Arrays e ArrayLists

Programação Orientada a Objetos — QXD0007

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

1º semestre/2022

Leitura deste tópico

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Capítulo 7** (Arrays e ArrayLists) do livro Java Como Programar, Décima Edição.

Arrays

Array

- É uma estrutura de dados que armazena uma coleção de valores do mesmo tipo sequencialmente na memória.
- Em Java, arrays são objetos. Logo, eles são do tipo referência.
  - Os elementos de um array podem ser de tipo nativo ou do tipo referência.

Sintaxe da declaração de um array: <tipo> [ ] <nome da variável>;

Exemplo: `int[] arr;`

`arr` ainda não foi inicializado

Arrays — Instanciação

- Um array pode ser instanciado usando o operador `new`:

```
int[] arr = new int[100];
```

Arrays — Instanciação

- Um array pode ser instanciado usando o operador `new`:

```
int[] arr = new int[100];
```

- O valor que determina o tamanho do array deve ser um inteiro:

```
int n = 100;
int[] arr = new int[n];
```

Arrays — Instanciação

- Um array pode ser instanciado usando o operador `new`:

```
int[] arr = new int[100];
```

- O valor que determina o tamanho do array deve ser um inteiro:

```
int n = 100;
int[] arr = new int[n];
```

- Para acessar um elemento específico no array, usamos o nome da referência para o array e o índice da posição do elemento no array.
  - Em Java, os elementos de um array de tamanho n são numerados de 0 a $n - 1$, assim como em C/C++.

Arrays — Instanciação

- Um array pode ser instanciado usando o operador `new`:

```
int[] arr = new int[100];
```

- O valor que determina o tamanho do array deve ser um inteiro:

```
int n = 100;
int[] arr = new int[n];
```

- Para acessar um elemento específico no array, usamos o nome da referência para o array e o índice da posição do elemento no array.
  - Em Java, os elementos de um array de tamanho n são numerados de 0 a $n - 1$, assim como em C/C++.

- Todo array sabe o seu tamanho e o armazena na variável **length**, que é um **atributo** do objeto array. A expressão `arr.length` devolve o tamanho do array `arr`.

Arrays — Instanciação

- Podemos instanciar um array e inicializar seus elementos com uma lista inicializadora, que é uma lista de elementos separados por vírgula e fechada com colchetes.

  Exemplo: `int[] array = {10, 20, 30, 40, 50};`

- Uma lista inicializadora também pode ser usada desta forma:

  ```
  int[] array = new int[] {10, 20, 30, 40, 50};
  ```

Problema Exemplo

- Fazer um programa para ler um número inteiro N e a altura de N pessoas. Armazene as N alturas em um vetor. Em seguida, mostrar a altura média dessas pessoas.

- O cálculo da altura média deve ser feito por uma função chamada `media`.

Input:	**Output:**
3 1.72 1.56 1.80	AVERAGE HEIGHT = 1.69

O laço `for each`

- O Java possui uma estrutura de repetição (laço) que permite iterar sobre os elementos de um array sem usar uma variável contadora, o for each.

  Sintaxe:

```
for (<Tipo> variavel :  nomeDoArray) {
    ..comandos..
}
```

O laço `for each`

- O Java possui uma estrutura de repetição (laço) que permite iterar sobre os elementos de um array sem usar uma variável contadora, o for each.

Sintaxe:

```
for (<Tipo> variavel :  nomeDoArray) {
    ..comandos..
}
```

Exemplo:

```
int[] array = {5, 78, 23, 10};
for (int element :  array) {
    System.out.println(element);
}
```

O laço `for each`

- O Java possui uma estrutura de repetição (laço) que permite iterar sobre os elementos de um array sem usar uma variável contadora, o for each.

  Sintaxe:

```
for (<Tipo> variavel :  nomeDoArray) {
    ..comandos..
}
```

  Exemplo:

```
int[] array = {5, 78, 23, 10};
for (int element :  array) {
    System.out.println(element);
}
```

- O for each só pode ser usado para obter o valor dos elementos do array, ele não pode modificá-los. Se seu programa precisar iterar e modificar o valor dos elementos, use o for tradicional ou outro laço.

