



Senac

Estrutura de Dados

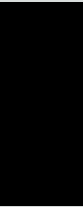
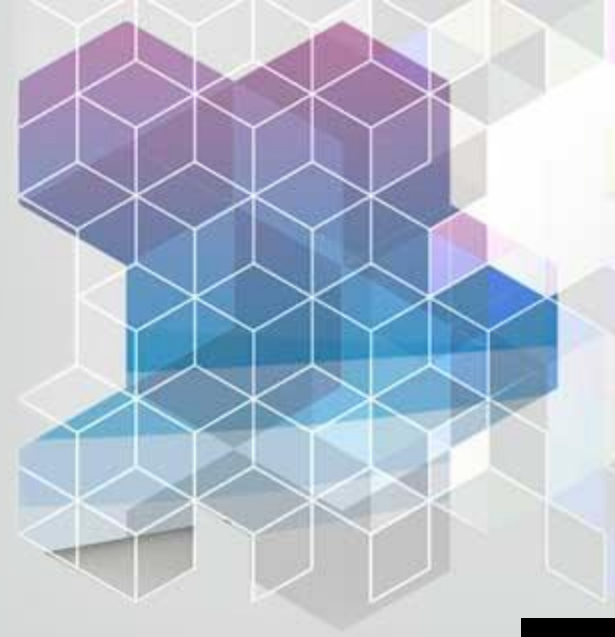
Curso Tecnologia em Análise de Desenvolvimento de Sistemas

Aula 03

Prof. Claudio Benossi

1. Unidade

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



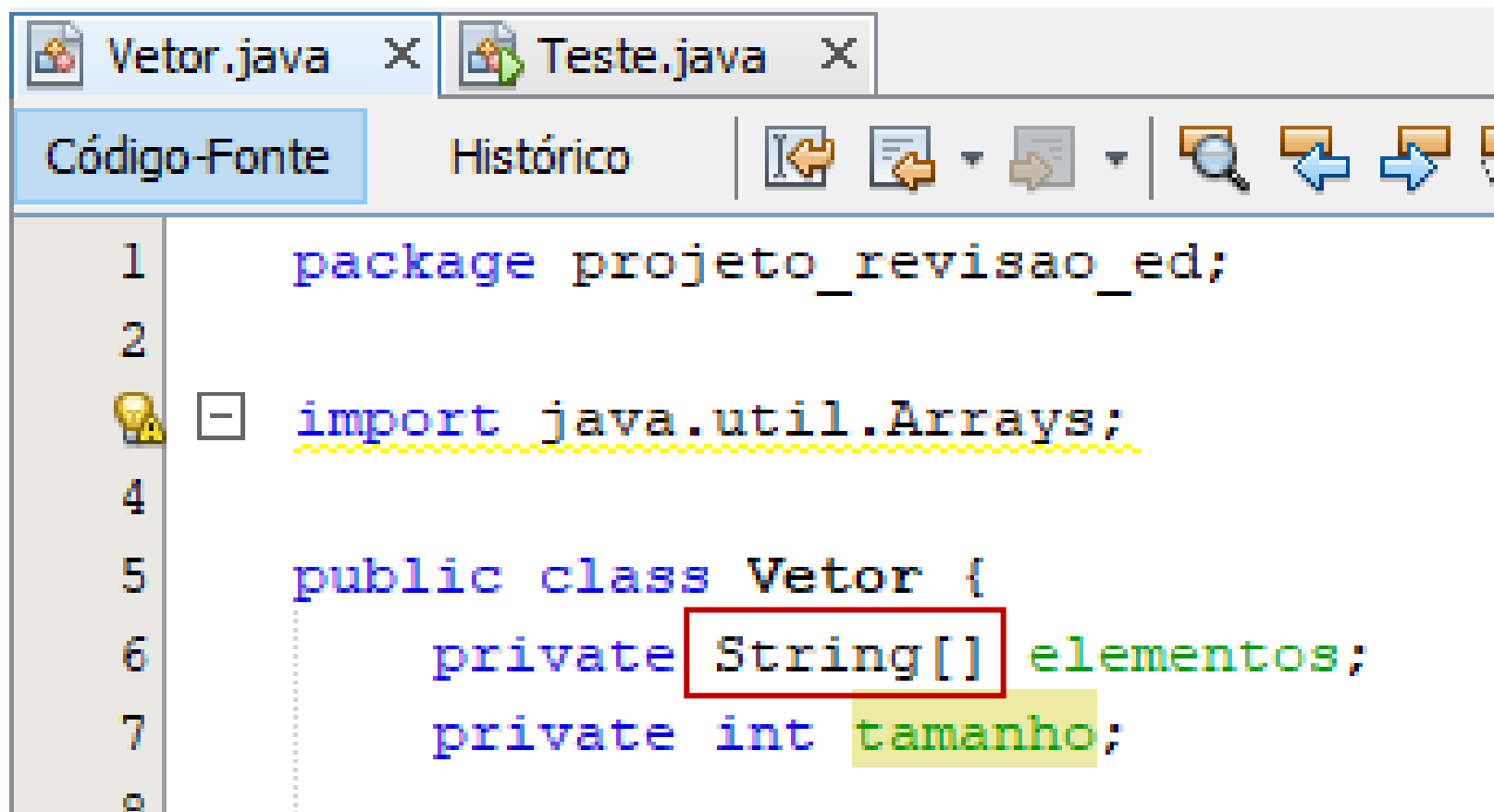
Arranjos – Array (Vetores ou Matrizes)

Pessoal em nossos exemplos anteriores estamos trabalhando com um vetor do tipo **String**, porém podemos fazer uma simples modificação em nosso código que permite criar nosso vetor com diferentes tipos de dados, basta passar o parâmetro **String** para **Object**.



