ArrayList em Java

Listas dinâmicas: criar, adicionar, percorrer, alterar, remover, pesquisar e ordenar

Estruturas de dados básicas | Material de estudo e prática

Nome do aluno: _____

Turma: _____

Data: _____

1. Objetivos desta apostila

Ao final desta apostila, você deverá ser capaz de:

- Explicar o que é um ArrayList e por que ele resolve as limitações dos arrays.
- Criar um ArrayList usando Generics e entender por que tipos primitivos precisam de classes wrapper.
- Adicionar, acessar, alterar e remover elementos com add(), get(), set() e remove().
- Percorrer uma lista com for, for-each e while.
- Pesquisar elementos com contains(), indexOf() e lastIndexOf().
- Ordenar e inverter listas com Collections.sort() e Collections.reverse().
- Montar um programa com menu (inserir, listar, alterar, excluir e pesquisar) usando ArrayList e Scanner.

2. Introdução ao ArrayList

Da ideia do array à necessidade de algo melhor

Você já estudou arrays, como em `int[] notas = new int[5];`. Eles funcionam muito bem quando sabemos, de antemão, quantos elementos vamos guardar. Mas muitos programas reais não sabem isso: quantos itens terá o carrinho de compras? Quantos alunos vão se matricular? Quantas tarefas o usuário vai anotar?

É aí que as **limitações dos arrays** aparecem:

- **Tamanho fixo:** o tamanho é definido na criação e nunca muda.
- **Sem adicionar nem remover de verdade:** para inserir um elemento no meio, é preciso deslocar os outros manualmente com um laço.
- **Para crescer, é preciso copiar:** a única saída é criar um array maior e copiar todos os elementos.
- **Posições sobrando:** se o array é grande demais, sobram posições vazias (0, false ou null) que o programador precisa controlar.

O que é o ArrayList

O `ArrayList` é uma classe da biblioteca padrão do Java (pacote `java.util`) que funciona como uma **lista dinâmica**: ela começa vazia, cresce quando você adiciona elementos e encolhe quando você remove. Por trás dos panos, ele usa um array interno e cuida sozinho de redimensioná-lo, então você não precisa se preocupar com isso.

Array × ArrayList

Característica	Array	ArrayList
Tamanho	Fixo, definido na criação	Dinâmico: cresce e encolhe
Tipos de elemento	Primitivos e objetos	Somente objetos (use wrappers para primitivos)
Tamanho atual	variavel.length (sem parênteses)	lista.size() (método)
Ler um elemento	v[i]	lista.get(i)
Alterar um elemento	v[i] = x;	lista.set(i, x);
Adicionar	Não existe (posição já existe)	lista.add(x);
Remover	Não existe	lista.remove(...);
Importação	Nenhuma (faz parte da linguagem)	import java.util.ArrayList;

Vantagens e desvantagens do ArrayList

Vantagens	Desvantagens
Tamanho dinâmico: você não precisa saber a quantidade de antemão.	Guarda apenas objetos: para int, double etc. é preciso usar os wrappers (Integer, Double...).
Métodos prontos: add, remove, contains, indexOf, sort...	Usa um pouco mais de memória e é um pouco mais lento que um array primitivo.
Código mais curto e legível para operações comuns.	Inserir ou remover no meio da lista obriga a deslocar os elementos seguintes (custo em listas muito grandes).

Quando utilizar cada um

Use um array quando...	Use um ArrayList quando...
a quantidade de elementos é conhecida e fixa (os 12 meses do ano, as 4 notas bimestrais).	a quantidade é desconhecida ou muda durante a execução (carrinho de compras, lista de alunos, tarefas).
você precisa de máximo desempenho com dados primitivos ou de matrizes.	você vai adicionar, remover ou pesquisar elementos com frequência.

> 💡 **Dica**
>
> Uma boa pergunta para decidir: "eu sei exatamente quantos elementos vou guardar?" Se sim, um array resolve. Se não, use ArrayList.

