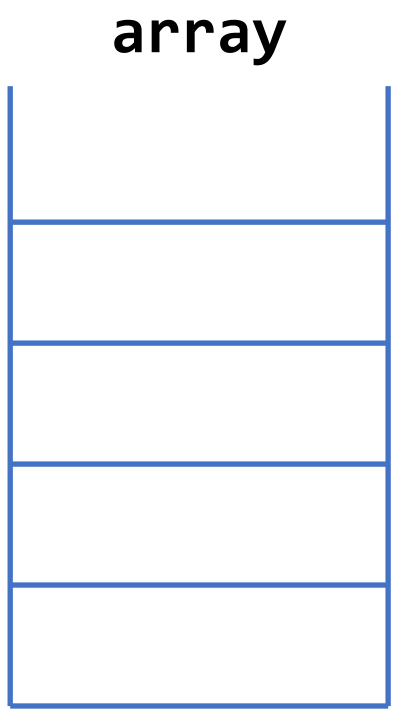
```
capacity = 4
top = -1
```




```
push(5);
```

Pilha em vetor

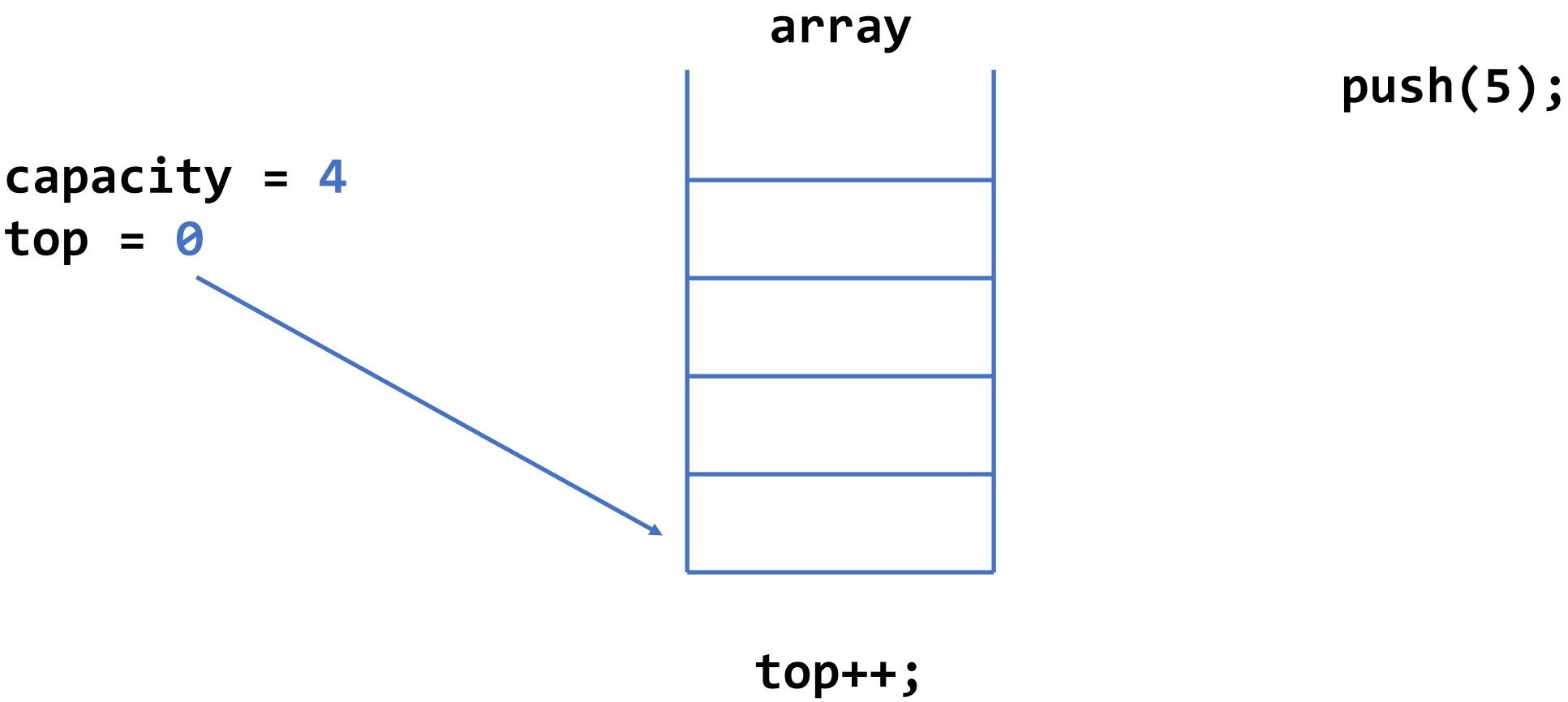
```
capacity = 4
top = -1
```




```
push(5);
```

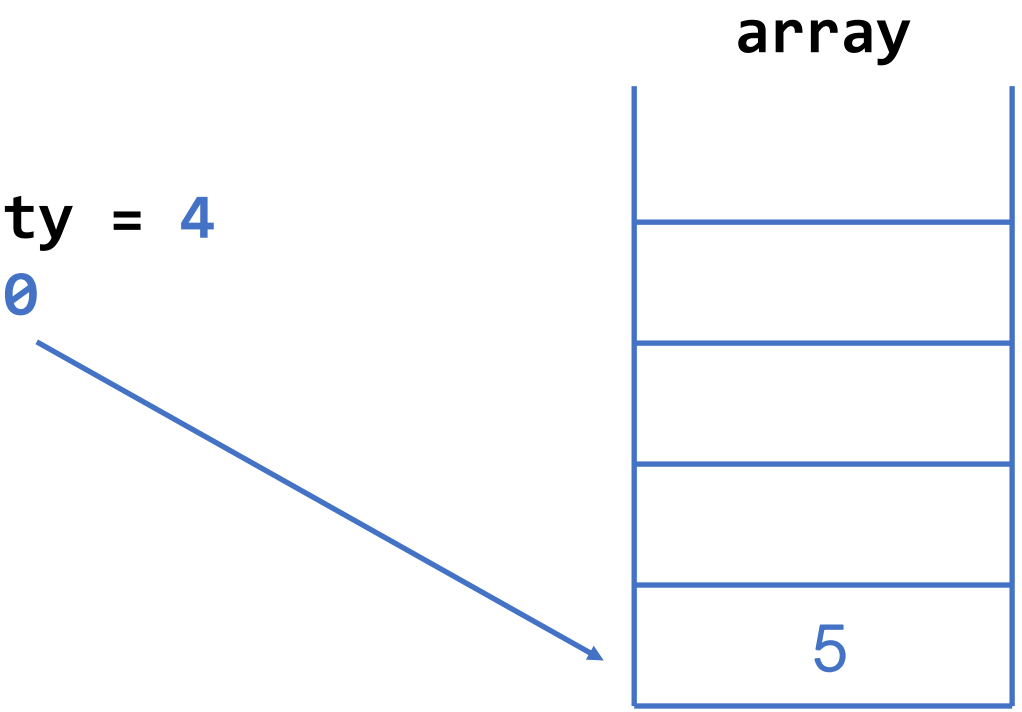
```
top++;
```

Pilha em vetor



Pilha em vetor

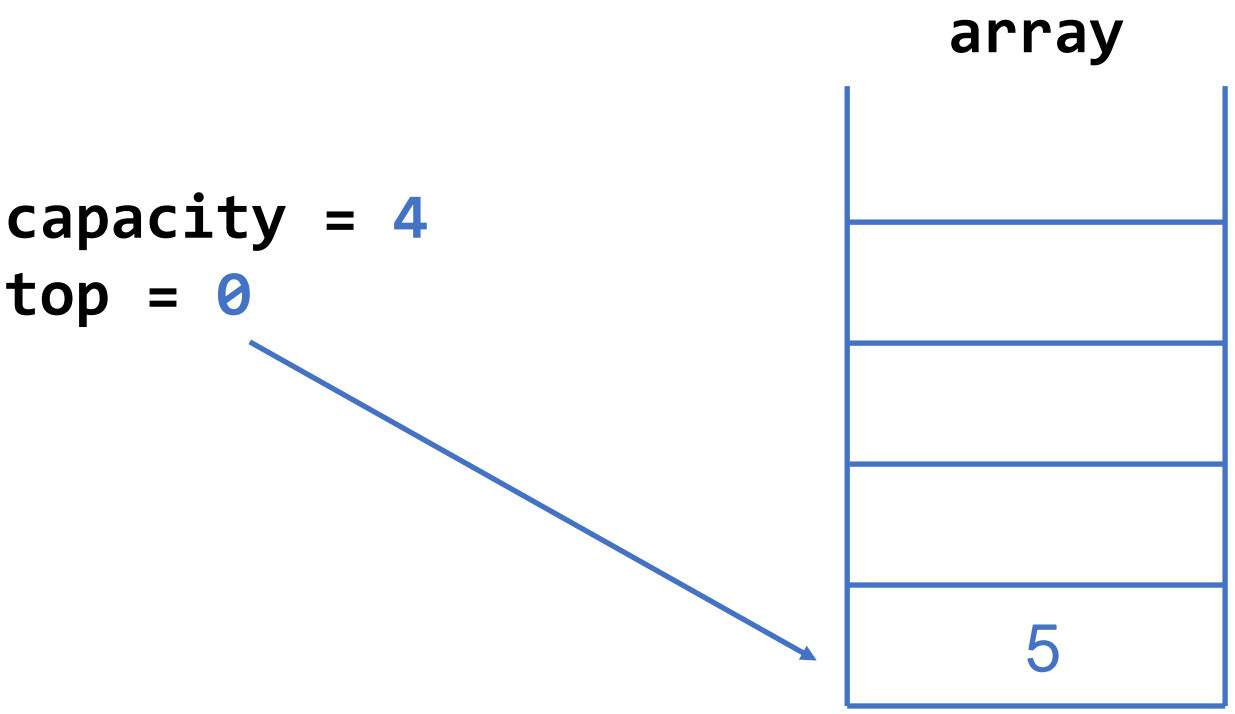
```
capacity = 4
top = 0
```




```
push(5);
```

