
# Estrutura de Dados

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

**Aula 02**

**Prof. Claudio Benossi**

# 1.
# Unidade

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

# Estrutura de Dados

## O que são algoritmos?

- Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.
- Uma ferramenta para resolver um problema computacional bem especificado.
- O enunciado do problema especifica em termos gerais a entrada e a saída do problema

# Estrutura de Dados

**Entrada:**

V[]={31, 41, 59, 26, 41, 58}

**Saída esperada:**

V[]={26, 31, 41, 41, 58, 59}

Instância do problema

Entrada que satisfaz a quaisquer restrições impostas no enunciado do problema.

Um algoritmo é dito **CORRETO** se gerar uma saída correta para cada instância do problema.

# Estrutura de Dados

Algoritmos eficientes.

**Medida de eficiência**: velocidade (quanto tempo um algoritmo demora para produzir seu resultado).

# Estrutura de Dados

**Problemas de instância**

Encontrar a média dos elementos de um vetor A[1 . . n] de números.

Uma das instâncias deste problema consiste em encontrar a média dos elementos do vetor A[]= {876, -145, 323, 112, 221}.

Outra instância deste problema consiste em encontrar a média dos elementos do vetor B[]={100, 120, 350, 440, 50, 600, 200}. O tamanho de uma instância de um problema é a quantidade de dados necessária para descrever a instância. Neste caso, diz-se que A=|5| e B=|7|

# Arranjos – Array (Vetores ou Matrizes)

Em computação um Vetor (Array) ou arranjo é o nome de uma matriz unidimensional considerada a mais simples das estruturas de dados.

Vetor (array unidimensional) é uma variável que armazena várias variáveis do mesmo tipo.

No problema apresentado anteriormente, nós podemos utilizar um vetor de 50 posições para armazenar os nomes dos 50 alunos.

# Arranjos – Array (Vetores ou Matrizes)

Matriz (array multidimensional) é um vetor de vetores. Imagine uma matriz para armazenar as 4 notas de 50 alunos.

Ou seja, um vetor de 50 posições, e em cada posição do vetor, há outro vetor com 4 posições. Isso é uma matriz.

Cada item do vetor (ou matriz) é acessado por um (ou dois) número(s) chamado(s) de índice(s).

# Tipos Primitivos e Tipos Abstratos de Dados

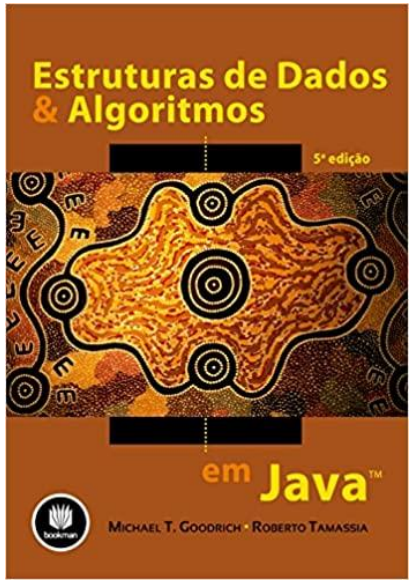
| Primitivos | Abstratos |
| --- | --- |
| int | Integer |
| boolean | Boolean |
| double | Double |
| float | Float |
| char | String |

- Estudante
- Professor
- Pessoa
- Pizza

# Arranjos – Array (Vetores ou Matrizes)

"Array (ou vetor) é a estrutura de dados mais simples que existe.

Um vetor armazena uma sequência de valores onde todos são do mesmo tipo"




Loiane Groner

# Arranjos – Array (Vetores ou Matrizes)

Imagine a necessidade de armazenar a temperatura média diária de todos os dias de um ano ...

Double tempDia001 = 25.7;

Double tempDia002 = 27.4;

Double tempDia003 = 29.3;

...

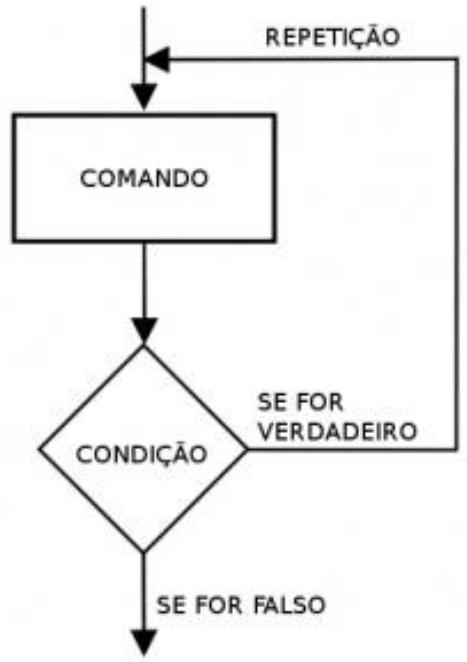
Double tempDia365 = 22.8;

**Imagine o tamanho do código desse programa a ser realizado, os recursos que serão utilizados???**

# Arranjos – Array (Vetores ou Matrizes)

**Solução:**

```
Double[] temperatura = new double[365]
temperatura[0] = 25.7;
temperatura[1] = 27.4;
temperatura[2]  = 29.3;
...
temperatura[364]  = 22.8;
```



# Arranjos – Array (Vetores ou Matrizes)


Implementação em Java

**Importante:**

- O uso da estrutura de repetição for
- O Uso do comando length

Projeto_revisao_ED.java

Código-Fonte | Histórico

```java
package projeto_revisao_ed;

public class Projeto_revisao_ED {

    public static void main(String[] args) {
        double[] temperatura = new double[365];
        temperatura[0] = 25.7;
        temperatura[1] = 27.4;
        temperatura[2] = 29.3;
        temperatura[3] = 31.4;
        temperatura[4] = 30.2;
        temperatura[5] = 29.6;

        System.out.println("O valor da temperatura do dia 3 é:" + temperatura[2]);
        System.out.println("O tamanho do vetor: " + temperatura.length);

        for (int i=0; i<temperatura.length; i++){
            System.out.println("O valor da temperatura do dia " + (i+1) + " é " + temperatura[i]);
        }

        for (double temp : temperatura){
            System.out.println(temp);
        }
    }
}
```

# Arranjos – Array (Vetores ou Matrizes)

Vamos falar sobre a manipulação de dados em um vetor


A **manipulação de dados** nada mais é que o método de coletar, limpar, processar e consolidar os **dados** para uso na análise. Simplificando, ele enriquece os **dados**, os transforma e melhora a precisão do resultado analítico. É uma etapa do processo analítico que consome uma quantidade significativa de tempo e esforço.

# Arranjos – Array (Vetores ou Matrizes)

Vamos fazer uma atividade com o conceito de Vetor:

- Criar uma aplicação na linguagem Java para solicitar ao usuário a média diária das temperaturas de um período de 7 dias;
- Verificar e informar:
  - Média da temperatura durante o peródo;
  - Quantos dias a temperatura ficou acima da média;
  - Quantos dias a temperatura ficou abaixo da média.

```
INICIO
        VAR flo_temperatura: ARRAY [7] numerico;
        VAR flo_soma, flo_media, int_contador, int_acima, int_abaixo: numerico;
        flo_soma = 0;
        PARA (int_contador = 1 ATÉ int_contador = 7, int_contador ++) FAÇA
             ESCREVER "Digite a " + int_contador + “º temperatura”;
             LER flo_temperatura[int_contador];
             flo_soma = flo_soma + flo_temperatura[int_contador];
         FIM PARA
         flo_media = flo_soma / 7;
         int_acima = 0;
         int_abaixo = 0;
         PARA (int_contador = 1 ATÉ int_contador = 7, int_contador ++) FAÇA
              SE (flo_temperatura[int_contador] > flo_media) ENTÃO
                   int_acima = int_acima + 1;
               FIM SE
               SE (flo_temperatura[int_contador] < flo_media) ENTÃO
                   int_abaixo = int_abaixo + 1;
               FIM SE
      FIM PARA
      ESCREVER “Total de dias que a temperatura ficou acima da média” + int_acima;
      ESCREVER “Total de dias que a temperatura ficou abaixo da média” + int_abaixo;
FIM
```

Projeto_temperatura.java

Código-Fonte | Histórico

```java
package projeto_temperatura;

import javax.swing.JOptionPane;

public class Projeto_temperatura {
    public static void main(String[] args) {
        double[] temperatura = new double[7];
        double soma, media;
        int contador, dias_acima, dias_abaixo;
        soma = 0;
        for(contador = 0; contador<7; contador ++){
            temperatura[contador] = Double.parseDouble(JOptionPane.showInputDialog("Digite a " + (contador + 1)
                    + "° temperatura: "));
            soma = soma + temperatura[contador];
        }
        media = soma / 7;
        dias_acima = 0;
        dias_abaixo = 0;
        for(contador = 0; contador<7; contador ++){
            if (temperatura[contador] > media){
                dias_acima = dias_acima + 1;
            }
            if (temperatura[contador] < media){
                dias_abaixo = dias_abaixo + 1;
            }
        }
        JOptionPane.showMessageDialog(null, "A média das temperaturas é " + media);
        JOptionPane.showMessageDialog(null, "A quantidade de dias acima da média é " + dias_acima);
        JOptionPane.showMessageDialog(null, "A quantidade de dias abaixo da média é " + dias_abaixo);
    }
}
```



Projeto_temperatura.java

Código-Fonte | Histórico

```java
package projeto_temperatura;

import java.util.Scanner;

public class Projeto_temperatura {
    public static void main(String[] args) {
        double[] temperatura = new double[7];
        double soma, media;
        int contador, dias_acima, dias_abaixo;
        soma = 0;
        Scanner dados = new Scanner(System.in);
        for(contador = 0; contador<7; contador ++){
            System.out.println("Digite a " + (contador + 1) + "° temperatura:");
            temperatura[contador] = dados.nextDouble();
            soma = soma + temperatura[contador];
        }
        media = soma / 7;
        dias_acima = 0;
        dias_abaixo = 0;
        for(contador = 0; contador<7; contador ++){
            if (temperatura[contador] > media){
                dias_acima = dias_acima + 1;
            }
            if (temperatura[contador] < media){
                dias_abaixo = dias_abaixo + 1;
            }
        }
        System.out.println("A média das temperaturas é " + media);
        System.out.println("A quantidade de dias acima da média é " + dias_acima);
        System.out.println("A quantidade de dias abaixo da média é " + dias_abaixo);
    }
}
```



Java

# Arranjos – Array (Vetores ou Matrizes)

Para entender os principais tópicos sobre manipulação de vetores vamos criar uma classe Vetor com alguns métodos.

# Arranjos – Array (Vetores ou Matrizes)

Um **método em Java** é equivalente a uma função, subrotina ou procedimento em outras linguagens de programação.

Não existe em **Java** o conceito de **métodos** globais.

Todos os **métodos** devem sempre ser definidos dentro de uma classe.

# Arranjos – Array (Vetores ou Matrizes)

Vamos criar uma classe em Java que além da estrutura de dados com o nosso **vetor,** vamos criar métodos:

- Método para adicionar um elemento (nesse caso será inserido no final do vetor);
- Também podemos criar um método para adicionar um elemento em uma posição especifica;
- Método para remover um elemento;
- Método para buscar um elemento (Pesquisa);
- Método para identificar o tamanho real do nosso vetor;
- E por fim um método para exibir o conteúdo do nosso vetor

Projeto_revisao_ED.java | Vetor.java

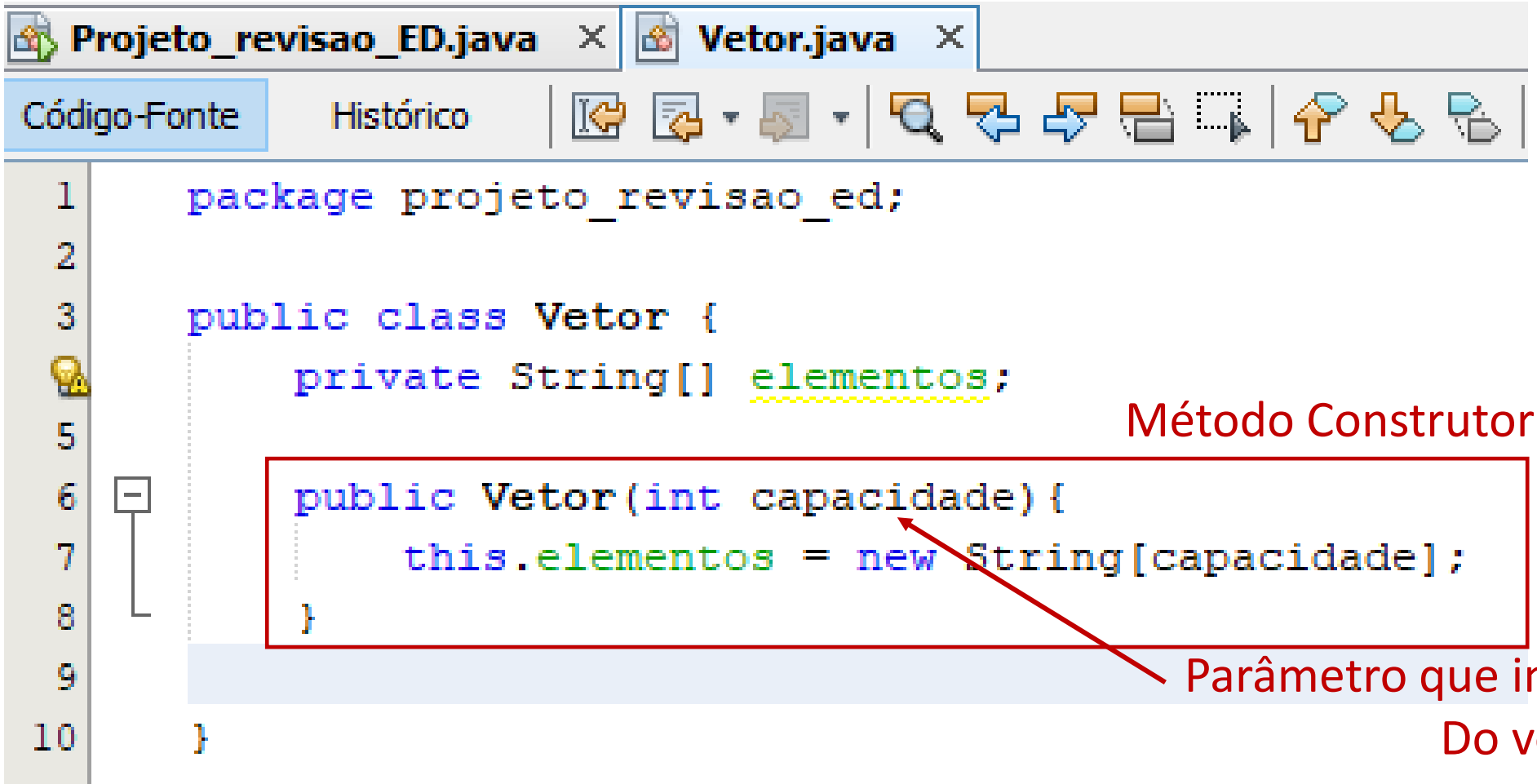
Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Vetor {
4       private String[] elementos;
5       public Vetor(int capacidade){
6           elementos = new String[capacidade];
7       }
8       public void adiciona(String elemento){
9
10      }
11      public void adiciona(int posicao, String elemento){
12
13      }
14      public void remove(int posicao){
15
16      }
17      public String busca(int posicao){
18
19      }
20      public int busca (String elemento){
21
22      }
23      public int tamanho(){
24
25      }
26      public String toString(){
27
28      }
29  }
```

Podemos ter um método com o mesmo, isso é uma **sobreposição**, o que vai diferenciar são os parâmetros passados pelo método

# Arranjos – Array (Vetores ou Matrizes)

Primeiro vamos criar a nossa classe Vetor:



# Arranjos – Array (Vetores ou Matrizes)

Primeiro vamos criar a nossa classe Vetor:

Projeto_revisao_ED.java | Vetor.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Vetor {
4       private String[] elementos;
5
6       public Vetor(int capacidade){
7           this.elementos = new String[capacidade];
8       }
9
10  }
```

**this** palavra-chave **Java é** usada para referenciar a instância atual do método no qual **é** usada. Aqui, isso se refere à variável de instância. ...
Isso chama o construtor da mesma classe **java** que possui um parâmetro .

# Arranjos – Array (Vetores ou Matrizes)

Vou criar um novo programa para testar a nossa classe através do método void main (ponto de partida para executar a nossa aplicação).

Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args){
5           Vetor vetor = new Vetor(5);
6       }
7   }
```

***Nesse casso vamos criar um vetor com 5 posições, somente.***

# Arranjos – Array (Vetores ou Matrizes)

## Vetor

| null | null | null | null | null |
|---|---|---|---|---|
| **Posição 0** | Posição 1 | Posição 2 | Posição 3 | Posição 4 |



Lembrando que quando eu crio um vetor do tipo **String** ele inicia todas as posições como **null** o valor inicial de uma String no Java, se for um vetor do tipo **boolean** terá o valor inicial igual a **False**, se for do tipo **int** o valor inicial será de **0**.

# Arranjos – Array (Vetores ou Matrizes)

Agora vamos adicionar elementos no final do nosso vetor, para isso vamos identificar a posição do vetor que está disponível.

Vamos voltar a nossa classe Vetor:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Vetor {
4      private String[] elementos;
5
6      public Vetor(int capacidade){
7          this.elementos = new String[capacidade];
8      }
9
10     public void adiciona(String elemento){
11         for(int i=0; i<this.elementos.length; i++){
12             if(this.elementos[i] == null){
13                 this.elementos[i] = elemento;
14                 break;
15             }
16         }
17     }
18
19 }
```

Estrutura de Repetição

Estrutura de decisão

# Arranjos – Array (Vetores ou Matrizes)

Vamos Testar:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args){
5           Vetor vetor = new Vetor(5);
6
7           vetor.adiciona("elemento 1");
8           vetor.adiciona("elemento 2");
9       }
10  }
```

| Ementо 1 | Elemento 2 | null | null | null |
|---|---|---|---|---|
| Posição 0 | Posição 1 | Posição 2 | Posição 3 | Posição 4 |

# Arranjos – Array (Vetores ou Matrizes)

Ok, **funciona**, porém quando trabalhos com um vetor muito grande será necessário fazer uma varredura do vetor isso pode ficar demorado e **não tão eficiente**, nesse caso é recomendado ***criar um atributo na minha classe para identificar a ultima posição ocupada***, dessa forma vamos **otimizar o processo**, nesse caso vamos ter o tamanho do vetor e o tamanho real ocupado no nosso vetor.

Vetor.java | Teste.java

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Vetor {
4      private String[] elementos;
5      private int tamanho;
6
7      public Vetor(int capacidade){
8          this.elementos = new String[capacidade];
9          this.tamanho = 0;
10     }
11
12     public void adiciona(String elemento)throws Exception{
13         if (this.tamanho < this.elementos.length){
14             this.elementos[this.tamanho] = elemento;
15             this.tamanho++;
16         } else {
17             throw new Exception("O Vetor já está cheio, "
18                     + "não é possível adiconar novos elementos");
19         }
20     }
```

Variável para determinar o tamanho atual do vetor

Vetor.java | Teste.java

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Vetor {
4      private String[] elementos;
5      private int tamanho;
6
7      public Vetor(int capacidade){
8          this.elementos = new Strir
9          this.tamanho = 0;
10     }
11
12     public void adiciona(String elemento)throws Exception{
13       if (this.tamanho < this.elementos.length){
14           this.elementos[this.tamanho] = elemento;
15           this.tamanho++;
16       } else {
17           throw new Exception("O Vetor já está cheio, "
18                   + "não é possível adiconar novos elementos");
19       }
20     }
```

## Comandos **throw** e **throws**

Imagine uma situação em que não é desejado que uma exceção seja tratada na própria classe ou método, mas sim em outro que venha lhe chamar.

Para solucionar tal situação utilizamos o comando **throws** na assinatura do método com a possível exceção que o mesmo poderá a vir lançar.

# Arranjos – Array (Vetores ou Matrizes)

Vamos Testar:

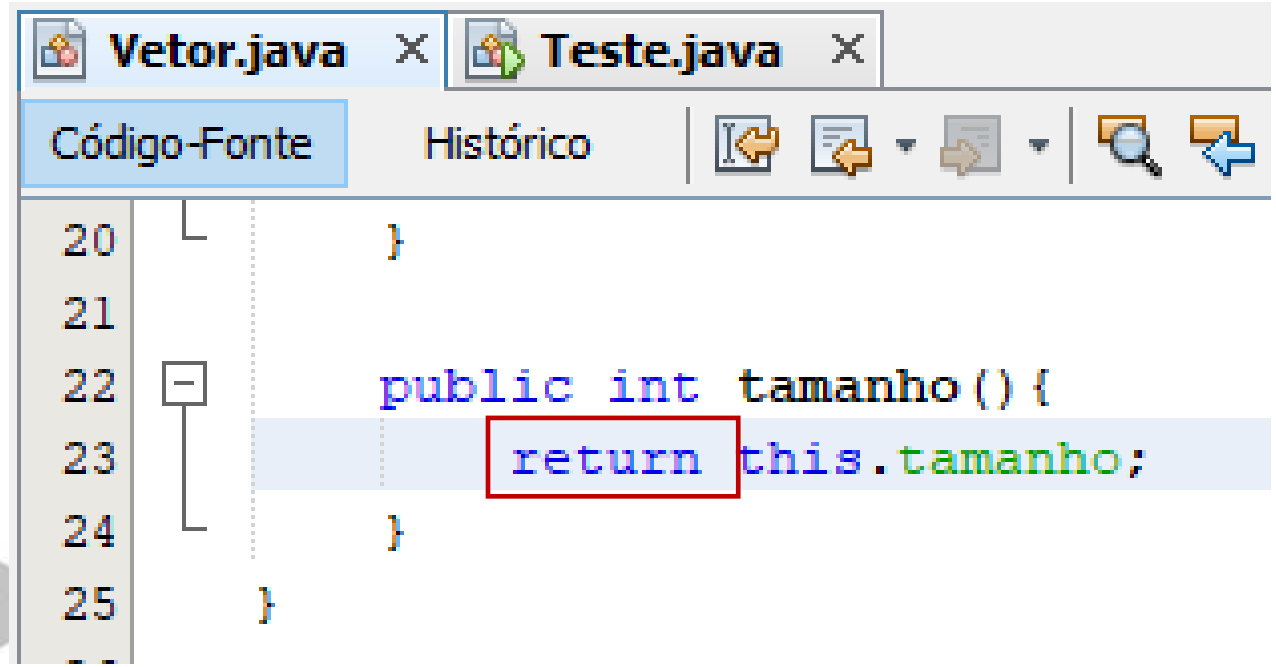
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args){
5           Vetor vetor = new Vetor(5);
6
7           try {
8               vetor.adiciona("elemento 1");
9               vetor.adiciona("elemento 2");
10          } catch (Exception e) {
12              e.printStackTrace();
            }
13      }
14  }
```

# Arranjos – Array (Vetores ou Matrizes)

Agora vamos implementar os métodos para informar o tamanho real de um vetor e como imprimir os elementos do nosso vetor:




Como o próprio nome diz, o **return serve** para retornar algo de dentro do método!
Sendo assim, todo método que não seja void está informando ao **Java** que ele vai retornar um valor e, por isso, obrigatoriamente deverá utilizar o **return** para devolver um valor!

# Arranjos – Array (Vetores ou Matrizes)

Observe que não criamos métodos de acesso como o **Get** e **Set** para forçar o controle interno da classe e evitar o usuário que vai utilizar a nossa classe não modifique essa informação.

Isso será visto em linguagem de programação, não em estrutura de dados!

# Arranjos – Array (Vetores ou Matrizes)

Vamos testar:

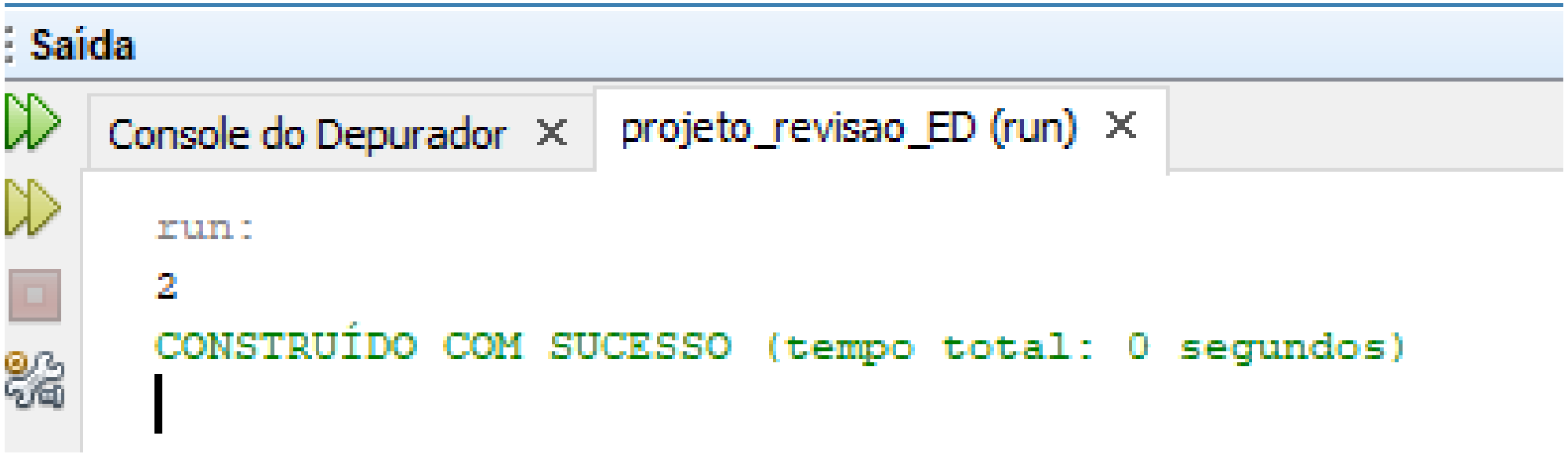
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args){
5           Vetor vetor = new Vetor(5);
6
7           try {
8               vetor.adiciona("elemento 1");
9               vetor.adiciona("elemento 2");
10          } catch (Exception e) {
12              e.printStackTrace();
13          }
14          System.out.println(vetor.tamanho());
15
16      }
17  }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:




Ok, mas agora queremos imprimir as informações armazenadas em nosso vetor nas posições ocupadas:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   import java.util.Arrays;
4
5   public class Vetor {
6       private String[] elementos;
7       private int tamanho;
8
9       public Vetor(int capacidade){...4 linhas }
13
14      public void adiciona(String elemento)throws Exception{...9 linhas }
23
24      public int tamanho(){...3 linhas }
27
28      public String toString(){
29          return Arrays.toString(elementos);
30      }
31  }
```

**Importando a Classe Arrays que tem uma série de métodos prontos para manipular as informações contidas em um vetor ou matriz.**

# Arranjos – Array (Vetores ou Matrizes)

Vamos testar:

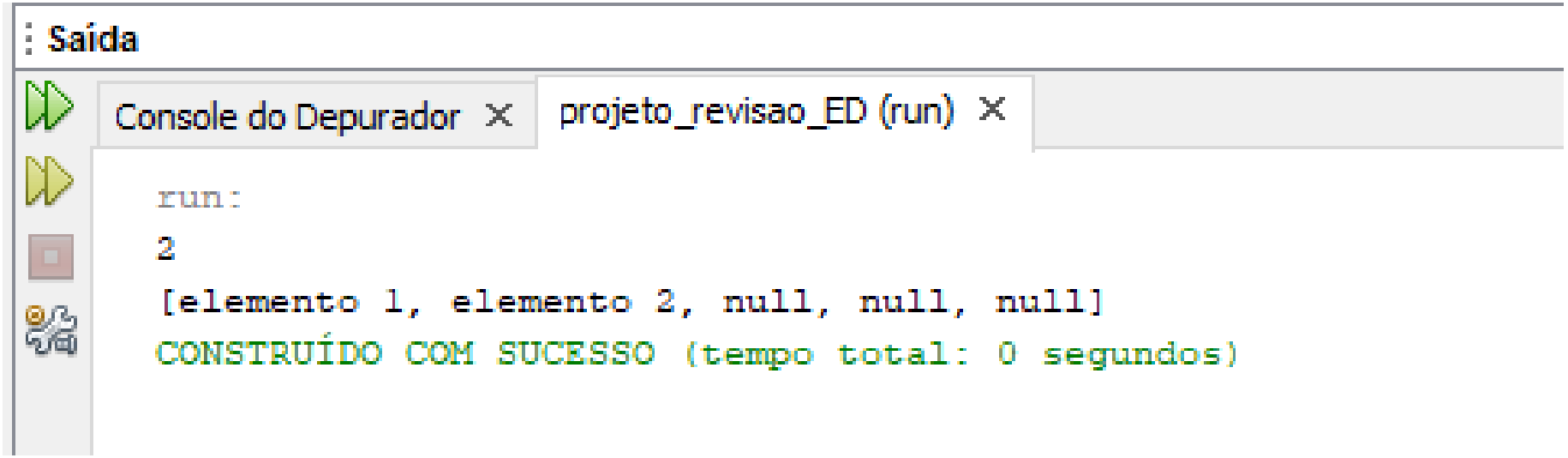
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args){
5           Vetor vetor = new Vetor(5);
6
7           try {
8               vetor.adiciona("elemento 1");
9               vetor.adiciona("elemento 2");
10          } catch (Exception e) {
12              e.printStackTrace();
13          }
14
15          System.out.println(vetor.tamanho());
16
17          System.out.println(vetor.toString());
18      }
    }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:




OK, porém, ele apresenta também os valores do vetor que estão com null, então vamos reescrever nosso método public String toString() para mostrar somente as posições com valores

Vetor.java | Teste.java

Código-Fonte | Histórico

```
28      @Override
29      public String toString(){
30          StringBuilder s = new StringBuilder();
31          s.append("[");
32
33          for(int i=0; i<this.tamanho-1; i++){
34              s.append(this.elementos[i]);
35              s.append(", ");
36          }
37
38          if(this.tamanho> 0){
39              s.append(this.elementos[this.tamanho-1]);
40          }
41
42          s.append("]");
43
44          return s.toString();
45      }
46  }
```

**@Override. É uma anotação marcadora que deve ser usada apenas com métodos. Serve para indicar que o método anotado está sobrescrevendo um método da superclasse.**

**O método append serve para adicionar mais conteúdo ao final de um StringBuffer.**

**A classe StringBuilder faz parte do pacote java. Essa classe permite criar e manipular dados de Strings dinamicamente, ou seja, podem criar variáveis de String modificáveis.**

# Arranjos – Array (Vetores ou Matrizes)

Vamos testar:

Outra Vez...

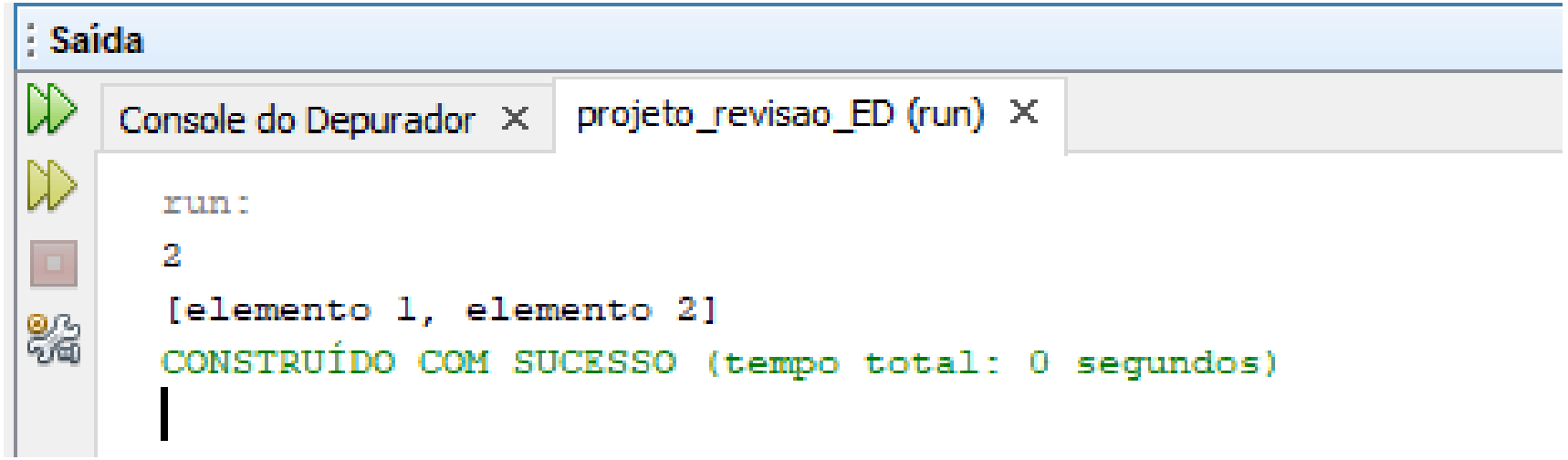
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args){
5           Vetor vetor = new Vetor(5);
6
7           try {
8               vetor.adiciona("elemento 1");
9               vetor.adiciona("elemento 2");
10          } catch (Exception e) {
12              e.printStackTrace();
13          }
14          System.out.println(vetor.tamanho());
15
16          System.out.println(vetor.toString());
17      }
18  }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:




Nesse caso o resultado fica muito melhor e mais *clean* sem apresentar um monte de valores *null*.

# Arranjos – Array (Vetores ou Matrizes)

OK estamos no caminho certo...

Agora vamos determinar como obter um elemento de uma determinada posição, para isso vamos criar o método de **busca** na classe **Vetor**.

# Arranjos – Array (Vetores ou Matrizes)

Método de Busca:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
46
47      public String busca(int posicao)throws Exception{
48          if(posicao >=0 && posicao < tamanho){
49              return elementos[posicao];
50          } else {
51              throw new Exception("Posição Inválida");
52          }
53      }
54  }
55
```

# Arranjos

Testando:

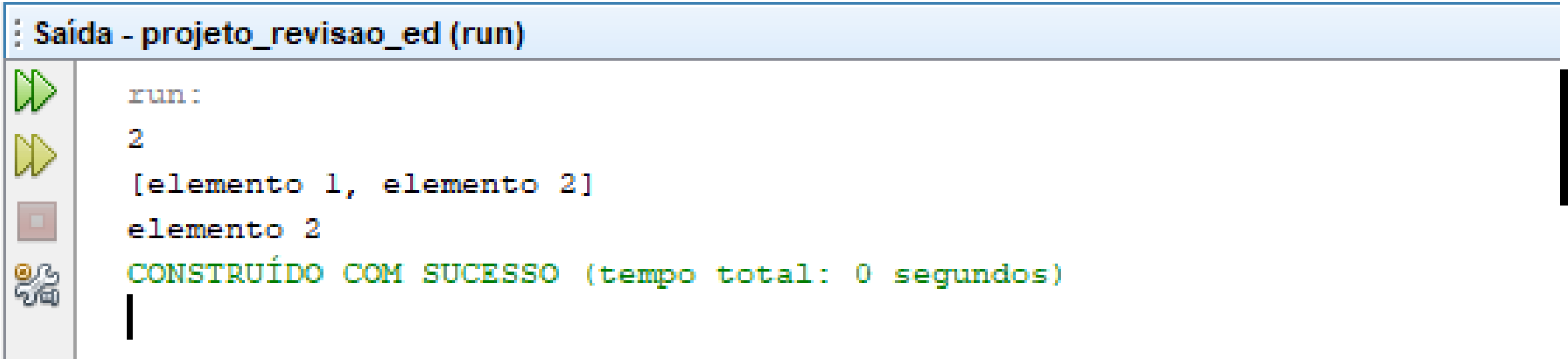
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Teste {
4      public static void main(String[] args) throws Exception{
5          Vetor vetor = new Vetor(5);
6
7          try {
8              vetor.adiciona("elemento 1");
9              vetor.adiciona("elemento 2");
10         } catch (Exception e) {
11             e.printStackTrace();
12         }
13
14         System.out.println(vetor.tamanho());
15
16         System.out.println(vetor.toString());
17
18         System.out.println(vetor.busca(1));
19     }
20 }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:



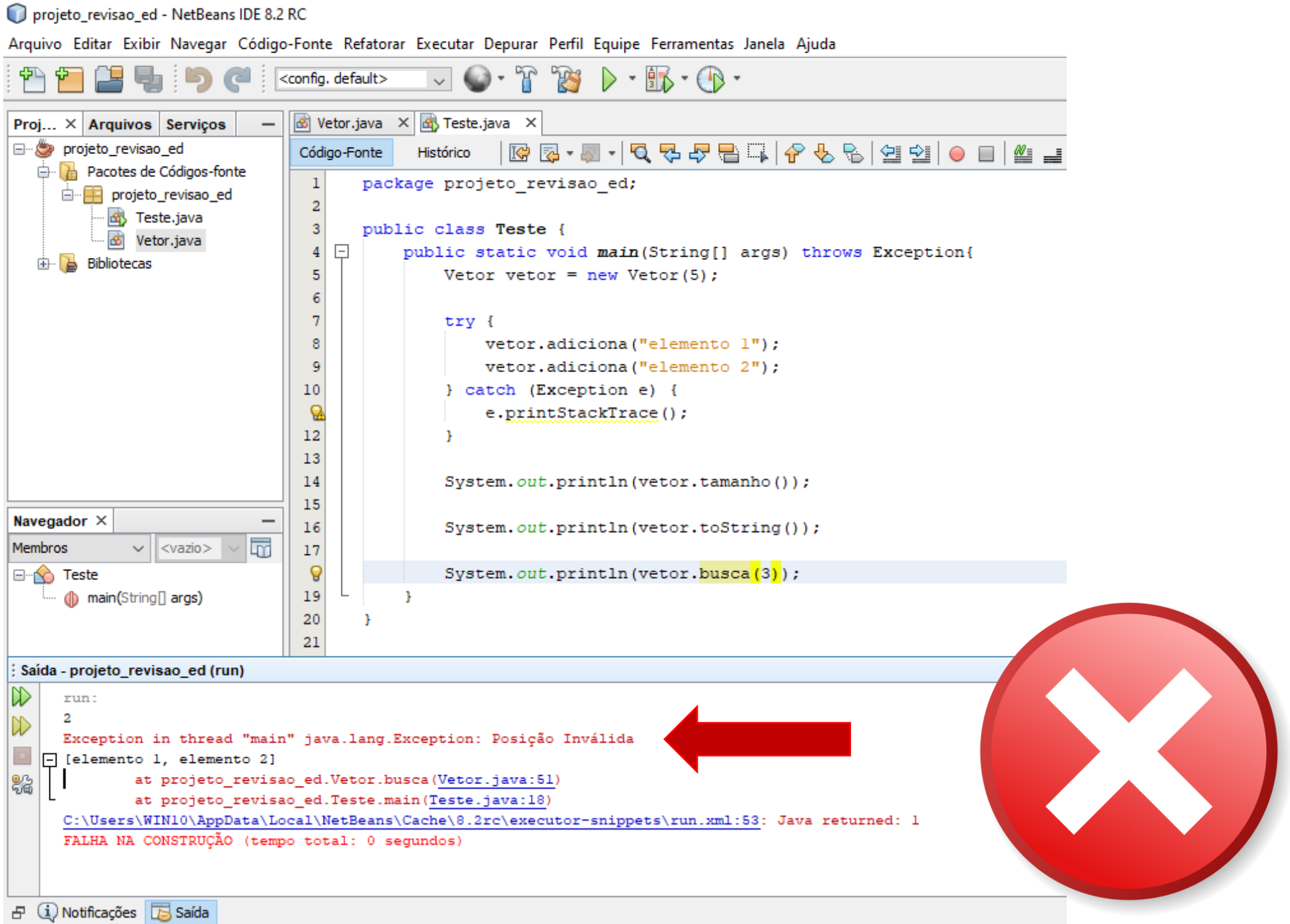
# Arranjos – Array (Vetores ou Matrizes)

Vamos provocar um erro propositalmente para testar a exceção quando solicitamos uma posição que não está preenchida.


Vetor.java | Teste.java

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Teste {
4      public static void main(String[] args) throws Exception{
5          Vetor vetor = new Vetor(5);
6
7          try {
8              vetor.adiciona("elemento 1");
9              vetor.adiciona("elemento 2");
10         } catch (Exception e) {
               e.printStackTrace();
12         }
13
14         System.out.println(vetor.tamanho());
15
16         System.out.println(vetor.toString());
17
           System.out.println(vetor.busca(3));
19     }
20 }
```


projeto_revisao_ed - NetBeans IDE 8.2 RC
Arquivo Editar Exibir Navegar Código-Fonte Refatorar Executar Depurar Perfil Equipe Ferramentas Janela Ajuda
<config. default>
Proj... Arquivos Serviços
projeto_revisao_ed
Pacotes de Códigos-fonte
projeto_revisao_ed
Teste.java
Vetor.java
Bibliotecas
Navegador
Membros
<vazio>
Teste
main(String[] args)
Vetor.java
Teste.java
Código-Fonte
Histórico
1 package projeto_revisao_ed;
2
3 public class Teste {
4 public static void main(String[] args) throws Exception{
5 Vetor vetor = new Vetor(5);
6
7 try {
8 vetor.adiciona("elemento 1");
9 vetor.adiciona("elemento 2");
10 } catch (Exception e) {
e.printStackTrace();
12 }
13
14 System.out.println(vetor.tamanho());
15
16 System.out.println(vetor.toString());
17
System.out.println(vetor.busca(3));
19 }
20 }
21
Saída - projeto_revisao_ed (run)
run:
2
Exception in thread "main" java.lang.Exception: Posição Inválida
[elemento 1, elemento 2]
at projeto_revisao_ed.Vetor.busca(Vetor.java:51)
at projeto_revisao_ed.Teste.main(Teste.java:18)
C:\Users\WIN10\AppData\Local\NetBeans\Cache\8.2rc\executor-snippets\run.xml:53: Java returned: 1
FALHA NA CONSTRUÇÃO (tempo total: 0 segundos)
Notificações
Saída

# Arranjos – Array (Vetores ou Matrizes)

Vamos fazer uma verificação para determinar se um determinado elemento existe no vetor, e em caso de positivo, ele deve informar a posição no vetor, para isso, vamos modificar o nosso método de busca.

Sabemos que existe vários algoritmos que podem ser utilizados para busca, porém vamos usar o mais simples nesse exemplo, que é a busca sequencial.

# Arranjos – Array (Vetores ou Matrizes)

Novo método de busca:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
54
55      public int busca1(String elemento){
56          for (int i=0; i<tamanho; i++){
57              if(elementos[i].equals(elemento)){
58                  return i;
59              }
60          }
61          return -1;
62      }
63  }
```

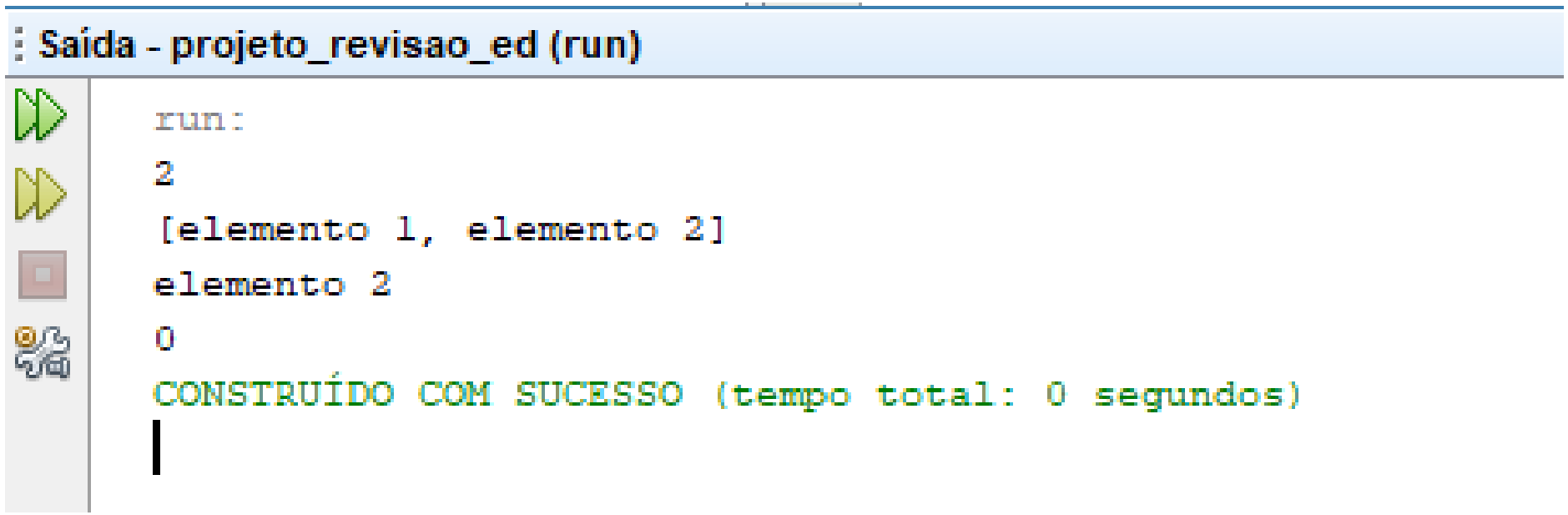
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args) throws Exception{
5           Vetor vetor = new Vetor(5);
6
7           try {
8               vetor.adiciona("elemento 1");
9               vetor.adiciona("elemento 2");
10          } catch (Exception e) {
11              e.printStackTrace();
12          }
13
14          System.out.println(vetor.tamanho());
15
16          System.out.println(vetor.toString());
17
18          System.out.println(vetor.busca(1));
19
20          System.out.println(vetor.buscal("elemento 1"));
21      }
22  }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:



# Arranjos – Array (Vetores ou Matrizes)

Vamos Adicionar um novo elemento em qualquer posição, imagine que eu queira inserir um valor na primeira posição, por exemplo:

| Elemento 1 | Elemento 2 | null | null | null |
|---|---|---|---|---|
| Posição 0 | Posição 1 | Posição 2 | Posição 3 | Posição 4 |



Caso eu queira inserir uma informação como “elemento 0” na posição 0 e mover demais valores para as posições seguintes, ficando como na imagem abaixo:

| Elemento 0 | Elemento 1 | Elemento 2 | null | null |
|---|---|---|---|---|
| Posição 0 | Posição 1 | Posição 2 | Posição 3 | Posição 4 |

# Arranjos – Array (Vetores ou Matrizes)

Vamos criar um novo método na classe Vetor:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
64      public boolean adicionaInicio(int posicao, String elemento)throws Exception{
65          if(posicao >=0 && posicao < tamanho){
66              for(int i=this.tamanho-1; i>posicao; i--){
67                  this.elementos[i+1] = this.elementos[i];
68              }
69              this.elementos[posicao] = elemento;
70              this.tamanho++;
71          } else {
72              throw new Exception("Posição Inválida");
73          }
74
75          return true;
76      }
77
78  }
```

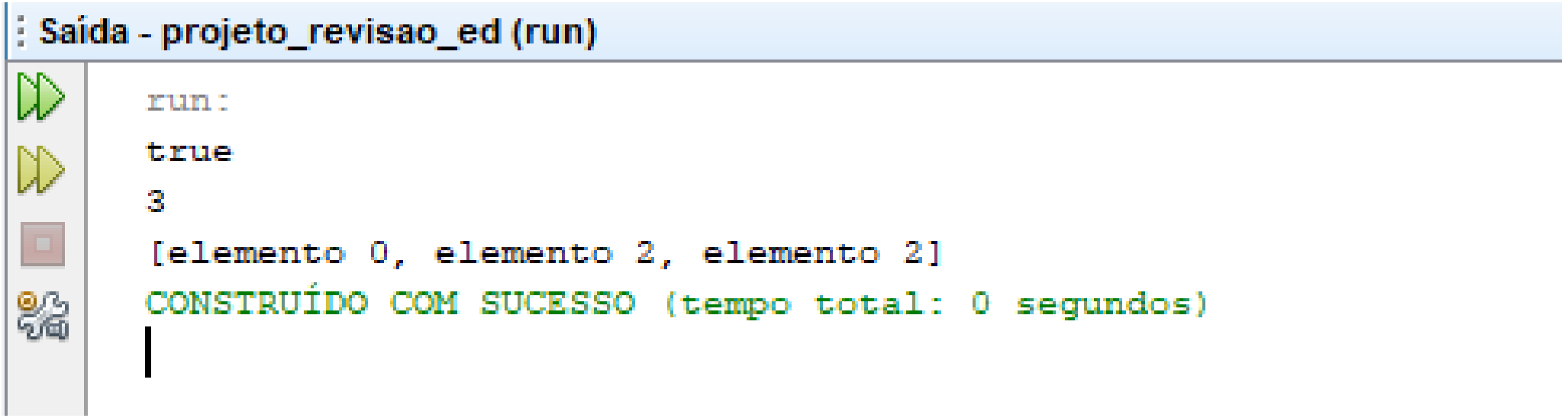
Vetor.java | **Teste.java**

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Teste {
4      public static void main(String[] args) throws Exception{
5          Vetor vetor = new Vetor(5);
6
7          try {
8              vetor.adiciona("elemento 1");
9              vetor.adiciona("elemento 2");
10         } catch (Exception e) {
               e.printStackTrace();
12         }
13
           System.out.println(vetor.adicionaInicio(0, "elemento 0"));
15
16         System.out.println(vetor.tamanho());
17
18         System.out.println(vetor.toString());
19     }
20 }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:



# Arranjos – Array (Vetores ou Matrizes)

Agora imagine que durante a execução de suas tarefas você identificou uma necessidade de armazenar mais informações, ou seja, tem que aumentas o tamanho do vetor.



# Arranjos – Array (Vetores ou Matrizes)

Normalmente quando trabalhamos com vetor, nem sempre sabemos o tamanho exato, para isso, precisamos desenvolver códigos que possamos determinar o tamanho do nosso vetor de forma dinâmica, de acordo com a nossa necessidade.

# Arranjos – Array (Vetores ou Matrizes)

Como **não podemos modificar o nosso vetor depois de inicializado**, vamos ter que **criar um processo** para criar um novo vetor com maior capacidade (recomenda-se dobrar o tamanho quando ele estourar) e fazer uma copia dos elementos do vetor que acabou a capacidade para o novo vetor e depois vamos atribuir ao vetor inicial o novo vetor (sobrepor).

# Arranjos – Array (Vetores ou Matrizes)

Vamos criar um novo método na classe Vetor chamado **aumentaCapacidade()**:

Vetor.java | Teste.java

Código-Fonte | Histórico

```
78      private void aumentaCapacidade(){
79          if (this.tamanho == this.elementos.length){
80              String[] elementosNovos = new String[this.elementos.length * 2];
                for (int i=0; i<this.elementos.length; i++){
82                  elementosNovos[i] = this.elementos[i];
83              }
84              this.elementos = elementosNovos;
85          }
86      }
87  }
```

# Arranjos – Array (Vetores ou Matrizes)

Lembrando que esse código só vai ser executado quando atingir a capacidade máxima do vetor, então precisamos inserir esse método dentro dos métodos de adição de novos elementos.

# Arranjos – Array (Vetores ou Matrizes)

```
public void adiciona(String elemento)throws Exception{
    this.aumentaCapacidade();
    if (this.tamanho < this.elementos.length){
        this.elementos[this.tamanho] = elemento;
        this.tamanho++;
    } else {
        throw new Exception("O Vetor já está cheio, "
                + "não é possível adiconar novos elementos");
    }
}
```

# Arranjos – Array (Vetores ou Matrizes)

E também:

```
public boolean adicionaInicio(int posicao, String elemento)throws Exception{
    this.aumentaCapacidade();
    if(posicao >=0 && posicao < tamanho){
        for(int i=this.tamanho-1; i>posicao; i--){
            this.elementos[i+1] = this.elementos[i];
        }
        this.elementos[posicao] = elemento;
        this.tamanho++;
     } else {
        throw new Exception("Posição Inválida");
     }

    return true;
}
```

Teste:

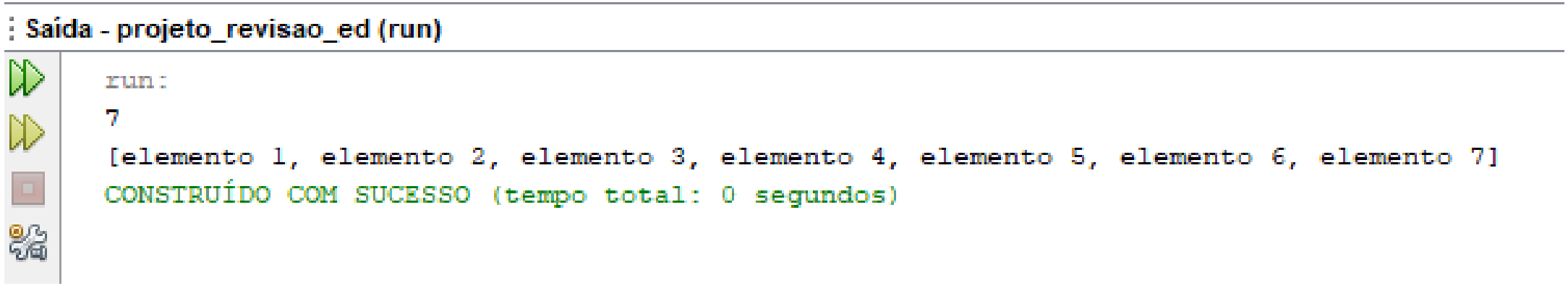
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1   package projeto_revisao_ed;
2
3   public class Teste {
4       public static void main(String[] args) throws Exception{
5           Vetor vetor = new Vetor(5);
6
7           try {
8               vetor.adiciona("elemento 1");
9               vetor.adiciona("elemento 2");
10              vetor.adiciona("elemento 3");
11              vetor.adiciona("elemento 4");
12              vetor.adiciona("elemento 5");
13              vetor.adiciona("elemento 6");
14              vetor.adiciona("elemento 7");
15          } catch (Exception e) {
16              e.printStackTrace();
17          }
18
19          System.out.println(vetor.tamanho());
20
21          System.out.println(vetor.toString());
22      }
23  }
24
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:



# Arranjos – Array (Vetores ou Matrizes)

Vamos agora remover um elemento do vetor:

Imagine que precisamos remover um elemento do vetor abaixo, no caso o "**Elemento 3**" que está na posição 2.



**vetor**

| Elemento 1 | Elemento 2 | Elemento 3 | Elemento4 | Elemento 5 | Elemento 6 | Elemento 7 | null | null | null |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Arranjos – Array (Vetores ou Matrizes)

Vamos criar o método remove():

Vetor.java | Teste.java

Código-Fonte | Histórico

```
89
90      public void remove(int posicao) throws Exception{
91          if(posicao >=0 && posicao < tamanho){
92              for(int i=posicao; i<this.tamanho-1; i++){
93                  this.elementos[i] = this.elementos[i+1];
94              }
95              this.tamanho--;
96          } else {
97              throw new Exception("Posição Inválida");
98          }
99      }
100 }
101
```

Teste:

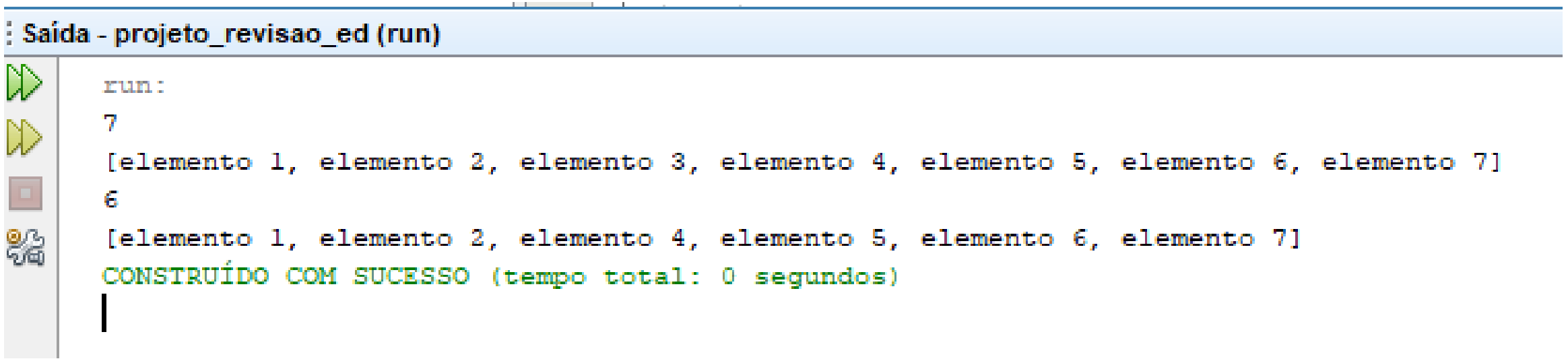
Vetor.java | Teste.java

Código-Fonte | Histórico

```
1  package projeto_revisao_ed;
2
3  public class Teste {
4      public static void main(String[] args) throws Exception{
5          Vetor vetor = new Vetor(5);
6
7          try {
8              vetor.adiciona("elemento 1");
9              vetor.adiciona("elemento 2");
10             vetor.adiciona("elemento 3");
11             vetor.adiciona("elemento 4");
12             vetor.adiciona("elemento 5");
13             vetor.adiciona("elemento 6");
14             vetor.adiciona("elemento 7");
15         } catch (Exception e) {
               e.printStackTrace();
17         }
18
19         System.out.println(vetor.tamanho());
20
21         System.out.println(vetor.toString());
22
23         vetor.remove(2);
24
25         System.out.println(vetor.tamanho());
26
27         System.out.println(vetor.toString());
28     }
29 }
```

# Arranjos – Array (Vetores ou Matrizes)

Resultado:



"O sucesso é a soma de pequenos esforços repetidos dia após dia"

**Robert Collier**

# Obrigado!

**Se precisar …**

Prof. Claudio Benossi

claudio.benossi@sp.senac.br