3. Criando um ArrayList

Importação

O `ArrayList` não faz parte do núcleo da linguagem: ele fica no pacote `java.util`. Por isso, é preciso importá-lo no início do arquivo, antes da declaração da classe:

```java
import java.util.ArrayList;
```

Declaração e instanciação

Veja o primeiro programa completo desta apostila. Nos próximos exemplos, vamos mostrar apenas os imports e o conteúdo do `main`.

```java
import java.util.ArrayList;

public class CriandoLista {
    public static void main(String[] args) {
        ArrayList<String> nomes = new ArrayList<>();
        ArrayList<Integer> notas = new ArrayList<>();
        ArrayList<Double> precos = new ArrayList<>();

        System.out.println(nomes);
        System.out.println(nomes.size());
    }
}
```

```
SAÍDA NO CONSOLE
[]
0
```

Observe cada parte de `ArrayList<String> nomes = new ArrayList<>();`:

- `ArrayList<String>`: o tipo da variável. Entre `< >` fica o **tipo dos elementos** que a lista vai guardar.
- `nomes`: o nome da variável.
- `new ArrayList<>()`: cria a lista. O `<>` (chamado de diamond) deixa o Java deduzir o tipo a partir da declaração.
- A lista nasce **vazia**: `size()` é 0 e `println(nomes)` mostra `[]`.

Conceito de Generics (<T>)

Os **Generics** permitem dizer ao Java qual é o tipo dos elementos da lista. O `T` de `<T>` significa "Type" (tipo): ele é um espaço reservado que você preenche com `String`, `Integer`, `Double` e assim por diante.

Isso traz segurança: se a lista foi declarada como `ArrayList<String>`, o compilador **recusa** qualquer tentativa de adicionar um número nela. O erro aparece na hora de compilar, e não só quando o programa já estiver rodando.

Por que não existe ArrayList<int>? Classes wrapper

Os Generics só aceitam **classes**, e `int`, `double`, `boolean` etc. são tipos primitivos. Para cada primitivo existe uma classe correspondente, chamada **wrapper** ("embrulho"):

Tipo primitivo	Classe wrapper	Exemplo de lista
int	Integer	ArrayList<Integer>
double	Double	ArrayList<Double>
boolean	Boolean	ArrayList<Boolean>
char	Character	ArrayList<Character>
long	Long	ArrayList<Long>

Na prática você quase não percebe a diferença, porque o Java converte automaticamente entre o primitivo e o wrapper. Essa conversão tem nome: **autoboxing** (de `int` para `Integer`) e **unboxing** (de `Integer` para `int`).

JAVA

```java
import java.util.ArrayList;

// ArrayList<int> lista = new ArrayList<>();   // ERRO: tipo primitivo não pode
ArrayList<Integer> lista = new ArrayList<>();

lista.add(5);                // autoboxing: int vira Integer
int x = lista.get(0);        // unboxing: Integer vira int
System.out.println(x + 1);
```

SAÍDA NO CONSOLE

```
6
```

> 💡 **Dica**
>
> Se você tentar escrever `ArrayList<int>`, o compilador acusa erro. Sempre troque o primitivo pelo wrapper: `ArrayList<Integer>`.

4. Adicionando elementos

O método `add()` tem duas formas:

- `add(elemento)`: adiciona **no final** da lista.
- `add(índice, elemento)`: **insere** na posição indicada, empurrando os elementos seguintes uma posição para a direita.

JAVA

```
import java.util.ArrayList;

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 NO CONSOLE

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

Veja o que aconteceu com os índices depois de `frutas.add(1, "Laranja")`:

Índice	0	1	2	3
Antes	Maçã	Banana	Uva	(vazio)
Depois	Maçã	Laranja	Banana	Uva

O comportamento dos índices

- Os índices começam em **0**, como nos arrays.
- Ao inserir no meio, **os elementos da direita são deslocados** e seus índices aumentam em 1.
- Em `add(índice, elemento)`, o índice pode ir de 0 até `size()` (inclusive). Usar `size()` equivale a adicionar no final. Um índice maior que `size()` causa `IndexOutOfBoundsException`.
- Imprimir a lista inteira com `System.out.println(lista)` mostra todos os elementos entre colchetes, separados por vírgula. É ótimo para conferir o resultado enquanto você programa.

5. Acessando elementos e descobrindo o tamanho

get() e size()

- `get(índice)` devolve o elemento da posição indicada, sem removê-lo.

- `size()` devolve **quantos elementos** a lista tem naquele momento.
- O último elemento está sempre em `size() - 1`.

JAVA

```java
import java.util.ArrayList;

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 NO CONSOLE

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

size() ou length?