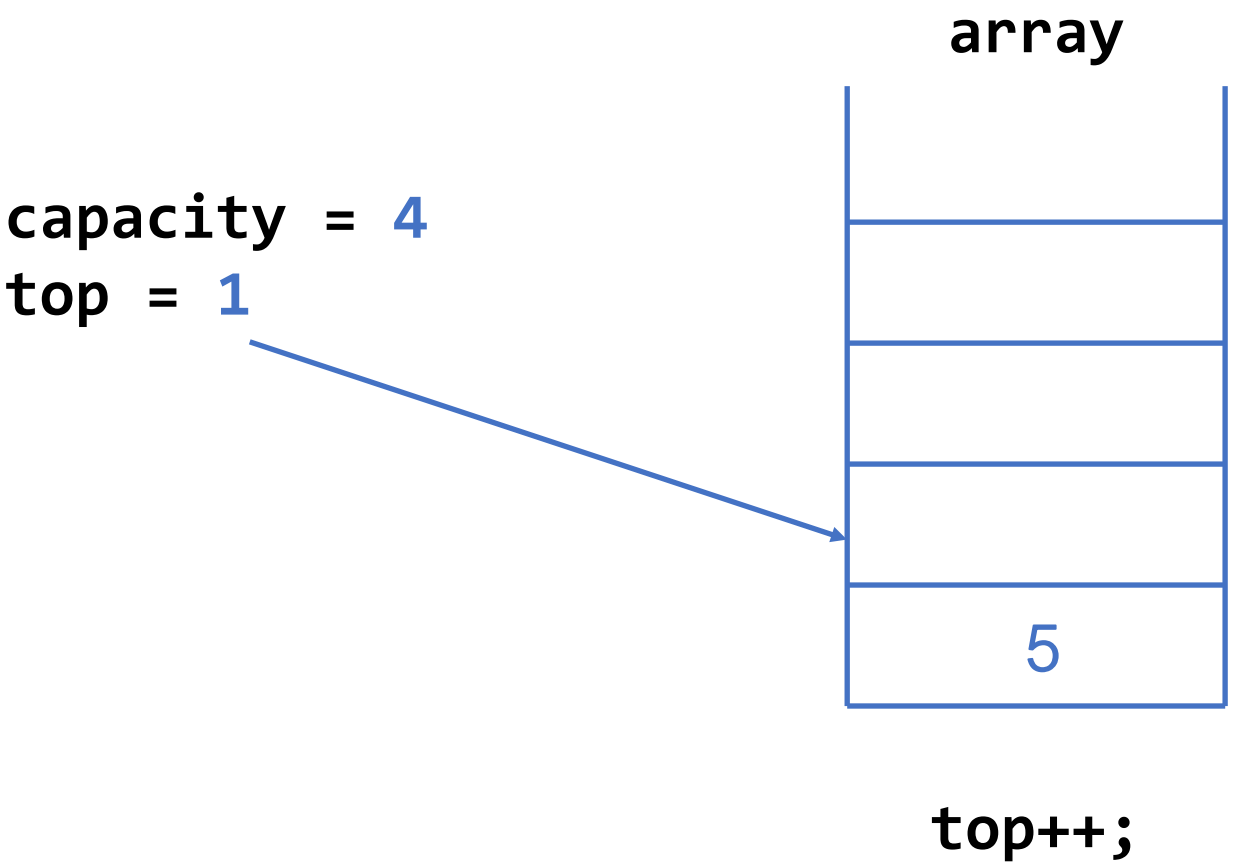
```
array[0] = 5;
```

Pilha em vetor




```
push(5);
push(2);
```

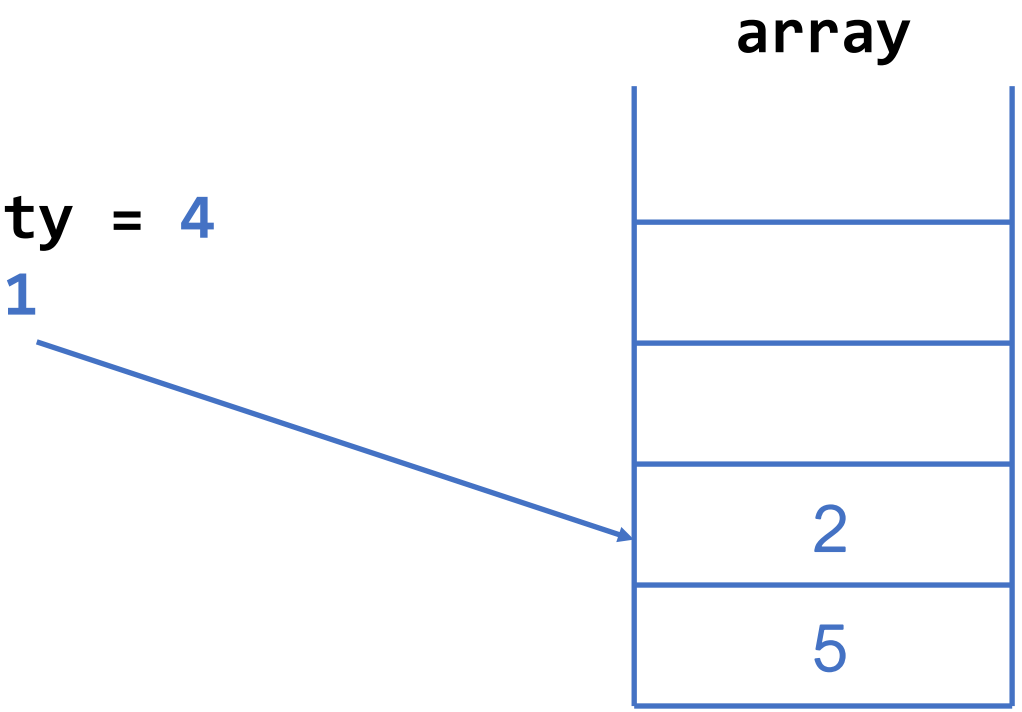
Pilha em vetor




```
push(5);
push(2);
```

Pilha em vetor

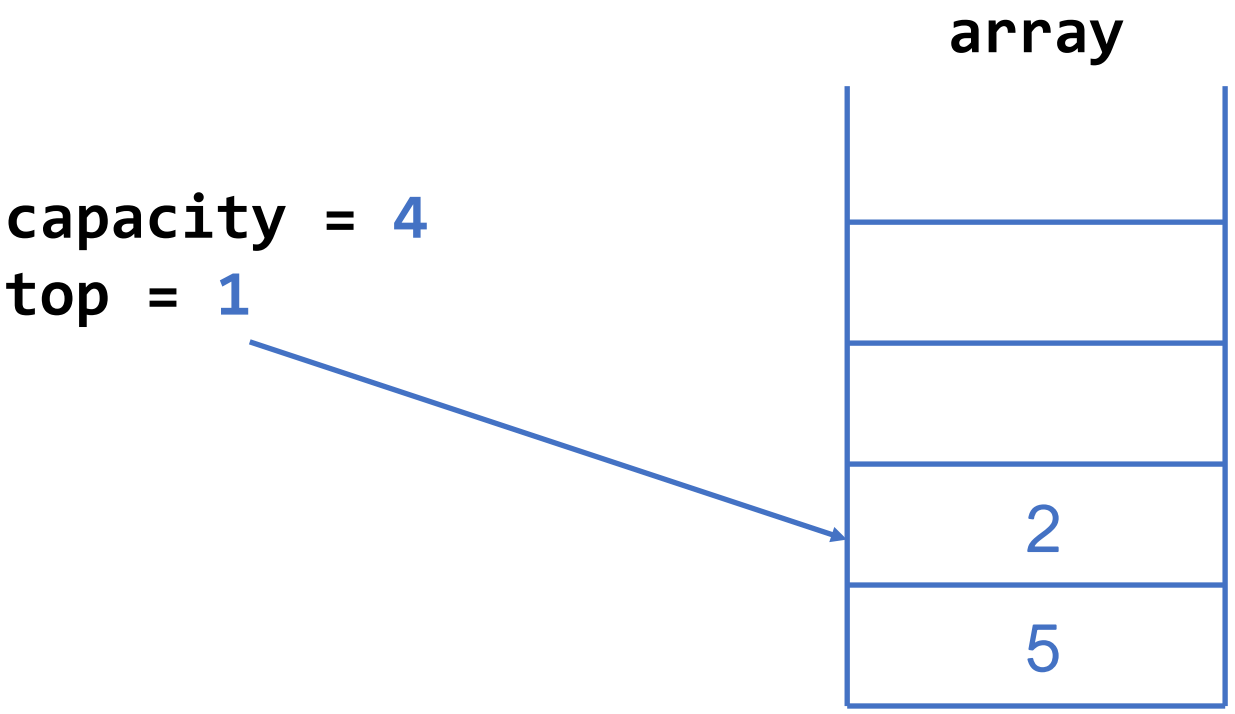
```
capacity = 4
top = 1
```




```
push(5);
push(2);
```

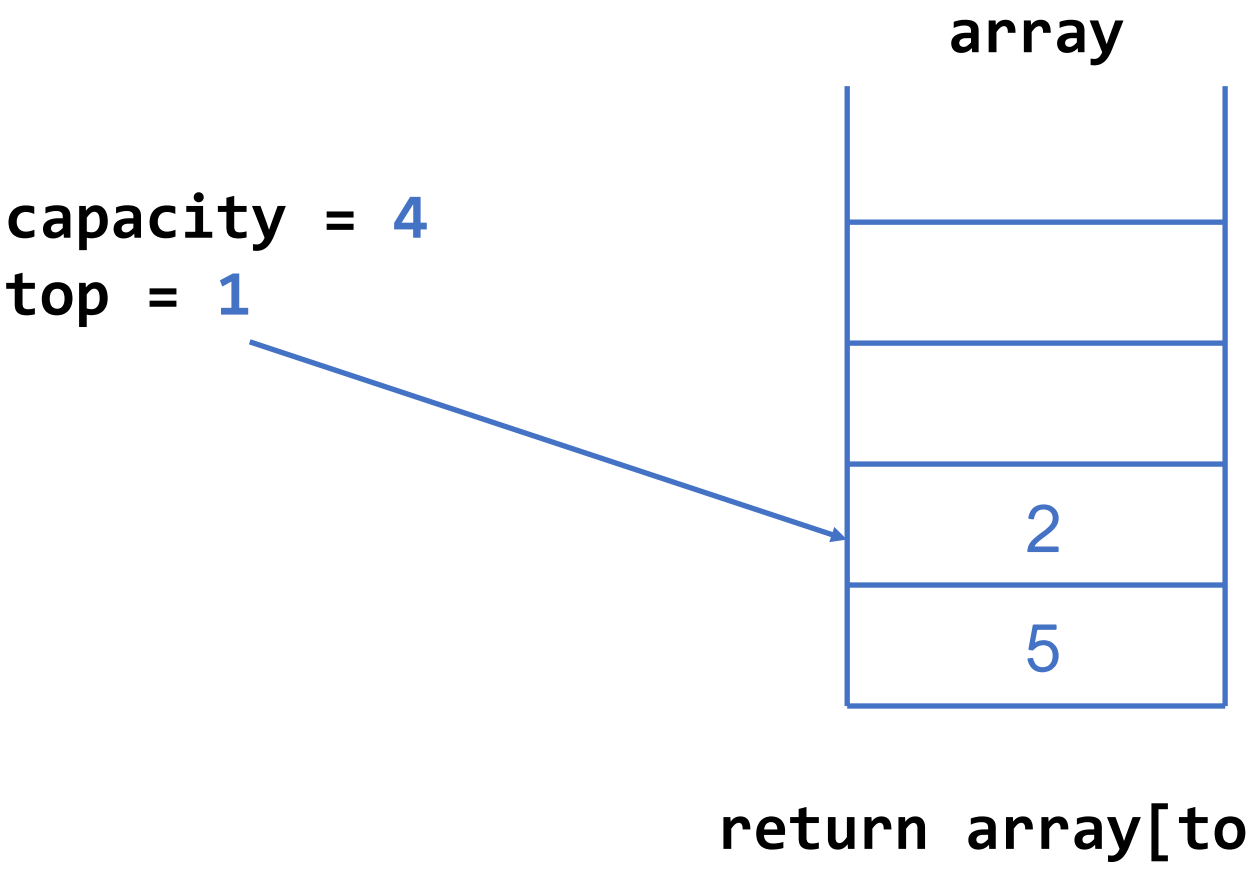
```
array[top] = 2;
```

Pilha em vetor




```
push(5);
push(2);
top();
```

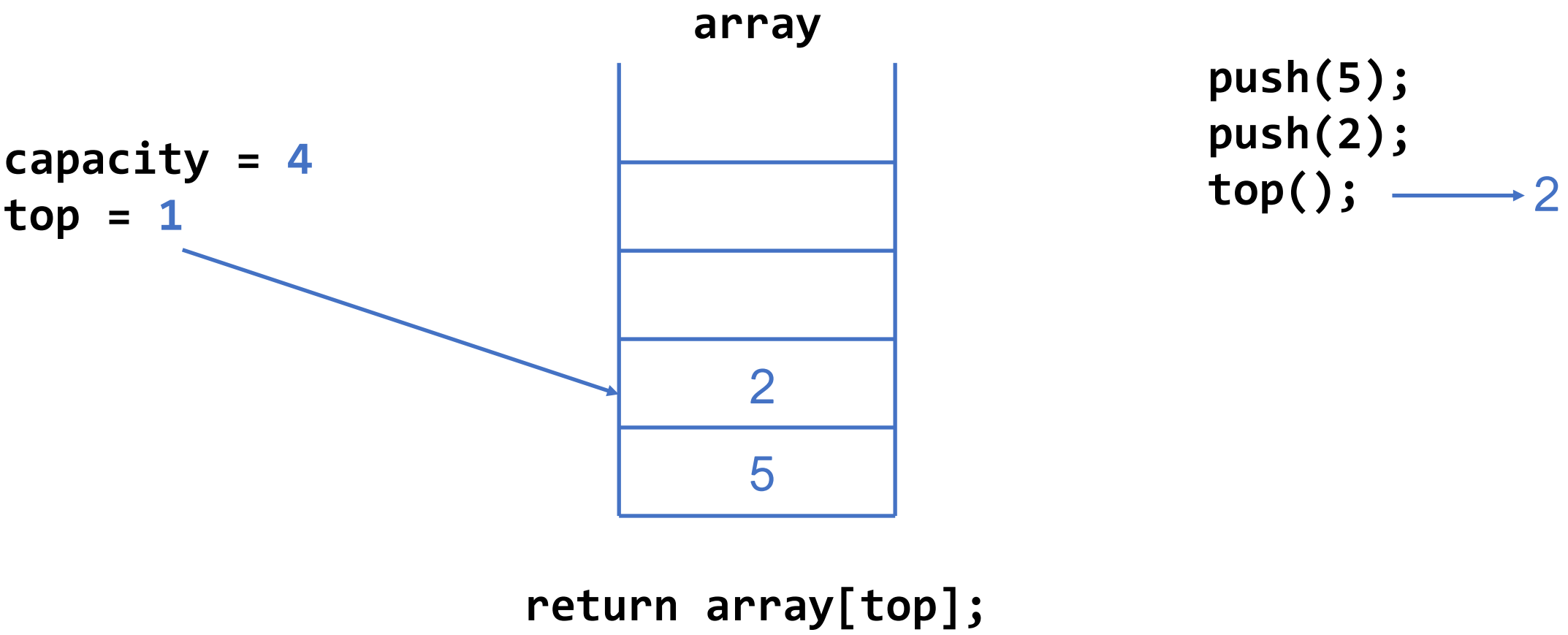
Pilha em vetor



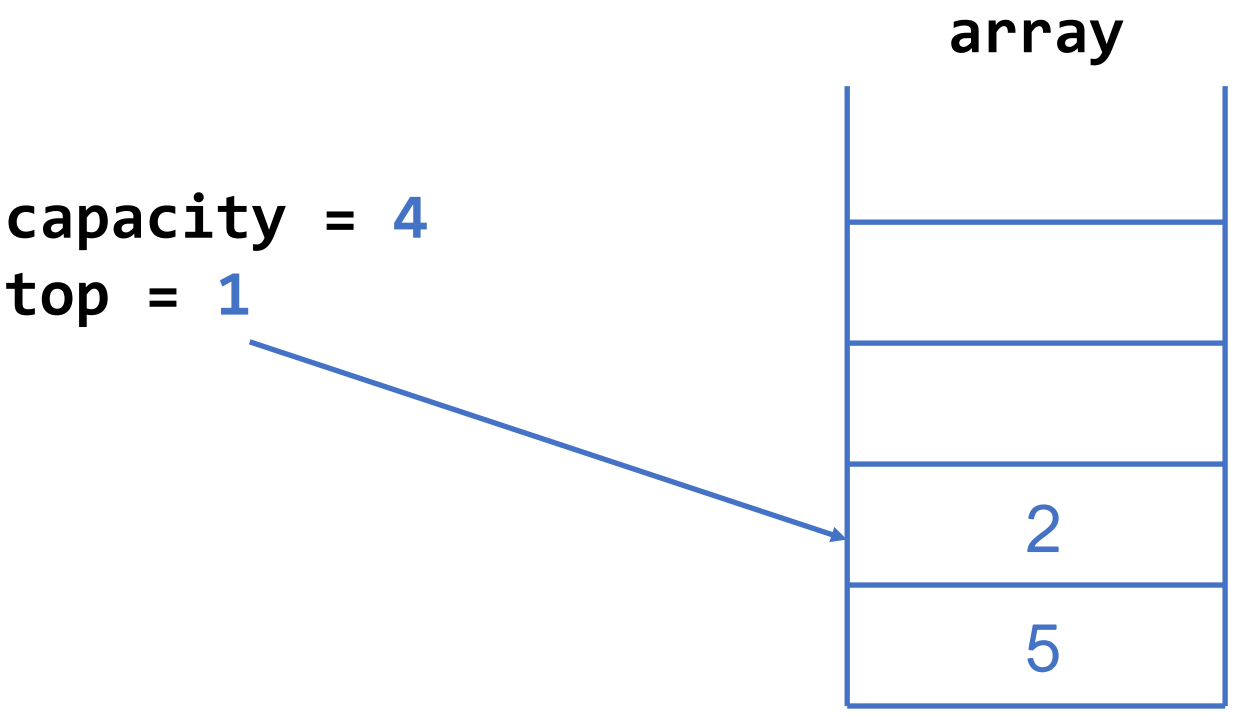

```
push(5);
push(2);
top();
```

```
return array[top];
```

Pilha em vetor

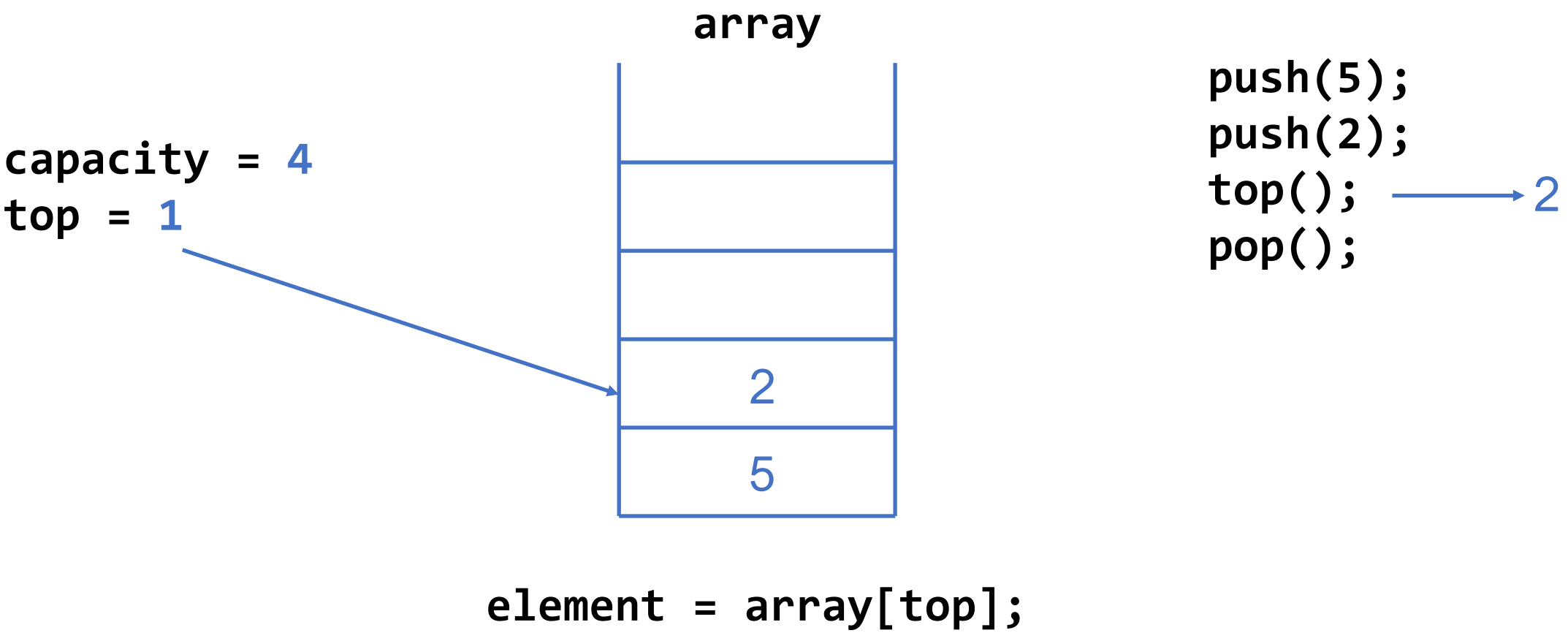


Pilha em vetor

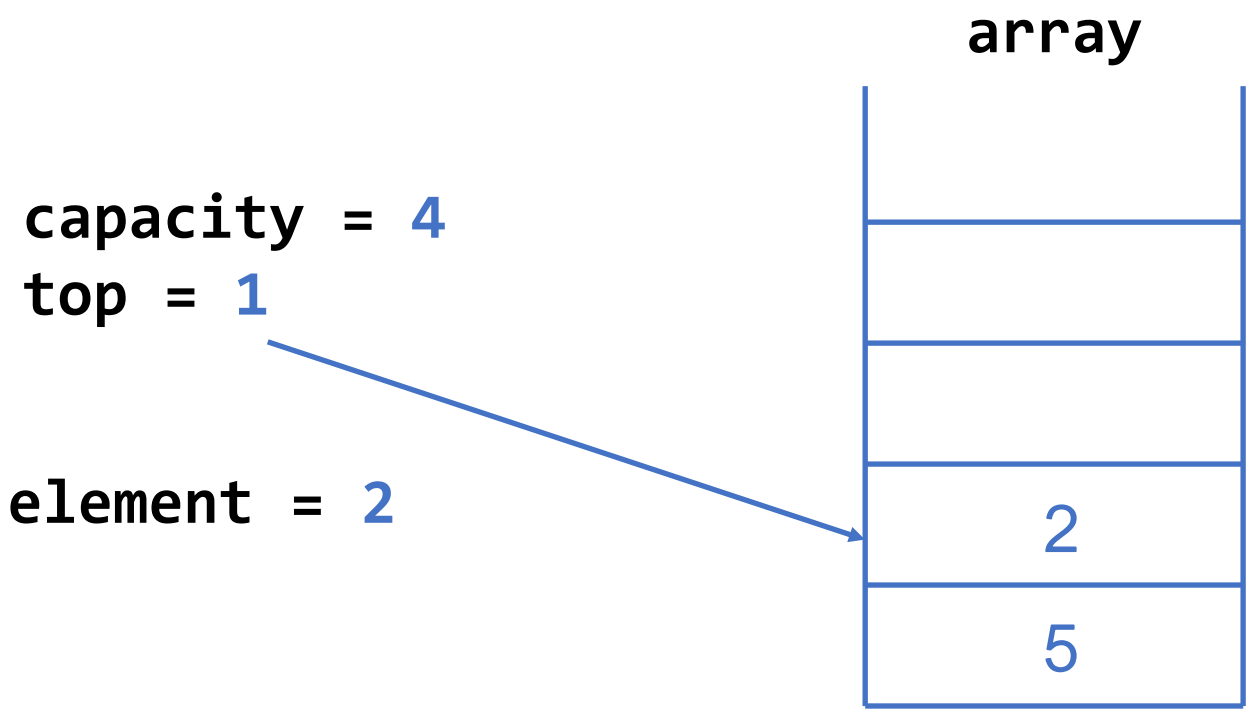



```
push(5);
push(2);
top();  → 2
pop();
```

Pilha em vetor



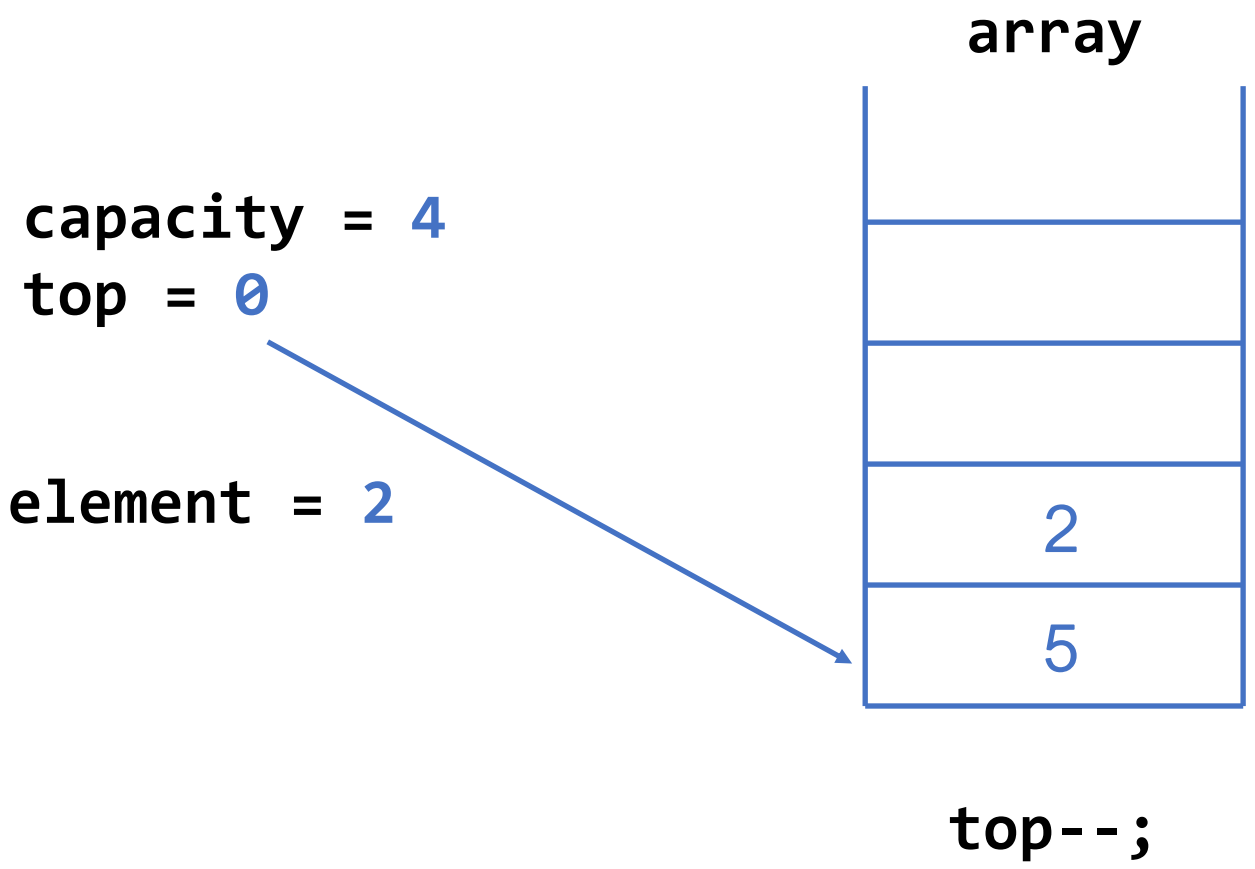
Pilha em vetor




```
push(5);
push(2);
top();  → 2
pop();
```

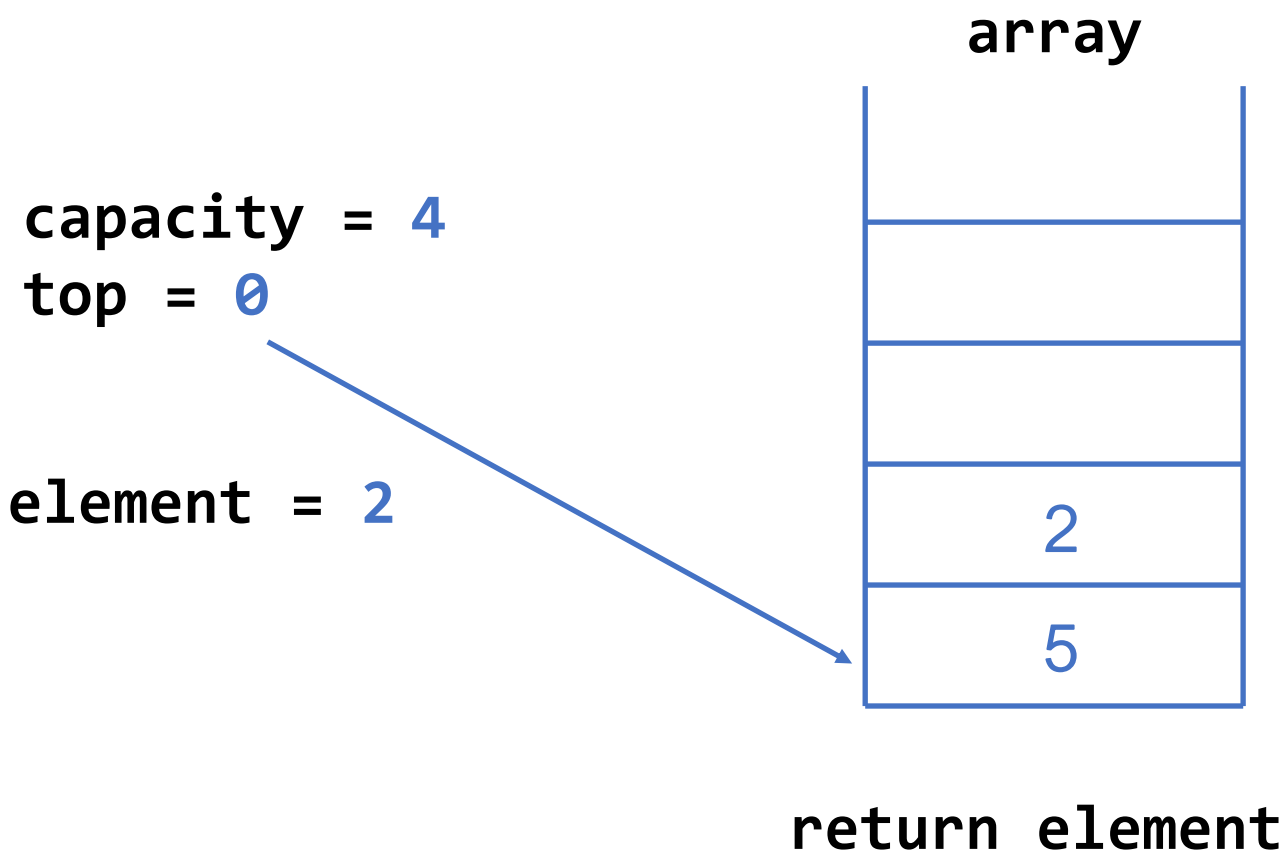
```
element = array[top];
```

Pilha em vetor



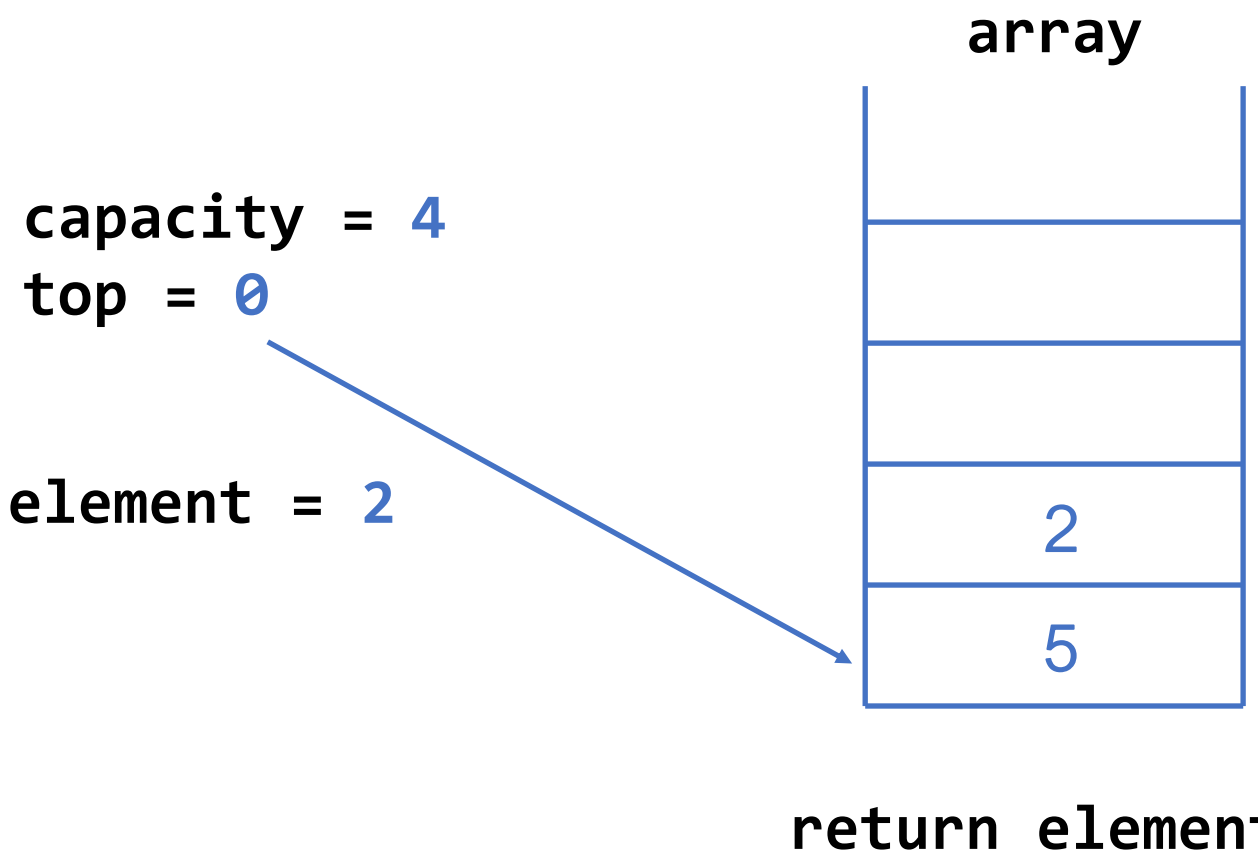

```
push(5);
push(2);
top();  → 2
pop();
```

Pilha em vetor




```
push(5);
push(2);
top();  → 2
pop();  → 2
```

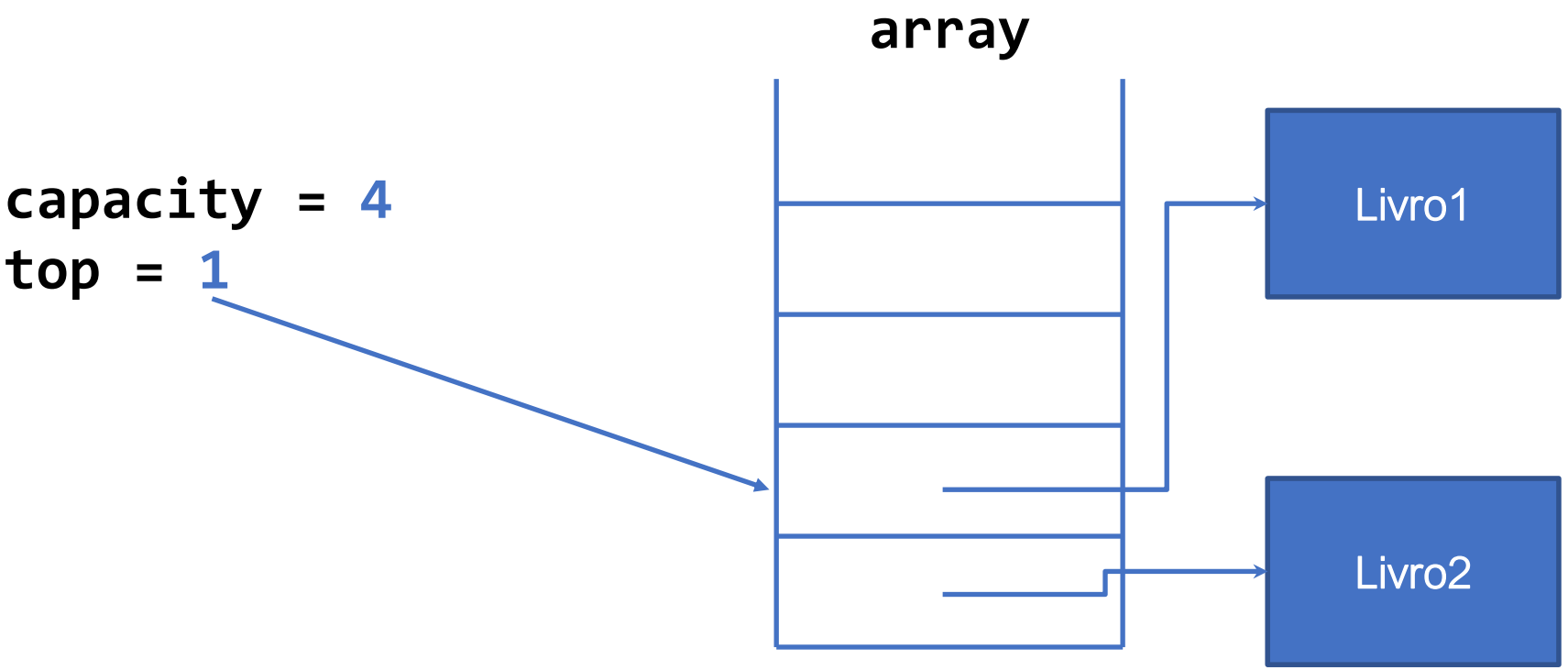
Pilha em vetor




```
push(5);
push(2);
top();  → 2
pop();  → 2
```

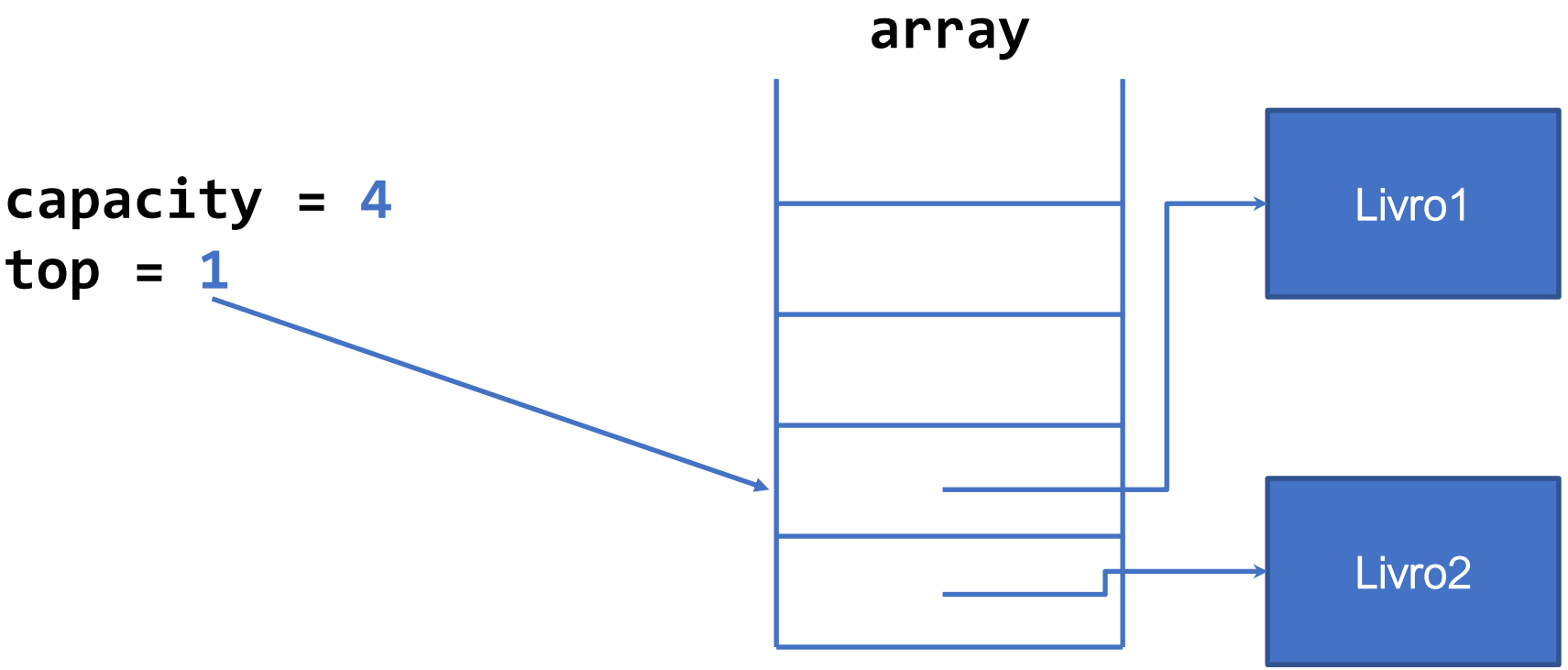
O que falta neste processo ?

Pilha em vetor (de livros)



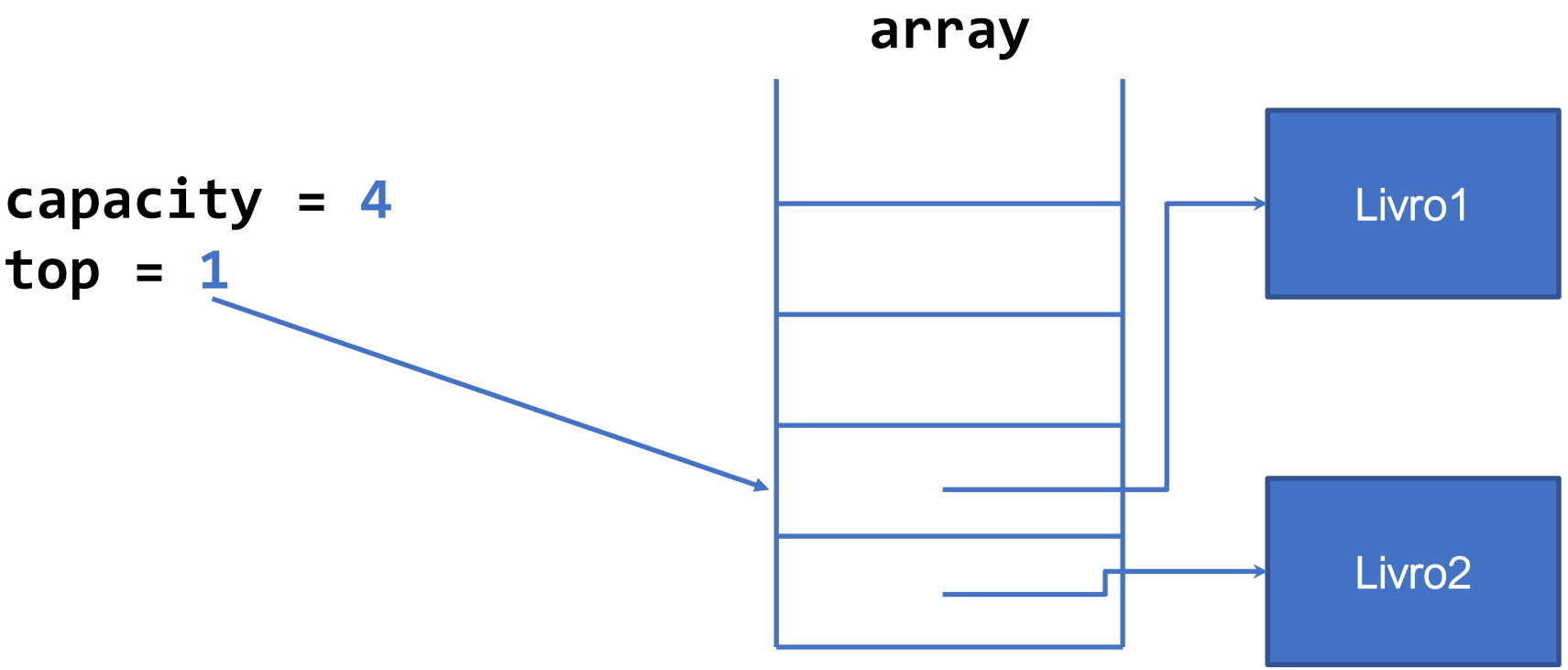

```
push(Livro2);
push(Livro1);
```

Pilha em vetor (de livros)




```
push(Livro2);
push(Livro1);
pop();
```

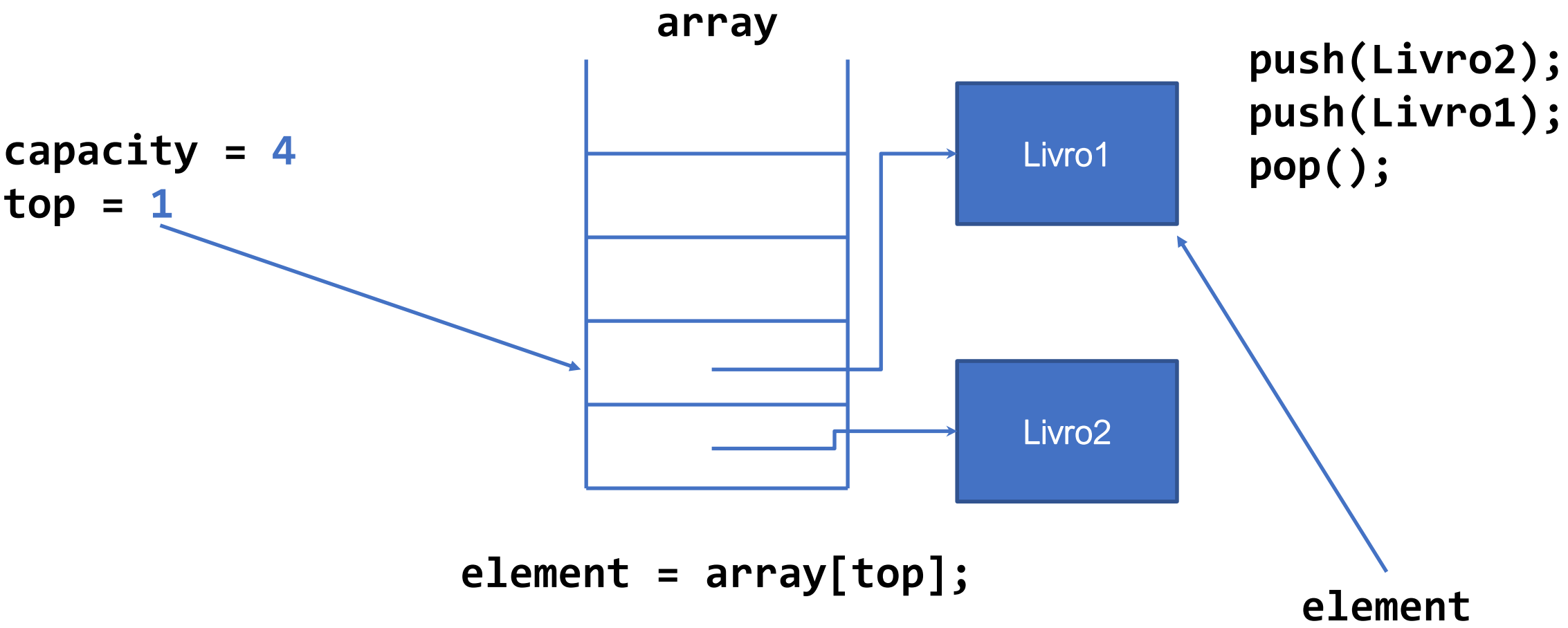
Pilha em vetor (de livros)



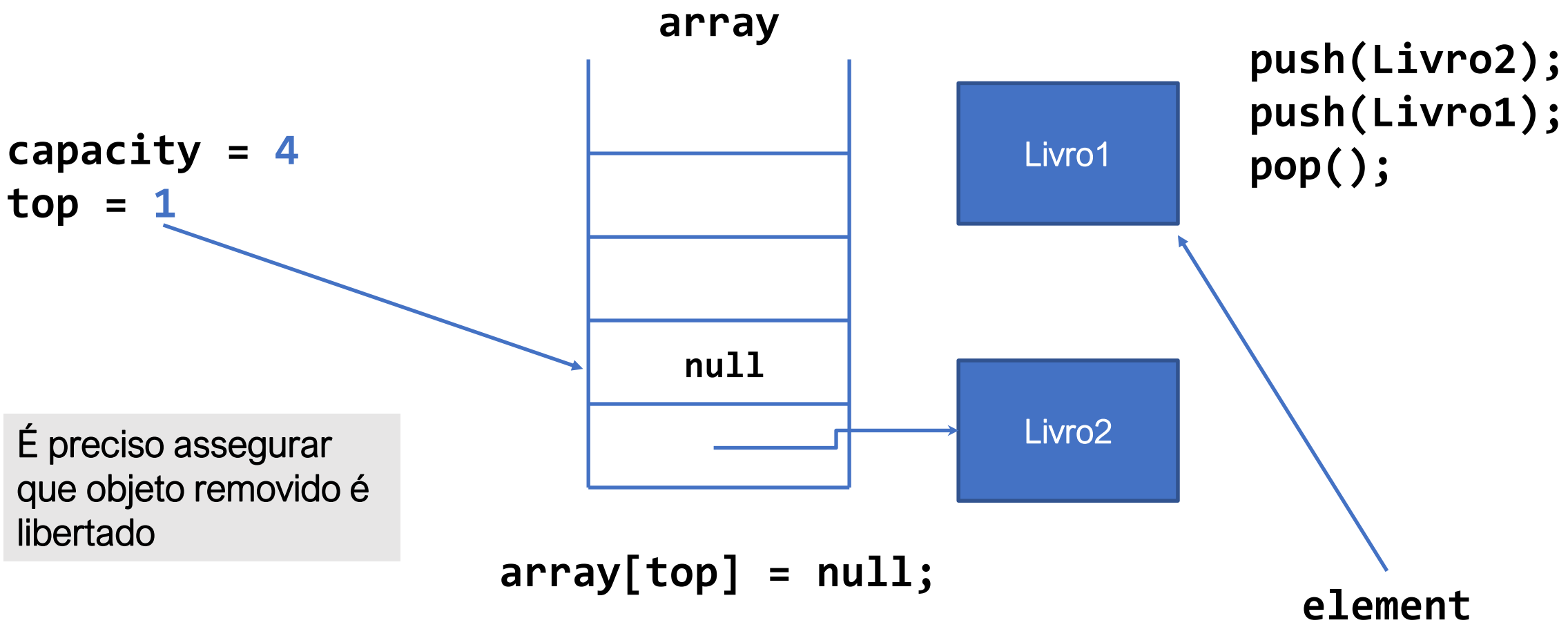

```
push(Livro2);
push(Livro1);
pop();
```

```
element = array[top];
```

Pilha em vetor (de livros)

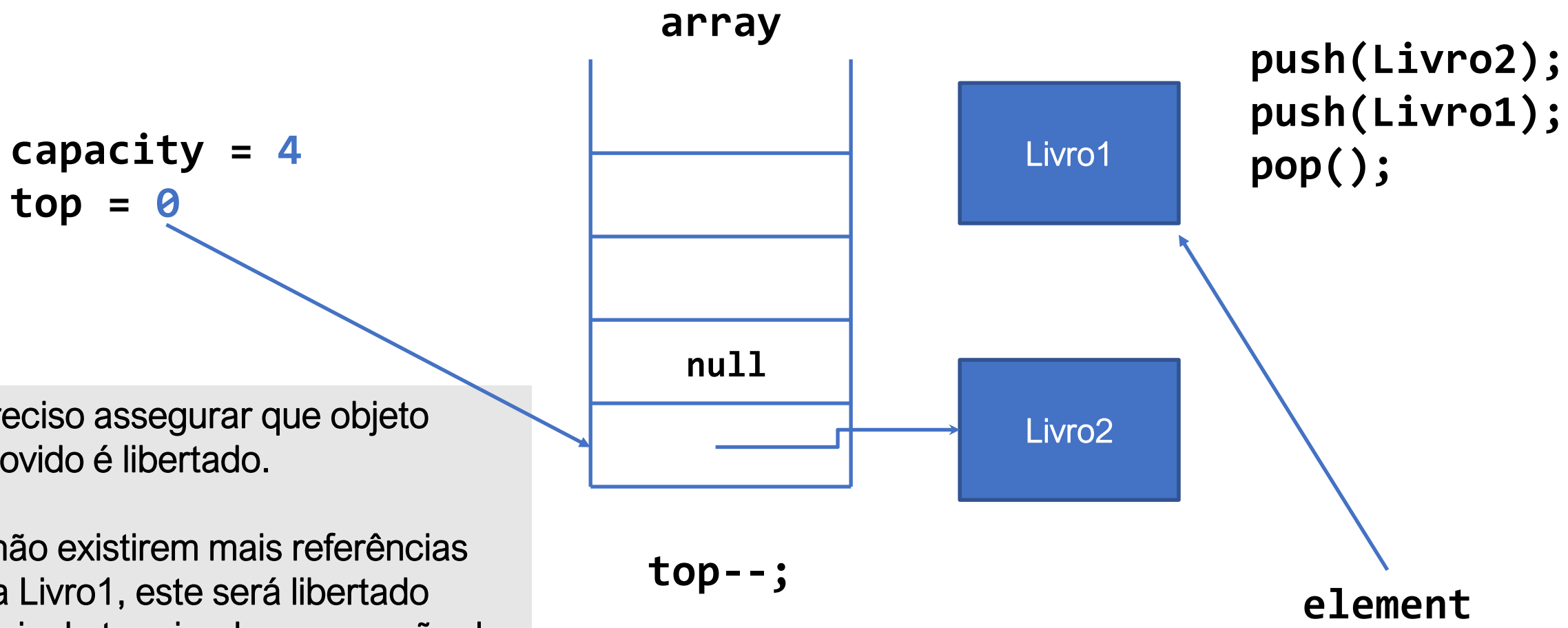


Pilha em vetor (de livros)



É preciso assegurar que objeto removido é libertado

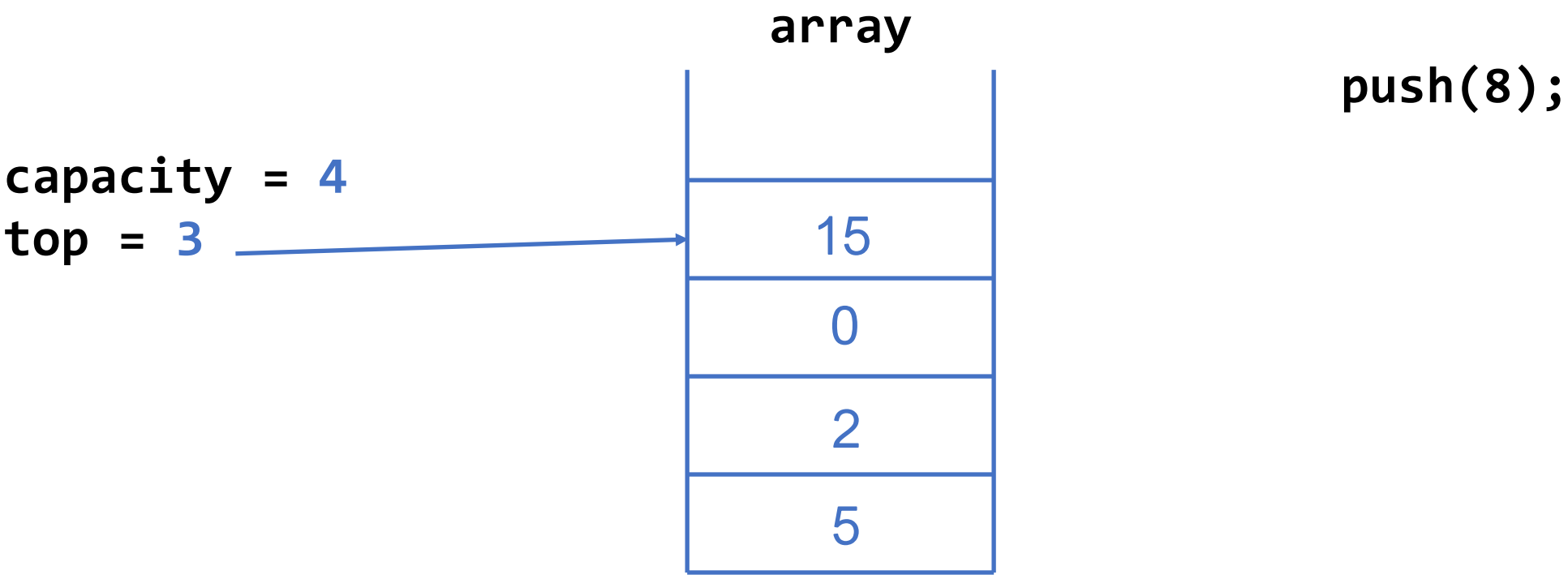
Pilha em vetor (de livros)



É preciso assegurar que objeto removido é libertado.

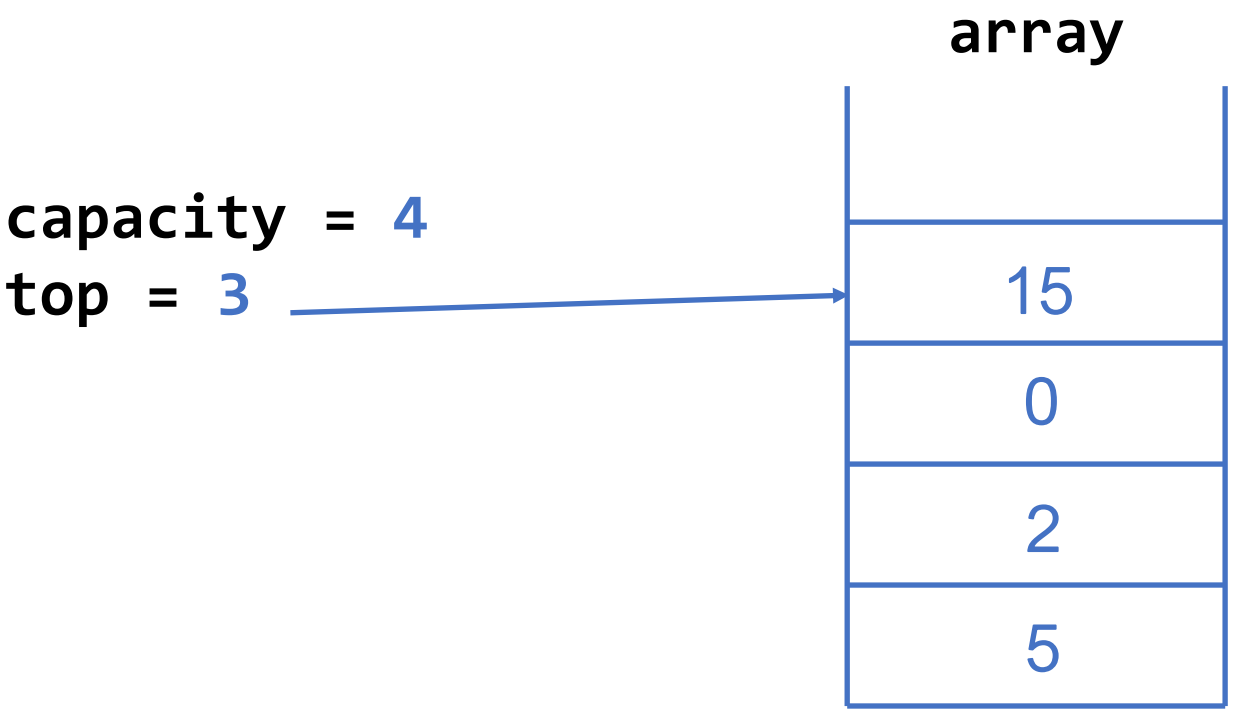
Se não existirem mais referências para Livro1, este será libertado depois de terminada a execução do método.

Pilha em vetor – vetor cheio




```
push(8);
```

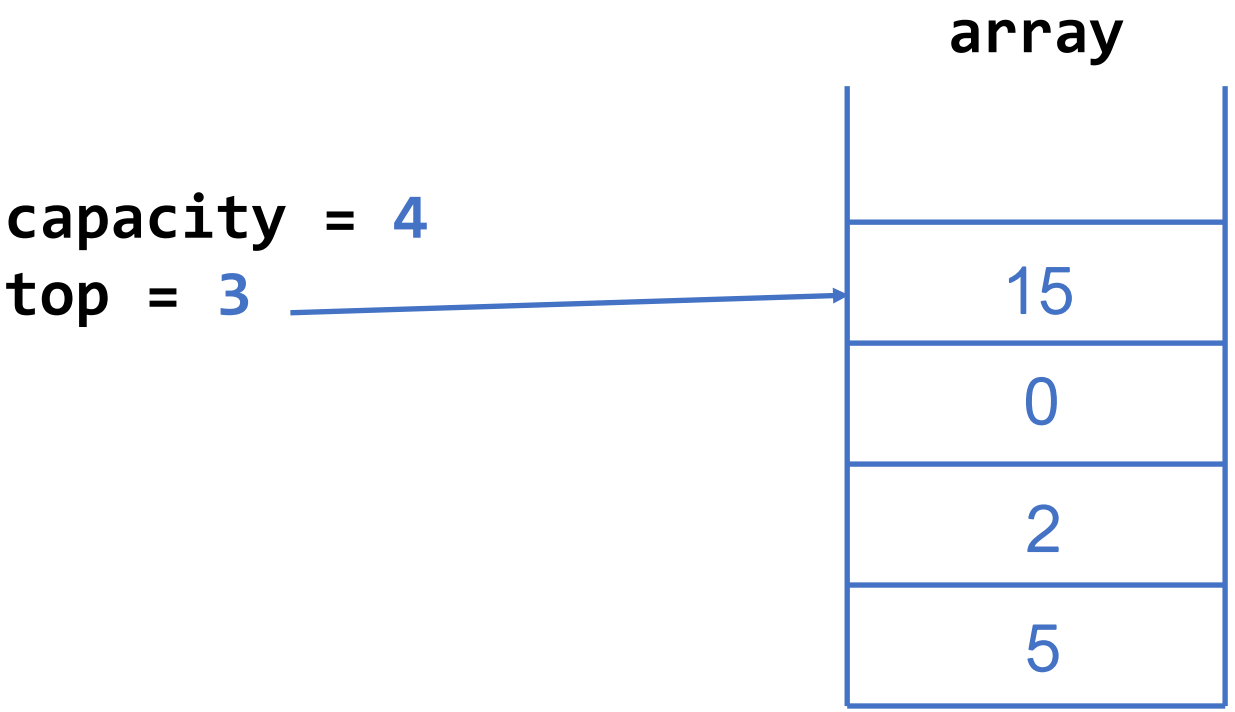
Pilha em vetor – vetor cheio




```
push(8);
```

O que acontece neste caso ?

Pilha em vetor – vetor cheio

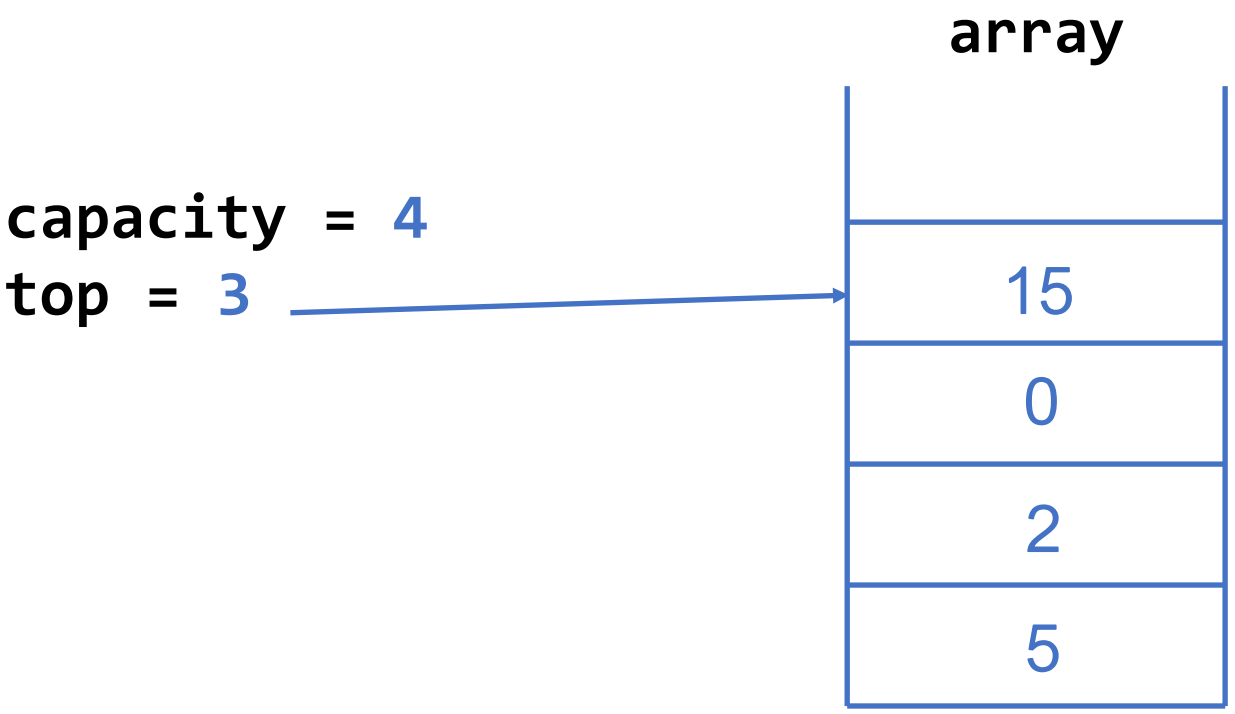



```
push(8);
```

O que acontece neste caso ?

Lançamento de exceção!!

Pilha em vetor – vetor cheio




```
push(8);
```

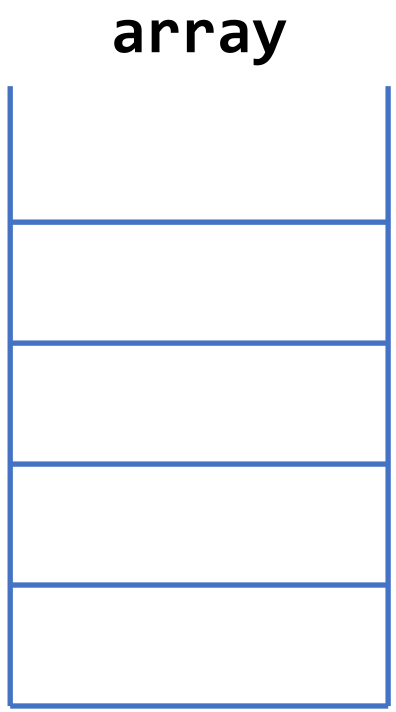
O que acontece neste caso ?

Lançamento de exceção!!

Esta exceção não faz parte do TAD Pilha, pois não está associada a uma pré-condição da operação do TAD, mas sim à implementação em vetor.

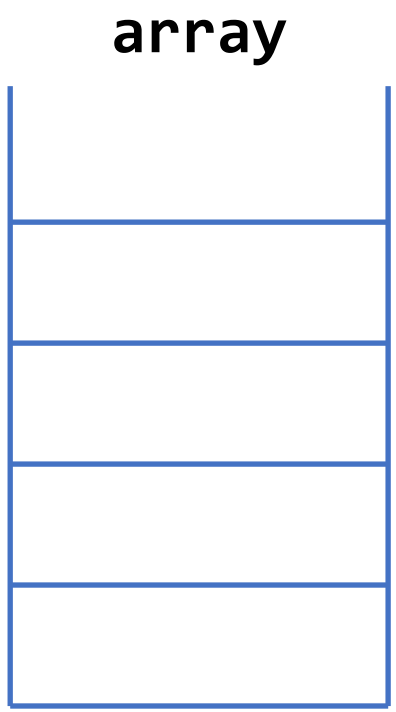
Pilha em vetor – vetor vazio

```
capacity = 4
top = -1
```



Pilha em vetor – vetor vazio

```
capacity = 4
top = -1
```

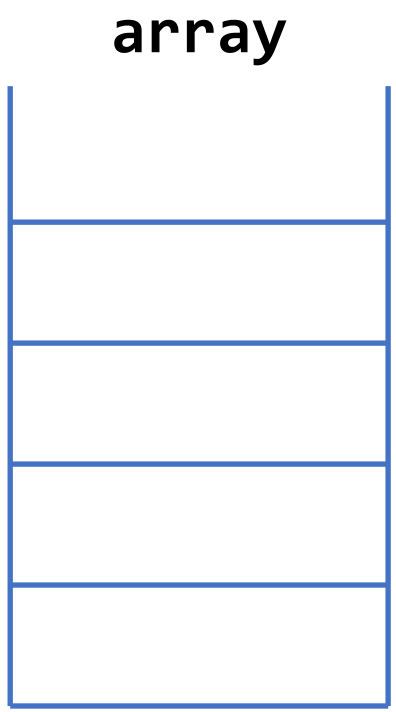



```
pop();
```

O que acontece neste caso ?

Pilha em vetor – vetor vazio

```
capacity = 4
top = -1
```




```
pop();
```

O que acontece neste caso ?

Lançamento de exceção!!

Esta exceção faz parte do TAD Pilha, e está associada a uma pré-condição da operação do TAD, pelo que deve ser considerada em qualquer implementação do TAD

Classe Pilha em Vetor (1)

```
package dataStructures;

public class StackInArray<E> implements Stack<E> {

   // Default capacity of the stack.
   public static final int DEFAULT_CAPACITY = 1000;

   // Memory of the stack: an array.
   protected E[] array;

   // Index of the element at the top of the stack.
   protected int top;
```

Classe Pilha em Vetor (2)

```
@SuppressWarnings("unchecked")
public StackInArray( int capacity ) {
   // Compiler gives a warning.
   array = (E[]) new Object[capacity];
   top = -1;
}

public StackInArray( ) {
   this(DEFAULT_CAPACITY);
}
```

O aviso vem do facto de não ser possível verificar, em tempo de compilação, os tipos dos dados que vão ser inseridos no vetor.

Para submissão no Mooshak, utilizar `@SuppressWarnings`

Classe Pilha em Vetor (3)

```
// Returns true iff the stack contains no elements.
public boolean isEmpty( ){
   return top == -1;
}

// Returns true iff the stack cannot contain more elements.
public boolean isFull( ){
   return this.size() == array.length;
}

// Returns the number of elements in the stack.
public int size( ){
   return top + 1;
}
```

Classe Pilha em Vetor (4)

```
// Returns the element at the top of the stack.
// Requires: size() > 0
public E top( ) throws EmptyStackException {
   if ( this.isEmpty() )
      throw new EmptyStackException();
   return array[top];
}

// Inserts the specified element onto the top of the stack.
// Requires: size() < array.length
public void push( E element ) throws FullStackException {
   if ( this.isFull() )
      throw new FullStackException();
   top++;
   array[top] = element;
}
```

Classe Pilha em Vetor (5)

```
 // Removes and returns the element at the top of the stack.
 // Requires: size() > 0
 public E pop( ) throws EmptyStackException {
   if ( this.isEmpty() )
      throw new EmptyStackException();
   E element = array[top];
   array[top] = null;  // For garbage collection.
   top--;
   return element;
 }
} // End of StackInArray.
```

Classes de Exceções da Pilha

```
package dataStructures;

public class EmptyStackException extends RuntimeException{
}

public class FullStackException extends RuntimeException{
}
```

RuntimeException: superclasse das excepções que podem acontecer durante a operação normal da máquina virtual Java. São excepções de aplicação e não são graves.

Estas excepções são *unchecked*. Não é necessário declará-las na cláusula **throws** dos métodos, mesmo que elas possam acontecer durante a execução dos mesmos. Podem propagar-se para fora do método, sendo capturadas no local certo onde isso deve acontecer.

Interface para TAD DevolucoesExemplar

Relembremos: Devoluções de exemplares de livros

- No momento da devolução de um exemplar na Biblioteca, a funcionária poderá não ter tempo para repor o livro juntamente com os exemplares disponíveis
- O processo normal será que todos os exemplares devolvidos fiquem guardados juntos, empilhados ao lado da receção da biblioteca
- Assim, o processo de devolução tem dois passos:
  - Receber o exemplar, o que implica inserir o exemplar numa pilha de exemplares a repor. A funcionária poderá receber até 10 devoluções antes de fazer, obrigatoriamente as reposições dos exemplares.
  - Mais tarde, com tempo, repor o exemplar no seu lugar. Neste caso, a funcionária irá, naturalmente, retirar o exemplar que está no topo da pilha de reposição, alterar a informação que lhe está associada (tornando-o disponível para empréstimo) e reinseri-lo no conjunto de cópias disponíveis.

Interface CopyReturn

```
package library;

public interface CopyReturn {

   int size();

   boolean isEmpty();

   //checks if the maximum number of returns has been reached
   boolean maxReturns();

   //copy is added do stack of books to return
   //Requires: !maxReturns()
   void receiveCopy(Copy copy) throws MaxReturnsException;

   //most recent copy to be returned is made available
   //Requires: !isEmpty()
   Copy returnCopy() throws NoReturnsException;

}
```

Tipo associado a Biblioteca (Library)

Implementação com Stack - CopyReturnClass (1)

```
package library;
import dataStructures.*;

public class CopyReturnsClass implements CopyReturn {
   public static final int MAX_RETURNS=10;
   protected Stack<Copy> returns;

   public CopyReturnsClass(){
      returns=new StackInArray<Copy>(MAX_RETURNS);
   }

   public int size(){
      return returns.size();
   }

   public boolean isEmpty(){
      return returns.isEmpty();
   }

   public boolean maxReturns(){
      return this.size()== MAX_RETURNS;
   }
```

Implementação com Stack - CopyReturnClass (1)

```
package library;
import dataStructures.*;

public class CopyReturnsClass implements CopyReturn {
   public static final int MAX_RETURNS=10;
   protected Stack<Copy> returns;

   public CopyReturnsClass(){
      returns=new StackInArray<>(MAX_RETURNS);
   }

   public int size(){
      return returns.size();
   }

   public boolean isEmpty(){
      return returns.isEmpty();
   }

   public boolean maxReturns(){
      return this.size()== MAX_RETURNS;
   }
```

Uso do operador diamante.

Para que serve ?

Implementação com Stack - CopyReturnClass (1)

```
package library;
import dataStructures.*;

public class CopyReturnsClass implements CopyReturn {
   public static final int MAX_RETURNS=10;
   protected Stack<Copy> returns;

   public CopyReturnsClass(){
      returns=new StackInArray<>(MAX_RETURNS);
   }

   public int size(){
      return returns.size();
   }

   public boolean isEmpty(){
      return returns.isEmpty();
   }

   public boolean maxReturns(){
      return this.size()== MAX_RETURNS;
   }
```

Uso do operador diamante.

Este operador permite inferir qual o tipo que será contido na criação do objeto `StackInArray`, a partir do tipo associado à variável `returns`.

Implementação com Stack - CopyReturnClass (2)

```
public boolean isFull() {
   return ((StackInArray<Copy>)returns).isFull();
}

//Requires: !this.maxReturns()
public void receiveCopy(Copy copy) throws
                  MaxReturnsException {
   if (this.maxReturns())
      throw new MaxReturnsException();
   returns.push(copy);
}


//Requires: !this.isEmpty()
public Copy returnCopy() throws NoReturnsException{
   if (this.isEmpty())
      throw new NoReturnsException();
   Copy c=returns.pop();
   ((LibrarianCopy)c).setSituation(true);
   return c;
}
```

Não é necessária na classe

Devia ser sempre possível ?

`returnCopy` actualiza o estado de `c` e devolve-o