JAVA STACK

É uma das estruturas de dados mais simples

A idéia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo.

Assim, quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo, e o único elemento que pode ser removido da pilha é o do topo.

Java Stack is a legacy Collection class.
It extends Vector class with five operations to support LIFO (Last In First Out). It is available in Collection API since Java 1.0.

As Vector implements List, Stack class is also a List implementation class but does NOT support all operations of Vector or List.
As Stack supports LIFO, it is also known as LIFO Lists.

Java Stack -> extends Vector

Java Stack é um objeto LIFO. Estende a classe Vector, mas suporta apenas cinco operações.

A classe Java Stack possui apenas um construtor vazio ou padrão.

Portanto, quando criamos uma pilha, inicialmente ela não contém itens que significam que a pilha está vazia.

A pilha internamente possui um ponteiro: TOP, que se refere à parte superior do elemento Stack.

Se Stack estiver vazio, TOP se refere ao local do primeiro elemento antes.

Se a pilha não estiver vazia, TOP se refere ao elemento top.

Operação de Stack

Os elementos da pilha são retirados na ordem inversa à ordem em que foram introduzidos: o primeiro que sai é o último que entrou (LIFO – last in, first out)

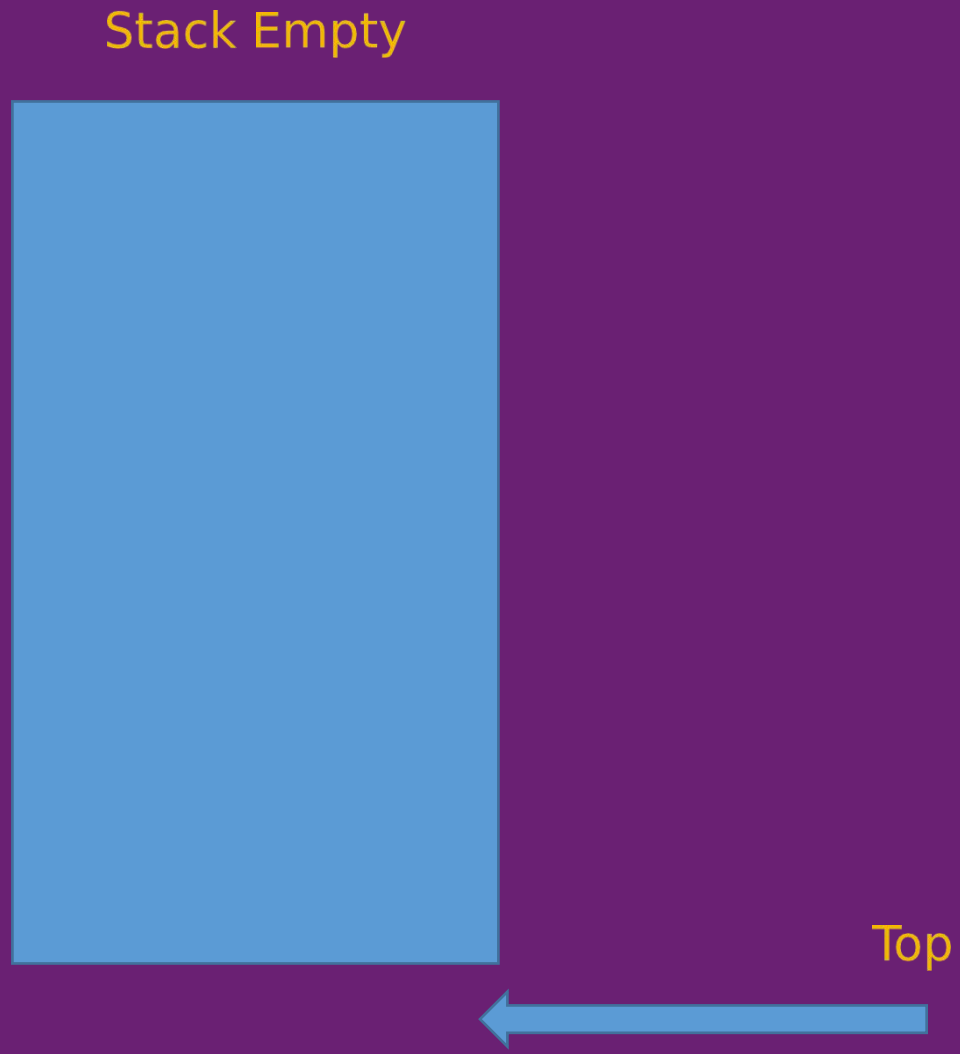
Existem duas operações básicas que devem ser implementadas numa estrutura de pilha:

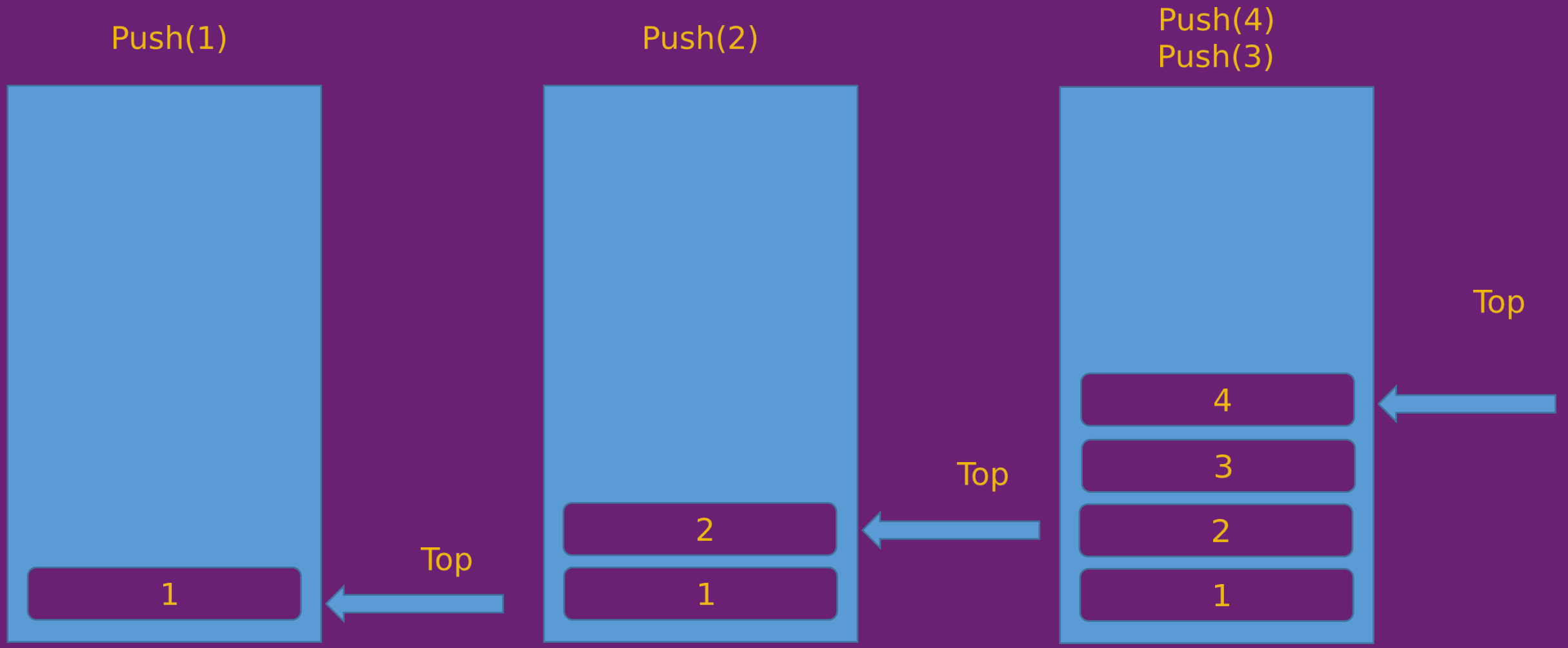
operação para empilhar (push) um novo elemento, inserindo-o no topo,