Essa é uma das confusões mais comuns de quem está começando. Cada tipo tem a sua forma:

Tipo	Como obter o tamanho	Observação
Array (int[], String[]...)	v.length	É um atributo: não tem parênteses.
String	texto.length()	É um método: tem parênteses.
ArrayList	lista.size()	É um método: tem parênteses. Nunca use .length.

Erro comum: IndexOutOfBoundsException

Se você pedir um índice que não existe, o programa é interrompido com uma exceção. Com 3 elementos, os índices válidos são 0, 1 e 2:

JAVA

```java
import java.util.ArrayList;

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

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

SAÍDA NO CONSOLE

```
Exception in thread "main" java.lang.IndexOutOfBoundsException:
Index 3 out of bounds for length 3
```

📌 **Regra para memorizar**

Em uma lista com `size()` elementos, os índices válidos vão de **0** até **size() - 1**. Pedir qualquer outro índice gera `IndexOutOfBoundsException`.

6. Percorrendo um ArrayList

Existem três formas de passar por todos os elementos de uma lista. As três mostram o mesmo conteúdo; muda a quantidade de código e o que você tem disponível dentro do laço.

JAVA

```java
import java.util.ArrayList;

ArrayList<String> cores = new ArrayList<>();
cores.add("Azul");
cores.add("Verde");
cores.add("Rosa");

// 1) for tradicional (tem o índice)
for (int i = 0; i < cores.size(); i++) {
    System.out.println(i + ": " + cores.get(i));
}

// 2) for-each (só o valor)
for (String cor : cores) {
    System.out.println(cor);
}

// 3) while
int pos = 0;
while (pos < cores.size()) {
    System.out.println(cores.get(pos));
    pos++;
}
```

SAÍDA NO CONSOLE

```
0: Azul
1: Verde
2: Rosa
Azul
Verde
Rosa
Azul
Verde
Rosa
```

Forma	Quando usar	Tem o índice?
for tradicional	Quando você precisa da posição (mostrar numerado, alterar ou remover por índice, percorrer de trás para frente).	Sim, na variável do laço (i).
for-each	Quando só quer ler cada valor, do início ao fim. É a forma mais simples e legível.	Não.
while	Quando a repetição depende de uma condição além de "acabar a lista" (por exemplo, parar ao achar algo).	Só se você criar e atualizar um contador.

⚠ **Atenção**

- Não adicione nem remova elementos de uma lista dentro de um **for-each**: o Java lança `ConcurrentModificationException`. Para remover enquanto percorre, use o for tradicional.
- Ao remover dentro de um for tradicional que anda para a frente, os elementos seguintes mudam de índice e alguns acabam "pulados". A solução clássica é percorrer de trás para frente (veja o exercício 10).

7. Alterando elementos

O método `set(índice, novoValor)` **substitui** o elemento da posição indicada e devolve o valor antigo. O tamanho da lista não muda.

JAVA

```
import java.util.ArrayList;

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 NO CONSOLE

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

📌 **add(índice, x) × set(índice, x)**

Com a lista `[A, B, C]` e o comando na posição 1:

- `add(1, "X")` **insere**: resultado `[A, X, B, C]` (a lista cresce).
- `set(1, "X")` **substitui**: resultado `[A, X, C]` (a lista mantém o tamanho).

Assim como no `get()`, o índice do `set()` precisa existir (de 0 a `size() - 1`). O `set()` nunca cria uma posição nova.

