# Programação Orientada a Objetos

Prof.ª Ma. Jessica Oliveira

Católica
Universidade Católica de Brasília

Aula 08 – 14/04/2025

# Estruturas de Dados e Manipulação de Coleções.

Universidade Católica de Brasília

# O que são coleções?

- Ao desenvolver um sistema, é comum precisarmos lidar com vários dados ao mesmo tempo: uma lista de livros, um conjunto de autores, um cadastro de usuários, entre outros.
- Para isso, o Java oferece um conjunto de estruturas chamadas coleções, que facilitam o armazenamento e a organização de grupos de objetos.
- Essas coleções fazem parte de um conjunto maior conhecido como *Framework* de Coleções do Java, que oferece soluções prontas para guardar, acessar, percorrer e manipular elementos de forma eficiente.
- Em vez de criar nossas próprias estruturas do zero ou depender de *arrays* com tamanho fixo, podemos usar essas ferramentas para tornar o código mais limpo, flexível e fácil de manter.

# Tipos principais de coleções.

# Listas (`List`)

- São estruturas que mantêm os elementos em uma **ordem definida e aceitam repetições**.
- Imagine uma fila de nomes onde a ordem importa: essa é a função de uma lista.
- Exemplos: `ArrayList`, `LinkedList`.

# Conjuntos (`Set`)

- Aqui, o importante é **garantir que nenhum valor se repita**. A ordem dos elementos não é prioridade.
- Útil quando queremos guardar dados únicos, como uma lista de usuários que votaram em uma enquete.
- Exemplo: `HashSet`.

# Mapas (Map)

- Funcionam como um dicionário. **Cada dado é associado a uma chave única**. Assim, ao procurar pela chave, obtemos o valor correspondente.
- Ideal para quando queremos acessar informações rapidamente, como um sistema de *login* que relaciona ID e nome do usuário.
- Exemplo: `HashMap`.

# Por que não usar apenas *arrays*?

- Os *arrays* tradicionais são úteis em algumas situações, mas têm limitações importantes:
  - O tamanho é fixo após a criação.
  - Não possuem métodos prontos para adicionar, remover ou buscar elementos de maneira prática.
  - Não são tão flexíveis para trabalhar com diferentes tipos de dados e operações.
- As coleções surgem justamente para resolver esses problemas, oferecendo uma alternativa mais moderna e funcional.

# Conhecendo as estruturas mais usadas.

# ArrayList.

- Funciona como uma lista que pode crescer conforme os dados vão sendo adicionados. Permite acessar os elementos diretamente por um número (índice).
- É uma das estruturas mais comuns e úteis no dia a dia.
- Exemplo:

```
ArrayList<String> nomes = new ArrayList<>();
nomes.add("Ana");
nomes.add("Carlos");
System.out.println(nomes.get(0)); // Exibe "Ana"
```

# ArrayList

- Útil quando você precisa de uma lista ordenada de elementos, em que a ordem de inserção é importante e o acesso aos dados por número de posição (índice) precisa ser rápido.
- Por que usar?
  - A leitura por índice é muito rápida.
  - A ordem dos dados é mantida.
  - É simples de usar e funciona bem para a maioria dos casos.

# LinkedList

- Também organiza os elementos em sequência, mas de uma forma diferente: cada item conhece o anterior e o próximo.
- Por isso, inserir ou remover valores no meio da lista é mais rápido do que no `ArrayList`.
- Exemplo:

```
LinkedList<String> fila = new LinkedList<>();
fila.add("João");
fila.addFirst("Maria");
fila.addLast("Pedro");
```

# `LinkedList`

- Bom para quando há muitas alterações na lista, como inserções ou exclusões no meio ou no início. Embora seja parecida com o `ArrayList`, ela é mais eficiente nessas operações.
- Por que usar?
  - Melhor desempenho quando há muitas inserções e remoções.
  - Útil para estruturas como filas, pilhas e históricos.

# HashSet

- É ideal para guardar valores sem repetições.
- Se tentarmos inserir um item que já existe, ele será ignorado automaticamente.
- Exemplo:

```
HashSet<String> cores = new HashSet<>();
cores.add("Azul");
cores.add("Verde");
cores.add("Azul"); // Não será adicionado de novo
```

# HashSet

- Útil quando o mais importante é garantir que não haja elementos duplicados, sem se preocupar com a ordem em que foram adicionados.
- Por que usar?
  - Garante que os dados sejam únicos.
  - Muito eficiente para verificações rápidas: `contains()` retorna rápido.
  - Ideal para sistemas de validação, filtros ou históricos sem repetições.

# HashMap

- Relaciona uma chave a um valor. Usamos a chave para encontrar rapidamente o que precisamos.
- Serve muito bem para cadastros e buscas específicas.
- Exemplo:

```
HashMap<Integer, String> usuarios = new HashMap<>();
usuarios.put(1, "Jessica");
usuarios.put(2, "Hélder");
System.out.println(usuarios.get(2)); // Exibe "Hélder"
```

# HashMap

- Bom para quando você precisa acessar dados rapidamente por um identificador exclusivo (chave). O mapa permite associar informações a uma chave específica.
- Por que usar?
  - É extremamente rápido para buscar informações.
  - Muito útil em cadastros, autenticações, filtros por categoria e relacionamentos chave-valor.
  - Substitui buscas manuais e estruturas improvisadas.

# Como percorrer coleções.

# for comum

```
for (int i = 0; i < lista.size(); i++) {
    System.out.println(lista.get(i));
}
```

# for-each (mais simples)

```
for (String nome : lista) {
    System.out.println(nome);
}
```

# Iterator (mais seguro em certos casos)

```
Iterator<String> it = lista.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

# Boas escolhas e cuidados ao usar coleções.

- Ao lidar com coleções, algumas práticas ajudam a evitar problemas e tornar o código mais eficiente:
  - Pense no que você precisa antes de escolher a estrutura. Vai acessar por índice? Use `ArrayList`. Precisa evitar dados repetidos? `HashSet`. Precisa de chave e valor? `HashMap`.
  - Evite modificar a lista enquanto estiver percorrendo com `for-each`. Use `Iterator` se precisar remover elementos nesse processo.
  - Antes de tentar acessar elementos, verifique se a coleção está vazia.
  - Use métodos prontos das coleções como `add()`, `remove()`, `contains()` e `get()` para evitar escrever código desnecessário.

# Comparando as estruturas.

| Estrutura | Permite repetição? | Mantém a ordem? | Quando usar? |
|---|---|---|---|
| ArrayList | Sim | Sim | Quando importa a ordem e acesso rápido. |
| LinkedList | Sim | Sim | Quando há muitas inserções e remoções. |
| HashSet | Não | Não | Quando é necessário evitar duplicações. |
| HashMap | Chaves únicas | Não | Quando se precisa buscar algo por chave. |

# Vamos para a prática?

# Microprojeto 08

🎯 **Objetivo:** utilizar coleções para armazenar e manipular dados.

Católica
Universidade Católica de Brasília

# Para o pós-aula...

- Implementar um `ArrayList`, `HashSet` ou `HashMap`.
- Criar métodos para ordenar e buscar elementos.
- Comparar diferentes estruturas de dados para melhor desempenho.
- Relatório técnico comparando as estruturas utilizadas e justificando escolhas.

# Dúvidas?

jessica.oliveira@p.ucb.br

Católica
Universidade Católica de Brasília