



**Senac**

# Estrutura de Dados

**Curso Tecnologia em Análise de Desenvolvimento de Sistemas**

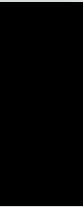
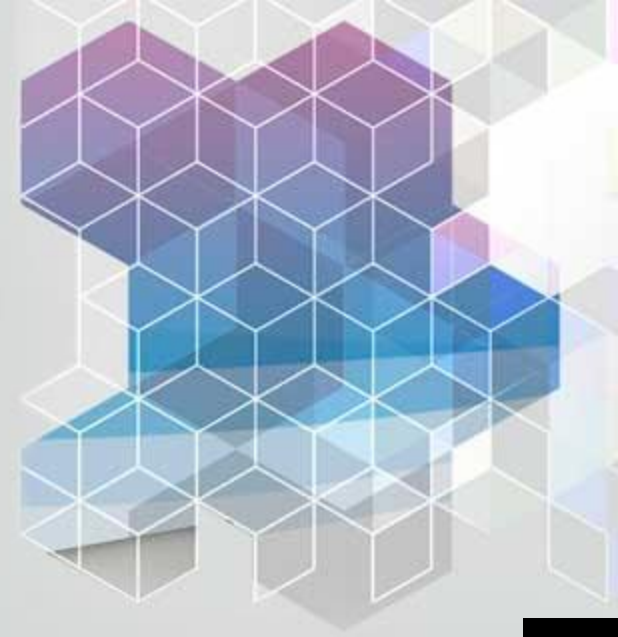
**Aula 04**

**Prof. Claudio Benossi**



# 1. Unidade

**Vetores, Matrizes, TAD e  
Alocação Dinâmica de  
Memória**



# Listas e Pilhas (Stacks)

Vamos falar hoje sobre Lista e Pilhas (Stacks).

Para muitas aplicações é necessário impor restrições de acesso aos dados armazenados em uma determinada estrutura de dados.



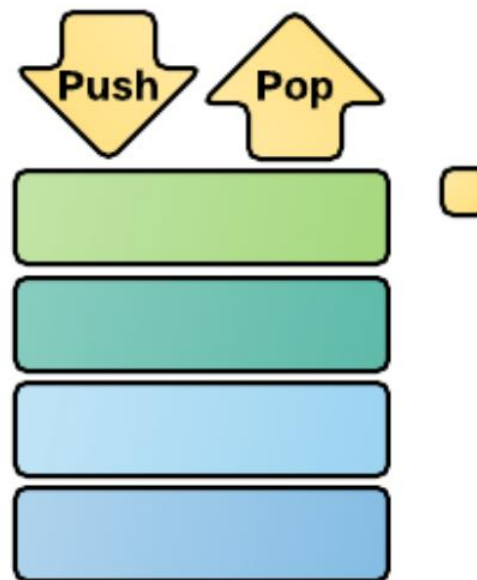
# Listas e Pilhas (Stacks)

Essas restrições podem vir a ser benéficas por:

- ✓ Aliviar a necessidade de usar estruturas com mais detalhes.
- ✓ Permitem implementações mais simples e flexíveis, já que menos operações são suportadas.

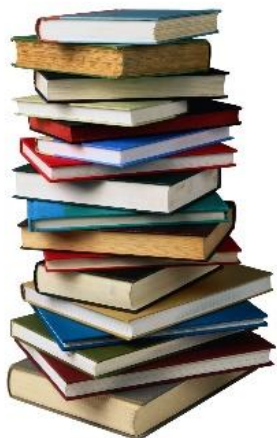
PILHAS - Last In First Out (LIFO).

FILAS - First In First Out (FIFO).



# Listas e Pilhas (Stacks)

Como uma pilha funciona como estrutura de dados?



Fazendo uma analogia com livros, pratos ou toalhas, podemos escolher qual nós queremos, afinal estão todos no mesmo local, porém um sobre o outro.

# Listas e Pilhas (Stacks)

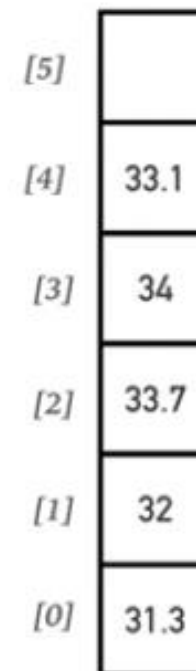
Com relação a estrutura de dados podemos ter:

*Vetor, Array, Lista*



*tamanho = 5*

*Pilha*



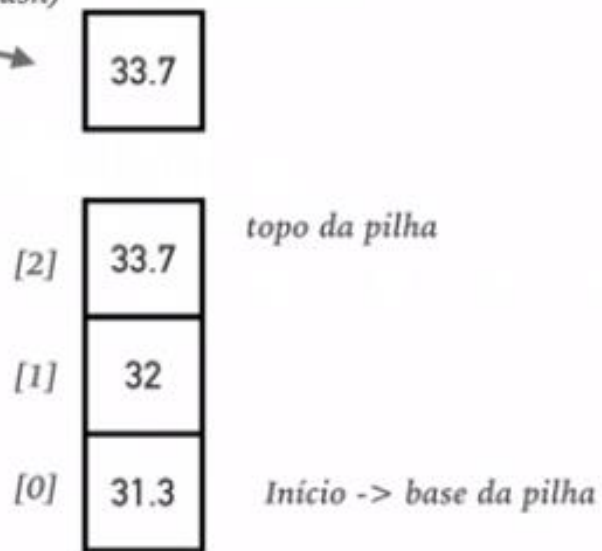
*Início -> base da pilha*

# Listas e Pilhas (Stacks)

Como funciona a Pilha:

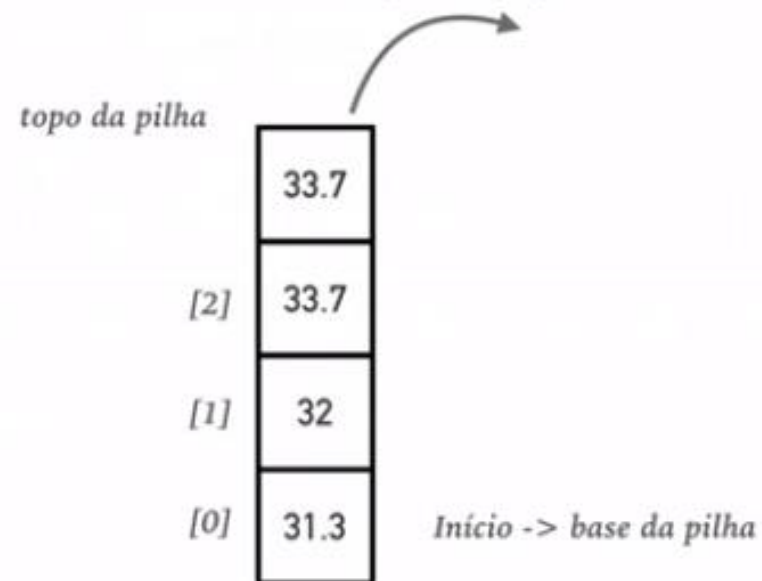
*Pilha*

empilhar (push)



*Pilha*

desempilhar (pop)



**LIFO – Last In First Out** (Último a entrar, primeiro a sair).



# Listas e Pilhas (Stacks)

Com relação ao comportamento da pilha em relação a estrutura de dados temos dois itens importantes que são o push e pop, quando precisamos empilhar adicionamos elementos no topo da pilha, ou seja no final do vetor e quando precisamos desempilhar, nos removemos o ultimo elemento no topo da pilha.



**LIFO** – Last In First Out (Último a entrar, primeiro a sair).






# Listas e Pilhas (Stacks)

Com relação ao comportamento da pilha em relação a estrutura de dados temos dois itens importantes que são o push e pop, quando precisamos empilhar adicionamos elementos no topo da pilha, ou seja no final do vetor e quando precisamos desempilhar, nos removemos o ultimo elemento no topo da pilha.