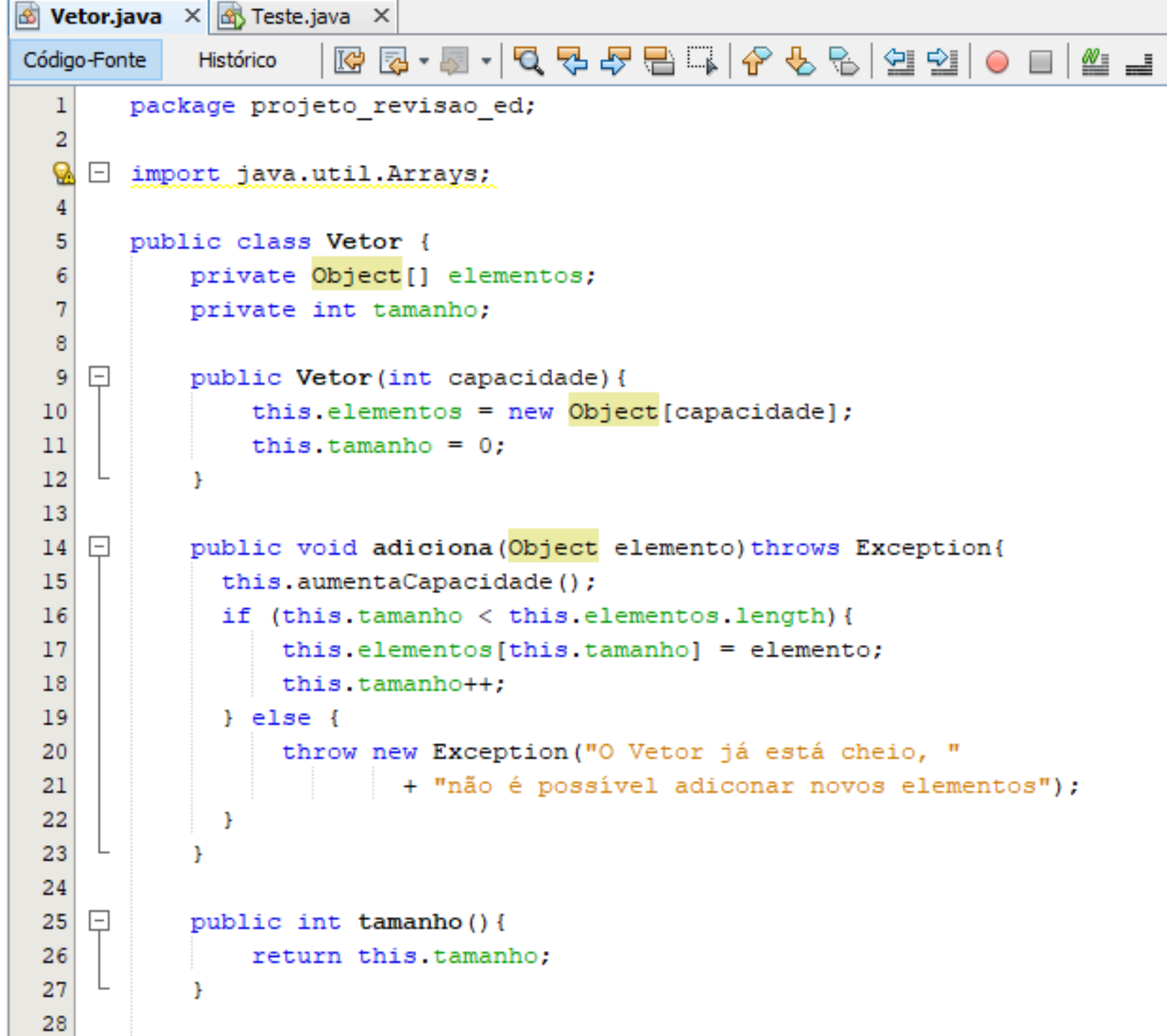
Arranjos – Array (Vetores ou Matrizes)

Nosso código conforme a aula anterior:



```
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 }
```

Modificar:



```
1 package projeto_revisao_ed;
2
3 import java.util.Arrays;
4
5 public class Vetor {
6     private Object[] elementos;
7     private int tamanho;
8
9     public Vetor(int capacidade) {
10         this.elementos = new Object[capacidade];
11         this.tamanho = 0;
12     }
13
14     public void adiciona(Object elemento) throws Exception {
15         this.aumentaCapacidade();
16         if (this.tamanho < this.elementos.length) {
17             this.elementos[this.tamanho] = elemento;
18             this.tamanho++;
19         } else {
20             throw new Exception("O Vetor já está cheio, "
21                 + "não é possível adicionar novos elementos");
22         }
23     }
24
25     public int tamanho() {
26         return this.tamanho;
27     }
28 }
```

