Coleções – Visão Geral

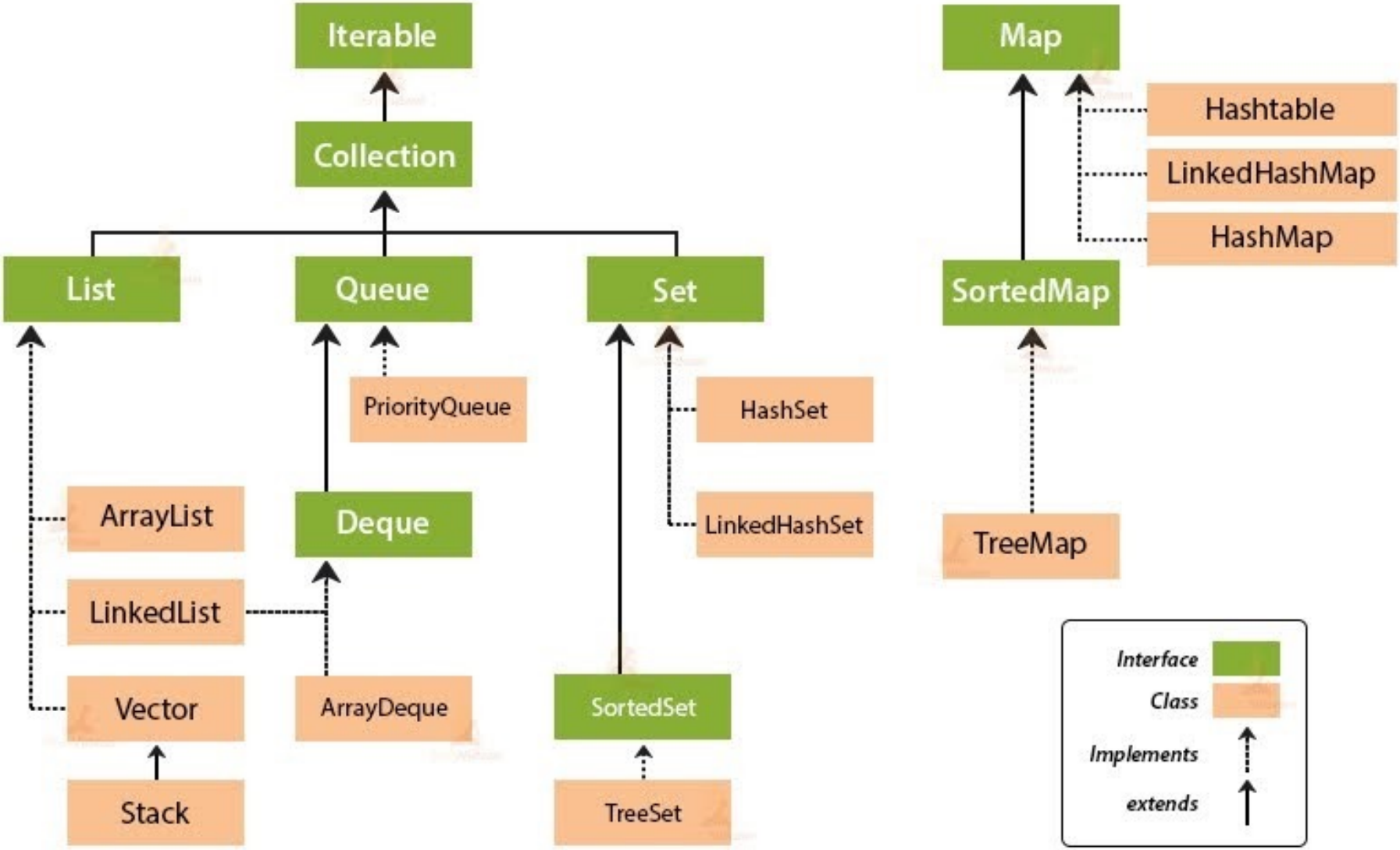
Profa. Karen Selbach Borges

Última Atualização: Jun/2021

The Collections Framework

- É uma arquitetura unificada para a representação e manipulação de coleções.
  - Uma coleção é um objeto que reúne múltiplos elementos (objetos) em uma única unidade.
  - As coleções são usadas para armazenar, recuperar, manipular e transmitir dados de um método para outro.
- Esta arquitetura é baseada numa hierarquia de interfaces e num conjunto de classes que implementam estas interfaces.
- A finalidade básica destas interfaces é permitir que as coleções sejam manipuladas independentemente dos detalhes de sua representação.

Collection Framework Hierarchy in Java



http://www.mauda.com.br/?p=2300

Framework

- As interfaces, e as classes que as implementam, foram projetadas de modo a serem o mais genéricas possíveis, ou seja, aceitarem qualquer tipo de objeto.
- Na documentação da API do framework, isso aparece na forma de:
  - <E> para elementos
  - <K> para chave
  - <V> para valor

Generics

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.util

Interface Collection<E>

Type Parameters:

`E - the type of elements in this collection`

All Superinterfaces:

`Iterable<E>`

All Known Subinterfaces:

`BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>`

Generics

- Até a versão 5 do JDK, ao trabalhar com coleções de objetos por exemplo, era possível misturar objetos em uma coleção.

```
public class TestaLista {
  public static void main(String[] argumentos) {
    List a = new ArrayList();
    a.add("um");
    a.add(new Integer(2));
    a.add(3);
  }
}
```

- Apesar de parecer um recurso bastante flexível, esse tipo de código tornava “quase impossível” recuperar os objetos contidos na coleção.

