Programação I

Coleções e Estruturas de Dados

Samuel da Silva Feitosa

Aula 23
2022/1

Coleções

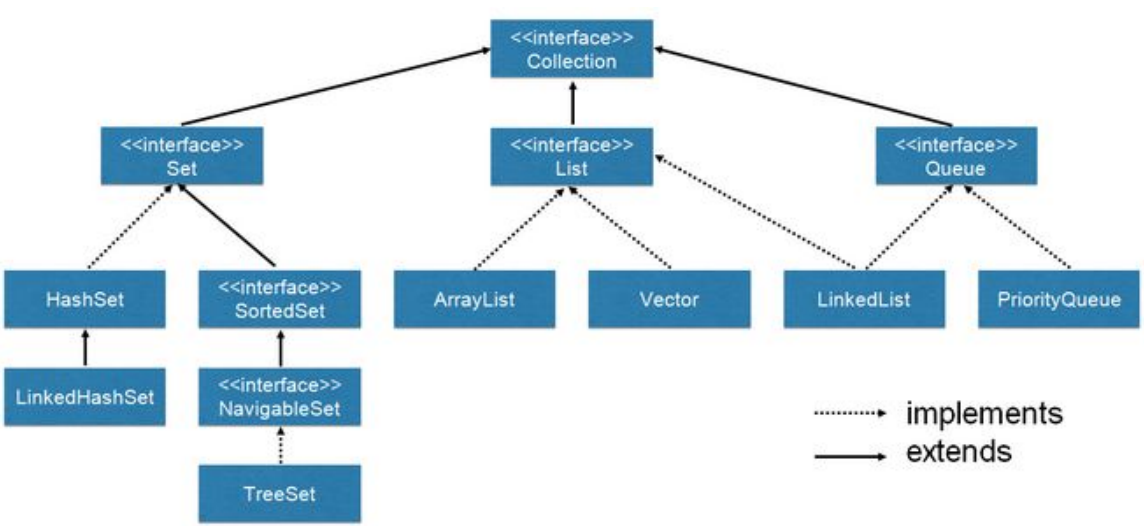
- Framework de coleções ou *collections*.
  - Um framework é um conjunto correlato de classes projetadas e testadas para serem utilizadas de modo efetivo em um grande número de projetos.
  - Provê a redução de esforços de aprendizagem, projeto e programação, além de ganhos em qualidade e desempenho.
- Uma coleção pode ser entendida como a representação de um grupo de objetos.
  - É uma espécie de container que agrupa objetos.

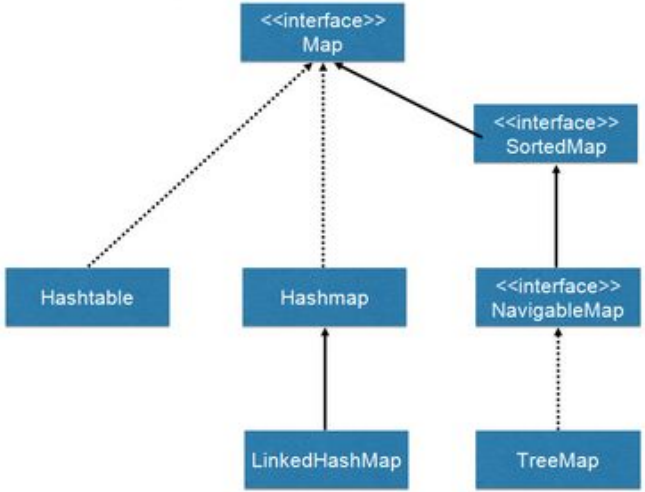
Collections Framework

- Reescrito na versão 5 para utilizar genéricos.
- É uma arquitetura para representar e manipular coleções de objetos de qualquer tipo.
  - Inclui estruturas de dados e algoritmos de qualidade.
- Permite que o desenvolvedor se concentre em seus objetivos.
- Collections é dividido em:
  - **Interfaces principais:** distinguem os tipos de coleções.
  - **Implementações:** classes disponíveis para uso.
  - **Algoritmos:** aplicáveis às coleções existentes.

Interfaces Principais

- Conjunto básico de interfaces.
  - Especificações de tipos abstratos.
  - Conjuntos, Listas, Filas, Mapas (chave-valor).





Interface Collection<E>

- Define uma coleção simples, sem observar detalhes de ordenação ou duplicação de elementos.
- Especifica um conjunto de métodos que podem ser divididos em operações elementares, de conjuntos e de conversão.

	Interfaces			
	Set<E>	List<E>	Queue<E>	Map<K,V>
Implementações	HashSet<E> LinkedHashSet<E> TreeSet<E>	ArrayList<E> LinkedList<E>	ArrayDeque<E> PriorityQueue<E>	EnumMap<K,V> HashMap<K,V> LinkedHashMap<K,V> TreeMap<K,V>

Operações elementares

- **add** e **remove** efetuam a adição e remoção individual de objetos.
- **contains** determina a existência de um objeto.
- **isEmpty** indica se a coleção está vazia.
- **size** indica a quantidade de objetos contidos.
- **iterator** retorna um Iterator que possibilita navegar pelos elementos da coleção.
- Possui também **addAll**, **removeAll**, **containsAll**, **clear**, **retainAll**.
- **toArray** converte a coleção em formato de array.

Iterable<E> e Iterator<E>

- Essas interfaces foram criadas para prover a navegação entre os elementos de quaisquer coleções.
- Todas as implementações de coleções possuem o método iterator().
  - Isto define a semântica para navegação dos elementos de coleções por meio de código genérico.
- Método forEach(Consumer<? super T> action)
  - Executa a ação definida pelo Consumer<T> em cada elemento disponível.

Exemplo Iterable<E> e Iterator<E>

```java
public static void main(String[] args) {
    List<String> lista = Arrays.asList("Banana", "Laranja", "Pera", "Uva");

    // Imprime todos os elementos
    lista.forEach(System.out::println);

    // Imprime elementos usando expressão lambda e filtro
    lista.forEach(e -> {
        if (e.length() > 5) {
            System.out.println(e);
        }
    });

    // Imprime elementos usando Iterator e filtro
    Iterator<String> it = lista.iterator();
    while (it.hasNext()) {
        String e = it.next();
        if (e.startsWith("P")) {
            System.out.println(e);
        }
    }
}
```

Interface Set<E> (conjuntos)

- Coleção que não permite elementos duplicados.
- Operações definidas para o tipo Set exibem comportamento das operações matemáticas.

Notação Matemática	Operação	Notação Programática
$z \in A$	z “pertence a” A	A.contains(z)
$A \supset B$	A “contém” B	A.containAll(B)
$A \cup B$	A “união” B	A.addAll(B)
$A \cap B$	A “intersecção” B	A.retainAll(B)
A - B	“diferença entre” A e B	A.removeAll(B)

Exemplo Set<E>

```
public static void main(String[] args) {
    Set<Integer> A = new HashSet<>();
    Set<Integer> B = new HashSet<>();

    A.addAll(Arrays.asList(1, 2, 4, 5));
    B.addAll(Arrays.asList(1, 4, 6));

    // Exibe conjuntos
    System.out.println("A " + A + " Size: " + A.size());
    System.out.println("B " + B + " Size: " + B.size());
    // Conjunto cópia de A
    Set<Integer> copiaA = new HashSet<>(A);
    // Testa operações
    System.out.println("A contém B? " + A.contains(B));
    A.addAll(B);
    System.out.println("A união B = " + A + " Size: " + A.size());
    A = new HashSet<>(copiaA); // como A foi alterado, recria A
    A.retainAll(B);
    System.out.println("A intersec B = " + A + " Size: " + A.size());
    A = new HashSet<>(copiaA); // como A foi alterado, recria A
    A.removeAll(B);
    System.out.println("A diferença B = " + A + " Size: " + A.size());
}
```

Interface List<E>

- Sequência de informações organizada conforme a ordem de inserção.
  - Permite acesso direto e aleatório.
  - Permite elementos duplicados.
- Diversas operações:
  - **get** e **remove**: obtêm ou removem um elemento.
  - **set** e **add**: especificam ou adicionam um elemento.
  - **addAll**: adiciona a coleção a partir de uma posição.
  - **indexOf** e **lastIndexOf**: determinam a primeira ou a última posição de um dado elemento.
  - **subList**: retorna um objeto List contendo os elementos existentes entre as posições dadas.

Interface List<E>

- Mais operações:
  - **iterator** e **listIterator**: retornam um objeto do tipo Iterator especializado na navegação em sequências.
- É possível utilizar algoritmos da classe Collections como sort (ordenar), shuffle (embaralhar) e *binarySearch* (busca binária).
- As coleções incluem duas implementações concretas: *ArrayList* e *LinkedList*.

Exemplo List<E>

```java
public static void main(String[] args) {
    // Criando uma Lista do tipo ArrayList
    List<String> aulas = new ArrayList<>();
    List<String> ultimas;

    aulas.add("Apresentação da disciplina");
    aulas.add("A linguagem Java");
    aulas.add("Estruturas de controle");
    aulas.add("Sintaxe e semântica");

    System.out.println(aulas);

    ultimas = aulas.subList(2, aulas.size());
    // For comum
    for (int i = 0; i < ultimas.size(); i++) {
        System.out.println("Ultimas: " + ultimas.get(i));
    }

    aulas.remove(0);
    // For especial para listas
    for (String aula : aulas) {
        System.out.println("Aula: " + aula);
    }
}
```

Interface Queue<E>

- Define uma coleção linear conhecida como Fila.
  - Estrutura geralmente usada para organizar elementos antes de seu processamento.
  - Filas podem ser usadas como estruturas FIFO (First In First Out), em que o primeiro elemento (head) é retirado antes dos demais.
- Principais operações:
  - **add**, **remove** e **element**: insere, retira e acessa o primeiro elemento da fila.
  - **offer**, **poll**, **peek**: similar as operações anteriores, porém retornando um valor especial em caso de falha.

Exemplo Queue<E>

```
public static void main(String[] args) {
    Queue<Integer> filaCaixa = new LinkedList<>();
    Scanner sc = new Scanner(System.in);
    int op;
    while (true) {
        System.out.println("Menu de Opções: ");
        System.out.println("1 - Adicionar pessoa na fila");
        System.out.println("2 - Atender cliente");
        System.out.println("0 - Sair");
        System.out.print("Escolha: ");
        op = sc.nextInt();
        if (op == 0) break;
        switch (op) {
            case 1: {
                System.out.println("Informe o código da pessoa: ");
                int cod = sc.nextInt();
                filaCaixa.add(cod);
                break;
            }
            case 2: {
                System.out.println("Atendendo: " + filaCaixa.remove());
                break;
            }
            default: {
                System.out.println("Opção inválida!");
                break;
            }
        }
    }
}
```

Interface Map<K,V>

- Permite a definição de uma lista de chave/valor.
  - Uma chave é um objeto que não possui duplicatas.
- As operações elementares dos mapas são:
  - **size**: retorna o número de associações (chave/valor).
  - **isEmpty**: verifica se está vazio.
  - **put**: insere uma nova associação (chave/valor).
  - **get**: obtém o valor de uma dada chave.
  - **remove**: elimina a associação indicada.
  - **containsKey** e **containsValue**: determina a existência da chave ou valor dado.
  - **clear** e **putAll**: limpa e inclui associações de um mapa.

Interface Map<K,V>

- As coleções incluem três concretas para Map<K,V>:
  - **HashMap**: uma tabela com indexação baseada em hashing.
  - **TreeMap**: uma árvore binária simétrica, cujos elementos são mantidos ordenados.
  - **LinkedHashMap**: uma tabela hashing combinada com uma lista ligada.
- O conjunto de entradas do HashMap (um Set de objetos Map.Entry) pode ser percorrido usando Iterator.

Exemplo Map<K,V>

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Map<String,Integer> hm = new HashMap<>();
    int cont = 0;
    String nome = "";
    while (!nome.equals("*")) {
        System.out.print("Digite um nome ('*' finaliza): ");
        nome = sc.nextLine();
        if (nome.equals("*")) continue;
        hm.put(nome, cont++);
        System.out.println(hm);
    }
    do {
        System.out.print("Digite um nome para excluir ('*' finaliza): ");
        nome = sc.nextLine();
        if (nome.equals("*")) continue;
        if (hm.containsKey(nome)) {
            hm.remove(nome);
            System.out.println("Nome removido.\n" + hm);
        }
        else {
            System.out.println("Nome não encontrado.\n" + hm);
        }
    } while (!nome.equals("*"));
    sc.close();
}
```

Considerações Finais

- Nesta aula estudamos as principais interfaces e classes concretas do Framework de Coleções do Java.
- Vimos exemplos com Sets, Lists, Queues e Maps.
  - Cada um destes containers tem características próprias, e podem ser aplicados para diferentes tipos de problemas.
- Cada interface possui diferentes implementações, que diferem principalmente nos mecanismos internos de implementação.
  - Para cada tarefa, uma das implementações pode ter desempenho superior.

Exercícios

1. Escreva um programa que leia números, reais ou inteiros, fornecidos pelo usuário, armazenando-os em uma implementação de Set<E>. Após a entrada de dados, use o Iterator<E> disponível para exibir todos os elementos fornecidos.
2. Escreva um programa que armazene nomes e telefones em um Map<K,V>. O programa também deve permitir a recuperação de um telefone a partir de um nome (pesquisa exata) ou todos os nomes e telefones cujos nomes começam com um prefixo dado.