8. Removendo elementos

O `remove()` também tem duas formas, e é importante não confundi-las:

- `remove(int índice)`: remove o elemento daquela posição e **devolve** o elemento removido.
- `remove(Object valor)`: remove a **primeira ocorrência** do valor e devolve `true` se encontrou ou `false` se não existia.

Remover por índice e por valor

JAVA

```java
import java.util.ArrayList;

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 NO CONSOLE

```
[Bruno, Diego]
```

Depois de qualquer remoção, os elementos seguintes **sobem uma posição** para ocupar o espaço, e o `size()` diminui em 1.

Cuidado com Integer e remove()

Em um `ArrayList<Integer>` existe uma pegadinha: `remove(1)` é interpretado como **índice** 1, e não como o valor 1. Para remover pelo valor, é preciso avisar ao Java que o número é um objeto `Integer`:

JAVA

```
import java.util.ArrayList;

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

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

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

SAÍDA NO CONSOLE

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

📌 **Regra para memorizar**

Em `ArrayList<Integer>`: `remove(2)` remove a **posição** 2. `remove(Integer.valueOf(2))` remove o **valor** 2.

Esvaziando a lista: isEmpty() e clear()

- `isEmpty()` devolve `true` se a lista não tem nenhum elemento (é o mesmo que `size() == 0`, só que mais legível).
- `clear()` remove **todos** os elementos de uma vez.

JAVA

```
import java.util.ArrayList;

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 NO CONSOLE

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

9. Pesquisando elementos

Método	O que faz	Retorno
contains(x)	Verifica se o valor existe na lista.	true ou false
indexOf(x)	Procura a primeira ocorrência de x.	O índice, ou -1 se não achar
lastIndexOf(x)	Procura a última ocorrência de x.	O índice, ou -1 se não achar

JAVA

```java
import java.util.ArrayList;

ArrayList<String> cidades = new ArrayList<>();
cidades.add("Niteroi");
cidades.add("Rio");
cidades.add("Petropolis");
cidades.add("Rio");

System.out.println(cidades.contains("Rio"));
System.out.println(cidades.contains("Sao Paulo"));
System.out.println(cidades.indexOf("Rio"));
System.out.println(cidades.lastIndexOf("Rio"));
System.out.println(cidades.indexOf("Sao Paulo"));
```

SAÍDA NO CONSOLE

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

- O valor **-1** significa "não encontrado". Como -1 nunca é um índice válido, ele serve de sinal: `if (lista.indexOf(x) != -1)`.
- Quando o valor aparece só uma vez, `indexOf()` e `lastIndexOf()` devolvem o mesmo número.
- A comparação de Strings considera maiúsculas e minúsculas: `"rio"` é diferente de `"Rio"`.

10. Ordenação

A classe `Collections` (também do pacote `java.util`) oferece métodos prontos para trabalhar com listas. Lembre-se de importá-la: `import java.util.Collections;`.

- `Collections.sort(lista)`: ordena em ordem **crescente** (números do menor para o maior; Strings em ordem alfabética).
- `Collections.reverse(lista)`: **inverte** a ordem em que os elementos estão naquele momento.

JAVA
```java
import java.util.ArrayList;
import java.util.Collections;

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 NO CONSOLE
```
[2, 4, 7, 9]
[9, 7, 4, 2]
```

Para obter uma lista em ordem **decrescente**, combine os dois: primeiro `sort`, depois `reverse`.

Diferença entre ordenar e inverter

Inverter **não ordena**. O `reverse()` apenas vira a lista de ponta-cabeça, seja ela qual for:

JAVA
```java
import java.util.ArrayList;
import java.util.Collections;

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

Collections.reverse(nomes);   // só inverte a ordem atual
System.out.println(nomes);

Collections.sort(nomes);      // ordena de A a Z
System.out.println(nomes);
```

SAÍDA NO CONSOLE
```
[Bruno, Ana, Carla]
[Ana, Bruno, Carla]
```

> ⚠ **Atenção**
>
> - Os dois métodos **alteram a própria lista**: não é criada uma cópia ordenada.
> - Em Strings, as letras maiúsculas vêm antes das minúsculas na ordenação: `[Ana, Zeca, bruno]`. Por isso, mantenha um padrão de capitalização ao cadastrar nomes.
> - O `Collections.sort()` funciona direto para String, Integer, Double etc. Para ordenar objetos criados por você, seria preciso definir o critério de comparação, assunto para um próximo momento.

