Programação II

Aula 03

Evandro J.R. Silva[1]

[1] Bacharelado em Ciência da Computação
Estácio Teresina

Sumário

Introdução

- Aula baseada em Java Tutotials by Oracle:
  - Collections.

Collections

- Uma *coleção*, ou *container*, é um objeto que agrupa múltiplos objetos.
- Em Java, as *Collections* (ou Coleções) são utilizadas para armazenar, recuperar, manipular e permitir *comnunição* entre dados agregados.
- Nós já somos familiarizados com algumas classes, principalmente `ArrayList`.

Collections

- Uma *coleção*, ou *container*, é um objeto que agrupa múltiplos objetos.
- Em Java, as *Collections* (ou Coleções) são utilizadas para armazenar, recuperar, manipular e permitir *comnunição* entre dados agregados.
- Nós já somos familiarizados com algumas classes, principalmente `ArrayList`.
- O *collections framework* do Java é uma arquitetura unificada para representação e manipulação de coleções. Todos os *framewoks* contêm:

Collections

- Uma *coleção*, ou *container*, é um objeto que agrupa múltiplos objetos.
- Em Java, as *Collections* (ou Coleções) são utilizadas para armazenar, recuperar, manipular e permitir *comnunição* entre dados agregados.
- Nós já somos familiarizados com algumas classes, principalmente `ArrayList`.
- O *collections framework* do Java é uma arquitetura unificada para representação e manipulação de coleções. Todos os *framewoks* contêm:
  - **Interfaces**;
  - **Implementações** [das interfaces] — basicamente esrturura de dados reutilizáveis (lembra das classes genéricas?);
  - **Algoritmos** — os métodos (por exemplo: `sort` ou `search`).

Collections

- Alguns benefícios de utilizar Collections
  - **Reduz o esforço de programação**: ao prover estruturas de dados e algoritmos úteis, o Framework Collections permite ao desenvolvedor se concentrar nas partes mais importantes do programa em vez de lidar com as *infraestruturas* do programa. Ao facilitar a interoperabilidade entre APIs não relacionadas, o programador é dispensado de programar objetos para adaptação ou conversão de código para conectar APIs.

Collections

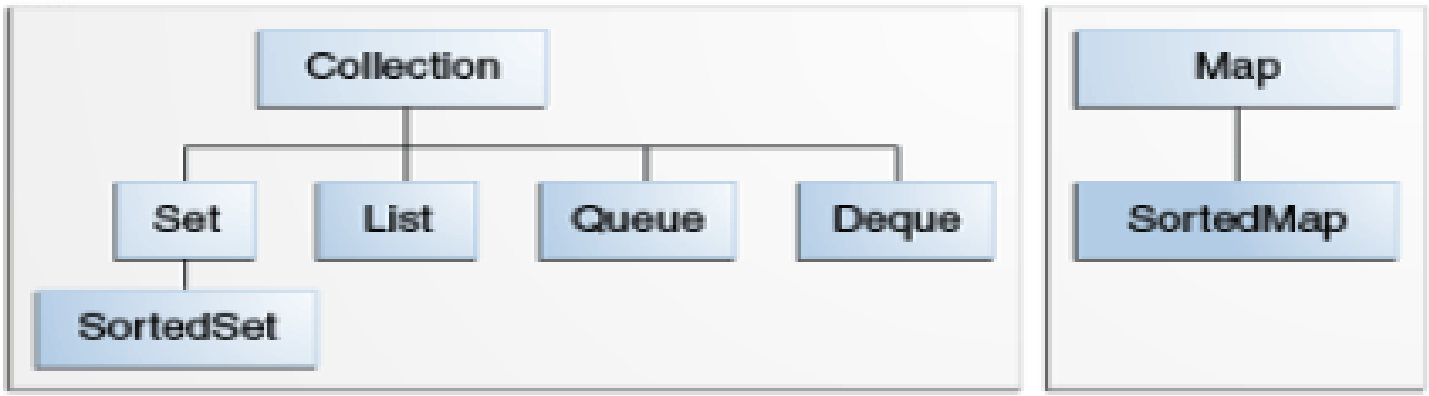
- Alguns benefícios de utilizar Collections
  - **Aumenta a velocidade e qualidade de programção**: o JFC provê implementações de algoritmos e estruturas de dados de alto desempenho e alta qualidade. As várias implementações de cada interface são intercambiáveis, portanto um programa pode ser facilmente aperfeiçoado ao se utilizar diferentes implementações de coleções. Uma vez que não há necessidade de desenvolver as estruturas de dados, o desenvolvedor terá mais tempo para melhorar a qualidade e o desempenho do seu programa.

Collections

- Alguns benefícios de utilizar Collections
  - **Reduz o esforço para desenvolver novas APIs**: os desenvolvedores não precisam reinventar a roda toda vez que forem criar uma API que utilize Collections. É possível utilizar as interfaces padrão.

Interfaces

- O núcleo de interfaces encapsula diferentes tipos de coleções, de forma que elas possam ser manipuladas independentemente dos detalhes de suas representações.



Interfaces

- `Collection`: raiz da hierarquia. É o denominador comum de todas as coleções implementadas, e pode ser utilizada como o mais genérico possível alguma coleção possa ser. Não existe uma implementação direta desta interface.

Interfaces

- `Collection`: raiz da hierarquia. É o denominador comum de todas as coleções implementadas, e pode ser utilizada como o mais genérico possível alguma coleção possa ser. Não existe uma implementação direta desta interface.
- `Set`: uma coleção que não pode conter elementos duplicados.

Interfaces

- `Collection`: raiz da hierarquia. É o denominador comum de todas as coleções implementadas, e pode ser utilizada como o mais genérico possível alguma coleção possa ser. Não existe uma implementação direta desta interface.
- `Set`: uma coleção que não pode conter elementos duplicados.
- `List`: uma coleção ordenada (também chamada de `sequência`). Pode conter elementos duplicados.

Interfaces

- `Collection`: raiz da hierarquia. É o denominador comum de todas as coleções implementadas, e pode ser utilizada como o mais genérico possível alguma coleção possa ser. Não existe uma implementação direta desta interface.
- `Set`: uma coleção que não pode conter elementos duplicados.
- `List`: uma coleção ordenada (também chamada de `sequência`). Pode conter elementos duplicados.
- `Queue`: uma coleção usada para armazenar elementos antes de seu processamento. Tradução direta: *fila*. Provê operações adicionais de inserção, extração e inspeção em relação à interfeace `Collection`.

Interfaces

- `Collection`: raiz da hierarquia. É o denominador comum de todas as coleções implementadas, e pode ser utilizada como o mais genérico possível alguma coleção possa ser. Não existe uma implementação direta desta interface.
- `Set`: uma coleção que não pode conter elementos duplicados.
- `List`: uma coleção ordenada (também chamada de `sequência`). Pode conter elementos duplicados.
- `Queue`: uma coleção usada para armazenar elementos antes de seu processamento. Tradução direta: *fila*. Provê operações adicionais de inserção, extração e inspeção em relação à interfeace `Collection`.
- `Deque`: similar a `Queue`. A diferença é que `Queue` segue por padrão o conceito FIFO (*First In First Out*), ou seja, todo novo elemento é inserido no fim, e toda remoção acotence com o primeiro elemento da coleção. `Deque` por sua vez usa tanto FIFO quanto LIFO (*Last In First Out*) e todos os elementos podem ser inseridos e removidos em ambas as pontas.

Interfaces

- `Map`: mapeia chaves a valores. Não pode haver chaves duplicadas, e cada chave deve mapear pelo menos um valor.
- `SortedSet`: um conjunto (`Set`) com ordenação ascendente.
- `SortedMap`: um `Map` com chaves ordenadas de forma ascendente.

Implementações

- As implementações são os objetos usados para armazenar as coleções, ou seja, no nosso contexto, são as implementações das interfaces de coleção.
- As principais implementações (ou as mais comuns) são divididas entre aquelas de **propósito geral** e **propósito especial**.
- Lista de implementações de propósito geral:
  - `Set`: `HashSet`, `TreeSet` e `LinkedHashSet`.
  - `List`: `ArrayList` e `LinkedList`.
  - `Queue`: `PriorityQueue` e `LinkedList`.
  - `Deque`: `ArrayDeque` e `LinkedList`.
  - `Map`: `HashMap`, `TreeMap` e `LinkedHashMap`.
- Lista de implementações de propósito especial:
  - `Set`: `EnumSet` e `CopyOnWriteArraySet`.
  - `List`: `CopyOnWriteArrayList`.
  - `Queue`: Não possui de propórito especial, mas possui implementações concorrentes: `LinkedBlockingQueue`, `ArrayBlockingQueue`, `PriorityBlockingQueue`, `DelayQueue` e `SynchronousQueue`.
  - `Deque`: Mesma situação de `Queue`, tendo a implementação concorrente `LinkedBlockingDeque`.
  - `Map`: `EnumMap`, `WeakHashMap` e `IdentityHashMap`.

Implementações

- Outros tipos de implementações
  - **Concorrentes**: são implementações projetadas para suportar alta concorrência, tipicamente ao custo de um desempenho de thread única. Essas implementações fazem parte do pacote `java.util.concurrent`.

Implementações

- Outros tipos de implementações
  - **Empacotamento**: implementações que são usadas em combinação com outras implementações, normalmente as de propósito geral, para prover funcionalidades novas ou restritas.

Implementações

- Outros tipos de implementações
  - **Conveniência**: são mini-implementações, tipicamente disponibilizadas via métodos `static factory`, os quais proveem alternativas convenientes e eficientes para as implementações de propósito geral, para coleções especiais (por exemplo, conjuntos unitários).

Implementações

- Outros tipos de implementações

  - **Abstratas**: são implementações *esqueleto*, as quais facilitam a construção de implementações personalizadas. É um tópico avançado, não é particularmente difícil, porém somente poucas pessoas precisarão delas.

Algoritmos

- Alguns métodos nativamente implementados para coleções:
  - `sort`: serve para ordenar uma lista;
  - `shuffle`: serve para embaralhar os elementos de uma lista.
  - `reverse`: reverte a ordem dos elementos.
  - `fill`: sobrescreve cada elemento de uma lista com um determinado valor.
  - `frequency`: conta a quantidade de vezes um elemento ocorre em uma coleção.

Exercícios

- Fazer os exercícios do site: https://www.w3resource.com/java-exercises/collection/index.php.

FIM

- Você pode encontrar mais sobre o conteúdo dessa aula no livro **Java: Como Programar** do Deitel, oitava edição, capítulos 20 e 21. Disponível nas bibliotecas virtuais (creio que na bilbioteca física também).

Até a próxima!