operação para desempilhar (pop) um elemento, removendo-o do topo

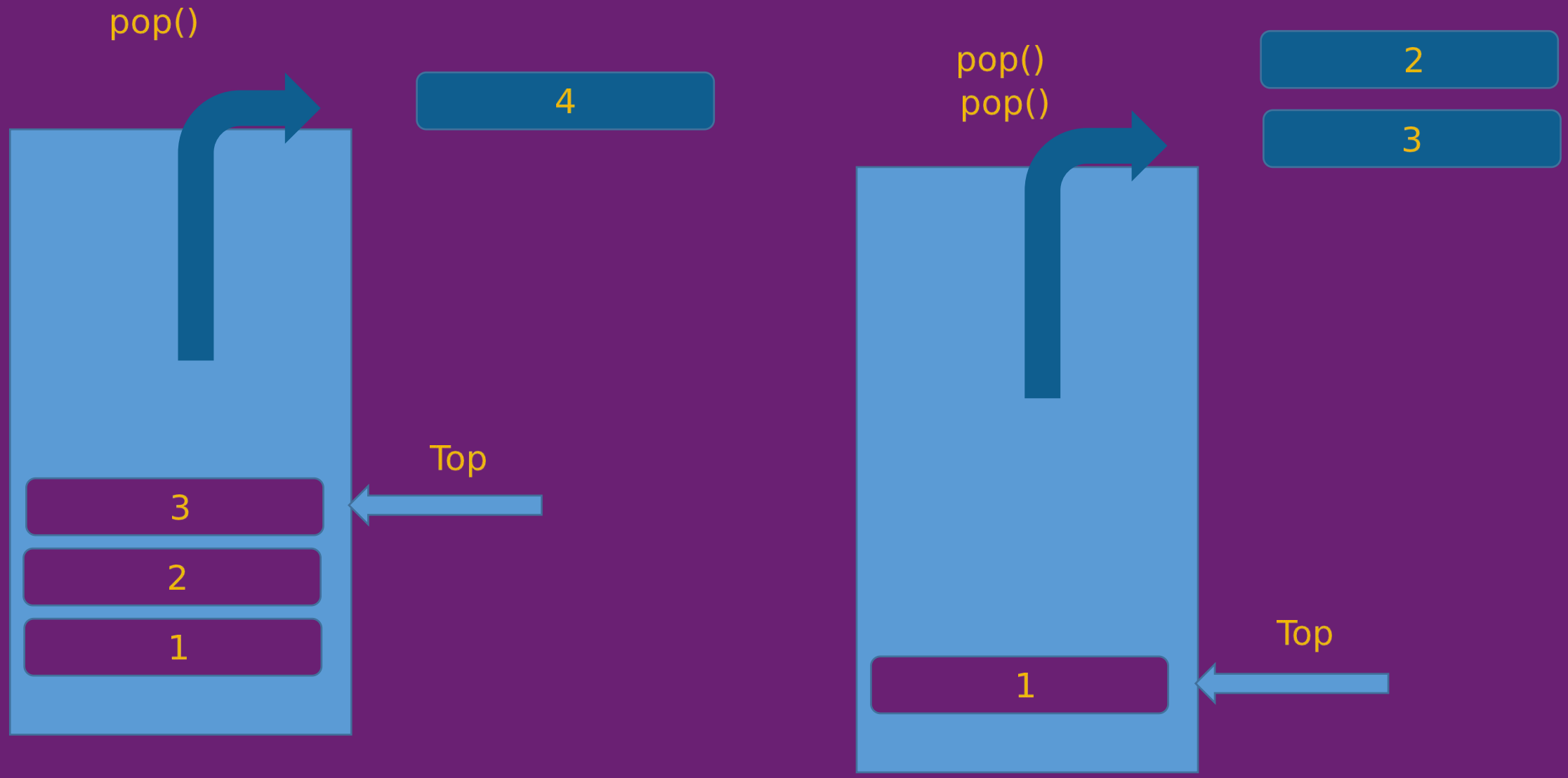
Java

Stack Empty





Operação de Stack Push



Operação de Stack Pop()

JAVA STACK

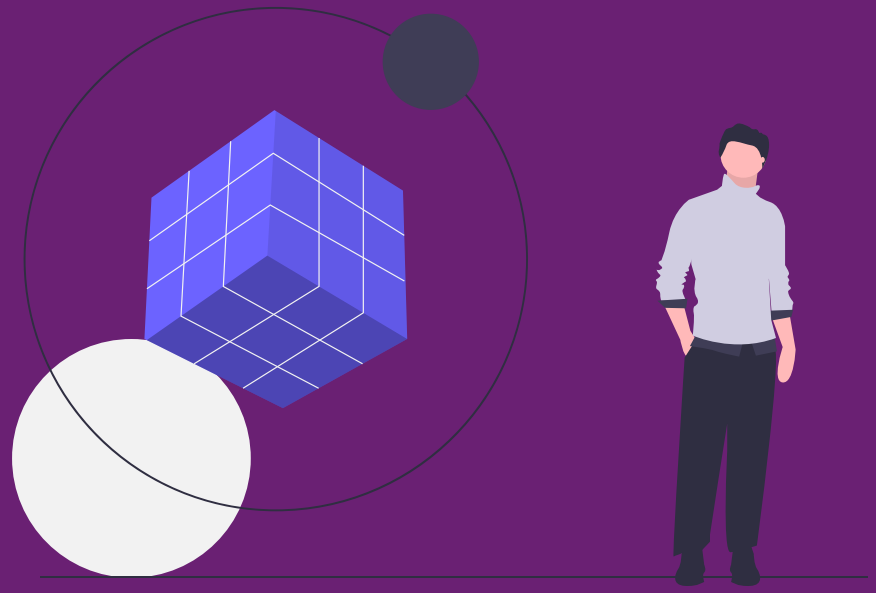
```
import java.util.Stack;

public class StackBasicExample {
    public static void main(String a[]){
        Stack<Integer> stack = new Stack<>();
        System.out.println("Empty stack : "  + stack);
        System.out.println("Empty stack : "  + stack.isEmpty());
        // Exception in thread "main" java.util.EmptyStackException
        // System.out.println("Empty stack : Pop Operation : "  + stack.pop());
        stack.push(1001);
        stack.push(1002);
        stack.push(1003);
        stack.push(1004);
        System.out.println("Peek : "  + stack.peek());
        System.out.println("Non-Empty stack : "  + stack);
        System.out.println("Non-Empty stack: Pop Operation : "  + stack.pop());
        System.out.println("Non-Empty stack : After Pop Operation : "  + stack);
        System.out.println("Non-Empty stack : search() Operation : "  + stack.search(1002));
        System.out.println("Non-Empty stack : "  + stack.isEmpty());
    }
}
```

ARRAY PARA STACK

```
import java.util.Stack;

public class ArrayToStackExample {
    public static void main(String a[]){
        Integer[] intArr = { 1001,1002,1003,1004};
        Stack<Integer> stack = new Stack<>();
        for(Integer i : intArr){
            stack.push(i);
        }
        System.out.println("Non-Empty stack : "  + stack);
    }
}
```