Exemplo: laço for each

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  public class Programa15 {
2      public static void main(String[] args) {
3          int[] array = {87, 78, 90, 45, 23, 17, 67};
4          int total = 0;
5
6          for(int number : array) {
7              total += number;
8          }
9
10         System.out.printf("Total: %d%n", total);
11     }
12 }
```

Arrays e métodos

Passando arrays para métodos

- Para que um método receba uma referência para um array através de uma chamada de método, a sua lista de parâmetros deve especificar um parâmetro do tipo array.
  - Sintaxe: <tipo> nome_do_metodo (<tipo>[ ] <identificador>)
  - Exemplo: void imprimeArray(int[ ] vetor)

Passando arrays para métodos

- Para que um método receba uma referência para um array através de uma chamada de método, a sua lista de parâmetros deve especificar um parâmetro do tipo array.
  - Sintaxe: <tipo> nome_do_metodo (<tipo>[ ] <identificador>)
  - Exemplo: void imprimeArray(int[ ] vetor)
- Para passar o array como argumento para o método, especifique apenas o nome do array, sem os colchetes:
  - Exemplo:
    int[ ] meuVetor = {1,2,3,4,5,6};
    imprimeArray(meuVetor);

Passando arrays para métodos

- Para que um método receba uma referência para um array através de uma chamada de método, a sua lista de parâmetros deve especificar um parâmetro do tipo array.
  - Sintaxe: <tipo> nome_do_metodo (<tipo>[ ] <identificador>)
  - Exemplo: void imprimeArray(int[ ] vetor)
- Para passar o array como argumento para o método, especifique apenas o nome do array, sem os colchetes:
  - Exemplo:
    int[ ] meuVetor = {1,2,3,4,5,6};
    imprimeArray(meuVetor);
- Todo objeto array sabe o seu tamanho, portanto não é preciso passar o tamanho do array como argumento. Além disso, o que é passado para o método é uma referência para o objeto array e não o seu valor.

Listas de argumentos com tamanhos variados

- Note que o método `printf` permite uma lista de argumentos de tamanho variável. Em Java, podemos escrever métodos com essa característica.

Listas de argumentos com tamanhos variados

- Note que o método `printf` permite uma lista de argumentos de tamanho variável. Em Java, podemos escrever métodos com essa característica.

- Em uma lista de parâmetros de um método, um tipo seguido por três pontos (...) indica que o método pode receber uma quantidade variável de argumentos daquele tipo.
  - Exemplo: `double media(double... numeros)`

Listas de argumentos com tamanhos variados

- Note que o método `printf` permite uma lista de argumentos de tamanho variável. Em Java, podemos escrever métodos com essa característica.

- Em uma lista de parâmetros de um método, um tipo seguido por três pontos (...) indica que o método pode receber uma quantidade variável de argumentos daquele tipo.
  - Exemplo: `double media(double... numeros)`

- Java trata a lista de argumentos de tamanho variável como se fosse um array, cujos elementos são todos do mesmo tipo.
  - No exemplo acima, o parâmetro `numeros` é tratado dentro do método como um array de doubles.

A classe `Arrays`

A classe `Arrays` do pacote `java.util`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Java possui a classe `Arrays` que fornece diversos métodos estáticos para manipulações comuns de arrays, como:
  - ordenação de array, busca binária, comparação de arrays, preenchimento de um vetor, etc.

- Importe a classe Arrays no início do seu programa:
  `import java.util.Arrays;`

- API do Java: `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Arrays.html`

Métodos estáticos da classe `Arrays`

Na tabela abaixo, o vetor `array` e a variável `x` são do mesmo tipo.

Métodos	**Descrição**
String `toString`(array)	Exibe os elementos do array como uma String.
void `sort`(array)	Ordena array em ordem crescente.
void `sort`(array, int from, int to)	Ordenar uma faixa do vetor em ordem crescente de `from` para `to-1` .
int `binarySearch`(array, x)	Busca a chave x em array. Retorna o índice da chave se ela for encontrada no array; ou um número negativo caso contrário.
boolean `equals`(array1, array2)	Recebe dois arrays. Retorna true caso sejam iguais e false caso contrário.
void `fill`(array, valor)	Preenche todo o array com determinado valor.
void `fill`(array, int from, int to, x)	Preenche o intervalo de `from` até `to-1` com o valor x.
`copyOf`(array, int total)	Retorna uma cópia do array. O argumento total é o número de elementos a serem copiados.
`copyOfRange`(array, int from, int to)	Retorna uma cópia do array de `from` até `to-1`

- **Obs.:** O array usado na função `binarySearch` deve estar ordenado.

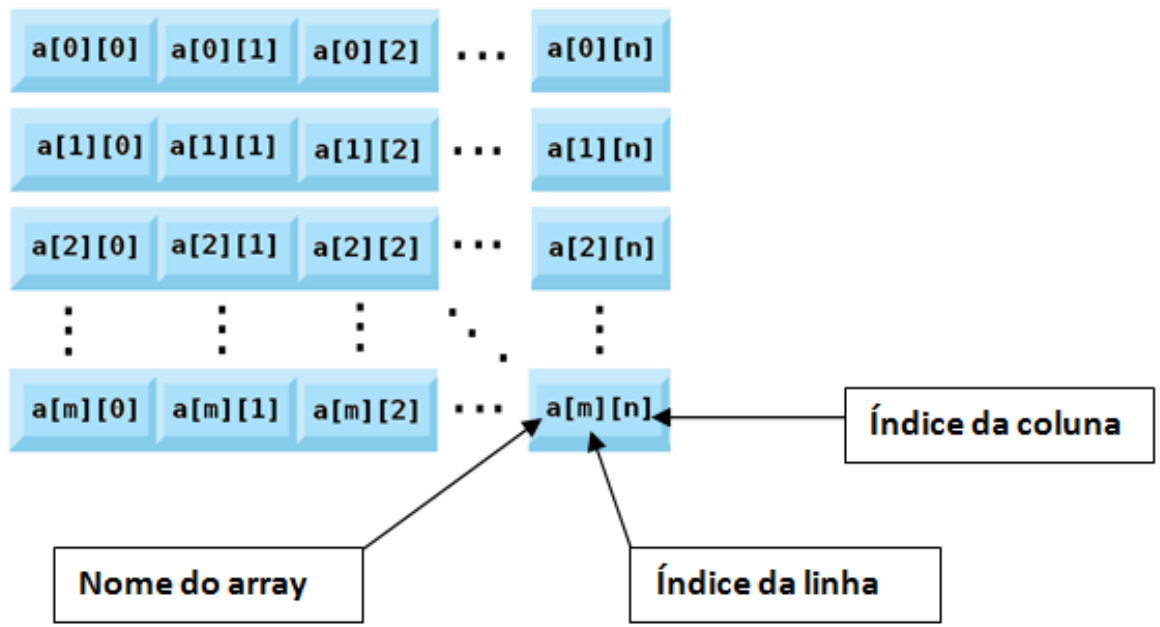
Exemplo

- Analisar o arquivo `ArrayManipulations.java`

Arrays bidimensionais (Matrizes)

Arrays Bidimensionais

- Esse tipo de array é declarado como tendo duas dimensões e é usado para representar tabelas de valores que consistem em informações organizadas em linhas e colunas.
- Exemplo de declaração: `int[][] matriz = new int[4][5];`



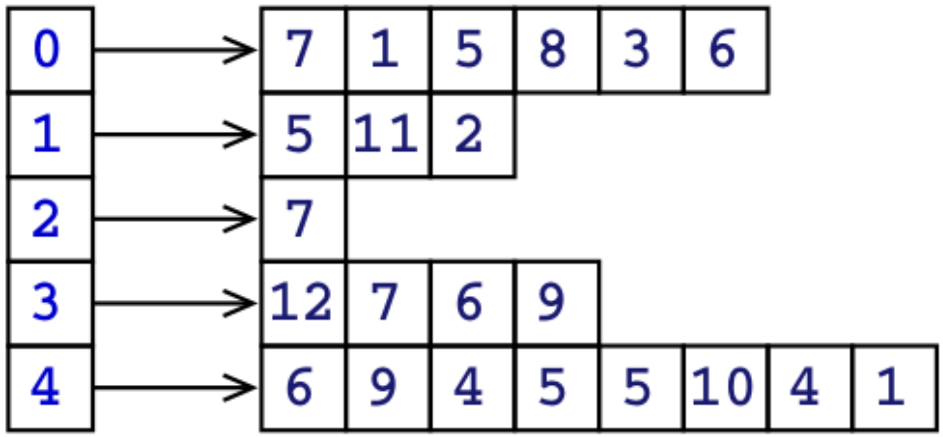
Problema Exemplo 3

- Fazer um programa para ler um número inteiro N e uma matriz de ordem N contendo números inteiros. Em seguida, mostrar a diagonal principal e a quantidade de valores negativos da matriz.

Input:	**Output:**
3 5 -3 10 15 8 2 7 9 -4	Main diagonal: 5 8 -4 Negative numbers = 2

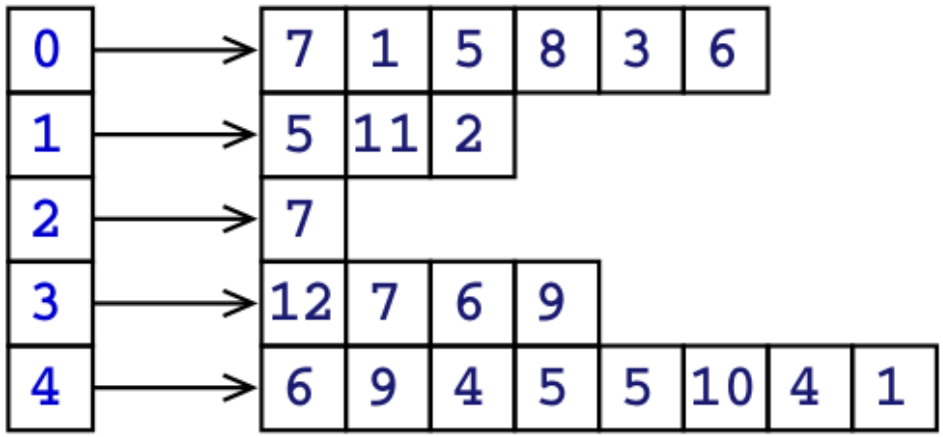
Matrizes irregulares

- Uma matriz é irregular se ela não têm o mesmo número de colunas para cada linha



Matrizes irregulares

- Uma matriz é irregular se ela não têm o mesmo número de colunas para cada linha




