Franklin Parrini
Educação e Tecnologia

LINGUAGEM DE PROGRAMAÇÃO · JAVA

ArrayList em Java

Listas dinâmicas na prática: criar, percorrer, alterar, remover, pesquisar e ordenar

java.util.ArrayList | 4 horas



Objetivos



Entender

O que é um ArrayList e quando usá-lo no lugar de um array.



Manipular

Adicionar, acessar, alterar, remover, pesquisar e ordenar elementos.



Aplicar

Construir um programa com menu usando ArrayList e Scanner.

O problema dos arrays

E se eu não sei quantos itens vou guardar?

- Tamanho fixo: é definido na criação e nunca muda.
- Não existe adicionar nem remover de verdade.
- Para crescer, é preciso criar outro array e copiar tudo.
- Posições sobrando precisam ser controladas à mão.



CONCEITO

O que é um ArrayList?

Uma lista dinâmica da biblioteca padrão do Java (pacote java.util). Começa vazia, cresce quando você adiciona e encolhe quando você remove. Por dentro, usa um array e cuida sozinha de redimensioná-lo.



Guarda objetos: String, Integer, Double... e também classes criadas por você.

COMPARAÇÃO

Array × ArrayList

Característica	Array	ArrayList
Tamanho	Fixo	Dinâmico
Elementos	Primitivos e objetos	Somente objetos (wrappers)
Tamanho atual	`v.length`	`lista.size()`
Ler	`v[i]`	`lista.get(i)`
Alterar	`v[i] = x;`	`lista.set(i, x);`
Adicionar / remover	Não existe	`add()` / `remove()`
Importação	Nenhuma	`import java.util.ArrayList;`

Vantagens e desvantagens

Vantagens

- Tamanho dinâmico
- Métodos prontos: add, remove, contains, sort...
- Código mais curto e legível

Desvantagens

- Só guarda objetos: primitivos pedem wrappers
- Um pouco mais de memória e mais lento que o array
- Inserir/remover no meio desloca elementos

DECISÃO

Quando utilizar cada um

Use um array quando...

- a quantidade é conhecida e fixa (12 meses, 4 notas bimestrais)
- precisa de máximo desempenho com primitivos ou de matrizes

Use um ArrayList quando...

- a quantidade é desconhecida ou muda (carrinho, lista de alunos)
- vai adicionar, remover ou pesquisar com frequência

Pergunta-chave: eu sei exatamente quantos elementos vou guardar?

Importação, declaração e instanciação

```
import java.util.ArrayList;

ArrayList<String>  nomes  = new ArrayList<>();
ArrayList<Integer> notas  = new ArrayList<>();
ArrayList<Double>  precos = new ArrayList<>();
```

- Entre < > fica o tipo dos elementos (Generics: o T de Type).
- O <> do new deixa o Java deduzir o tipo.
- A lista nasce vazia: size() é 0 e println mostra [].

Por que não existe ArrayList<int>?

Primitivo	Wrapper
`int`	`Integer`
`double`	`Double`
`boolean`	`Boolean`
`char`	`Character`
`long`	`Long`

Generics só aceitam classes.

```
ArrayList<int>      ✗
ArrayList<Integer>  ✓
```

O Java converte sozinho entre int e Integer: autoboxing (int → Integer) e unboxing (Integer → int).

```
lista.add(5);   int x = lista.get(0);   // sem conversões manuais
```