11. ArrayList + Scanner: programa com menu

Vamos juntar tudo em um programa clássico: um **cadastro com menu**, que permite inserir, listar, alterar, excluir e pesquisar nomes. Cada opção do menu usa um método que você já conhece:

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

O programa completo

MENUARRAYLIST.JAVA

```
import java.util.ArrayList;
import java.util.Scanner;

public class MenuArrayList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<String> nomes = new ArrayList<>();
        int opcao;

        do {
            System.out.println();
            System.out.println("=== MENU ===");
            System.out.println("1 - Inserir");
            System.out.println("2 - Listar");
            System.out.println("3 - Alterar");
            System.out.println("4 - Excluir");
            System.out.println("5 - Pesquisar");
            System.out.println("0 - Sair");
            System.out.print("Opção: ");
```

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

            switch (opcao) {
                case 1:
                    System.out.print("Nome: ");
                    nomes.add(sc.nextLine());
                    System.out.println("Cadastrado!");
                    break;

                case 2:
                    if (nomes.isEmpty()) {
                        System.out.println("Lista vazia.");
                    } else {
                        for (int i = 0; i < nomes.size(); i++) {
                            System.out.println(i + " - " + nomes.get(i));
                        }
                    }
                    break;

                case 3:
                    System.out.print("Posição a alterar: ");
                    int posAlterar = Integer.parseInt(sc.nextLine());
                    if (posAlterar >= 0 && posAlterar < nomes.size()) {
                        System.out.print("Novo nome: ");
                        nomes.set(posAlterar, sc.nextLine());
                        System.out.println("Alterado!");
                    } else {
                        System.out.println("Posição inválida.");
                    }
                    break;

                case 4:
                    System.out.print("Nome a excluir: ");
                    String alvo = sc.nextLine();
                    if (nomes.remove(alvo)) {
                        System.out.println("Excluído!");
                    } else {
                        System.out.println("Nome não encontrado.");
                    }
                    break;

                case 5:
                    System.out.print("Nome a pesquisar: ");
                    String busca = sc.nextLine();
                    int achado = nomes.indexOf(busca);
                    if (achado >= 0) {
                        System.out.println("Encontrado na posição " + achado);
                    } else {
                        System.out.println("Nome não encontrado.");
                    }
                    break;

                case 0:
                    System.out.println("Até logo!");
                    break;
```

```
                default:
                    System.out.println("Opção inválida.");
            }
        } while (opcao != 0);

        sc.close();
    }
}
```

Como o programa funciona

- O **do-while** repete o menu até o usuário digitar 0. Ele é ideal aqui porque o menu precisa aparecer pelo menos uma vez.
- O **switch** escolhe o que fazer de acordo com a opção digitada. Cada `case` termina com `break`.
- A lista `nomes` fica fora do laço, para que os dados **continuem guardados** entre uma opção e outra.
- Em Alterar, o `if` confere se a posição existe antes de chamar `set()`. Sem essa validação, uma posição inválida causaria `IndexOutOfBoundsException`.

Por que Integer.parseInt(sc.nextLine())?

Existe um problema clássico quando se mistura `nextInt()` com `nextLine()`. O `nextInt()` lê só o número e **deixa o ENTER** (a quebra de linha) no buffer. O `nextLine()` seguinte encontra esse ENTER e devolve uma linha vazia, "pulando" a leitura do texto, como se o programa ignorasse o que você ia digitar.

A solução mais simples é ler **sempre** com `nextLine()` e converter o número com `Integer.parseInt()`, como no programa acima. Assim o buffer nunca fica com sobras.

▶ Exemplo de execução (início)

```
=== MENU ===
1 - Inserir
2 - Listar
3 - Alterar
4 - Excluir
5 - Pesquisar
0 - Sair
Opção: 1
Nome: Ana
Cadastrado!

(o menu aparece de novo...)
Opção: 2
0 - Ana
```

> ⚠ **Atenção**
>
> O programa assume que o usuário digita números válidos no menu. Se alguém digitar uma letra, o `Integer.parseInt()` lança `NumberFormatException`. Tratar esse caso exige `try/catch`, assunto para depois.

Para ir além: ArrayList de objetos

Se a sua turma já estudou classes, o uso mais comum do `ArrayList` no dia a dia é guardar **objetos criados por você**, e não apenas Strings e números. Veja uma lista de alunos:

JAVA

```
import java.util.ArrayList;

class Aluno {
    String nome;
    double nota;

    Aluno(String nome, double nota) {
        this.nome = nome;
        this.nota = nota;
    }

    @Override
    public String toString() {
        return nome + " (" + nota + ")";
    }
}

public class ListaDeAlunos {
    public static void main(String[] args) {
        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 NO CONSOLE

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

- `ArrayList<Aluno>` guarda objetos da classe `Aluno`. Todos os métodos desta apostila (add, get, remove, for-each...) funcionam do mesmo jeito.

- O método `toString()` define como o objeto aparece quando a lista inteira é impressa. Sem ele, o Java mostraria um código de memória pouco útil.

12. Exercícios de fixação

Resolva no espaço abaixo de cada questão (ou no computador, testando o código). O gabarito está no final da apostila.

1) Cite duas limitações dos arrays que o ArrayList resolve.

---

2) Escreva o código (com o import) que cria um ArrayList de Strings chamado tarefas e adiciona "Estudar", "Treinar" e "Dormir". Depois, imprima a lista.

---

3) Qual é a saída do trecho abaixo?

```java
ArrayList<String> l = new ArrayList<>();
l.add("A");
l.add("B");
l.add("C");
l.add(1, "X");
System.out.println(l);
System.out.println(l.size());
```

---

4) Qual é a saída do trecho abaixo? (Acompanhe a lista passo a passo.)

```java
ArrayList<Integer> n = new ArrayList<>();
n.add(5);
n.add(10);
n.add(15);
n.add(20);
n.set(0, 50);
n.remove(2);
n.remove(Integer.valueOf(20));
System.out.println(n);
```

---

5) Há um erro nesta linha. Explique o problema e escreva a versão correta:

```java
ArrayList<int> idades = new ArrayList<>();
```

---

6) Há um erro na última linha. Explique o problema e escreva a versão correta:

```java
ArrayList<String> nomes = new ArrayList<>();
nomes.add("Ana");
System.out.println(nomes.length);
```

---

7) Dado um ArrayList<Integer> chamado valores, escreva um trecho que calcule e mostre a soma de todos os elementos (use for-each).

---

8) Dado um ArrayList<String> chamado nomes, escreva um trecho que mostre em que posição está o nome "Marcos" ou a mensagem "Marcos não está na lista".

---

9) Dada a lista [8, 3, 10, 1], escreva o trecho que a deixa em ordem decrescente e diga o que será impresso.

---

10) Desafio de atenção: escreva um trecho que remova de um ArrayList<Integer> todos os números negativos. Dica: percorra a lista de trás para frente com um for tradicional.

⚠ **Erros comuns**

- Usar `ArrayList<int>` em vez de `ArrayList<Integer>`.
- Escrever `lista.length` ou `lista[0]` em vez de `lista.size()` e `lista.get(0)`.
- Esquecer o `import java.util.ArrayList;` (ou o `import java.util.Collections;`).
- Usar um índice igual a `size()` no `get()` ou no `set()` (o último índice é `size() - 1`).
- Confundir `remove(índice)` com `remove(valor)` em listas de Integer.
- Remover elementos dentro de um for-each.
- Confundir `Collections.reverse()` (inverter) com `Collections.sort()` (ordenar).

13. Desafio final: lista de compras

Crie um programa com menu (do-while + switch + Scanner) para gerenciar uma **lista de compras** usando `ArrayList<String>`. O programa deve oferecer as opções:

- 1 - Adicionar item
- 2 - Listar itens (numerados a partir de 1; avisar se a lista estiver vazia)
- 3 - Remover item (informando o nome; avisar se não existir)
- 4 - Pesquisar item (dizer se está ou não na lista)
- 5 - Ordenar de A a Z
- 6 - Mostrar a quantidade de itens
- 0 - Sair

Use somente `nextLine()` para ler os dados (e `Integer.parseInt()` para a opção do menu). Teste o programa com a lista vazia, com itens repetidos e com uma opção inválida.

💡 **Dica**

Comece pelo esqueleto do menu da seção 11 e vá acrescentando um case por vez, testando a cada passo.

14. Gabarito

Questão 1

Duas entre as limitações: o tamanho é fixo; não é possível adicionar ou remover elementos de verdade (inserir no meio exige deslocar tudo manualmente); para crescer é preciso criar um array maior e copiar; podem sobrar posições vazias.

Questão 2

JAVA

```java
import java.util.ArrayList;

// dentro do main:
ArrayList<String> tarefas = new ArrayList<>();
tarefas.add("Estudar");
tarefas.add("Treinar");
tarefas.add("Dormir");
System.out.println(tarefas);
```

SAÍDA NO CONSOLE

```
[Estudar, Treinar, Dormir]
```

Questão 3

Saída (a lista é [A, B, C] e o X entra na posição 1, empurrando os demais):

SAÍDA NO CONSOLE

```
[A, X, B, C]
4
```

Questão 4

Passo a passo: [5, 10, 15, 20] → set(0, 50) → [50, 10, 15, 20] → remove(2) remove a posição 2, que guarda o 15 → [50, 10, 20] → remove(Integer.valueOf(20)) remove o valor 20 → resultado:

SAÍDA NO CONSOLE

```
[50, 10]
```

Questão 5

Tipos primitivos não podem ser usados em Generics. Correção: `ArrayList<Integer> idades = new ArrayList<>();`

Questão 6

O ArrayList não tem `length`. O tamanho é obtido com o método `size()`. Correção: `System.out.println(nomes.size());`

Questão 7

JAVA

```
ArrayList<Integer> valores = new ArrayList<>();
valores.add(4);
valores.add(10);
valores.add(6);

int soma = 0;
for (int v : valores) {
    soma += v;
}
System.out.println("Soma: " + soma);
```

SAÍDA NO CONSOLE

```
Soma: 20
```

Questão 8

JAVA

```
ArrayList<String> nomes = new ArrayList<>();
nomes.add("Ana");
nomes.add("Marcos");
nomes.add("Lia");

int pos = nomes.indexOf("Marcos");
if (pos != -1) {
    System.out.println("Marcos está na posição " + pos);
} else {
    System.out.println("Marcos não está na lista");
}
```

SAÍDA NO CONSOLE

```
Marcos está na posição 1
```

Questão 9

JAVA

```
ArrayList<Integer> v = new ArrayList<>();
v.add(8);
v.add(3);
v.add(10);
v.add(1);

Collections.sort(v);
Collections.reverse(v);
System.out.println(v);
```

SAÍDA NO CONSOLE

```
[10, 8, 3, 1]
```

Questão 10

JAVA

```java
ArrayList<Integer> v = new ArrayList<>();
v.add(5);
v.add(-2);
v.add(-7);
v.add(8);
v.add(-1);

for (int i = v.size() - 1; i >= 0; i--) {
    if (v.get(i) < 0) {
        v.remove(i);
    }
}
System.out.println(v);
```

SAÍDA NO CONSOLE

```
[5, 8]
```

Percorrer de trás para frente evita o problema dos índices que mudam depois de cada remoção: remover o elemento i só desloca elementos que já foram verificados.