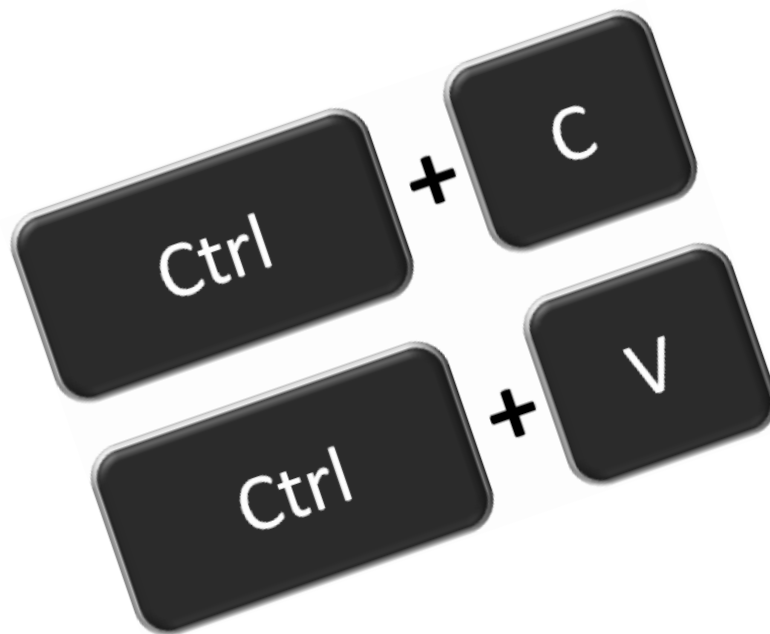
Arranjos – Array (Vetores ou Matrizes)

Agora você pode testar passando como parâmetros diferentes tipos de dados...



Arranjos – Array (Vetores ou Matrizes)

OK, então vamos criar uma nova classe em nosso pacote com o nome de VetorObjeto que a principio é uma copia da nossa classe Vetor, porém alterando o parâmetro String para Object.



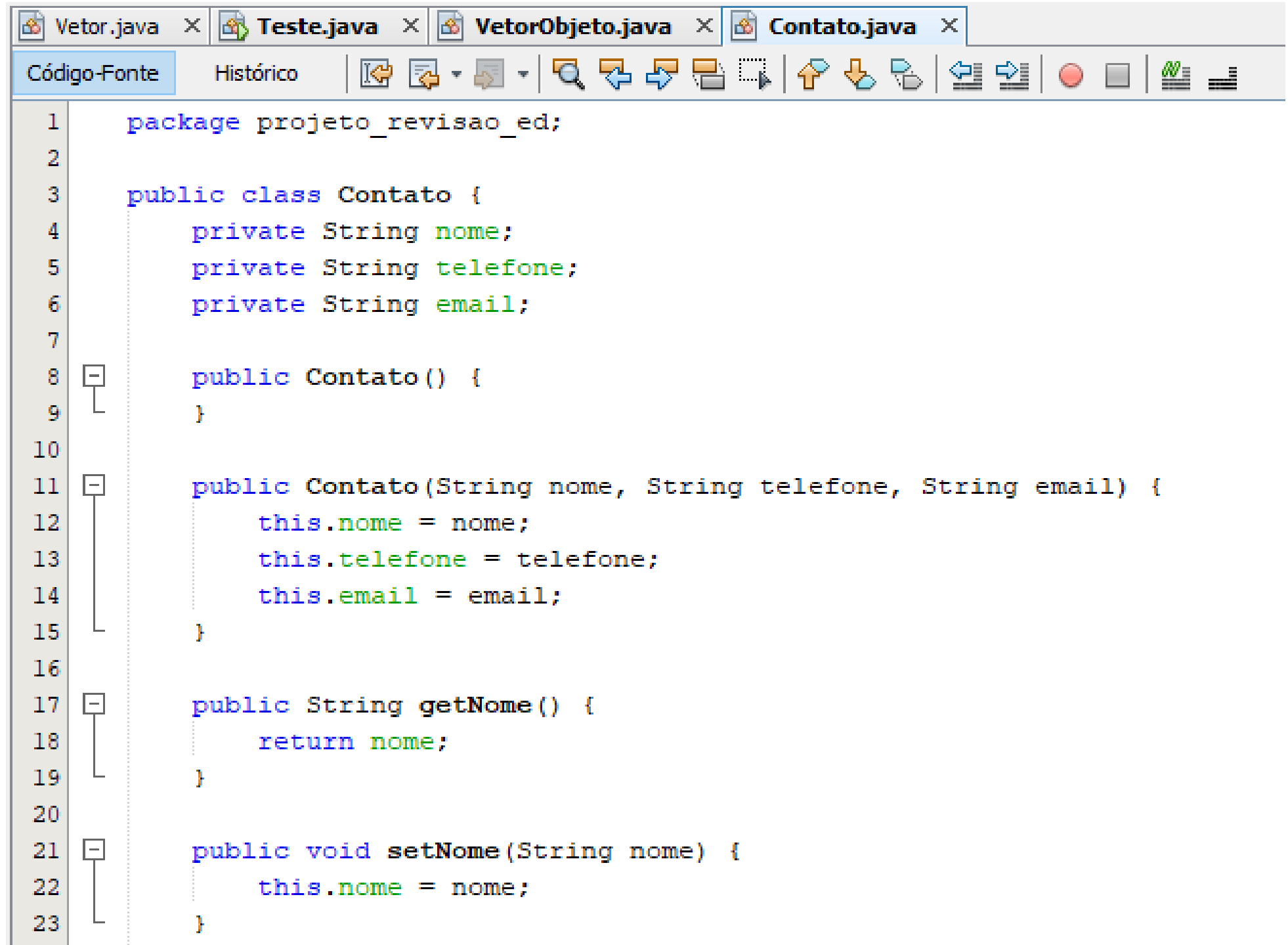
```
1 package projeto_revisao_ed;
2
3 import java.util.Arrays;
4
5 public class VetorObjeto {
6     private Object[] elementos;
7     private int tamanho;
8
9     public VetorObjeto(int capacidade) {
10         this.elementos = new Object[capacidade];
11         this.tamanho = 0;
12     }
13
14     public void adiciona(Object elemento) throws Exception {
15         this.aumentaCapacidade();
16         if (this.tamanho < this.elementos.length) {
17             this.elementos[this.tamanho] = elemento;
18             this.tamanho++;
19         } else {
20             throw new Exception("O Vetor já está cheio, "
21                 + "não é possível adicionar novos elementos");
22         }
23     }
24
25     public int tamanho() {
26         return this.tamanho;
27     }
```




Arranjos – Array (Vetores ou Matrizes)

Agora para melhorar nosso exemplo, vamos criar uma classe **Contado**, contendo os atributos **nome**, **telefone** e **email**, todos do tipo **String**, será necessário criar também os métodos de acesso **Get** e **Set**, assim como o **construtor** e um método **toString** para organizar os dados na hora da exibição dos dados.

1ª parte



```
1 package projeto_revisao_ed;
2
3 public class Contato {
4     private String nome;
5     private String telefone;
6     private String email;
7
8     public Contato() {
9     }
10
11     public Contato(String nome, String telefone, String email) {
12         this.nome = nome;
13         this.telefone = telefone;
14         this.email = email;
15     }
16
17     public String getNome() {
18         return nome;
19     }
20
21     public void setNome(String nome) {
22         this.nome = nome;
23     }
```

2º parte

```
24  
25 - public String getTelefone() {  
26     return telefone;  
27 }  
  
28  
29 - public void setTelefone(String telefone) {  
30     this.telefone = telefone;  
31 }  
32  
33 - public String getEmail() {  
34     return email;  
35 }  
36  
37 - public void setEmail(String email) {  
38     this.email = email;  
39 }  
40  
41 @Override  
42 - public String toString() {  
43     return "Contato{" + "nome=" + nome + ", telefone=" + telefone + ", email=" + email + '}';  
44 }  
45  
46 }
```

Arranjos – Array (Vetores ou Matrizes)

Agora para testar, vamos criar alguns contatos, chamar os métodos para verificar o tamanho do vetor e para exibir os dados inseridos.





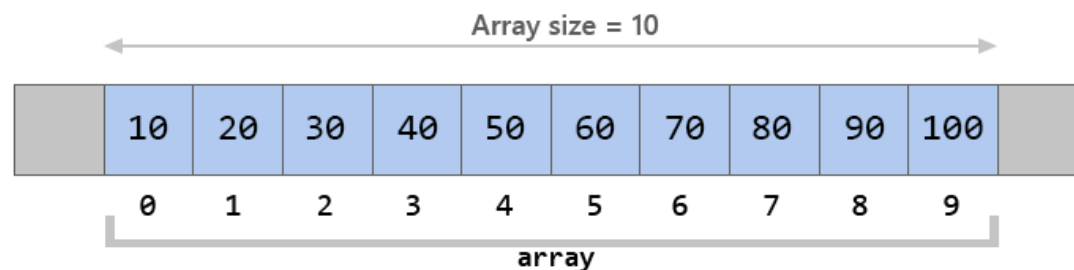
```
1 package projeto_revisao_ed;
2
3 public class Teste {
4     public static void main(String[] args) throws Exception{
5         VetorObjeto vetor = new VetorObjeto(5);
6
7         Contato c1 = new Contato("Claudio","9999-2525","claudio@gmail.com");
8         Contato c2 = new Contato("Daniel","9999-7213","daniel@gmail.com");
9         Contato c3 = new Contato("Miguel","9999-0101","miguel@gmail.com");
10
11         try {
12             vetor.adiciona(c1);
13             vetor.adiciona(c2);
14             vetor.adiciona(c3);
15         } catch (Exception e) {
16             e.printStackTrace();
17         }
18
19         System.out.println("Tamanho do vetor: " + vetor.tamanho());
20
21         System.out.println(vetor);
22     }
23 }
```

Arranjos – Array (Vetores ou Matrizes)

Resultado:

```
run:
Tamanho do vetor: 3
[Contato{nome=Claudio, telefone=9999-2525, email=claudio@gmail.com}, Contato{nome=Daniel, telefone=9999-7213, email=daniel@gmail.com}, Contato{nome=Miguel, telefone=9999-0101, email=miguel@gmail.com}]
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Arranjos – Array (Vetores ou Matrizes)



Para encerrar o assunto Array, vamos falar sobre o **ArraList**, uma das funcionalidades mais utilizadas quando falamos em estrutura de dados com vetor.




Arranjos – Array (Vetores ou Matrizes)

Podemos dizer que **ArrayList** é uma classe para coleções.

Uma classe genérica (generic classes), para ser mais exato.

Coleções mesmo, de qualquer tipo de 'coisa' ou dado, não somente de tipos primitivos.





Arranjos – Array (Vetores ou Matrizes)

A classe **ArrayList** é uma implementação da interface List que utiliza um vetor para armazenar elementos.

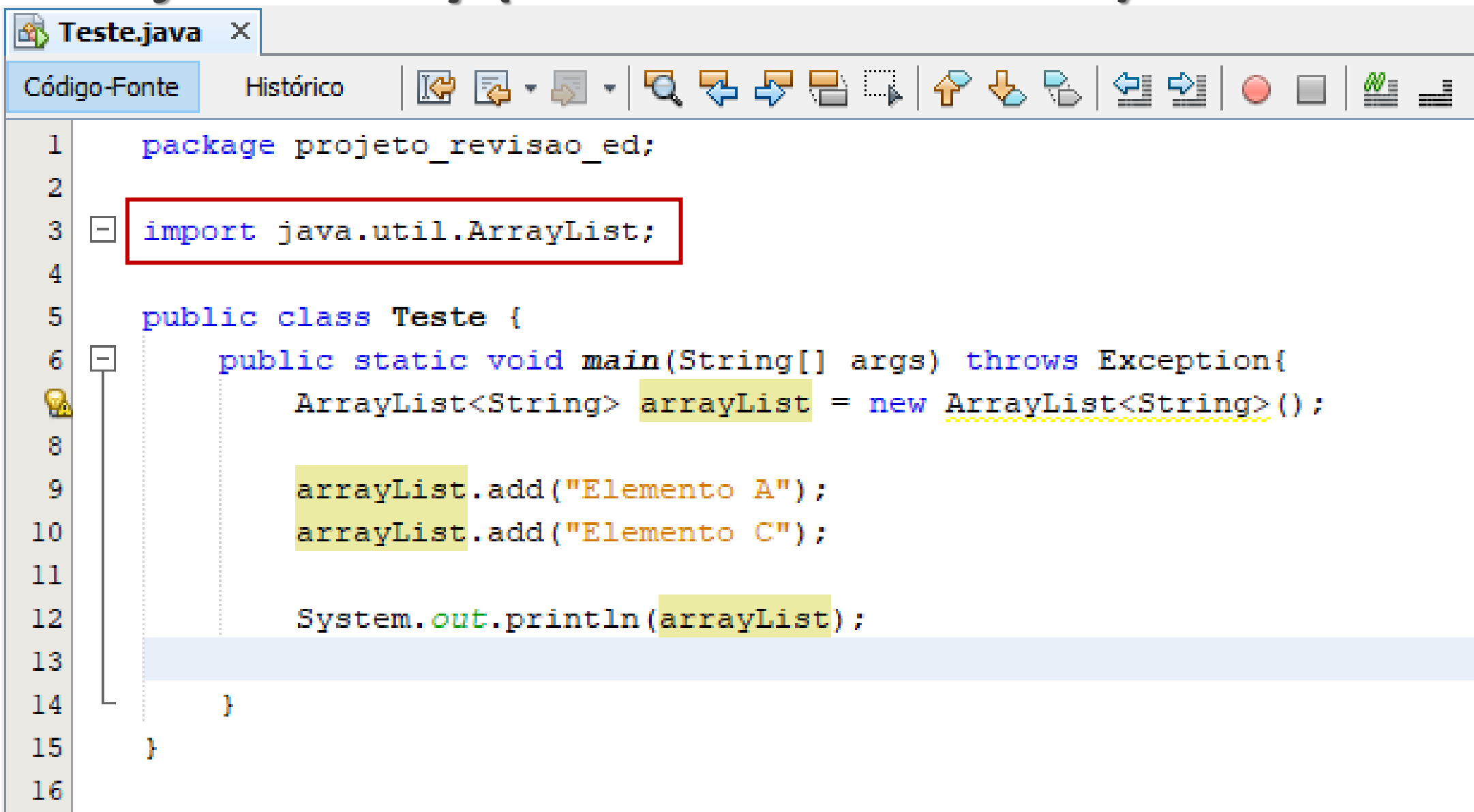
O vetor interno da classe **ArrayList** é recriado quando há remoções de elemento, adições de elemento no fim da lista além da capacidade dimensionada e adições de elementos que não no final da lista.

Arranjos – Array (Vetores ou Matrizes)

A classe **ArrayList** já vem com vários métodos prontos na linguagem Java, vamos conferir alguns ...



Arranjos – Array (Vetores ou Matrizes)



The screenshot shows an IDE window titled "Teste.java". The interface includes a toolbar with icons for file operations (open, save, print), navigation (back, forward, search), and development (run, debug, test). The code editor displays the following Java code:

```
1 package projeto_revisao_ed;
2
3 import java.util.ArrayList;
4
5 public class Teste {
6     public static void main(String[] args) throws Exception{
7         ArrayList<String> arrayList = new ArrayList<String>();
8
9         arrayList.add("Elemento A");
10        arrayList.add("Elemento C");
11
12        System.out.println(arrayList);
13
14    }
15 }
16
```

The code is syntactically correct. The line `import java.util.ArrayList;` is highlighted with a red rectangular box. The line `ArrayList<String> arrayList = new ArrayList<String>();` is highlighted with a yellow background. The lines `arrayList.add("Elemento A");` and `arrayList.add("Elemento C");` are also highlighted with a yellow background. The line `System.out.println(arrayList);` is highlighted with a yellow background. The line `System.out.println(arrayList);` is highlighted with a light blue background. The line `System.out.println(arrayList);` is highlighted with a light blue background.

Arranjos – Array (Vetores ou Matrizes)

O método **add** insere o elemento dentro do vetor, semelhante ao método **adicionar** que criamos nos exemplos anteriores.

Resultado:

⋮ Saída - projeto_revisao_ed (run)



run:



[Elemento A, Elemento C]



CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

|



Arranjos – Array (Vetores ou Matrizes)

Também posso solicitar a adição de um novo elemento numa posição específica e ele ajusta automaticamente a estrutura do nosso vetor.



```
1 package projeto_revisao_ed;
2
3 import java.util.ArrayList;
4
5 public class Teste {
6     public static void main(String[] args) throws Exception{
7         ArrayList<String> arrayList = new ArrayList<String>();
8
9         arrayList.add("Elemento A");
10        arrayList.add("Elemento C");
11
12        System.out.println(arrayList);
13
14        arrayList.add(1, "Elemento B");
15
16        System.out.println(arrayList);
17    }
18 }
19 }
```

Arranjos – Array (Vetores ou Matrizes)

Resultado:

```
⋮ Saída - projeto_revisao_ed (run)

run:
[Elemento A, Elemento C]
[Elemento A, Elemento B, Elemento C]
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
|
```

Arranjos – Array (Vetores ou Matrizes)

Também podemos fazer buscas em nosso vetor com o método **contains**.





```
1 package projeto_revisao_ed;
2
3 import java.util.ArrayList;
4
5 public class Teste {
6     public static void main(String[] args) throws Exception{
7         ArrayList<String> arrayList = new ArrayList<String>();
8
9         arrayList.add("Elemento A");
10        arrayList.add("Elemento C");
11
12        System.out.println(arrayList);
13
14        arrayList.add(1, "Elemento B");
15
16        System.out.println(arrayList);
17
18        boolean existe = arrayList.contains("Elemento C");
19        if (existe){
20            System.out.println("Elementos encontrado no vetor");
21        } else {
22            System.out.println("Elementos não existe no vetor");
23        }
24    }
25 }
```

Arranjos – Array (Vetores ou Matrizes)

Resultado:

```
⋮ Saída - projeto_revisao_ed (run)

run:
[Elemento A, Elemento C]
[Elemento A, Elemento B, Elemento C]
Elementos encontrado no vetor
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Caso eu queira encontrar a posição no elemento no vetor também é possível.

Teste.java

Código-Fonte Histórico

```
1 package projeto_revisao_ed;
2 import java.util.ArrayList;
3
4 public class Teste {
5     public static void main(String[] args) throws Exception{
6         ArrayList<String> arrayList = new ArrayList<String>();
7
8         arrayList.add("Elemento A");
9         arrayList.add("Elemento C");
10        System.out.println(arrayList);
11        arrayList.add(1, "Elemento B");
12
13        System.out.println(arrayList);
14
15        boolean existe = arrayList.contains("Elemento C");
16        if (existe){
17            System.out.println("Elementos encontrado no vetor");
18        } else {
19            System.out.println("Elementos não existe no vetor");
20        }
21
22        int posicao = arrayList.indexOf("Elemento C");
23        if (posicao > -1){
24            System.out.println("Elementos encontrado na posição nº " + posicao);
25        } else {
26            System.out.println("Elementos não existe no vetor");
27        }
28    }
29 }
```

Arranjos – Array (Vetores ou Matrizes)

Resultado:

∴ Saída - projeto_revisao_ed (run)



run:



[Elemento A, Elemento C]



[Elemento A, Elemento B, Elemento C]