ADICIONANDO

add(): no final e em uma posição

```
ArrayList<String> frutas = new ArrayList<>();
frutas.add("Maçã");
frutas.add("Banana");
frutas.add("Uva");
System.out.println(frutas);

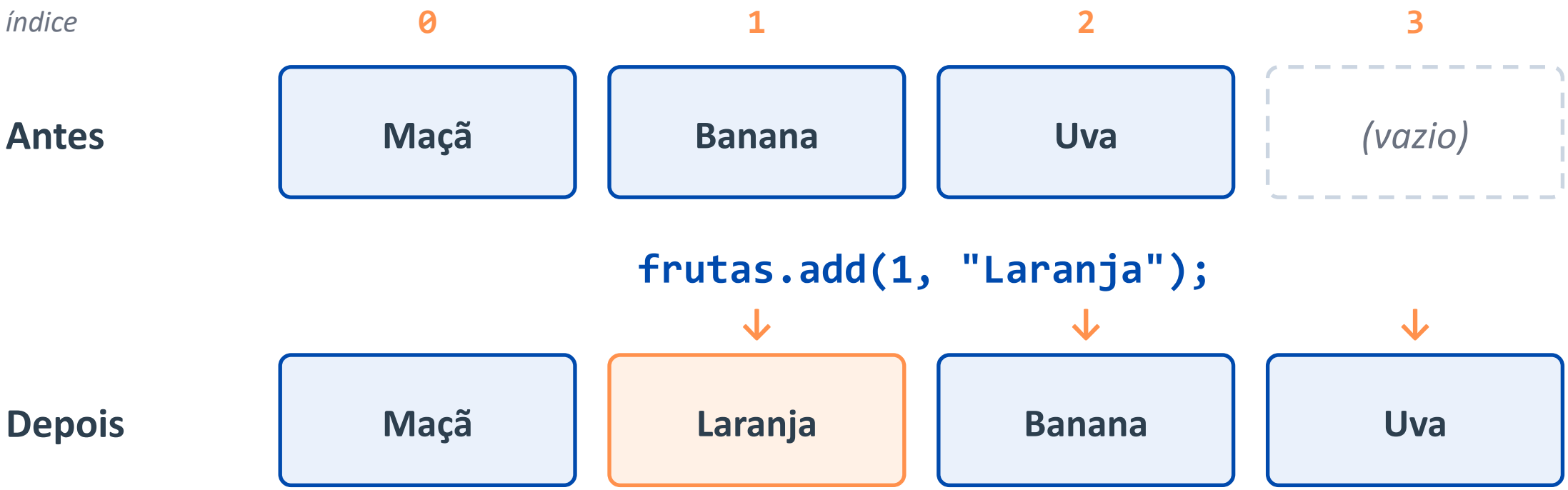
frutas.add(1, "Laranja");
System.out.println(frutas);
```

SAÍDA

```
[Maçã, Banana, Uva]
[Maçã, Laranja,
Banana, Uva]
```

add(x) vai para o final. add(índice, x) insere na posição e empurra os demais para a direita.

O que acontece com os índices



Banana e Uva foram deslocadas uma posição para a direita (ganharam +1 no índice).

ACESSANDO

get() e size()

```
ArrayList<String> frutas = new ArrayList<>();
frutas.add("Maçã");
frutas.add("Banana");
frutas.add("Uva");

System.out.println(frutas.get(0));
System.out.println("Total: " + frutas.size());

for (int i = 0; i < frutas.size(); i++) {
    System.out.println(i + " -> " + frutas.get(i));
}
```

SAÍDA

```
Maçã
Total: 3
0 -> Maçã
1 -> Banana
2 -> Uva
```

O primeiro elemento é get(0); o último é get(lista.size() - 1).

ATENÇÃO

size() ou length?

Tipo	Tamanho	Observação
Array	`v.length`	Atributo: sem parênteses
String	`texto.length()`	Método: com parênteses
ArrayList	`lista.size()`	Método: nunca use .length

Dica: se o tipo é ArrayList, o tamanho é sempre size().

ERRO COMUM

IndexOutOfBoundsException

```
ArrayList<String> frutas = new ArrayList<>();
frutas.add("Maçã");
frutas.add("Banana");
frutas.add("Uva");

System.out.println(frutas.get(3));  // índices: 0, 1, 2
```

```
Exception: IndexOutOfBoundsException: Index 3 out of bounds for length 3
```

Com size() elementos, os índices válidos vão de 0 até size() - 1.

for tradicional e for-each

```
// for tradicional: tem o índice
for (int i = 0; i < cores.size(); i++) {
    System.out.println(i + ": " + cores.get(i));
}

// for-each: só o valor
for (String cor : cores) {
    System.out.println(cor);
}
```

for → quando precisa da posição (numerar, alterar, remover, percorrer de trás para frente)

for-each → quando só quer ler cada valor: é o mais simples e legível

while e um cuidado importante

```
int pos = 0;
while (pos < cores.size()) {
    System.out.println(cores.get(pos));
    pos++;
}
```

O while é útil quando a repetição depende de outra condição, além do fim da lista.



