Java API Collections

CST em Análise e Desenvolvimento de Sistemas

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Campus São José

Licenciamento

Slides licenciados sob Creative Commons “Atribuição 4.0 Internacional”

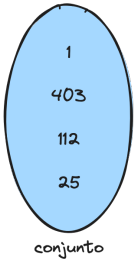
Motivação

- Suponha que você precise armazenar uma lista de nomes de pessoas. Nesta lista, você pode querer adicionar, remover ou pesquisar nomes.
- O uso de um vetor seria uma solução simples, como no exemplo abaixo:

```
String [] nomes = new String[1000];
nomes[0] = "João";
nomes[1] = "Maria";
// ...
```

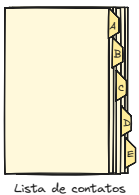
- Seria fácil adicionar novos nomes, mas e se você quisesse saber quantos nomes foram armazenados? Ou se não quisesse armazenar nomes duplicados? Ou ainda, se quisesse remover um nome específico?

Grupos naturais de objetos



Conjunto de números naturais

- Conjunto não ordenado de elementos
- Não permite elementos duplicados



Lista de contatos

- Permite elementos duplicados
- Oferece uma forma rápida de acessar elementos

Coleção em Java

- **Coleção é um objeto** que **representa um conjunto de objetos** dentro de uma única unidade
  - Permite armazenar, obter e manipular dados agregados com facilidade
- Coleção é um **tipo de dado abstrato** que representa um conjunto de objetos
  - baralho
  - pasta de e-mails
  - catálogo telefônico
  - …

Java Collections Framework I

https://docs.oracle.com/en/java/javase/21/core/java-collections-framework.html

Benefícios do uso do *Java Collections Framework*

- Redução de tempo de desenvolvimento
- Redução de complexidade do código
- Aumentar a eficiência e qualidade código
- Aumentar a portabilidade do código

Java Collections Framework II

https://docs.oracle.com/en/java/javase/21/core/java-collections-framework.html

- **Interfaces**
  - Tipos de dados abstratos para representar coleções de objetos
  - Ex: `List`, `Set`, `Queue`, `Map`
- **Implementações**
  - Estruturas de dados para armazenar e manipular grupos de objetos
  - Ex: `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`, `HashMap`, `TreeMap`

Java Collections Framework III

https://docs.oracle.com/en/java/javase/21/core/java-collections-framework.html

- **Algoritmos**
  - Operações comuns para manipular coleções de objetos
  - Ex: Busca, ordenação, embaralhamento, agrupamento, etc.
- **Classes utilitárias**
  - `Collections` – Métodos estáticos para operações comuns em coleções
  - `Arrays` – Métodos estáticos para operações comuns em vetores
  - `Comparator` – Interface funcional para ordenação de objetos

Java Collections Framework I

Interfaces e suas implementações

- **List**
  - Coleção ordenada e permite elementos duplicados, sendo possível acessar elementos por índice
  - Implementações: `ArrayList`, `LinkedList`, `Vector`, `Stack`
- **Set**
  - Segue a abstração de conjuntos matemáticos e não permite elementos duplicados e não mantém a ordem de inserção
  - Implementações: `HashSet`, `LinkedHashSet`, `TreeSet`

Java Collections Framework II

Interfaces e suas implementações

- **Queue**
  - Fila que ordena elementos para serem processados posteriormente, por exemplo, FIFO (primeiro que entra é o primeiro que sai)
  - Implementações: `LinkedList`, `ArrayDeque`, `PriorityQueue`
- **Map**
  - Mapeia chaves para valores (também conhecido como dicionário)
  - Não permite chaves duplicadas e cada chave mapeia somente um valor
  - Implementações: `HashMap`, `LinkedHashMap`, `TreeMap`

Qual coleção usar?

Você precisa de uma coleção...

- que mantenha a ordem de inserção e permita elementos duplicados?
- que permita acessar elementos por índice?
- que não permita elementos duplicados?
- que permita armazenar nulos?
- que oferte operações de busca, inserção e remoção eficientes?
- que opere em ambientes concorrentes? (*multi-thread*)

Exemplos de uso

Os exemplos a seguir são simplificados e visam apenas apresentar a sintaxe básica de uso das coleções. Como ainda não foi apresentado o conceito de herança e polimorfismo na disciplina, optou-se na pela declaração de variáveis utilizando classes concretas ao invés de interfaces.