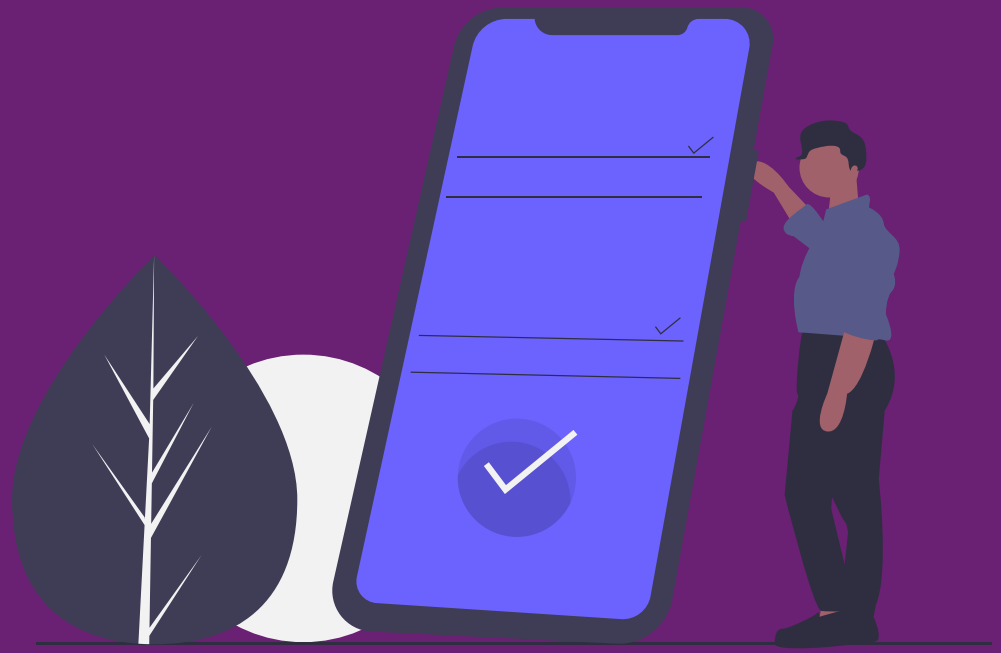
ArrayList para Stack

```
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class ListToStackExample {
    public static void main(String a[]){
        Stack<Integer> stack = new Stack<>();
        List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);

        System.out.println("Non-Empty stack addAll Operation : "  + stack.addAll(list));
        System.out.println("Non-Empty stack : "  + stack);
    }
}
```

Stack para ArrayList



```
iimport java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class StackBasicExample {
    public static void main(String a[]){
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);

        List<Integer> list = new ArrayList<>();
        list.addAll(stack);

        System.out.println("Non-Empty stack : "  + stack);
        System.out.println("Non-Empty List : "  + list);
    }
}
```