Cuidado

Não adicione nem remova elementos dentro de um for-each: o Java lança ConcurrentModificationException.

Para remover durante a repetição, use o for tradicional, de trás para frente.

ALTERANDO

set(): substituindo valores

```
ArrayList<String> nomes = new ArrayList<>();
nomes.add("Ana");
nomes.add("Bruno");
nomes.add("Carla");

String antigo = nomes.set(1, "Beatriz");
System.out.println("Substituído: " + antigo);
System.out.println(nomes);
```

SAÍDA

```
Substituído: Bruno
[Ana, Beatriz, Carla]
```

set(índice, novoValor) troca o elemento e devolve o valor antigo. O tamanho da lista não muda.

COMPARAÇÃO

add(índice, x) × set(índice, x)

Lista inicial: [A, B, C]

Comando	O que faz	Resultado	Tamanho
`add(1, "X")`	Insere e desloca	`[A, X, B, C]`	4
`set(1, "X")`	Substitui	`[A, X, C]`	3

Inserir aumenta a lista. Substituir mantém o tamanho.

REMOVENDO

remove(): por índice e por valor

```
ArrayList<String> nomes = new ArrayList<>();
nomes.add("Ana");
nomes.add("Bruno");
nomes.add("Carla");
nomes.add("Diego");

nomes.remove(0);  // por índice
nomes.remove("Carla");  // por valor
System.out.println(nomes);
```

SAÍDA

```
[Bruno, Diego]
```

remove(int) devolve o elemento removido; remove(Object) devolve true/false.

Cuidado com Integer e remove()

```
ArrayList<Integer> numeros = new ArrayList<>();
numeros.add(10);
numeros.add(20);
numeros.add(30);

numeros.remove(1);  // índice 1 (o 20)
System.out.println(numeros);

numeros.remove(Integer.valueOf(10));  // valor 10
System.out.println(numeros);
```

SAÍDA

```
[10, 30]
[30]
```

remove(1) usa o número como ÍNDICE. Para remover o VALOR, use Integer.valueOf().

REGRA PRÁTICA

`remove(2)` **remove a POSIÇÃO 2**

`remove(Integer.valueOf(2))` **remove o VALOR 2**

Em ArrayList<Integer>, um número simples dentro do remove() é tratado como índice.

REMOVENDO

isEmpty() e clear()

```
ArrayList<String> tarefas = new ArrayList<>();
System.out.println(tarefas.isEmpty());

tarefas.add("Estudar");
tarefas.add("Treinar");
System.out.println(tarefas.isEmpty());
System.out.println(tarefas);

tarefas.clear();
System.out.println(tarefas.size());
```

SAÍDA

```
true
false
[Estudar, Treinar]
0
```

isEmpty() é o mesmo que size() == 0. clear() remove todos os elementos de uma vez.

contains(), indexOf() e lastIndexOf()

```
// [Niteroi, Rio, Petropolis, Rio]
cidades.contains("Rio")
cidades.contains("Sao Paulo")
cidades.indexOf("Rio")
cidades.lastIndexOf("Rio")
cidades.indexOf("Sao Paulo")
```

RETORNO

```
true
false
1
3
-1
```

-1 significa "não encontrado". A busca diferencia maiúsculas de minúsculas: "rio" ≠ "Rio".

Collections.sort() e Collections.reverse()

```
ArrayList<Integer> numeros = new ArrayList<>();
numeros.add(7);
numeros.add(2);
numeros.add(9);
numeros.add(4);

Collections.sort(numeros);
System.out.println(numeros);

Collections.reverse(numeros);
System.out.println(numeros);
```

SAÍDA

```
[2, 4, 7, 9]
[9, 7, 4, 2]
```

Precisa de: import java.util.Collections; Os dois métodos alteram a própria lista.