```
int[][] mat = new int[5][];
mat[0] = new int[6];
mat[1] = new int[3];
mat[2] = new int[1];
mat[3] = new int[4];
mat[4] = new int[8];
```

Classes Empacotadoras e seus atributos e métodos estáticos

Classes empacotadoras (*Wrapper classes*)

- Todos os oito tipos nativos do Java possuem uma classe correspondente denominada classe empacotadora que pertencem ao pacote `java.lang`.

Primitive type	Wrapper type
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

Classes empacotadoras (*Wrapper classes*)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Todos os oito tipos nativos do Java possuem uma classe correspondente denominada classe empacotadora que pertencem ao pacote `java.lang`.

Primitive type	Wrapper type
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- Classes empacotadoras possuem atributos e métodos estáticos que podem ser úteis em várias aplicações.

Classes empacotadoras são necessárias

- Todas as estruturas de dados do Collection Framework do Java e as classes do pacote java.util trabalham apenas com objetos e, portanto, as classes empacotadoras serão necessárias caso você queira armazenar tipos de dados nativos.

Criando objetos de classes empacotadoras

Classes empacotadoras possuem o método estático `valueOf` que recebe como argumento um valor de tipo nativo e retorna um objeto da sua respectiva classe empacotadora.

Criando objetos de classes empacotadoras

Classes empacotadoras possuem o método estático `valueOf` que recebe como argumento um valor de tipo nativo e retorna um objeto da sua respectiva classe empacotadora.