Elementos encontrado no vetor

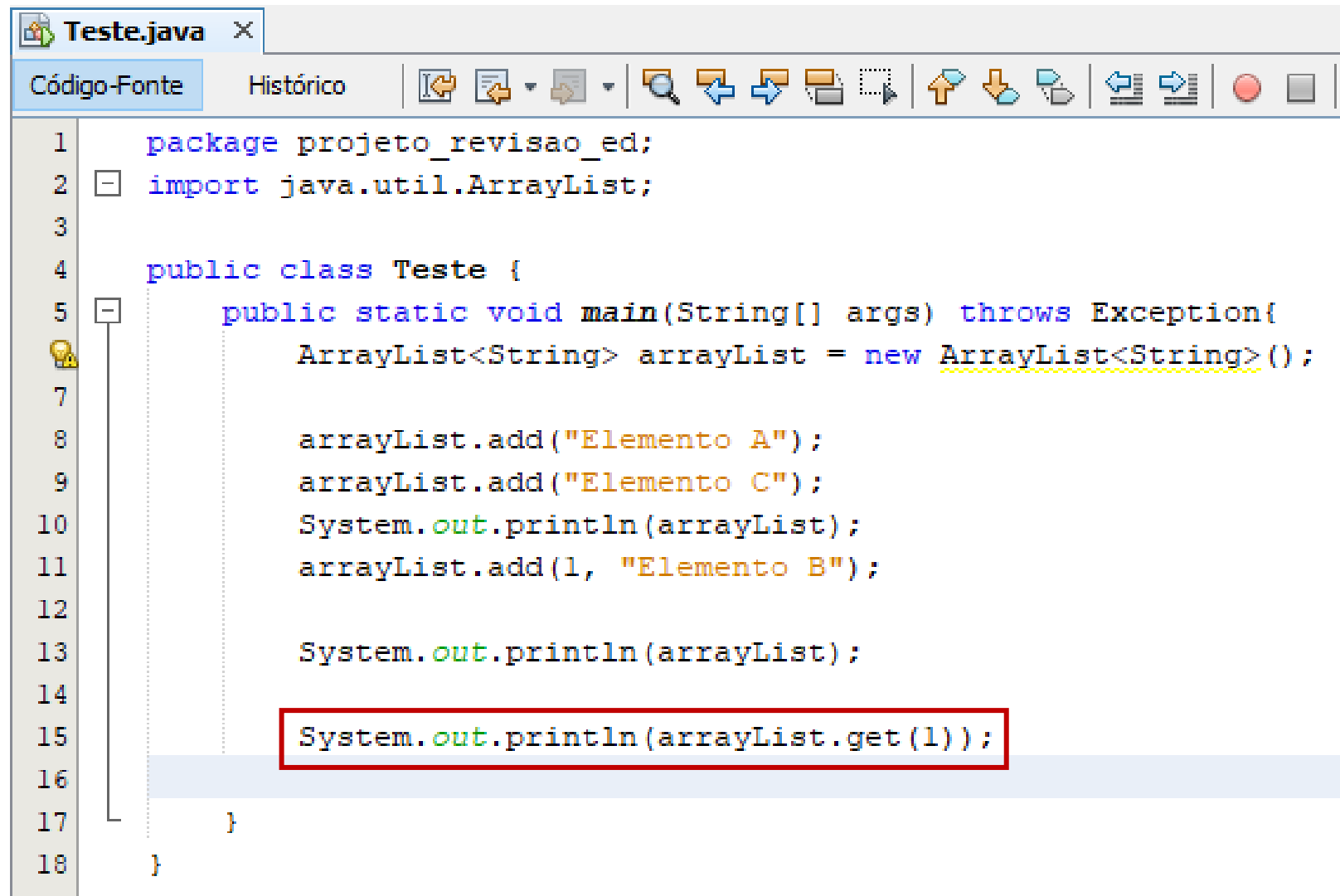
Elementos encontrado na posição nº 2

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

|

Arranjos – Array (Vetores ou Matrizes)

Método de busca por posição `arr.asList.get()`:



```
1 package projeto_revisao_ed;
2 import java.util.ArrayList;
3
4 public class Teste {
5     public static void main(String[] args) throws Exception{
6         ArrayList<String> arrayList = new ArrayList<String>();
7
8         arrayList.add("Elemento A");
9         arrayList.add("Elemento C");
10        System.out.println(arrayList);
11        arrayList.add(1, "Elemento B");
12
13        System.out.println(arrayList);
14
15        System.out.println(arrayList.get(1));
16    }
17 }
18 }
```

Arranjos – Array (Vetores ou Matrizes)

Resultado:

⋮ Saída - projeto_revisao_ed (run)



run:



[Elemento A, Elemento C]

[Elemento A, Elemento B, Elemento C]



Elemento B



CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

|

Arranjos – Array (Vetores ou Matrizes)

Agora vamos **remover** um item do nosso **ArrayList** com o **método remove**, que pode ser usado por índice ou por objeto:





```
1 package projeto_revisao_ed;
2 import java.util.ArrayList;
3
4 public class Teste {
5     public static void main(String[] args) throws Exception{
6         ArrayList<String> arrayList = new ArrayList<String>();
7
8         arrayList.add("Elemento A");
9         arrayList.add("Elemento C");
10        System.out.println(arrayList);
11        arrayList.add(1, "Elemento B");
12
13        System.out.println(arrayList);
14
15        arrayList.remove(0);
16        arrayList.remove("Elemento C");
17
18        System.out.println(arrayList);
19    }
20 }
```


Arranjos – Array (Vetores ou Matrizes)

Resultado:

⋮ Saída - projeto_revisao_ed (run)



run:



[Elemento A, Elemento C]



[Elemento A, Elemento B, Elemento C]