Desafio final: lista de compras (uma solução possível)

LISTADECOMPRAS.JAVA

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class ListaDeCompras {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<String> itens = new ArrayList<>();
        int opcao;

        do {
            System.out.println();
            System.out.println("=== LISTA DE COMPRAS ===");
            System.out.println("1 - Adicionar item");
            System.out.println("2 - Listar itens");
            System.out.println("3 - Remover item");
            System.out.println("4 - Pesquisar item");
            System.out.println("5 - Ordenar de A a Z");
            System.out.println("6 - Mostrar quantidade");
            System.out.println("0 - Sair");
            System.out.print("Opção: ");
            opcao = Integer.parseInt(sc.nextLine());
```

```
            switch (opcao) {
                case 1:
                    System.out.print("Item: ");
                    itens.add(sc.nextLine());
                    break;

                case 2:
                    if (itens.isEmpty()) {
                        System.out.println("A lista está vazia.");
                    } else {
                        for (int i = 0; i < itens.size(); i++) {
                            System.out.println((i + 1) + ". " + itens.get(i));
                        }
                    }
                    break;

                case 3:
                    System.out.print("Item a remover: ");
                    if (itens.remove(sc.nextLine())) {
                        System.out.println("Removido!");
                    } else {
                        System.out.println("Item não encontrado.");
                    }
                    break;

                case 4:
                    System.out.print("Item a pesquisar: ");
                    if (itens.contains(sc.nextLine())) {
                        System.out.println("Está na lista.");
                    } else {
                        System.out.println("Não está na lista.");
                    }
                    break;

                case 5:
                    Collections.sort(itens);
                    System.out.println("Lista ordenada!");
                    break;

                case 6:
                    System.out.println("Itens na lista: " + itens.size());
                    break;

                case 0:
                    System.out.println("Boas compras!");
                    break;

                default:
                    System.out.println("Opção inválida.");
            }
        } while (opcao != 0);

        sc.close();
    }
}
```

Outras soluções são válidas, desde que usem os métodos corretos e tratem os casos de lista vazia e de item não encontrado.

15. Resumo para estudar antes da prova

- ArrayList é uma **lista dinâmica** do pacote `java.util`; precisa de `import java.util.ArrayList;`.
- Declaração: `ArrayList<Tipo> nome = new ArrayList<>();`. O tipo entre `< >` (Generics) deve ser uma **classe**: use `Integer`, `Double`, `Boolean`... (wrappers) no lugar de int, double, boolean.
- Índices começam em **0**; o último é `size() - 1`. Índice inválido gera `IndexOutOfBoundsException`.
- O tamanho é `size()` (método), nunca `length`.
- `add(x)` adiciona no final; `add(i, x)` insere e desloca os demais; `set(i, x)` substitui sem mudar o tamanho.
- `remove(i)` remove por posição; `remove(valor)` remove a primeira ocorrência. Em `ArrayList<Integer>`, use `remove(Integer.valueOf(x))` para remover pelo valor.
- `contains()` devolve true/false; `indexOf()` e `lastIndexOf()` devolvem a posição ou **-1** se não achar.
- Percorrer: for tradicional (com índice), for-each (só o valor) ou while. Não remova elementos dentro de um for-each.
- `Collections.sort()` ordena de forma crescente; `Collections.reverse()` só inverte a ordem atual. Para decrescente: sort e depois reverse.
- Em menus com Scanner, leia tudo com `nextLine()` e converta com `Integer.parseInt()` para evitar o problema do ENTER no buffer.

Tabela de métodos

Método	O que faz
add(x) / add(i, x)	Adiciona no final / insere na posição i
get(i)	Devolve o elemento da posição i
set(i, x)	Substitui o elemento da posição i
remove(i) / remove(x)	Remove por posição / remove a primeira ocorrência do valor
size()	Quantidade de elementos
isEmpty()	true se a lista está vazia
clear()	Remove todos os elementos
contains(x)	true se o valor existe
indexOf(x) / lastIndexOf(x)	Primeira / última posição do valor (ou -1)
Collections.sort(lista)	Ordena em ordem crescente

Método	O que faz
Collections.reverse(lista)	Inverte a ordem atual