Generics

- Se caso algum erro de *cast* acontecesse, esse só seria visto em tempo de execução, quando seria emitido um “ClassCastException”.

```java
import java.util.*;
public class main
{
    // tip: arguments are passed via the field below this editor
  public static void main(String[] argumentos) {
    List a = new ArrayList();
    a.add("um");
    a.add(new Integer(2));
    a.add(3.0);
    Iterator it = a.iterator();
    while (it.hasNext()){
       String s = (String) it.next();
       System.out.println(s);
    }
  }
}
```

Execute Mode, Version, Inputs & Arguments

Execute

Result

CPU Time: 0.26 sec(s), Memory: 30500 kilobyte(s)compiled and executed

```
um

Exception in thread "main" java.lang.ClassCastException:
    at main.main(main.java:12)
```

Generics

- Assim, o código de exemplo deve receber a tipagem indicada em vermelho.

```
public class TestaLista {
  public static void main(String[] argumentos) {
    List<Integer> a = new ArrayList<Integer>();
    a.add(1);
    a.add(2);
    a.add(3);
    Integer n = a.get(1);
  }
}
```

Os valores são automaticamente encapsulados em um Integer

O cast, que era necessário para recuperar o objeto, desaparece.

BREVE DESCRIÇÃO DE CADA UMA DAS INTERFACES

Interface Collection

- Representa uma abstração para coleções de objetos.
- Fornece métodos para manipulação de grupos de objetos:
  - Adição de elementos
  - Remoção de elementos
  - Teste de continência
  - Listagem de todos os elementos (em ordem arbitrária)

Interface List

- Implementa um conjunto de objetos que permite objetos duplicados.
- Implementa uma seqüência de objetos: controle sobre a posição em que o objeto é inserido através de um índice
- Exemplo:
  - Conjunto de idades de um grupo de pessoas
  - Conjunto de datas de aniversários

Interface Queue

- Implementa uma fila, no esquema FIFO (*First-In First-Out*)
- Cada fila possui um elemento *head* que representa o primeiro da fila e é o alvo de operações como *peek* (recuperar sem remover) e *poll* (recuperar e remover)
- Exemplo:
  - Senhas para atendimento de caixa de banco

Interface Deque

- É uma estrutura similar à fila (Queue), mas que permite adicionar e remover por ambos os lados.
- Possui os elementos:
  - *head,* que representa o primeiro da fila
  - *tail*, que representa o último da fila
- Pode ser usado para representar um pilha: estrutura que insere e remove no fim da fila - LIFO (*Last-In First-Out*)

Interface Set

- Implementa coleções que não podem conter objetos duplicados.
- Exemplos :
  - conjunto de disciplinas cursadas por um aluno
  - conjunto de cartas em um jogo de poker

Interface SortedSet

- É uma especialização de Set
- Implementa um conjunto adicional de métodos para ordenar de forma ascendente os elementos.
- Exemplo de uso :
  - Lista telefônica

Interface Map

- Manipula grupos de objetos, onde cada objeto está associado a uma chave de busca.
- Não permite duas chaves iguais.
- Exemplo de uso :
  - Catálogo de produtos (cod x descrição)

Interface SortedMap

- É uma especialização de Map
- Implementa um conjunto adicional de métodos para organizar os elementos conforme a ordenação ascendente das chaves.
- Exemplo de uso :
  - Dicionário (palavra x significado)

Sobre a Ordenação

- Para que a ordenação seja possível é necessário que as classes, cujos objetos serão ordenados, implementem as interfaces Comparable ou Comparator.

Interface Comparable

- Pertence ao pacote java.lang
- Classes que implementam essa interface devem implementar o método compareTo, cuja lógica é a seguinte:
  - Considere x e y as informações a serem comparadas,
  - Se x<y então retorna um valor negativo
  - Se x>y então retorna um valor positivo
  - Se x==y então retorna zero.

Interface Comparable


```
public class Country implements Comparable<Country>{
    private int countryId;

    @Override
    public int compareTo(Country country) {
       if (this.countryId < country.countryId) return -1;
       else if  (this.countryId > country.countryId) return 1;
              else return 0 ;
    }
}
```

Interface Comparator

- Pertence ao pacote java.útil
- Classes que implementam essa interface devem implementar o método *compare*, cuja lógica é a mesma da interface Comparable.

Interface Comparator


```
public class CountrySortByIdComparator implements Comparator<Country>{

    @Override
    public int compare(Country country1, Country country2) {
      if (country1.getCountryId() < country2. getCountryId() ) return -1;
      else if  (country1.getCountryId() > country2. getCountryId()) return 1;
            else return 0 ;
    }
}
```

Comparable x Comparator

	Comparable (java.lang)	**Comparator (java.util)**
Arquitetura	A interface deve ser implementada pela classe cujos objetos serão ordenados. A ordenação deve seguir a lógica do sistema (por exemplo, ordenação de alunos pelo nome). Sistema fica mais engessado.	A interface é implementada por uma classe utilitária, responsável pela ordenação dos objetos. Essa ordenação pode ser realizada a partir de diferentes critérios (pelo id, pelo nome, pela idade, etc) Sistema fica mais flexível.
Implementação	int compareTo(Object o1) Compara this com o objeto o1 e retorna um inteiro, cujo valor será: positivo – this é maior que o1 zero – this é igual a o1 negativo – this é menor que o1	int compare(Object o1,Object o2) Compara os objetos o1 e o2 e retorna um inteiro, cujo valor será: positivo – o1 é maior que o2 zero – o1 é igual a o2 negativo – o1 é menor que o2