```
1  import java.util.Scanner;
2
3  public class CreateWrapper {
4      public static void main(String[] args) {
5          Integer myInt = Integer.valueOf(13);
6
7          System.out.println(myInt); // invoca toString()
8
9          int soma = myInt.intValue() + 3;
10
11         System.out.println(soma); // imprime 16
12     }
13 }
```

Wrappers são imutáveis

- Objetos de classe empacotadora guardam o valor de tipo nativo em um atributo declarado como `private final`.

- Uma vez atribuído um valor a esse atributo, não será possível modificá-lo.

Boxing e unboxing em classes empacotadoras

Boxing (ou empacotamento)

- **Autoboxing:** é o processo de conversão automática de um tipo de dado nativo em um objeto da sua classe empacotadora correspondente.

Boxing (ou empacotamento)

- **Autoboxing:** é o processo de conversão automática de um tipo de dado nativo em um objeto da sua classe empacotadora correspondente.

- Desde o Java 5, não precisamos mais usar o método estático `valueOf` das classes empacotadoras para converter os tipos primitivos em objetos.

- Desde o Java 9, os construtores das classes empacotadoras **estão descontinuados**, o que significa que não devemos usar construtores para criar objetos dessas classes.
  - O mecanismo de autoboxing faz isso automaticamente ou, se você preferir, pode usar o método estático `valueOf`.

Criando objetos de classes empacotadoras

usando o mecanismo de boxing

```
1  public class CreateWrapperBoxing {
2      public static void main(String[] args) {
3
4          Double myDouble = 5.99; // criando um novo Double
5
6          System.out.println(myDouble); // chama toString()
7
8          Double ref = 5.99;
9
10         if(ref == myDouble)
11             System.out.println("são o mesmo objeto");
12         else
13             System.out.println("objetos distintos");
14
15         if(ref.equals(myDouble))
16             System.out.println("possuem mesmo valor");
17         else
18             System.out.println("possuem valores distintos");
19     }
20 }
```

Unboxing (ou desempacotamento)

- **Unboxing:** é o processo de conversão automática de um objeto de classe empacotadora em seu respectivo tipo nativo.

- O compilador do Java aplica *unboxing* quando um objeto de uma classe empacotadora é:
  - Passado como parâmetro para um método que espera um valor do tipo nativo correspondente.
  - Atribuído a uma variável do tipo nativo correspondente.

Unboxing — Exemplo


```
1  public class WrapperUnboxing {
2      public static void main(String[] args) {
3          Double myDouble = -4567.0; // boxing
4
5          double x = myDouble; // unboxing
6
7          System.out.println("Absolute value of " + myDouble
8              + " = " + Math.abs(x));
9      }
10 }
```

Classe Integer – Métodos não-estáticos

Métodos	**Descrição**
int compareTo(Integer a)	Compara dois valores Integer numericamente. Retorna 0 se x == y, menor que 0 se x <y; e maior que 0 se x >y
int intValue()	Retorna o valor deste Integer como um int
long longValue()	Retorna o valor deste Integer como um long
byte byteValue()	Retorna o valor deste Integer como um byte
short shortValue()	Retorna o valor deste Integer como um short
double doubleValue()	Retorna o valor deste Integer como um double
String toString()	Retorna uma String representando este Integer
boolean equals(Object obj)	Compara este Integer com o objeto obj
int hashCode()	Retorna um hash code para este Integer

- Consulte a API do java para as demais classes Wrappers.
- Analisar o arquivo WrapperExemplo2.java

Classe Integer

Atributos Estáticos

- `int MIN_VALUE`: constante que dá o menor valor que um int pode ter
- `int MAX_VALUE`: constante que dá o maior valor que um int pode ter
- `int BYTES`: o número de bytes usados para representar um int
- `int SIZE`: o número de bits usados para representar um int

Classe Integer

Atributos Estáticos

- `int MIN_VALUE`: constante que dá o menor valor que um int pode ter
- `int MAX_VALUE`: constante que dá o maior valor que um int pode ter
- `int BYTES`: o número de bytes usados para representar um int
- `int SIZE`: o número de bits usados para representar um int