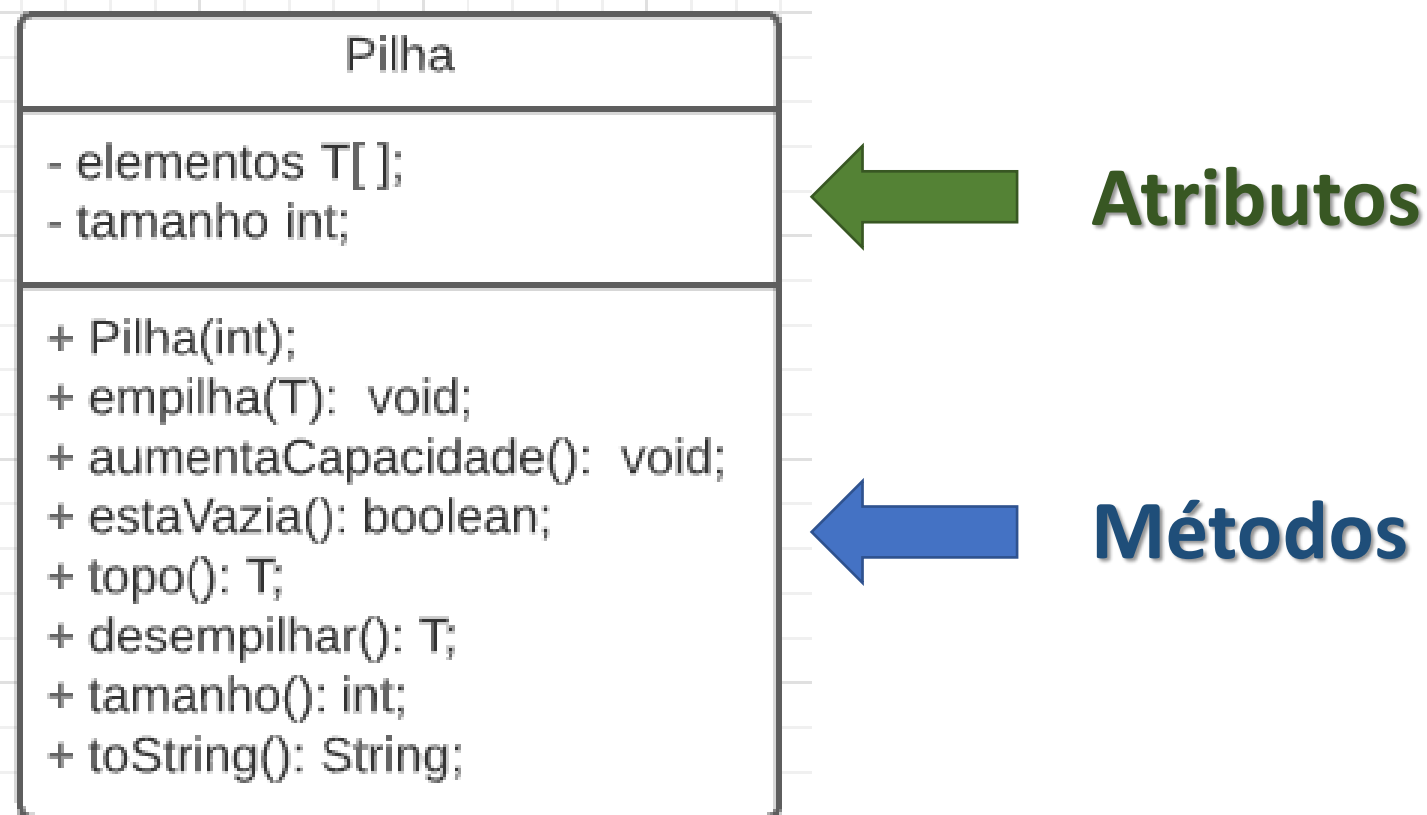
LIFO – Last In First Out (Último a entrar, primeiro a sair).

Para entender melhor esse assunto, vamos realizar as seguintes tarefas:



# Listas e Pilhas (Stacks)

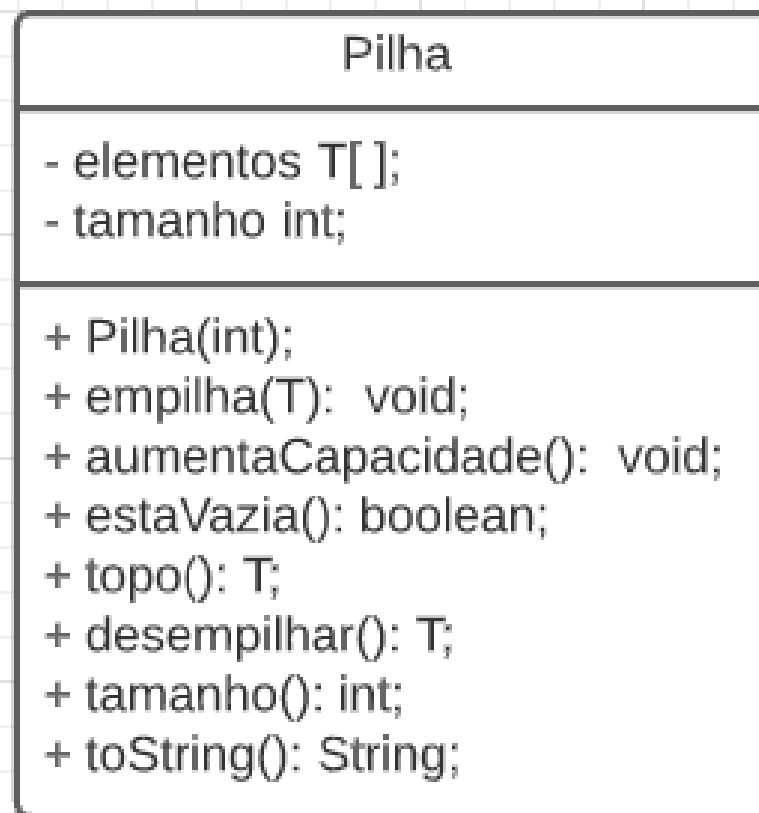
Para exemplificar o conceito de Pilha, vamos seguir a estrutura da classe abaixo:

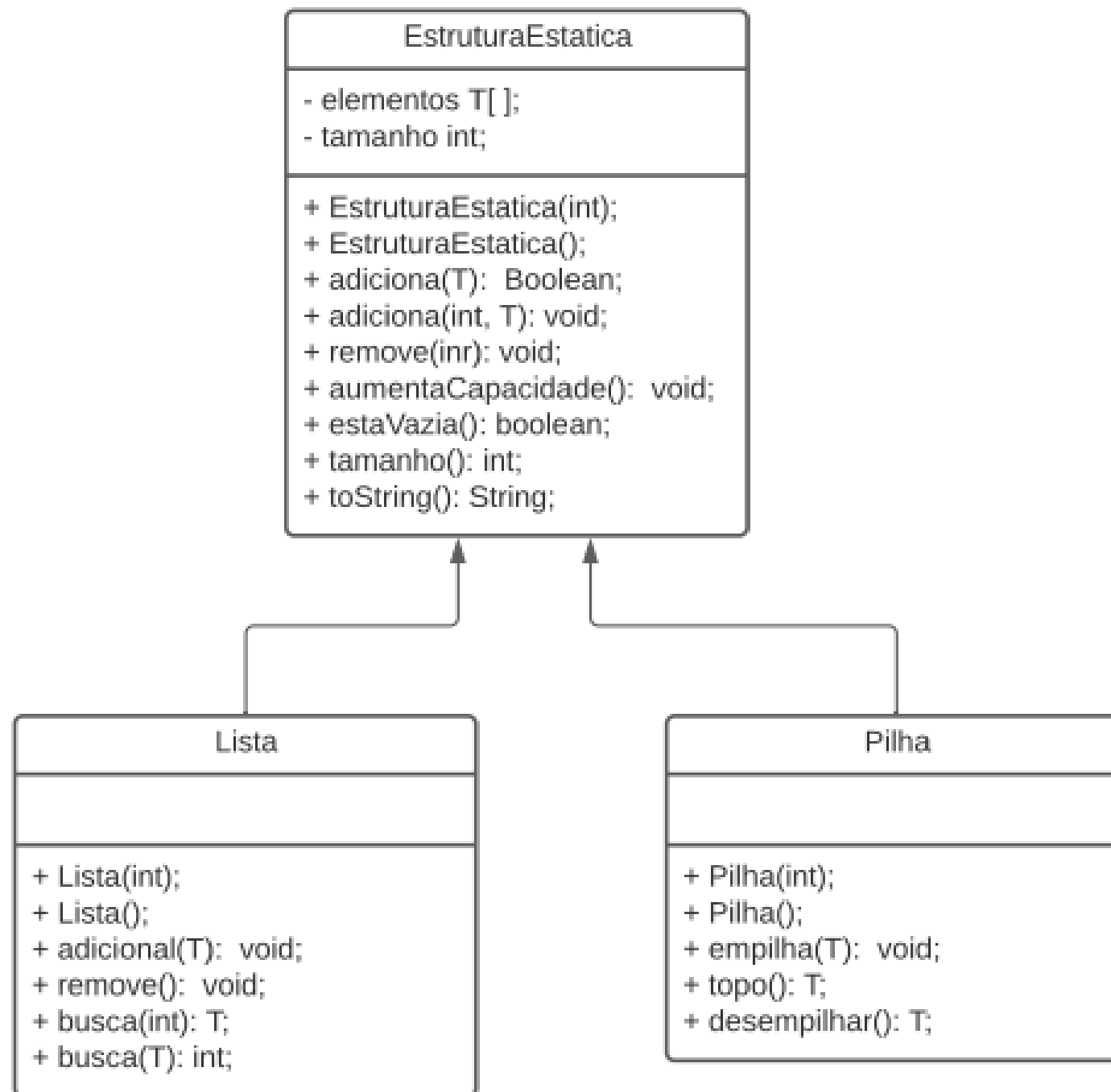


# Listas e Pilhas (Stacks)

**Obs.:** Vamos criar nosso vetor de forma generics “elementos T[ ]”, ou seja, pode ser declarado como qualquer objeto.

Porém para aproveitar nosso projeto e já falar também sobre Listas, vamos trabalhar com um conceito de **herança** já que Pilha e Lista **compartilham** alguns atributos e métodos:







# Listas e Pilhas (Stacks)

Vamos começar com a nosso código, primeiro a classe **EstruturaEstatica**, lembrando que vamos reutilizar alguns conceitos vistos na aula anterior sobre vetor.




```
1 package estruturaestatica;
2
3 public class EstruturaEstatica<T> {
4     public T[] elementos;
5     public int tamanho;
6     // método construtor com parâmetro
7     public EstruturaEstatica(int capacidade){
8         this.elementos = (T[]) new Object[capacidade];
9         this.tamanho = 0;
10    }
11    // método Construtor sem parâmetro
12    public EstruturaEstatica(){
13        this(10);
14    }
15    // método para adicionar elemento
16    public boolean adiciona(T elemento){
17        this.aumentaCapacidade();
18        if(this.tamanho < this.elementos.length){
19            this.elementos[this.tamanho] = elemento;
20            this.tamanho++;
21            return true;
22        }
23        return false;
24    }
```

```
25 // método para adicionar em qualquer posição
26 public boolean adiciona(int posicao, T elemento){
27     if(!(posicao >=0 && posicao < tamanho)){
28         throw new IllegalArgumentException("Posição Inválida");
29     }
30     this.aumentaCapacidade();
31     for(int i=this.tamanho-1; i>=posicao; i--){
32         this.elementos[i+1] = this.elementos[i];
33     }
34     this.elementos[posicao] = elemento;
35     this.tamanho++;
36
37     return true;
38 }
39 // método para aumentar a capacidade do vetor
40 public void aumentaCapacidade() {
41     if(this.tamanho == this.elementos.length){
42         T[] elementosNovos = (T[]) new Object[this.elementos.length];
43         for (int i=0; i<this.elementos.length; i++){
44             elementosNovos[i] = this.elementos[i];
45         }
46         this.elementos = elementosNovos;
47     }
48 }
```

```
49
50 public int tamanho() {
51     return this.tamanho;
52 }
53
54 @Override
55 public String toString() {
56     StringBuilder s = new StringBuilder();
57     s.append("[");
58
59     for(int i=0; i<this.tamanho-1; i++){
60         s.append(this.elementos[i]);
61         s.append(", ");
62     }
63
64     if(this.tamanho> 0){
65         s.append(this.elementos[this.tamanho-1]);
66     }
67
68     s.append("]");
69
70     return s.toString();
71 }
72
73 public boolean estaVazia(){
74     return this.tamanho == 0;
75 }
```






```
76  
77 ☐ public void remove(int posicao){  
78     if(!(posicao >=0 && posicao < tamanho)){  
79         throw new IllegalArgumentException("Posição Inválida");  
80     }  
81     for (int i=posicao; i<tamanho-1; i++){  
82         elementos[i] = elementos[i+1];  
83     }  
84     tamanho --;  
85 }  
86  
87
```

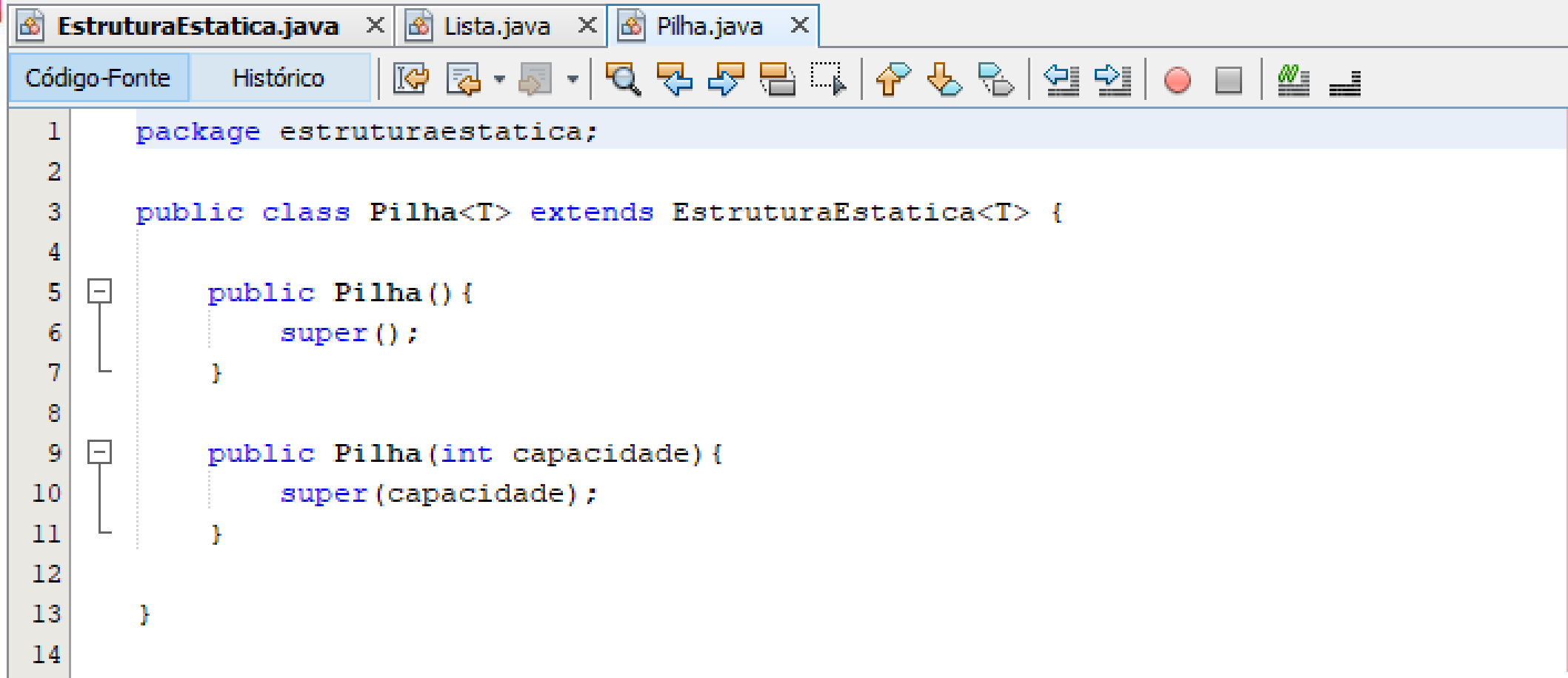
Essa é a Classe Pai ou super Classe.

Agora vamos criar as classes que vão herdar os atributos e os métodos as classes Listas e Pilhas.





```
1 package estruturaestatica;
2
3 public class Lista<T> extends EstruturaEstatica<T>{
4     // método Construtor com parâmetros o super proque está herdando ...
5     [-] public Lista(int capacidade) {
6         super(capacidade);
7     }
8     // método Contrutor sem Parâmetros
9     [-] public Lista() {
10         super();
11     }
12
13     [-] public boolean adiciona(T elemento){
14         return super.adiciona(elemento);
15     }
16
17     [-] public boolean adiciona(int posicao, T elemento){
18         return super.adiciona(posicao, elemento);
19     }
20
21 }
```

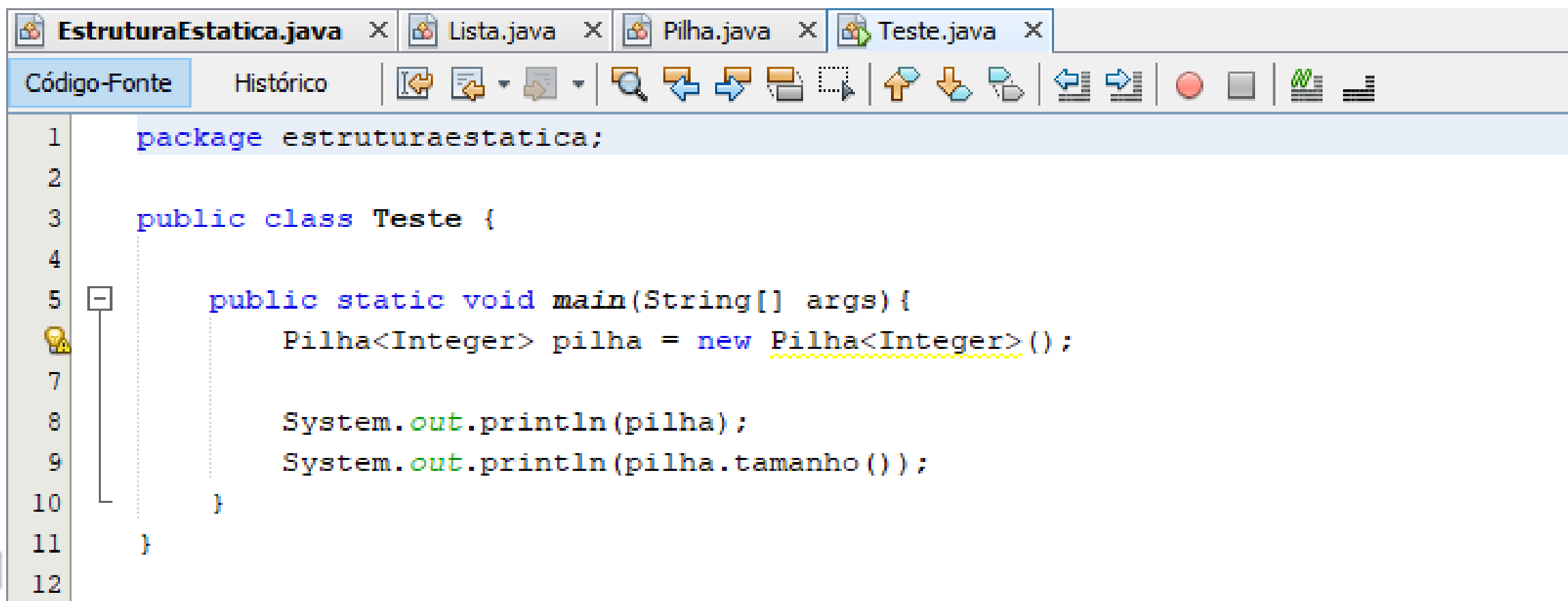


```
1 package estruturaestatica;
2
3 public class Pilha<T> extends EstruturaEstatica<T> {
4
5     public Pilha() {
6         super();
7     }
8
9     public Pilha(int capacidade) {
10         super(capacidade);
11     }
12
13 }
14
```

OK, sei que ainda falta alguns métodos, porém vamos testar nosso projeto, o que foi feito até agora, para isso vamos criar uma classe teste.

# Listas e Pilhas (Stacks)

## Classe Teste:



The screenshot shows an IDE window with four tabs: EstruturaEstatica.java, Lista.java, Pilha.java, and Teste.java. The Teste.java tab is active, showing the following code:

```
1 package estruturaestatica;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         Pilha<Integer> pilha = new Pilha<Integer>();
7
8         System.out.println(pilha);
9         System.out.println(pilha.tamanho());
10    }
11 }
12
```

The code is displayed with line numbers on the left. A light blue highlight is visible on line 1. A toolbar with various icons is located above the code editor.

# Listas e Pilhas (Stacks)

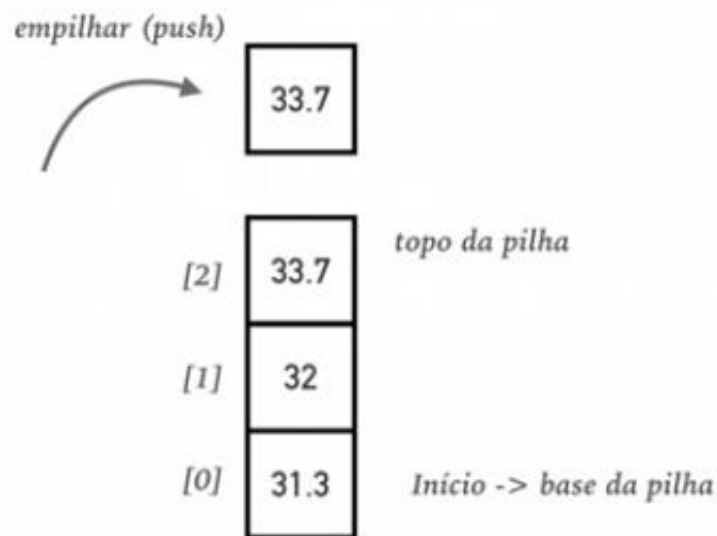
Resultado:

```
⋮ Saída - EstruturaEstatica (run)
run:
[]
0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
|
```

# Listas e Pilhas (Stacks)

OK agora que já temos nossa estrutura montada, vamos verificar como empilhar um elemento, lembrando que o comportamento da pilha - **LIFO** – **L**ast **I**n **F**irst **O**ut (Último a entrar, primeiro a sair).

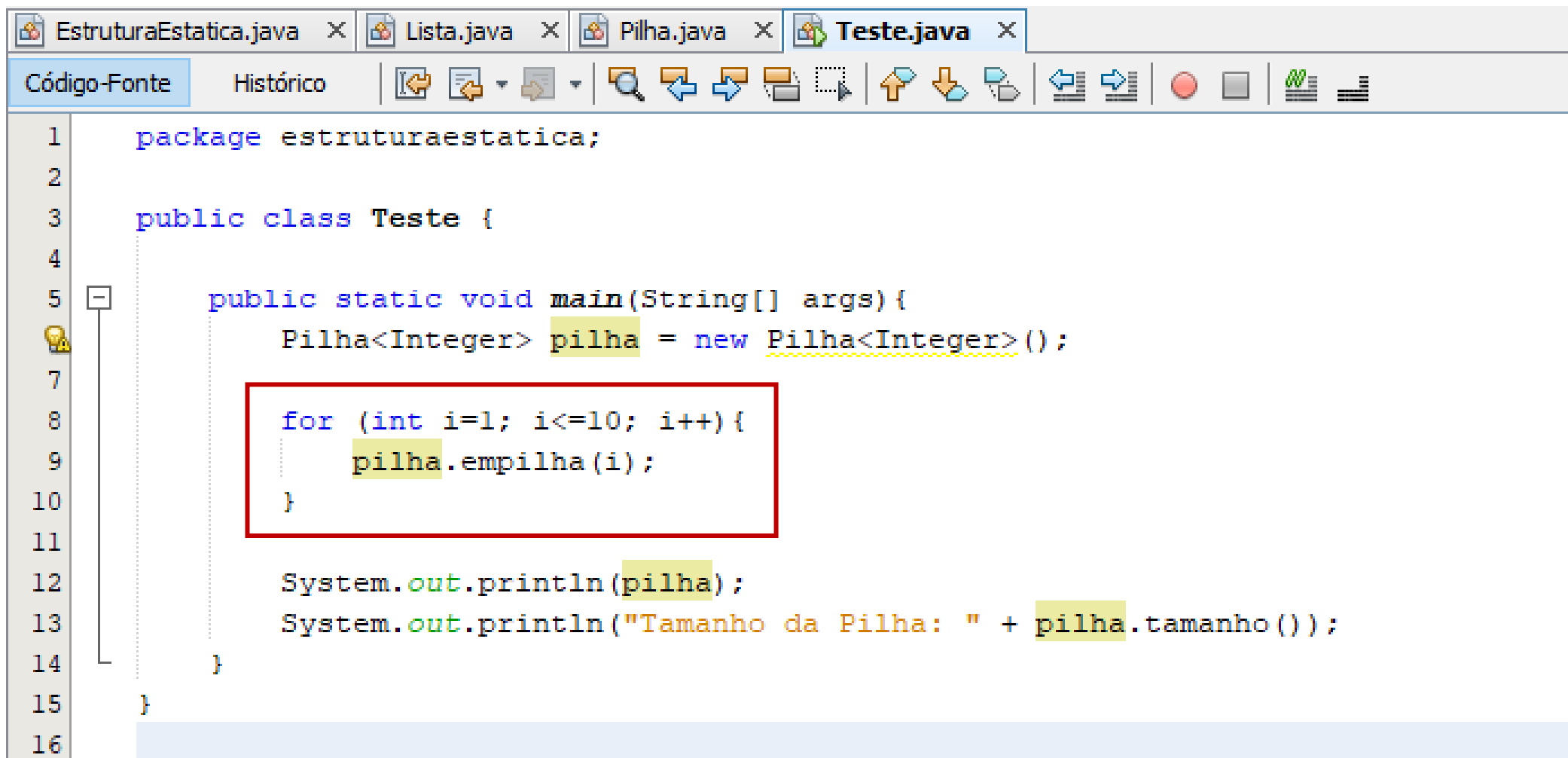
*Pilha*



```
1 package estruturaestatica;
2
3 public class Pilha<T> extends EstruturaEstatica<T> {
4
5     [- public Pilha() {
6         super();
7     }
8
9     [- public Pilha(int capacidade) {
10         super(capacidade);
11     }
12
13     [- public void empilha(T elemento) {
14         // dessa forma estamos reutilizando o código da classe pai
15         super.adiciona(elemento);
16     }
17
18
19 }
20
```

# Listas e Pilhas (Stacks)

Vamos Testar:



The screenshot shows an IDE with four tabs: EstruturaEstatica.java, Lista.java, Pilha.java, and Teste.java. The Teste.java tab is active, displaying the following code:

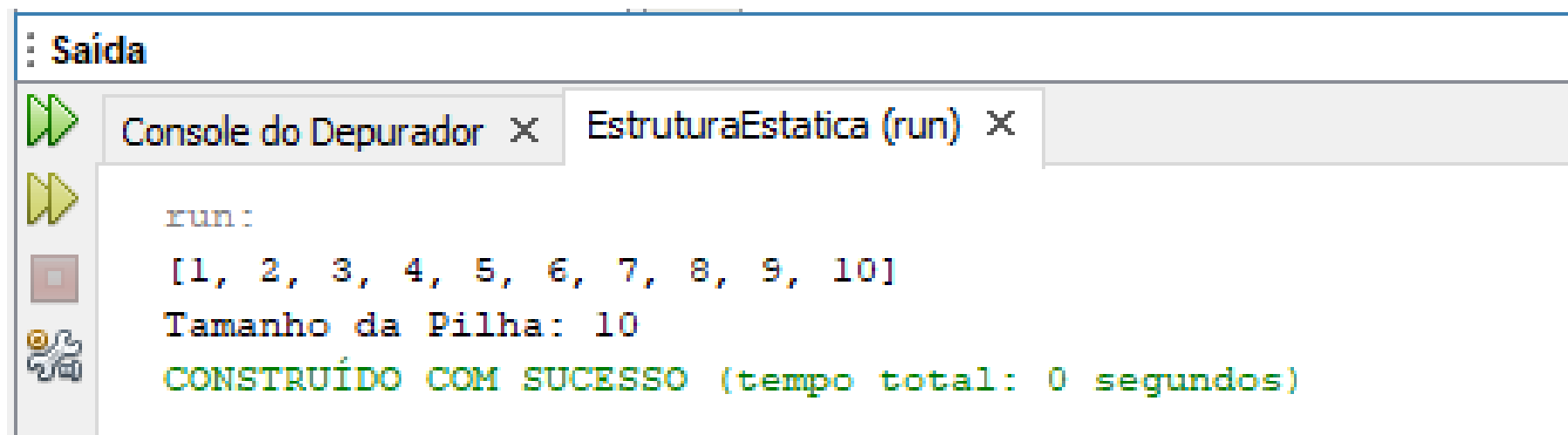
```
1 package estruturaestatica;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         Pilha<Integer> pilha = new Pilha<Integer>();
7
8         for (int i=1; i<=10; i++){
9             pilha.empilha(i);
10        }
11
12        System.out.println(pilha);
13        System.out.println("Tamanho da Pilha: " + pilha.tamanho());
14    }
15 }
16
```

The code is written in Java. It defines a package `estruturaestatica` and a class `Teste`. The `main` method creates a `Pilha<Integer>` object named `pilha`. It then pushes integers from 1 to 10 onto the stack using the `empilha` method. Finally, it prints the stack object and its size using the `tamanho` method.



# Listas e Pilhas (Stacks)

Resultado:



```
run:  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Tamanho da Pilha: 10  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



# Listas e Pilhas (Stacks)

Lembrando que colocamos como parâmetro inicial para criar nosso vetor com 10 posições, porém se dentro da estrutura de repetição for colocarmos:

```
for (int i=1; i<=15; i++){
```

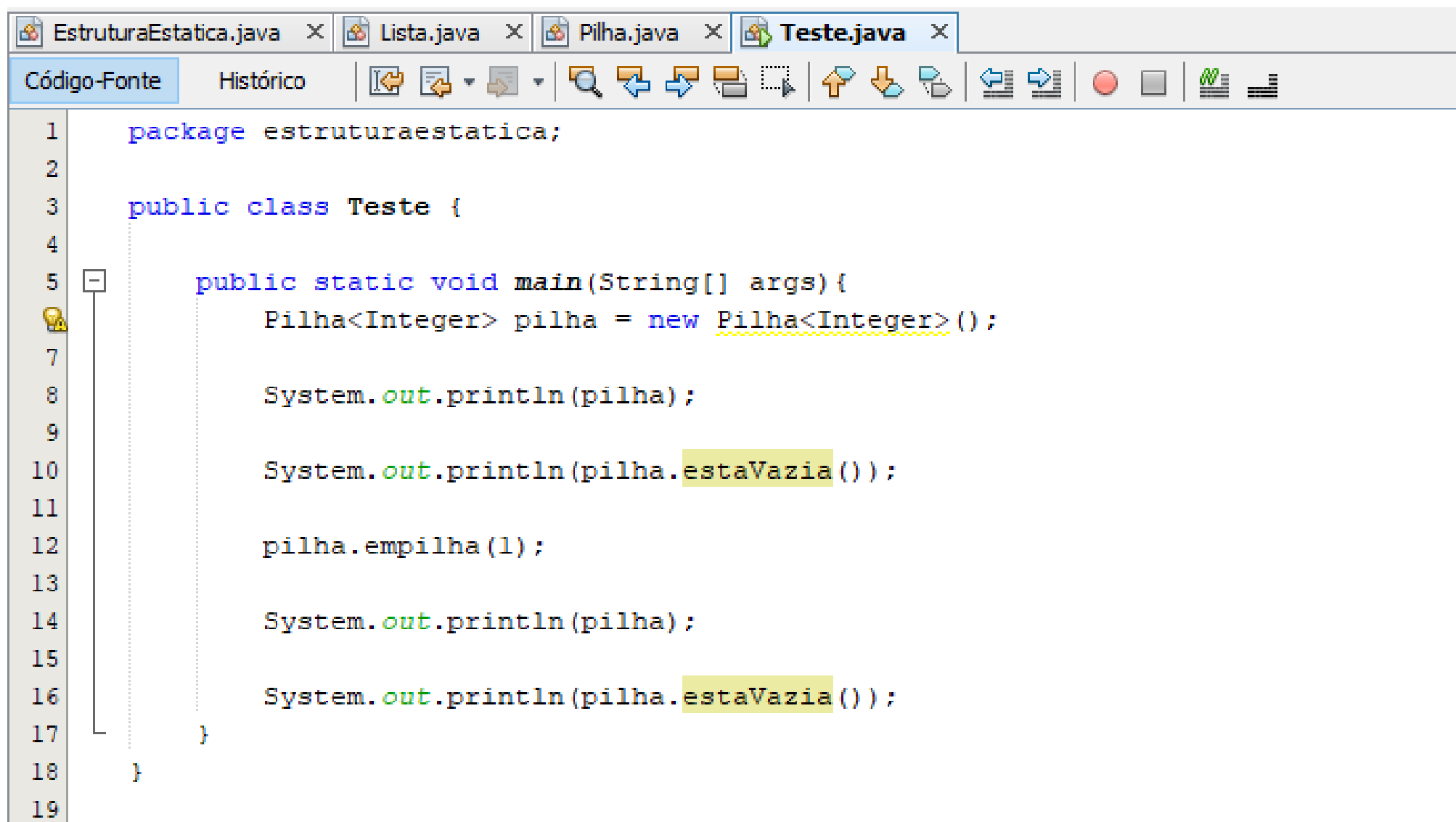
Não teríamos problemas, pois ele verifica o tamanho e aumenta de forma automática.

# Listas e Pilhas (Stacks)



Aproveitando para lembrar que a estrutura de dados pilha é muito utilizada pelas próprias linguagens de programação como C#, Java, C, C++, JavaScript, etc., todas elas utilizam uma pilha interna, quando chamamos um método existe uma pilha (stack) de métodos, vale a pena pesquisar sobre o assunto e aproveito para indicar o fórum sobre o assunto mais famoso no mundo que é o **stackoverflow**, serve para tirar dúvidas, conhecer mais sobre o assunto - <https://stackoverflow.com>.

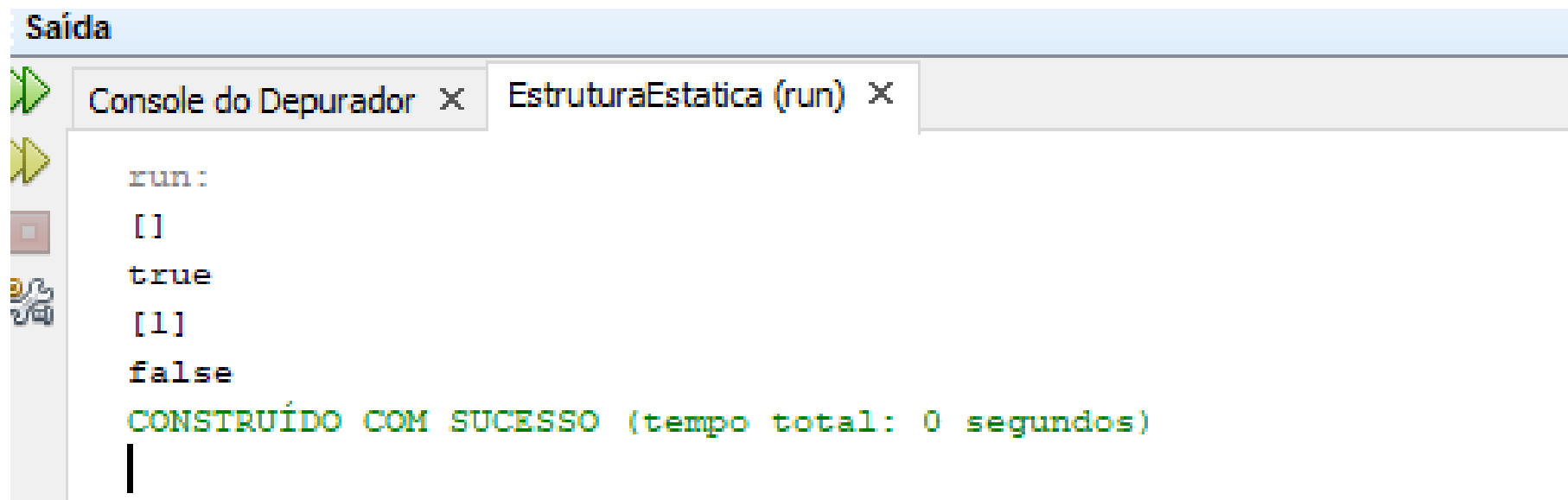
Agora vamos verificar se a nossa pilha está vazia:



```
1 package estruturaestatica;
2
3 public class Teste {
4
5     public static void main(String[] args){
6         Pilha<Integer> pilha = new Pilha<Integer>();
7
8         System.out.println(pilha);
9
10        System.out.println(pilha.estaVazia());
11
12        pilha.empilha(1);
13
14        System.out.println(pilha);
15
16        System.out.println(pilha.estaVazia());
17    }
18 }
19
```

# Listas e Pilhas (Stacks)

Resultado:

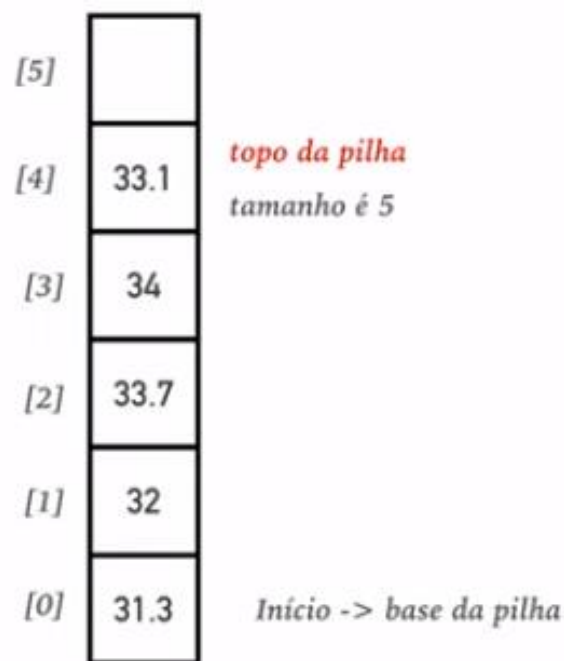


The screenshot shows a debugger's output window with a tab titled "Saída". Below the tab, there are two sub-tabs: "Console do Depurador" and "EstruturaEstatica (run)". The "Console do Depurador" tab is active, displaying the following output:

```
run :  
[]  
true  
[1]  
false  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)  
|
```

# Listas e Pilhas (Stacks)

Agora vamos verificar o elemento no topo da nossa pilha através do método **topo()**: *Pilha*



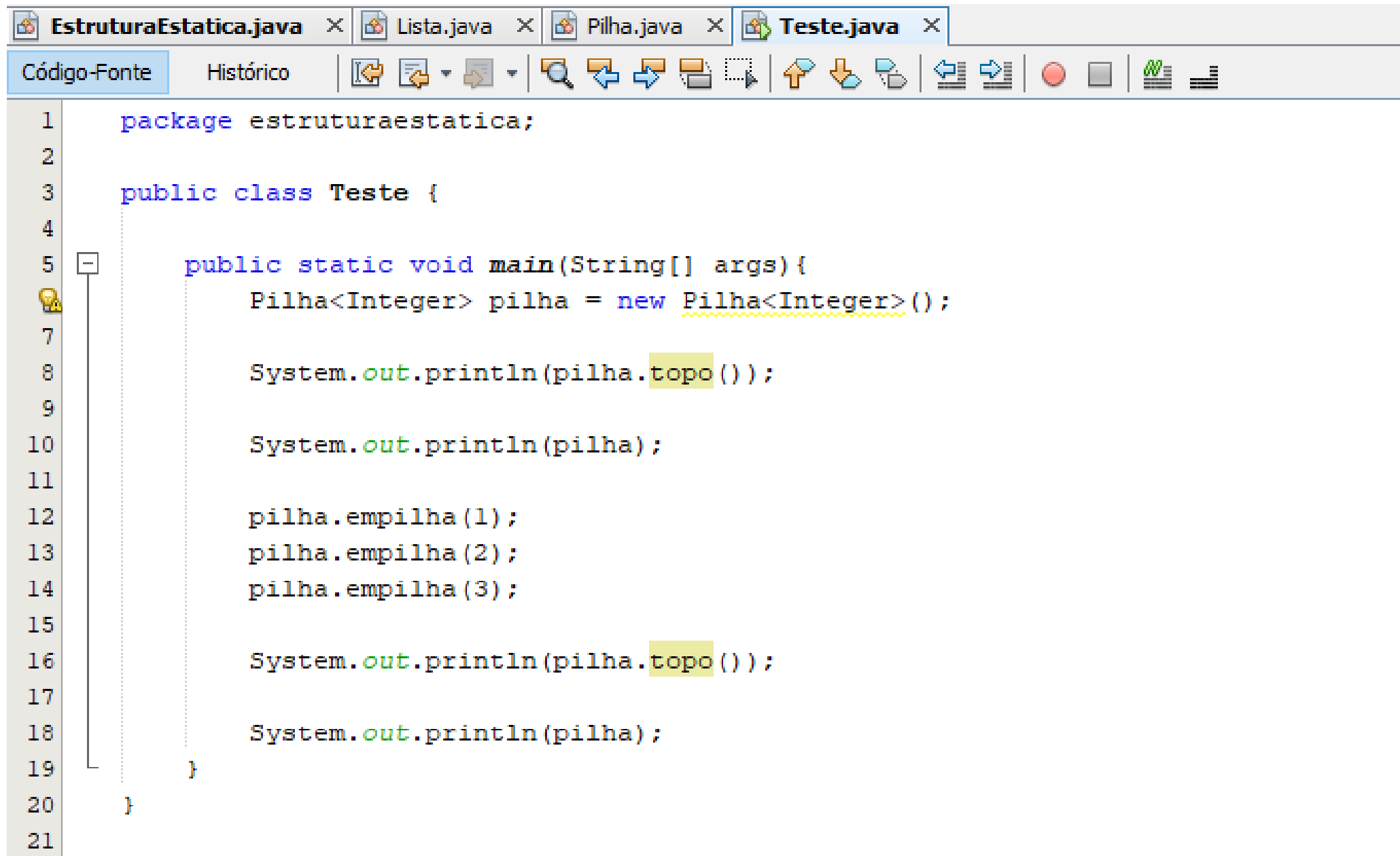
Vale lembrar que nesse método eu só vou visualizar a informação, não será inserido e nem removido nenhum elemento.

EstruturaEstatica.java x Lista.java x Pilha.java x Teste.java x

Código-Fonte Histórico

```
1 package estruturaestatica;
2
3 public class Pilha<T> extends EstruturaEstatica<T> {
4
5     public Pilha() {
6         super();
7     }
8
9     public Pilha(int capacidade) {
10         super(capacidade);
11     }
12
13     public void empilha(T elemento) {
14         // dessa forma estamos reutilizando o código da classe pai
15         super.adiciona(elemento);
16     }
17
18
19     public T topo() {
20         if(this.estaVazia()) {
21             return null;
22         } else {
23             return this.elementos[tamanho-1];
24         }
25     }
26 }
```

Teste:

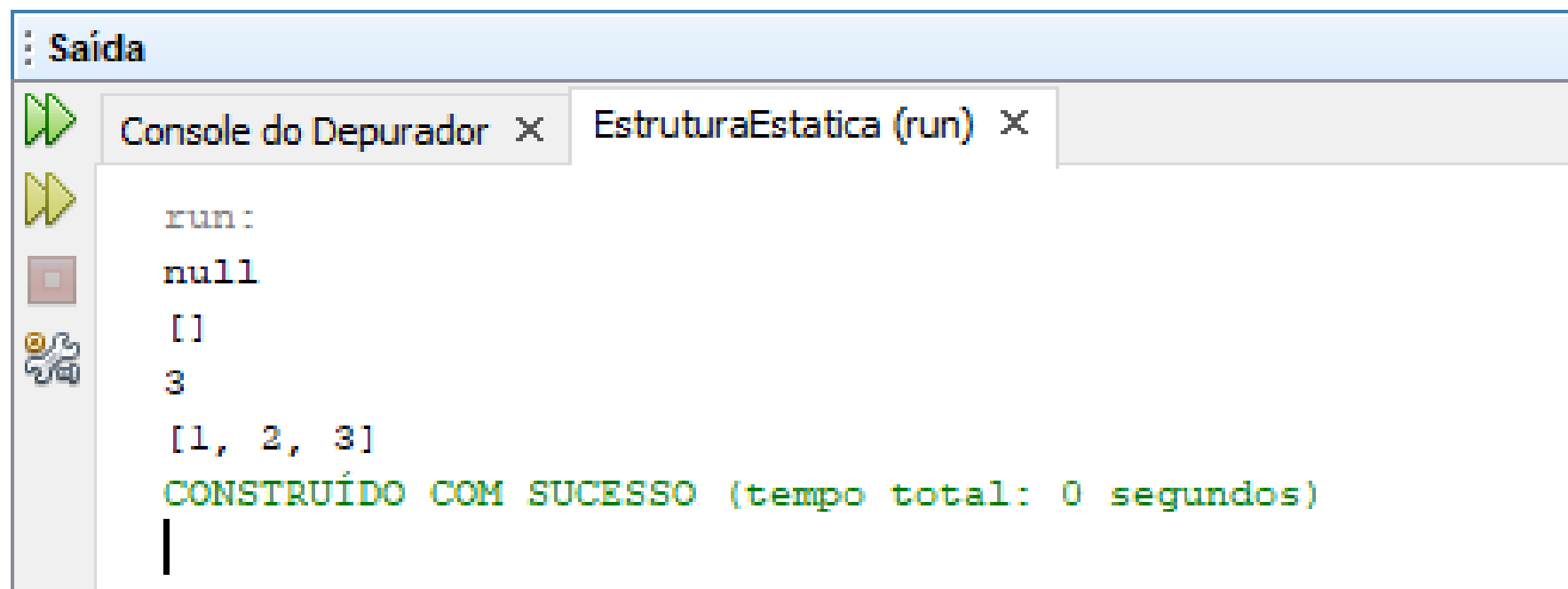


```
1 package estruturaestatica;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         Pilha<Integer> pilha = new Pilha<Integer>();
7
8         System.out.println(pilha.topo());
9
10        System.out.println(pilha);
11
12        pilha.empilha(1);
13        pilha.empilha(2);
14        pilha.empilha(3);
15
16        System.out.println(pilha.topo());
17
18        System.out.println(pilha);
19    }
20 }
21
```



# Listas e Pilhas (Stacks)

Resultado:

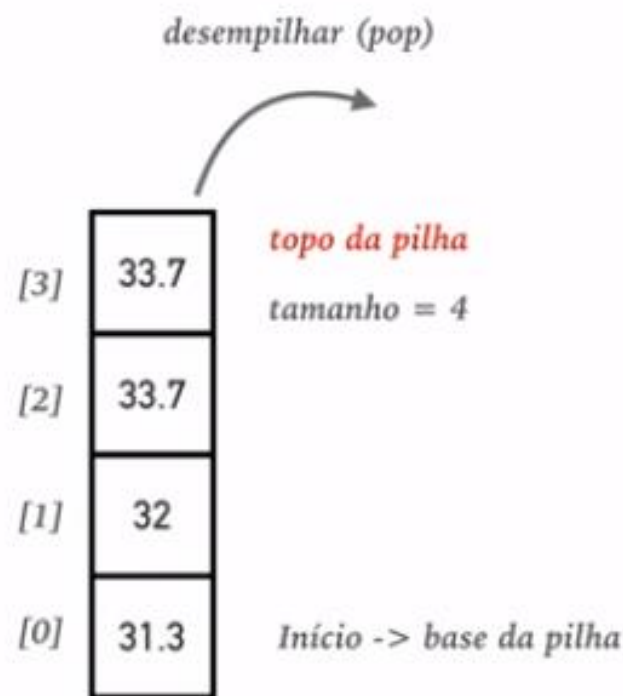


```
run:  
null  
[]  
3  
[1, 2, 3]  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)  
|
```

# Listas e Pilhas (Stacks)

Vamos agora criar o método **desempilhar()** ou seja remover um elemento da nossa pilha, respeitando a característica da pilha **LIFO** – **Last In First Out** (Último a entrar, primeiro a sair).

*Pilha*

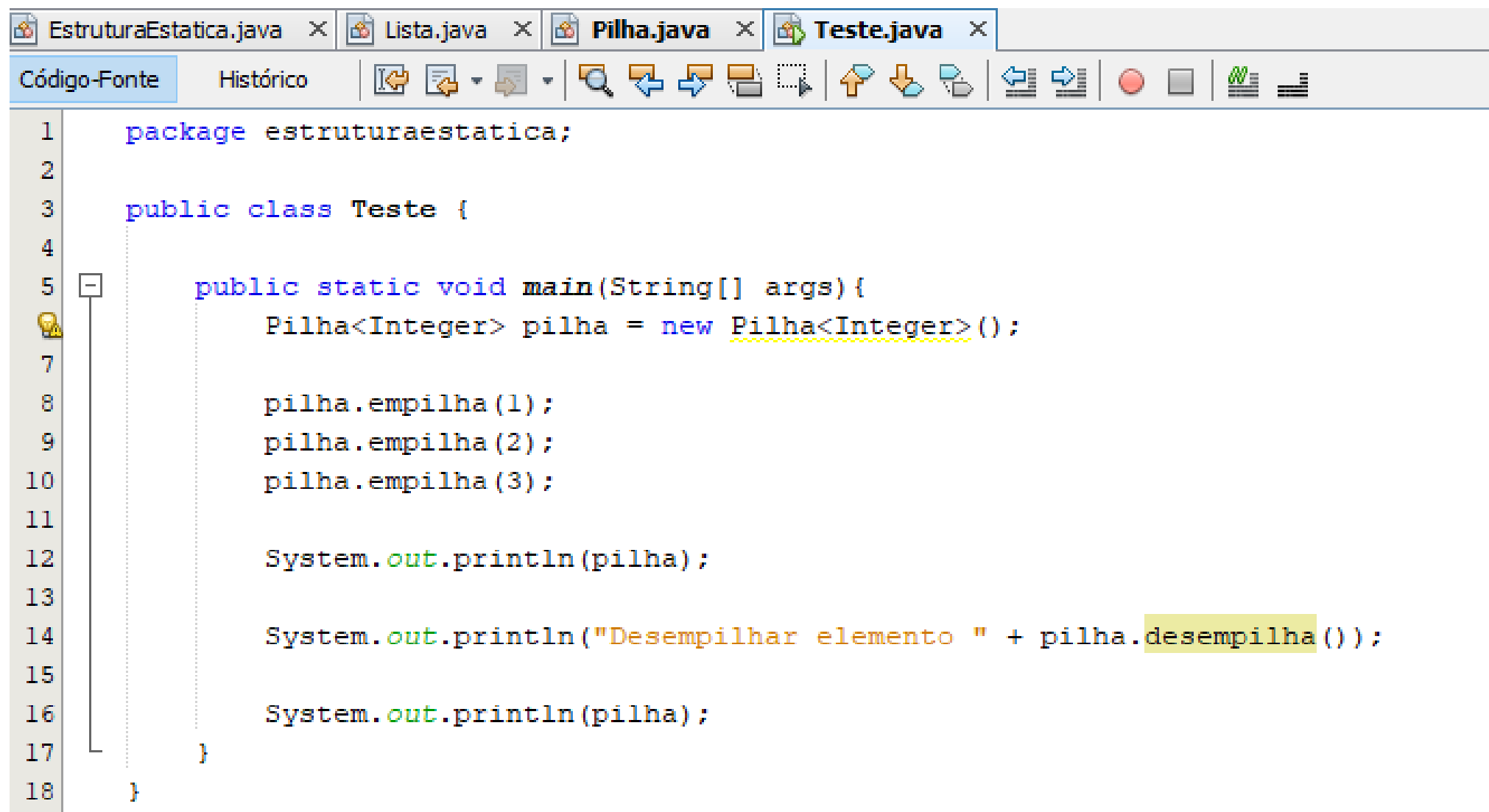




```
1 package estruturaestatica;
2
3 public class Pilha<T> extends EstruturaEstatica<T> {
4
5     public Pilha() {...3 linhas }
6
7
8     public Pilha(int capacidade) {...3 linhas }
9
10
11
12     public void empilha(T elemento) {...5 linhas }
13
14
15
16
17
18     public T topo() {...7 linhas }
19
20
21
22
23
24
25
26
27     public T desempilha() {
28         if(this.estaVazia()) {
29             return null;
30         }
31         T elemento = this.elementos[tamanho-1];
32         tamanho--;
33         return elemento;
34     }
35 }
36
```

# Listas e Pilhas (Stacks)

Teste:



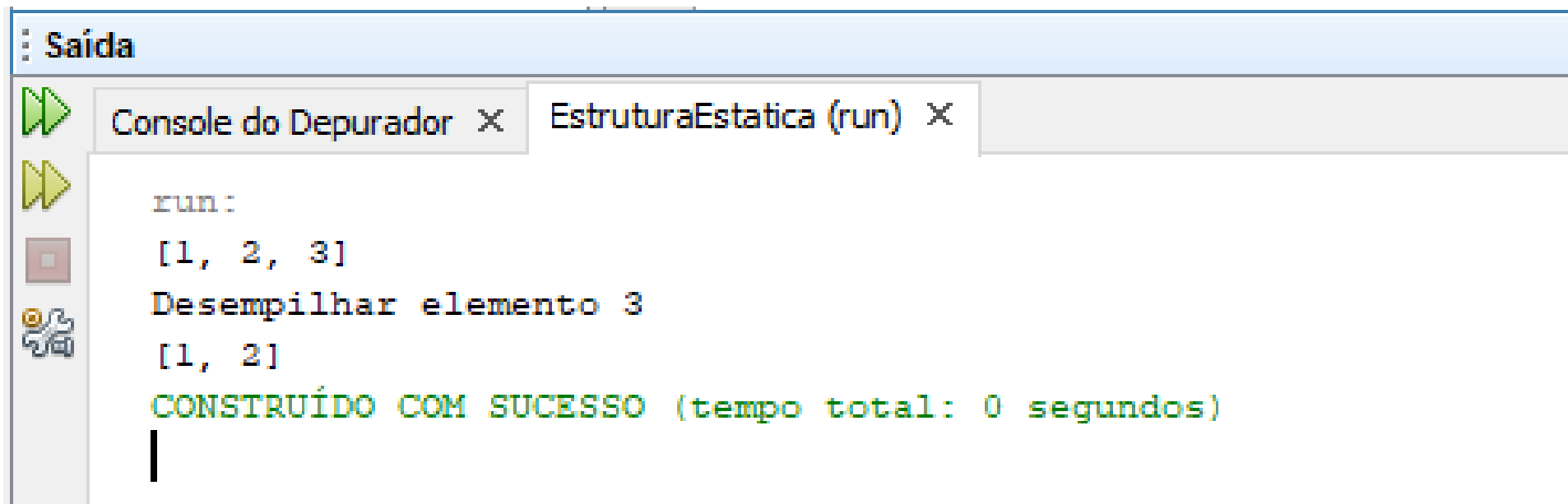
The screenshot shows an IDE with four tabs: EstruturaEstatica.java, Lista.java, Pilha.java, and Teste.java. The Teste.java tab is active, displaying the following code:

```
1 package estruturaestatica;
2
3 public class Teste {
4
5     public static void main(String[] args){
6         Pilha<Integer> pilha = new Pilha<Integer>();
7
8         pilha.empilha(1);
9         pilha.empilha(2);
10        pilha.empilha(3);
11
12        System.out.println(pilha);
13
14        System.out.println("Desempilhar elemento " + pilha.desempilha());
15
16        System.out.println(pilha);
17    }
18 }
```

The code defines a package `estruturaestatica` and a class `Teste`. The `main` method creates a `Pilha<Integer>` object, pushes the values 1, 2, and 3 onto the stack, prints the stack, pops an element (printing "Desempilhar elemento "), and prints the stack again.

# Listas e Pilhas (Stacks)

Resultado:



```
run:
[1, 2, 3]
Desempilhar elemento 3
[1, 2]
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
|
```

# Listas e Pilhas (Stacks)

Agora que já vimos a lógica do funcionamento de uma pilha, vamos conhecer a **API Java Stack**, que são bem semelhantes aos métodos desenvolvidos até agora nesse projeto.





```
1 package estruturaestatica;
2
3 import java.util.Stack;
4
5 public class Teste {
6
7     public static void main(String[] args){
8         Stack<Integer> stack = new Stack<Integer>();
9
10        // método para verificar se a pilha está vazia
11        System.out.println(stack.isEmpty());
12
13        // método para empilhar
14        stack.push(1);
15        stack.push(2);
16        stack.push(3);
17
18        // método para verificar o tamanho da pilha
19        System.out.println(stack.size());
20
21        // Método ToString para exibir a pilha
22        System.out.println(stack);
23    }
```

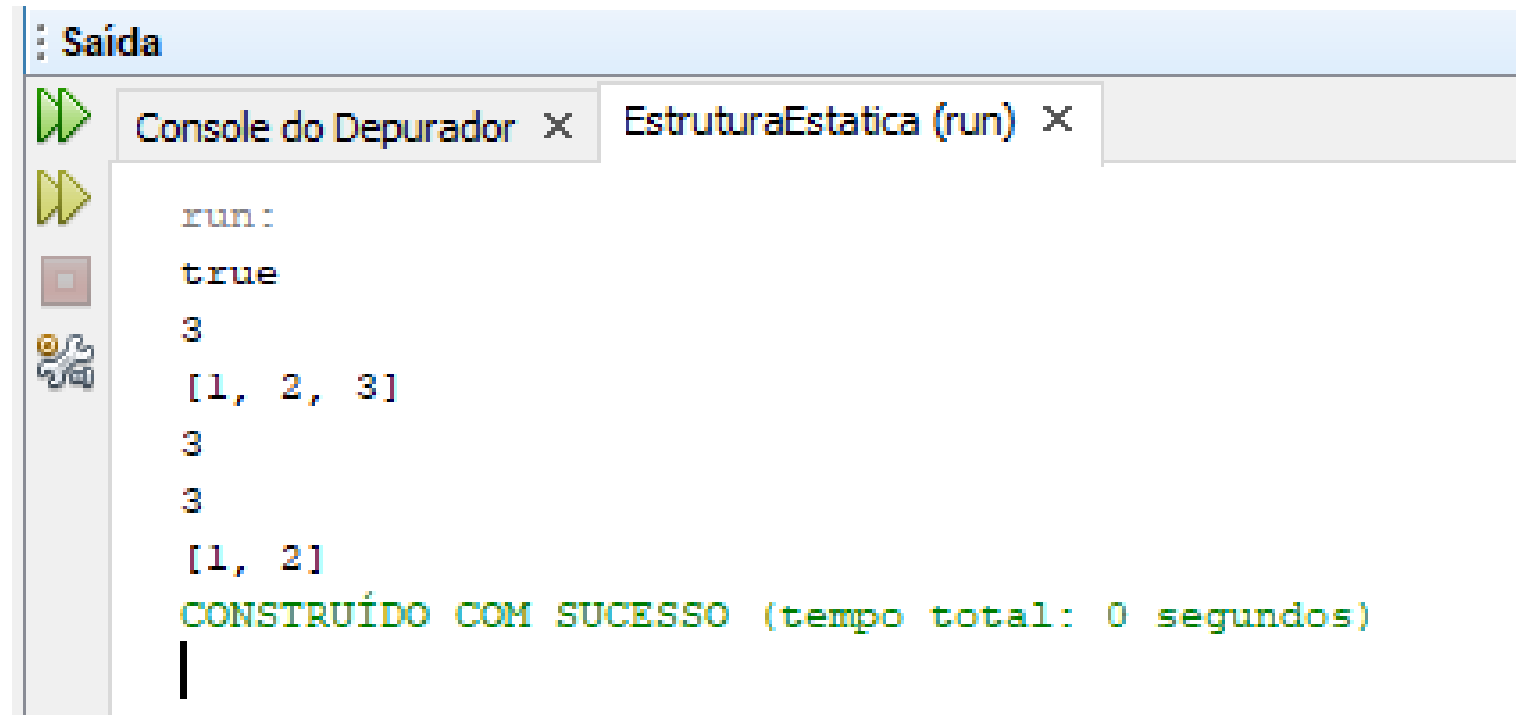


```
23
24 // Método para verificar o topo
25 System.out.println(stack.peek());
26
27 // Método para desempilhar
28 System.out.println(stack.pop());
29
30 // Método ToString para exibir a pilha
31 System.out.println(stack);
32
33
34 }
35 }
36
37
```



# Listas e Pilhas (Stacks)

Resultado:



```
run:
true
3
[1, 2, 3]
3
3
[1, 2]
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

"Quando a gente acha  
que tem todas as  
respostas,  
vem a vida e muda  
todas as perguntas"



**Luís Fernando Veríssimo**

# Obrigado!

**Se precisar ...**

Prof. Claudio Benossi

[claudio.benossi@sp.senac.br](mailto:claudio.benossi@sp.senac.br)