ArrayList

Armazena objetos em um vetor, cujo tamanho aumenta automaticamente

```java
// Criando uma lista de String
ArrayList<String> lista = new ArrayList<>(); // poderia ser de qualquer tipo, ex: Pessoa

// Adicionando elementos
lista.add("POO");
lista.add("IFSC");

// Obtendo o total de elementos na coleção
int total = lista.size();

// Convertendo para vetor
String[] v = (String[]) lista.toArray();

// Obtém elemento na posição 1
String nome = lista.get(1);

// Obtém elemento na última posição
String ultimo = lista.getLast();
```

ArrayList

Percorrendo elementos de uma coleção

```
ArrayList<Pessoa> lista = new ArrayList<>();

// Adicionando objetos do tipo Pessoa na lista
lista.add(new Pessoa("Ana", 40));
lista.add(new Pessoa("Juca", 30));
lista.add(new Pessoa("Maria", 25));

// Usando for-each para percorrer a lista
for(Pessoa p: lista){
    System.out.println(p);
}

// Usando forEach com lambda
lista.forEach(pessoa -> System.out.println(pessoa));

// Usando forEach com referência de método
lista.forEach(System.out::println);
```

ArrayList

Ordenação e embaralhamento de elementos

```
ArrayList<Integer> lista = new ArrayList<>();
lista.add(2);
lista.add(4);
lista.add(1);

// Ordenando com java.util.Comparator
lista.sort(Comparator.naturalOrder());

// Ordenando com Collections
Collections.sort(lista);

// Embaralhando elementos
Collections.shuffle(lista);

// Verifica se a lista está vazia
boolean vazia = lista.isEmpty();
```

ArrayList

Busca de elementos

```
Pessoa a = new Pessoa("Ana", 40)
ArrayList<Pessoa> lista = new ArrayList<>();
lista.add(a);
lista.add(new Pessoa("Juca", 30));
lista.add(new Pessoa("Maria", 25));

// Busca por um elemento cujo nome seja "Maria"
Pessoa p = lista.stream()
    .filter(pessoa -> pessoa.getNome().equals("Maria"))
    .findFirst()
    .orElse(null); // retorna null se não encontrar

if (p != null){ // se encontrou
    System.out.println(p);
}

// Verificando se um objeto, referenciado por a, está na lista
boolean contem = lista.contains(a);

// Removendo elemento se existir
lista.removeIf(elemento->elemento.getNome().equals("Juca"));
```

Set

Conjunto de elementos que não permite elementos duplicados

```
HashSet<String> conjunto = new HashSet<String>();

conjunto.add("A");
conjunto.add("C");
conjunto.add("B");

// Verificando se conseguiu adicionar um elemento
if (conjunto.add("A")){
    System.out.println("Adicionado");
} else {
    System.out.println("Não adicionado");
}

System.out.println(conjunto);

// Removendo todos elementos
conjunto.clear();
```

Queue

Armazena elementos em uma fila

```
Queue<String> fila = new LinkedList<>();

fila.add("Juca");
fila.add("Ana");
fila.add("Maria");

// Remove o primeiro elemento da fila. Retorna null se a fila estiver vazia.
String nome = fila.poll();
if (nome != null) {
    System.out.println(nome);
}
```

- `Queue` é uma interface que estende `Collection`, algumas implementações:
  - `LinkedList`
  - `ArrayDeque`
  - `PriorityQueue`

Map

Mapeia chaves para valores

```
HashMap<String, String> cores = new HashMap<>();

cores.put("Azul", "#0000FF");
cores.put("Vermelho", "#FF0000");
cores.put("Verde", "#00FF00");

String codigo = cores.get("Azul");
if (codigo != null) {
    System.out.println("Código da cor Azul: " + codigo);
} else {
    System.out.println("Cor não encontrada");
}

// Percorrendo o Map
cores.forEach((chave, valor) -> {
    System.out.println(chave + " - " + valor);
});

// Removendo um valor
cores.entrySet().removeIf(elemento -> elemento.getKey().equals("Vermelho"));
```

Algumas boas práticas com coleções I

- Utilize classes empacotadoras (`Integer`, `Double`, etc.) para tipos primitivos
  - Coleções não aceitam tipos primitivos, apenas objetos
- Utilize `isEmpty()` ao invés de `size() == 0`
- Utilize `contains()` ao invés de `indexOf()`
- Utilize `add()` ao invés de `add(index, elemento)`
- Utilize `removeIf()` ao invés de `remove()`
  - `removeIf()` aceita expressões lambda
- Não use loop `for` com índice para percorrer uma coleção, utilize `for-each` ou `forEach` com lambda

Algumas boas práticas com coleções II

```java
ArrayList<Integer> lista = new ArrayList<>();
lista.add(2);
lista.add(4);
lista.add(1);

// Não faça isso
for (int i = 0; i < lista.size(); i++){
    System.out.println(lista.get(i));
}

// Faça isso
for (String elemento: lista)
    System.out.println(elemento);

// ou faça isso com lambda
lista.forEach(elemento -> System.out.println(elemento));
```

Algumas boas práticas com coleções

Quando aprendermos sobre herança e polimorfismo, veremos o porquê

- Utilize a interface mais genérica possível
  - Programar para interfaces, não para implementações deixa o código mais flexível
- Objetos armazenados em coleções devem implementar `equals`, `hashCode` e `Comparable` (se necessário)
  - `equals` para comparar objetos e `hashCode` para calcular o código de espalhamento
  - A interface `Comparable` é usada para ordenar objetos