```
1 class IntegerExemplo {
2     public static void main(String[] args) {
3         System.out.println("MIN_VALUE: " + Integer.MIN_VALUE);
4         System.out.println("MAX_VALUE: " + Integer.MAX_VALUE);
5         System.out.println("BYTES: " + Integer.BYTES);
6         System.out.println("SIZE: " + Integer.SIZE);
7     }
8 }
```

Classe Integer

Alguns Métodos Públicos e Estáticos

- `int max(int a, int b)`: retorna o valor do maior parâmetro
- `int min(int a, int b)`: retorna o valor do menor parâmetro
- `int parseInt(String s)`: converte a String `s` para int
- `int parseInt(String s, int radix)`: converte o valor da String `s` na base `radix` para decimal e retorna o valor
- `String toString(int i)`: converte o int i para String

Classe Integer

Alguns Métodos Públicos e Estáticos

- `int max(int a, int b)`: retorna o valor do maior parâmetro
- `int min(int a, int b)`: retorna o valor do menor parâmetro
- `int parseInt(String s)`: converte a String `s` para int
- `int parseInt(String s, int radix)`: converte o valor da String `s` na base `radix` para decimal e retorna o valor
- `String toString(int i)`: converte o int i para String

Como exemplo, veja a classe `IntegerExemplo2.java`

Classe Integer

Alguns Métodos Públicos e Estáticos

- `int max(int a, int b)`: retorna o valor do maior parâmetro
- `int min(int a, int b)`: retorna o valor do menor parâmetro
- `int parseInt(String s)`: converte a String `s` para int
- `int parseInt(String s, int radix)`: converte o valor da String `s` na base `radix` para decimal e retorna o valor
- `String toString(int i)`: converte o int i para String

Como exemplo, veja a classe `IntegerExemplo2.java`

Para mais detalhes sobre a classe `Integer`, consulte a API do Java: `https://cr.openjdk.java.net/~iris/se/17/latestSpec/api/java.base/java/lang/Integer.html`

Demais classes empacotadoras

- Todas as demais classes empacotadoras possuem os atributos estáticos `MIN_VALUE`, `MAX_VALUE`, `BYTES` e `SIZE`, e também o método estático `toString`.