Queue em Java

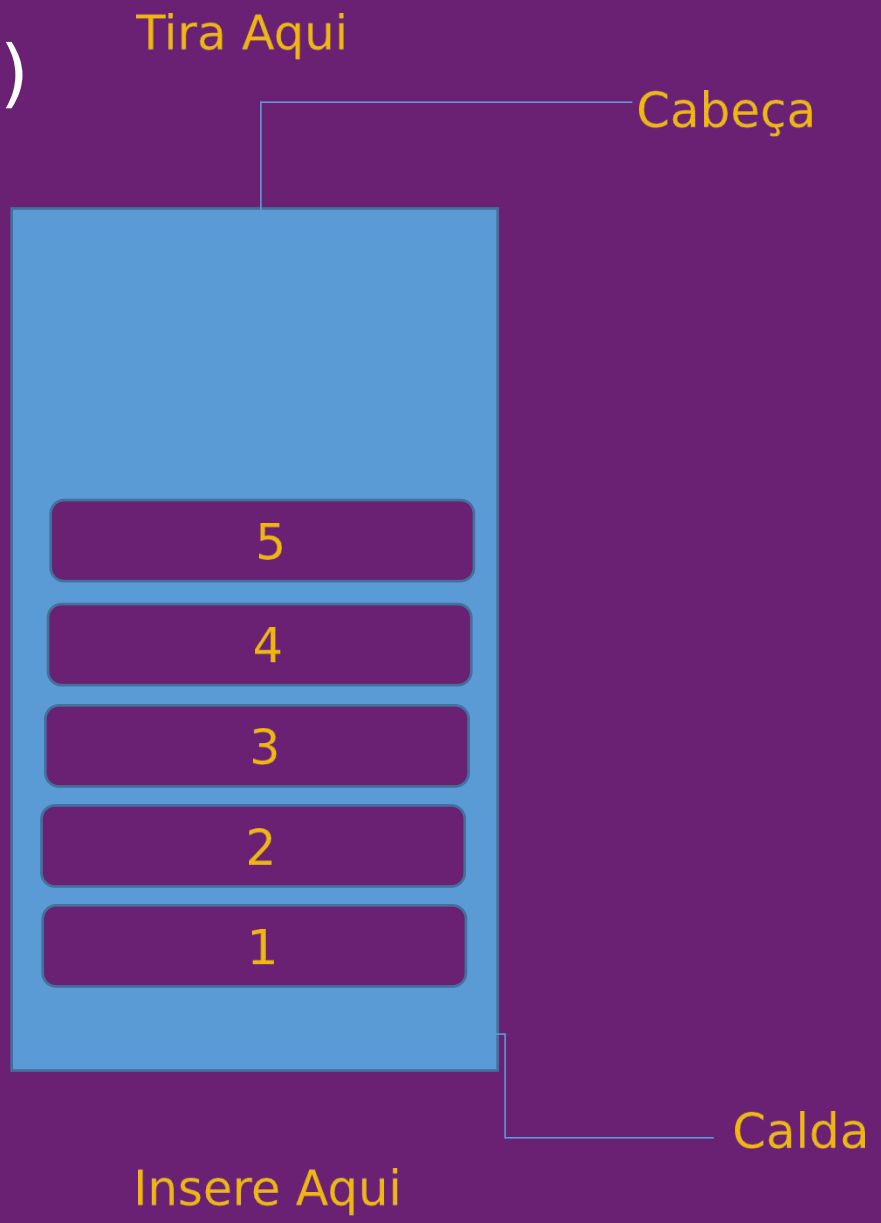
Queue In Java

Uma fila é uma estrutura de dados que segue o princípio do FIFO (primeiro a entrar, primeiro a sair), os elementos são inseridos no final da lista e excluídos do início da lista.
Essa interface está disponível no java.util.package e estende a Interface de coleção.

A fila suporta vários métodos, incluindo inserção e exclusão.
As filas disponíveis no java.util.package são conhecidas como Filas não ligadas,
enquanto as filas presentes no pacote java.util.concurrent são conhecidas, são Filas vinculadas.

Todas as filas, exceto o Deques, suportam a inserção no final e a exclusão pela frente.
O Deques suporta a inserção e exclusão de elementos nas duas extremidades.

Operação de Queue Pop()



Queue em Java

Implementação da fila Java

Para usar a interface da fila, precisamos instanciar uma classe concreta.
A seguir, estão as poucas implementações que podem ser usadas:

util.LinkedList
util.PriorityQueue

Como essas implementações não são seguras para threads, o PriorityBlockingQueue atua como uma alternativa para implementação segura de threads.

Exemplo 1

```
Queue q1 = new LinkedList();
Queue q2 = new PriorityQueue();
```

Queue em Java

Métodos na fila Java

add (): o método add () é usado para inserir elementos no final ou na cauda da fila. O método é herdado da interface Collection.

offer (): o método offer () é preferível ao método add (), pois insere o especificado elemento na fila sem violar nenhuma restrição de capacidade.

peek (): o método peek () é usado para examinar a frente da fila sem removê-la. Se a fila estiver vazia, ele retornará um valor nulo.

element (): se a fila estiver vazia, o método lançará NoSuchElementException.

remove (): o método remove () remove a frente da fila e a retorna. Lança NoSuchElementException se a fila estiver vazia.

poll (): o método poll () remove o início da fila e o retorna.

Se a fila estiver vazia, ele retornará um valor nulo.

Queue em Java

```
import java.util.*;

public class MainQueue {
	public static void main(String[] args) {
		Queue<String> q1 = new LinkedList<String>();
		q1.add("I");
		q1.add("Love");
		q1.add("Rock");
		q1.add("And");
		q1.add("Roll");
		System.out.println("Elements in Queue:" + q1);
		System.out.println("Removed element: " + q1.remove());
		System.out.println("Head: " + q1.element());
		System.out.println("poll(): "+q1.poll());
		System.out.println("peek(): "+q1.peek());
		System.out.println("Elements in Queue:"+q1);

	}
}
```

Maps em Java

Quando você instancia um HashMap a sua capacidade inicial é 16, ou seja, você consegue inserir até 16 elementos no Map, sem a necessidade de criar novas posições.

Caso você deseje, também pode instanciar um HashMap com mais ou menos de 16 posições, fica a seu critério e análise.

O atributo Load Factor está intrinsecamente ligado ao tamanho do HashMap, não é a toa que estamos explicando os dois juntos. O load factor é um atributo que mensura em que momento

o HashMap deve dobrar seu tamanho, ou seja, antes que você possa preencher as 16 posições, em algum momento o tamanho do HashMap irá dobrar de 16 para 32, vamos ver como na

Maps em Java

```
import java.util.HashMap;
import java.util.Map;

public class ExemploHashMap {

  public static void main (String args[]){

   Map<String,String> example = new HashMap<String,String>();
  example.put( "K1", new String( "V1" ));example.put( "K2", new String( "V2" )); example.put( "K3", new String( "V3" ));
  example.put( "K4", new String( "V4" )); example.put( "K5", new String( "V5" )); example.put( "K6", new String( "V6" ));
  example.put( "K7", new String( "V7" )); example.put( "K8", new String( "V8" )); example.put( "K9", new String( "V9" ));
  example.put( "K10", new String( "V10" )); example.put( "K11", new String( "V11" ));         example.put( "K12", new String( "V12" ));
  /*
   * LIMITE DE INSERÇÃO.
   * Aqui é o limite de acord com o load factor, ou seja,
   * quando o elemento 13 for inserido ocorrerá um Rehash na nossa lista.
   * */

   example.put( "K13", new String( "V13" ));

   System.out.println("Rehash ocorrendo agora ! Nosso HashMap terá tamanho igual a 32 a partir daqui");

  example.put( "K14", new String( "V14" ));
  example.put( "K15", new String( "V15" ));
  example.put( "K16", new String( "V16" ));

  }
}
```

Maps em Java

```
import java.util.HashMap;
import java.util.Map;


public class ExemploHashMap {

	public static void main (String args[]){

		Map<String,String> example = new HashMap<String,String>();
		example.put( "K1", new String( "V1" ));
		example.put( "K2", new String( "V2" ));
		example.put( "K3", new String( "V3" ));
		example.put( "K4", new String( "V4" ));
		example.put( "K5", new String( "V5" ));

		String keyToSearch = "K1";

		if ( example.containsKey( keyToSearch ) ) {
		 System.out.println("Valor da Chave "+keyToSearch+
		  " = "+example.get(keyToSearch));
		}else{
			System.err.println("Chave não existe");
		}
	}
}
```

Maps em Java

```
import java.util.HashMap;
import java.util.Map;

public class ExemploHashMap {

	public static void main(String args[]) {

		Map<String, String> example = new HashMap<String, String>();

		example.put("K1", new String("V1"));
		example.put("K2", new String("V2"));
		example.put("K3", new String("V3"));
		example.put("K4", new String("V4"));
		example.put("K5", new String("V5"));

		/*
		 * O método "keySet()" retorna um Set com todas as chaves do
		 * nosso HashMap, e tendo o Set com todas as Chaves,
		  * podemos facilmente pegar
		 * os valores que desejamos
		 * */

		for (String key : example.keySet()) {

			//Capturamos o valor a partir da chave
			String value = example.get(key);
			System.out.println(key + " = " + value);
		}

	}

}
```

Maps em Java

```
import java.util.HashMap;
import java.util.Map;

public class ExemploHashMap {

	public static void main(String args[]) {

		Map<String, String> example = new HashMap<String, String>();

		example.put("K1", new String("V1"));
		example.put("K2", new String("V2"));
		example.put("K3", new String("V3"));
		example.put("K4", new String("V4"));
		example.put("K5", new String("V5"));

		/*
		 * O método "keySet()" retorna um Set com todas as chaves do
		 * nosso HashMap, e tendo o Set com todas as Chaves,
		  * podemos facilmente pegar
		 * os valores que desejamos
		 * */

		for (String key : example.keySet()) {

			//Capturamos o valor a partir da chave
			String value = example.get(key);
			System.out.println(key + " = " + value);
		}

	}

}
```

THANKS