ATENÇÃO

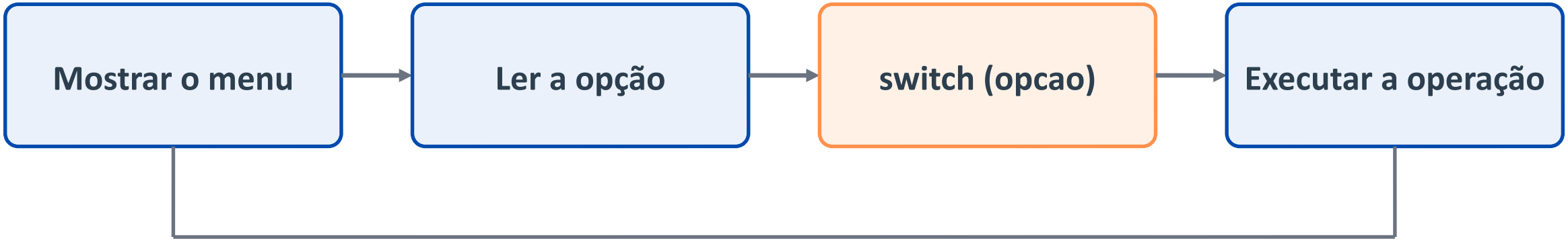
Ordenar ≠ Inverter

`sort()` coloca em ordem crescente / alfabética

`reverse()` só vira a ordem atual de ponta-cabeça

Para ordem decrescente: Collections.sort(lista) e depois Collections.reverse(lista).

ArrayList + Scanner: o programa com menu




```
do {
    // menu + leitura + switch
} while (opcao != 0);
```

Cada opção usa um método conhecido

Opção	Operação	Métodos
1	Inserir	`add()`
2	Listar	`isEmpty(), size(), get() e um for`
3	Alterar	`validar a posição com size() e usar set()`
4	Excluir	`remove(valor): devolve true/false`
5	Pesquisar	`indexOf()`
0	Sair	`encerra o do-while`

Esqueleto do menu (versão resumida)

```
do {
    // mostra o menu
    opcao = Integer.parseInt(sc.nextLine());
    switch (opcao) {
        case 1: nomes.add(sc.nextLine()); break;
        case 2: /* listar: for + get() */ break;
        case 3: /* alterar: set() */      break;
        case 4: nomes.remove(sc.nextLine()); break;
        case 5: /* pesquisar: indexOf() */ break;
    }
} while (opcao != 0);
```

O programa completo, com todas as validações, está na apostila (seção 11).

ARMADILHA CLÁSSICA

nextInt() seguido de nextLine()

O nextInt() lê só o número e deixa o ENTER no buffer. O nextLine() seguinte lê esse ENTER e devolve uma linha vazia, como se pulasse a digitação.

Solução: leia tudo com nextLine() e converta o número:

```
int opcao = Integer.parseInt(sc.nextLine());
```

PARA IR ALÉM

ArrayList de objetos

```
ArrayList<Aluno> turma = new ArrayList<>();
turma.add(new Aluno("Ana", 8.5));
turma.add(new Aluno("Bruno", 6.0));

for (Aluno a : turma) {
    System.out.println(a.nome + " tirou " + a.nota);
}
System.out.println(turma);
```

SAÍDA

```
Ana tirou 8.5
Bruno tirou 6.0
[Ana (8.5), Bruno
(6.0)]
```

Aluno é uma classe com nome, nota e toString(). Os métodos são os mesmos: add, get, remove, for-each...

REVISÃO

Os métodos que você precisa saber

Método	O que faz
`add(x)  /  add(i, x)`	Adiciona no final / insere na posição i
`get(i)`	Lê o elemento da posição i
`set(i, x)`	Substitui o elemento da posição i
`remove(i)  /  remove(x)`	Remove por posição / por valor
`size()  /  isEmpty()  /  clear()`	Quantidade / está vazia? / esvazia
`contains(x)`	true se o valor existe
`indexOf(x)  /  lastIndexOf(x)`	Primeira / última posição (ou -1)
`Collections.sort(lista)`	Ordena de forma crescente
`Collections.reverse(lista)`	Inverte a ordem atual

Resumo para levar da aula

- ArrayList é uma lista dinâmica: import java.util.ArrayList;
- ArrayList<Tipo>: use wrappers (Integer, Double...) no lugar de primitivos.
- Índices de 0 a size() - 1; tamanho com size(), nunca length.
- add, get, set, remove, contains, indexOf: o kit básico de operações.
- Collections.sort() ordena; Collections.reverse() só inverte.



Franklin Parrini
Educação e Tecnologia

Obrigado!

Agora é hora de praticar: exercícios da apostila e desafio da lista de compras