- Com exceção da classe `Character`, todas as demais também têm os métodos estáticos `max` e `min`.

- Do mesmo modo que a classe `Integer` tem o método estático `parseInt`, as demais classes empacotadoras (com exceção da classe `Character`) têm o seu método "parse" equivalente, obtido trocando-se Int pelo nome do tipo nativo correspondente.
  - Veja o programa `WrappersExemplo.java`

Exercício

A linguagem Java dispõe de um suporte nativo a vetores, que exige a definição de seu tamanho no momento da instanciação. Depois de instanciado, o tamanho do vetor não pode ser modificado. Escreva uma classe chamada **Vetor** cujos objetos simulem vetores de tamanho variável. A classe define os seguintes métodos:

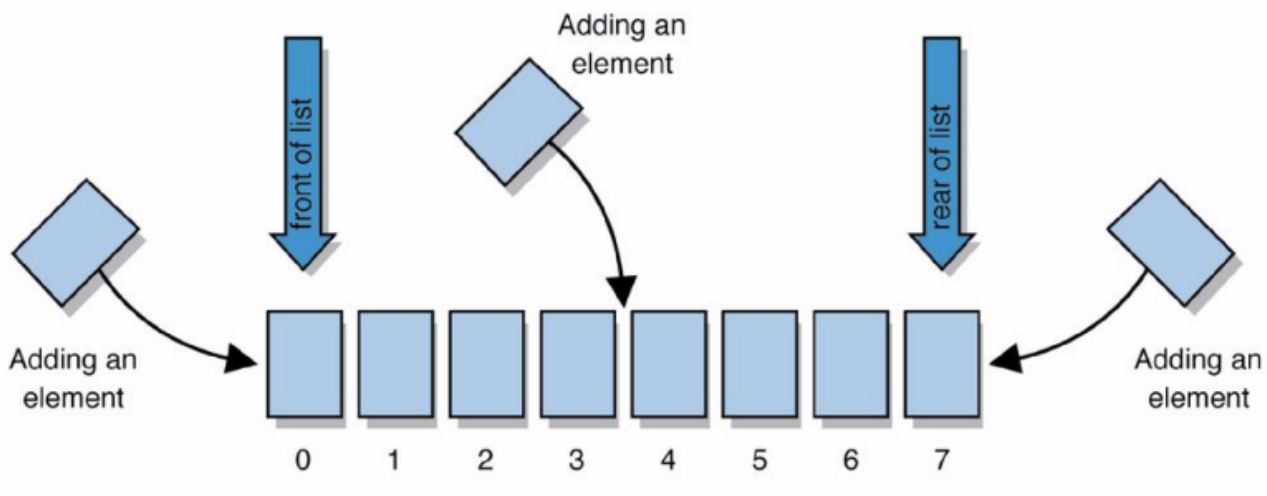
- **construtor(int n)**: recebe como parâmetro o tamanho inicial do vetor.
- **add(Integer i)**: recebe como parâmetro um Integer e o coloca na próxima posição disponível do vetor; note que o vetor cresce automaticamente, portanto, se a inserção ultrapassar o tamanho inicial estabelecido na criação, por exemplo, o vetor deve aumentar seu tamanho automaticamente.
- **get(int i)**: recebe como parâmetro uma posição do vetor e retorna o Integer que estiver naquela posição; se a posição não estiver ocupada ou ultrapassar o tamanho do vetor, este método retorna nulo.
- **size()**: retorna o número de elementos atualmente no vetor.
- **toString()**: retorna o Vetor como uma String.
- **equals(Vetor v))**: retorna true se este vetor é igual ao vetor v passado como parâmetro; retorna false caso contrário.

ArrayLists

ArrayLists

Introdução ao Collection Framework

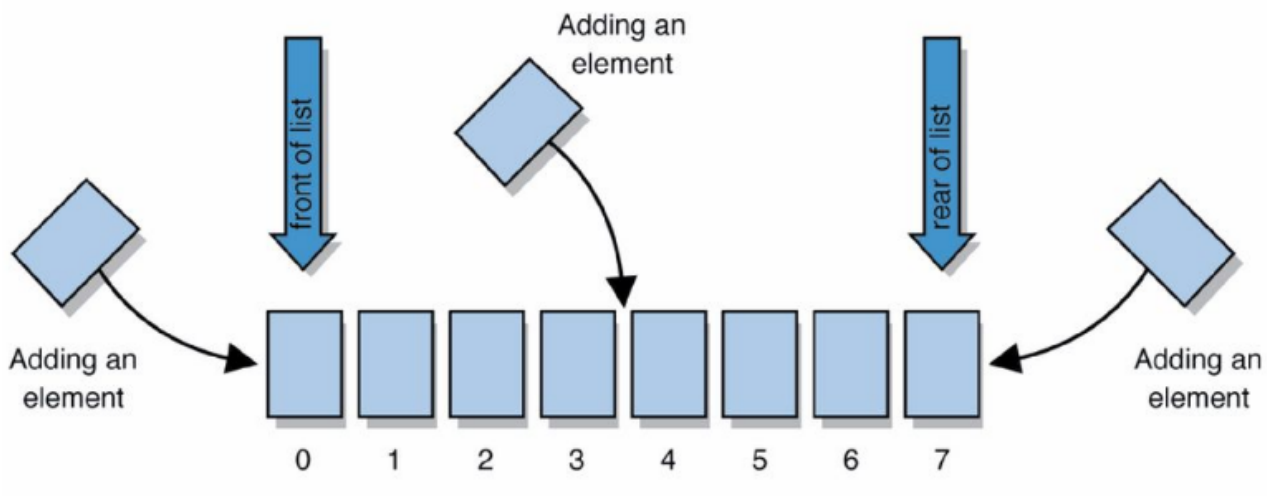
- Um ArrayList encapsula a estrutura de dados lista, que é uma estrutura de dados que pode diminuir e aumentar de tamanho automaticamente à medida que os elementos são adicionados ou removidos.



ArrayLists