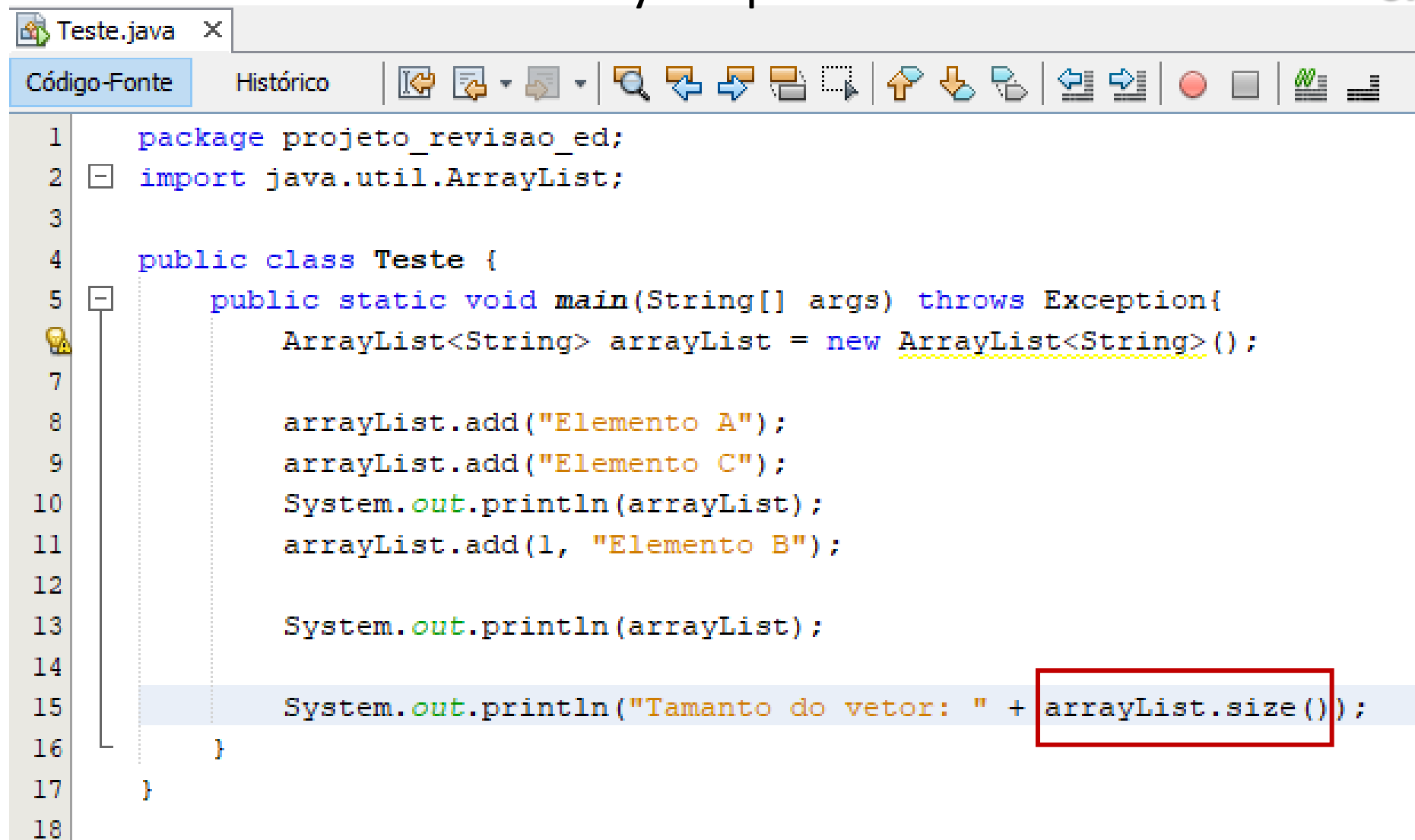
[Elemento B]

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

|

Arranjos – Array (Vetores ou Matrizes)

Para saber o tamanho do ArrayList podemos utilizar o método **size**:



The screenshot shows an IDE window titled 'Teste.java'. The code is as follows:

```
1 package projeto_revisao_ed;
2 import java.util.ArrayList;
3
4 public class Teste {
5     public static void main(String[] args) throws Exception{
6         ArrayList<String> arrayList = new ArrayList<String>();
7
8         arrayList.add("Elemento A");
9         arrayList.add("Elemento C");
10        System.out.println(arrayList);
11        arrayList.add(1, "Elemento B");
12
13        System.out.println(arrayList);
14
15        System.out.println("Tamanto do vetor: " + arrayList.size());
16    }
17 }
18
```

The line `arrayList.size()` on line 15 is highlighted with a red rectangular box.

Arranjos – Array (Vetores ou Matrizes)

Resultado:

```
⋮ Saída - projeto_revisao_ed (run)

run:
[Elemento A, Elemento C]
[Elemento A, Elemento B, Elemento C]
Tamanto do vetor: 3
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
|
```

Arranjos – Array (Vetores ou Matrizes)

Lembrando que ainda existem outros métodos que podemos utilizar no nosso dia-a-dia:

● add(String e)	boolean
● add(int index, String element)	void
● addAll(Collection<? extends String> c)	boolean
● addAll(int index, Collection<? extends String> c)	boolean
● clear()	void
● clone()	Object
● contains(Object o)	boolean
● containsAll(Collection<?> c)	boolean
● ensureCapacity(int minCapacity)	void
● equals(Object o)	boolean
● forEach(Consumer<? super String> action)	void
● get(int index)	String

● getClass()	Class<?>
● hashCode()	int
● indexOf(Object o)	int
● isEmpty()	boolean
● iterator()	Iterator<String>
● lastIndexOf(Object o)	int
● listIterator()	ListIterator<String>
● listIterator(int index)	ListIterator<String>
● notify()	void
● notifyAll()	void
● parallelStream()	Stream<String>
● remove(Object o)	boolean

.... Tem mais

Vamos pesquisar e descobrir para que serve cada um deles.

"A meta é se melhor
que ontem,
NÃO melhor que
ninguém!"



Autor desconhecido

Obrigado!

Se precisar ...

Prof. Claudio Benossi

claudio.benossi@sp.senac.br