Introdução ao Collection Framework

- Um ArrayList encapsula a estrutura de dados lista, que é uma estrutura de dados que pode diminuir e aumentar de tamanho automaticamente à medida que os elementos são adicionados ou removidos.



Internamente, um ArrayList é implementado usando um array.
A classe `ArrayList` pertence ao pacote `java.util`

ArrayLists

- Um ArrayList não pode armazenar tipos nativos, ele só pode armazenar objetos do tipo referência.

- Para armazenar dados de tipos nativos em um ArrayList, precisaremos usar as suas respectivas classes empacotadoras:
  - Byte
  - Short
  - Integer
  - Long
  - Float
  - Double
  - Character
  - Boolean

Criando um ArrayList

O tipo do objeto deve ser especificado usando o operador diamante <>:

Sintaxe: Arraylist<tipo> nome_da_variavel = new Arraylist<tipo>();

Criando um ArrayList

O tipo do objeto deve ser especificado usando o operador diamante <>:

Sintaxe: Arraylist<tipo> nome_da_variavel = new Arraylist<tipo>();

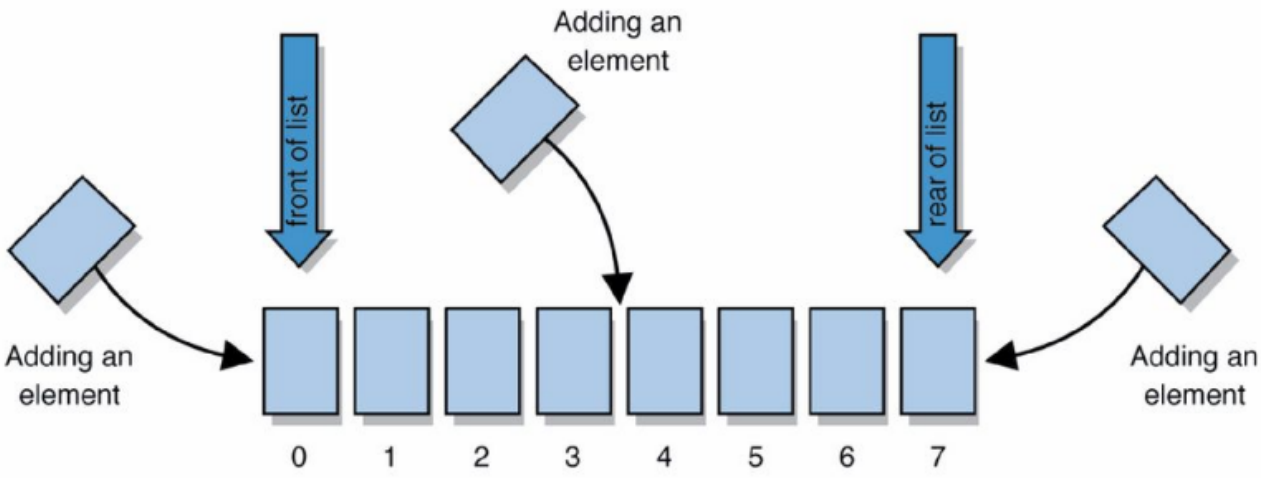
Exemplos:

- ArrayList<String> str1 = new arrayList<String>();
- ArrayList<String> str2 = new arrayList<String>(10);
- ArrayList<Integer> age = new ArrayList<Integer>(2);

Note que a classe empacotadora `Integer` foi usada no lugar do tipo primitivo `int`

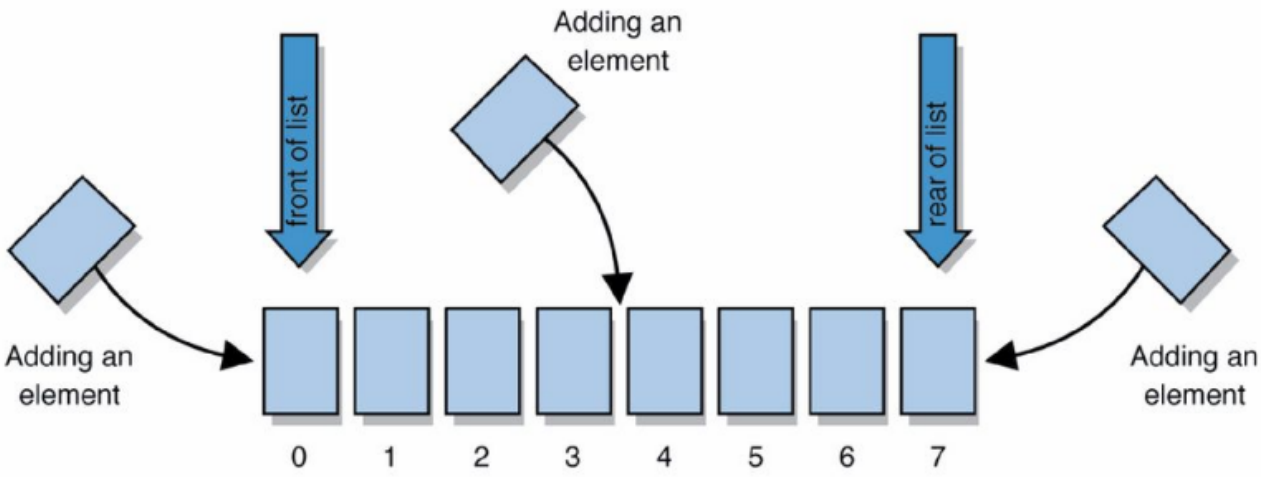
Adicionando elementos a um ArrayList

Elementos podem ser adicionados no início, no fim ou em qualquer outra posição.



Adicionando elementos a um ArrayList

Elementos podem ser adicionados no início, no fim ou em qualquer outra posição.

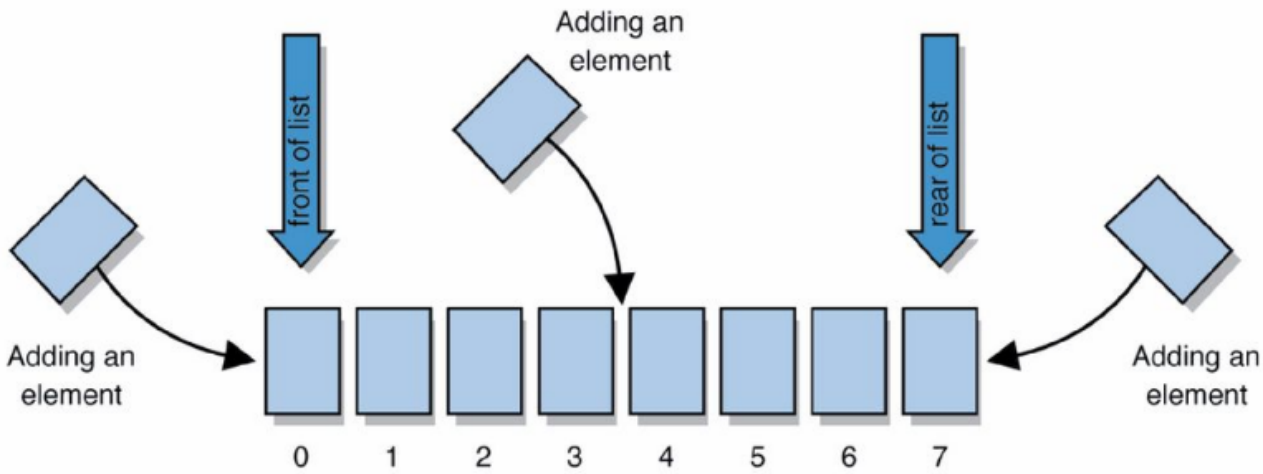


- `void add(NomeDaClasse elemento)`
  Adiciona elemento ao final do ArrayList

Adicionando elementos a um ArrayList

Universidade Federal do Ceará
Campus Quixadá

Elementos podem ser adicionados no início, no fim ou em qualquer outra posição.



- `void add(NomeDaClasse elemento)`
  Adiciona elemento ao final do ArrayList

  Exemplo:

```
ArrayList<Integer> myList = new ArrayList<Integer>();
myList.add(5);
myList.add(7);
```

Adicionando elementos a um ArrayList

- `void add(int index, NomeDaClasse elemento)`
  - Adiciona elemento na posição index do ArrayList.
  - O index deve ser maior que ou igual a zero e menor que ou igual ao número de elementos no ArrayList.

Adicionando elementos a um ArrayList

- `void add(int index, NomeDaClasse elemento)`
  - Adiciona elemento na posição index do ArrayList.
  - O index deve ser maior que ou igual a zero e menor que ou igual ao número de elementos no ArrayList.

  Exemplo:

```
ArrayList<Integer> myList = new ArrayList<Integer>();
myList.add(5);
myList.add(7);
myList.add(8);
myList.add(6, 1);
```

Removendo elementos a um ArrayList

- `boolean remove(NomeDaClasse elemento)`
  - Remove a primeira ocorrência de elemento do Arraylist.
    Retorna true se o elemento foi removido; false caso contrário.

- `void remove(int index)`
  - Remove o elemento na posição index do ArrayList

- `void clear()`
  - Remove todos os elementos

Acessando elementos a um ArrayList

- `void set(int index, NomeDaClasse elemento)`
  - Seta novo valor para o elemento na posição index do Arraylist.
  - O valor que estava na posição index é sobrescrito.

Acessando elementos a um ArrayList

- `void set(int index, NomeDaClasse elemento)`
  - Seta novo valor para o elemento na posição index do Arraylist.
  - O valor que estava na posição index é sobrescrito.

- `NomeDaClasse get(int index)`
  - Retorna o elemento na posição index do ArrayList.

Acessando elementos a um ArrayList

- `void set(int index, NomeDaClasse elemento)`
  - Seta novo valor para o elemento na posição index do Arraylist.
  - O valor que estava na posição index é sobrescrito.

- `NomeDaClasse get(int index)`
  - Retorna o elemento na posição index do ArrayList.

- `boolean equals(ArrayList<NomeDaClasse> lista)`
  - Determina se as duas listas contém os mesmos elementos, na mesma ordem.

Funcionalidades adicionais de um ArrayList

- `boolean isEmpty()`
  - Retorna true se o ArrayList não tem elementos.

Funcionalidades adicionais de um ArrayList

- `boolean isEmpty()`
  - Retorna true se o ArrayList não tem elementos.

- `int size()`
  - Retorna o número de elementos atualmente no ArrayList.

Funcionalidades adicionais de um ArrayList

- `boolean isEmpty()`
  - Retorna true se o ArrayList não tem elementos.
- `int size()`
  - Retorna o número de elementos atualmente no ArrayList.
- `T[] toArray(T[] arr)`
  - Retorna um array contendo todos os elementos do ArrayList na ordem correta.

Funcionalidades adicionais de um ArrayList

- `boolean isEmpty()`
  - Retorna true se o ArrayList não tem elementos.
- `int size()`
  - Retorna o número de elementos atualmente no ArrayList.
- `T[] toArray(T[] arr)`
  - Retorna um array contendo todos os elementos do ArrayList na ordem correta.
- `boolean contains(NomeDaClasse elemento)`
  - Testa se elemento é um componente deste ArrayList.

Funcionalidades adicionais de um ArrayList

- `boolean isEmpty()`
  - Retorna true se o ArrayList não tem elementos.
- `int size()`
  - Retorna o número de elementos atualmente no ArrayList.
- `T[] toArray(T[] arr)`
  - Retorna um array contendo todos os elementos do ArrayList na ordem correta.
- `boolean contains(NomeDaClasse elemento)`
  - Testa se elemento é um componente deste ArrayList.
- `String toString()`
  - Retorna uma representação em String do ArrayList.

Funcionalidades adicionais de um ArrayList

- `int indexOf(NomeDaClasse element)`
  - Retorna o índice da primeira ocorrência de element no arrayList. Retorna -1 se o elemento não for encontrado.

Funcionalidades adicionais de um ArrayList

- `int indexOf(NomeDaClasse element)`
  - Retorna o índice da primeira ocorrência de element no arrayList. Retorna -1 se o elemento não for encontrado.

- `int lastIndexOf(NomeDaClasse element)`
  - Retorna o índice da última ocorrência de element no arrayList. Retorna -1 se o elemento não for encontrado.

Funcionalidades adicionais de um ArrayList

- `void trimToSize()`
  - Reduz a capacidade do ArrayList para o tamanho atual da lista.

Funcionalidades adicionais de um ArrayList

- `void trimToSize()`
  - Reduz a capacidade do ArrayList para o tamanho atual da lista.

- `void ensureCapacity(int minCapacity)`
  - Aumenta a capacidade do ArrayList garantindo que ele possa conter pelo menos o número de elementos especificado pelo argumento `minCapacity`.

Funcionalidades adicionais de um ArrayList

- `void trimToSize()`
  - Reduz a capacidade do ArrayList para o tamanho atual da lista.

- `void ensureCapacity(int minCapacity)`
  - Aumenta a capacidade do ArrayList garantindo que ele possa conter pelo menos o número de elementos especificado pelo argumento `minCapacity`.

- Para mais detalhes, consulte a API do Java: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html

Exemplo

- Analisar a classe DemoArrayList.java.

Exercício

- Escreva um programa que lê um parágrafo e mostra as palavras individuais como uma lista.
  - Primeiro mostre todas as palavras.
  - Então, mostre todas as palavras em ordem reversa.
  - Então, mostre-as de modo que todas as palavras em plural estejam com letra maiúscula.
  - Por fim, mostre-as com todas as palavras em plural removidas.

Exercício

- Escreva um programa que lê um parágrafo e mostra as palavras individuais como uma lista.
  - Primeiro mostre todas as palavras.
  - Então, mostre todas as palavras em ordem reversa.
  - Então, mostre-as de modo que todas as palavras em plural estejam com letra maiúscula.
  - Por fim, mostre-as com todas as palavras em plural removidas.

Solução: Ver o arquivo Paragrafo.java

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM