Análise das Estruturas de Dados em Java

Por: Breno Copeland Pitanga
Nyckoll Hayanne Santos

O que é Java?

Linguagem de Programação de alto nível.

Estruturas de Dados em Java

São definidas como uma coleção de dados que oferece um meio eficaz de armazenar e organizar dados em um computador.

ED’s do Colletion:

LinkedList; Deque; ArrayList; Vector; Stack; PriorityQueue; EnumSet; HashSet; LinkedHashSet; TreeSet; ArrayDeque.

ED’s do Map:

EnumMap; HashMap; LinkedHashMap; IdentityHashMap; TreeMap; WeakHashMap.

- java.lang.**Object**
  - java.util.**AbstractCollection**<E> (implements java.util.Collection<E>)
    - java.util.**AbstractList**<E> (implements java.util.List<E>)
      - java.util.**AbstractSequentialList**<E>
        - java.util.**LinkedList**<E> (implements java.lang.Cloneable, java.util.Deque<E>, java.util.List<E>, java.io.Serializable)
      - java.util.**ArrayList**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)
      - java.util.**Vector**<E> (implements java.lang.Cloneable, java.util.List<E>, java.util.RandomAccess, java.io.Serializable)
        - java.util.**Stack**<E>
    - java.util.**AbstractQueue**<E> (implements java.util.Queue<E>)
      - java.util.**PriorityQueue**<E> (implements java.io.Serializable)
    - java.util.**AbstractSet**<E> (implements java.util.Set<E>)
      - java.util.**EnumSet**<E> (implements java.lang.Cloneable, java.io.Serializable)
      - java.util.**HashSet**<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)
        - java.util.**LinkedHashSet**<E> (implements java.lang.Cloneable, java.io.Serializable, java.util.Set<E>)
      - java.util.**TreeSet**<E> (implements java.lang.Cloneable, java.util.NavigableSet<E>, java.io.Serializable)
    - java.util.**ArrayDeque**<E> (implements java.lang.Cloneable, java.util.Deque<E>, java.io.Serializable)
  - java.util.**AbstractMap**<K,V> (implements java.util.Map<K,V>)
    - java.util.**EnumMap**<K,V> (implements java.lang.Cloneable, java.io.Serializable)
    - java.util.**HashMap**<K,V> (implements java.lang.Cloneable, java.util.Map<K,V>, java.io.Serializable)
      - java.util.**LinkedHashMap**<K,V> (implements java.util.Map<K,V>)
    - java.util.**IdentityHashMap**<K,V> (implements java.lang.Cloneable, java.util.Map<K,V>, java.io.Serializable)
    - java.util.**TreeMap**<K,V> (implements java.lang.Cloneable, java.util.NavigableMap<K,V>, java.io.Serializable)
    - java.util.**WeakHashMap**<K,V> (implements java.util.Map<K,V>)

Heranças de LinkedList

Methods inherited from class java.util.AbstractSequentialList

`iterator`

Methods inherited from class java.util.AbstractList

`equals, hashCode, listIterator, removeRange, subList`

Methods inherited from class java.util.AbstractCollection

`containsAll, isEmpty, removeAll, retainAll, toString`

Methods inherited from class java.lang.Object

`finalize, getClass, notify, notifyAll, wait, wait, wait`

Methods inherited from interface java.util.List

`containsAll, equals, hashCode, isEmpty, iterator, listIterator, removeAll, retainAll, subList`

Methods inherited from interface java.util.Deque

`iterator`

Vector X ArrayList X LinkedList

Vantagens:

Vector:

- Tamanho Dinâmico
- Preserva a ordem dos elementos
- Tem suporte para valores nulos
- Sincronizado
- Por ser antigo é uma boa alternativa para sistemas antigos

ArrayList:

- Tamanho Dinâmico
- Fácil de usar
- Acesso rápido
- Preserva a ordem dos elementos
- Tem suporte para valores nulos

LinkedList:

- Tamanho Dinâmico
- Inserção e remoção eficientes
- Iteração flexível

Vector X ArrayList X LinkedList

Desvantagens:

Vector:

- Mais lento que por ser sincronizado
- Fora de um ambiente com múltiplas threads as outras classes são melhores

ArrayList:

- Mais lento que um array estático
- Mais pesado que um array estático
- Não é seguro quando trabalhado em múltiplas threads
- Perde performance com muitos elementos

LinkedList:

- Possui uma performance inferior à classe ArrayList no assunto de acessar um elemento específico.
- É mais pesado que um ArrayList

Vector X ArrayList X LinkedList

Vector:

Main.java

```java
import java.util.Vector;

class Main {
  public static void main(String[] args) {
    Vector lv = new Vector(3, 10);

    lv.add(null);
    lv.add(3);
    lv.add(5);
    System.out.println(lv.capacity());
    System.out.println(lv);

    lv.add(7);
    System.out.println(lv.capacity());
    System.out.println(lv);

    lv.remove(0);
    System.out.println(lv.capacity());
    System.out.println(lv);

    lv.clear();
    System.out.println(lv.capacity());
    System.out.println(lv);

    lv.trimToSize();
    System.out.println(lv.capacity());
    System.out.println(lv);
  }
}
```

Vector X ArrayList X LinkedList

ArrayList

```java
import java.util.ArrayList;

class Main {
  public static void main(String[] args) {
    ArrayList<Integer> la = new ArrayList();
    System.out.println(la);

    la.add(7);
    la.add(3);
    la.add(5);

    System.out.println(la);

    la.remove(0);

    System.out.println(la);

    la.clear();

    System.out.println(la);
  }
}
```

Vector X ArrayList X LinkedList

LinkedList

```java
import java.util.LinkedList;
import java.util.ArrayList;

class Main {
  public static void main(String[] args) {
    LinkedList ll = new LinkedList();
    ArrayList<String> la = new ArrayList();

    la.add("Queridos");
    la.add("Colegas");

    System.out.println(ll);

    ll.add("Boa");
    ll.add("Tarde");

    System.out.println(ll);

    ll.addAll(la);

    System.out.println(ll);

    ll.clear();

    System.out.println(ll);
  }
```

Stack

Algumas Operações:

empty(); peek(); pop(); push(); search().

Main.java

```java
import java.util.*;
import java.util.Stack;

class Main {
  public static void main(String[] args) {
    Stack <Integer> es= new Stack();

    System.out.println(es.empty());
    System.out.println(es);

    es.push(1);
    es.push(2);
    es.push(3);
    es.push(4);
    es.push(5);

    System.out.println(es);

    System.out.println(es.peek());

    System.out.println("Posicao do 1 \n" + es.search(1));
    Iterator t = es.iterator();

    while (t.hasNext())
    {
      System.out.println("\nTirando " + es.peek() + " temos:");
      es.pop();
      System.out.println(es);
    }

  }
}
```

PriorityQueue

Regras de Funcionamento:

1. Não permite termos nulos
2. Não pode ser gerada para objetos não comparáveis
3. Possuem um tamanho dinâmico
4. Não são sincronizadas

Construtores de Inicialização(7):

1. Criar com a capacidade inicial padrão (11).
2. Criar contendo os elementos da coleção especificada.
3. Criar com a capacidade inicial especificada.
4. Criar com a capacidade inicial especificada que ordena seus elementos de acordo com o comparador específico.

Construtores de Inicialização(7):

5. Criar contendo os elementos da fila de prioridade.

6. Criar contendo os elementos no conjunto classificado especificado.

7. Criar com capacidade inicial padrão e cujos elementos são ordenados de acordo com o comparador especificado.

HashSet

Algumas Características:

- A estrutura de dados subjacente para HashSet é Hashtable.
- Não é garantido que os objetos inseridos no HashSet sejam inseridos na mesma ordem.
- Os objetos são inseridos com base em seu código hash.
- Elementos NULL são permitidos em HashSet.

Declaration of HashSet

```
public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable,
Serializable
```

where **E** is the type of elements stored in a HashSet.

LinkedHashSet

Subclasse da anterior HashSet, com o adicional de uma lista duplamente encadeada ligando tudo por baixo a fim de manter uma ordem de entrada.

TreeSet

Trata-se de uma implementação da classe de Interface set fazendo uso de uma árvore balanceada.

LinkedHashSet

Constructor Summary

Constructors

Constructor	Description
`LinkedHashSet()`	Constructs a new, empty linked hash set with the default initial capacity (16) and load factor (0.75).
`LinkedHashSet(int initialCapacity)`	Constructs a new, empty linked hash set with the specified initial capacity and the default load factor (0.75).
`LinkedHashSet(int initialCapacity, float loadFactor)`	Constructs a new, empty linked hash set with the specified initial capacity and load factor.
`LinkedHashSet(Collection<? extends E> c)`	Constructs a new linked hash set with the same elements as the specified collection.

TreeSet

Constructor Summary

Constructors

Constructor	Description
`TreeSet()`	Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
`TreeSet(Collection<? extends E> c)`	Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.
`TreeSet(Comparator<? super E> comparator)`	Constructs a new, empty tree set, sorted according to the specified comparator.
`TreeSet(SortedSet<E> s)`	Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.

ArrayDeque

Constructor Summary

Constructors

Constructor	Description
`ArrayDeque()`	Constructs an empty array deque with an initial capacity sufficient to hold 16 elements.
`ArrayDeque(int numElements)`	Constructs an empty array deque with an initial capacity sufficient to hold the specified number of elements.
`ArrayDeque(Collection<? extends E> c)`	Constructs a deque containing the elements of the specified collection, in the order they are returned by the collection's iterator.

- java.util.**AbstractMap**<K,V> (implements java.util.Map<K,V>)
  - java.util.**EnumMap**<K,V> (implements java.lang.Cloneable, java.io.Serializable)
  - java.util.**HashMap**<K,V> (implements java.lang.Cloneable, java.util.Map<K,V>, java.io.Serializable)
    - java.util.**LinkedHashMap**<K,V> (implements java.util.Map<K,V>)
  - java.util.**IdentityHashMap**<K,V> (implements java.lang.Cloneable, java.util.Map<K,V>, java.io.Serializable)
  - java.util.**TreeMap**<K,V> (implements java.lang.Cloneable, java.util.NavigableMap<K,V>, java.io.Serializable)
  - java.util.**WeakHashMap**<K,V> (implements java.util.Map<K,V>)

Map Interface

Trata-se de uma classe de interface, onde como características temos a associação entre elemento e chave.

EnumMap

- EnumMap é uma coleção ordenada;
- São mantidos na ordem natural de suas chaves;
- é uma implementação Map de alto desempenho, mais rápida que o HashMap;
- EnumMap é representado internamente como arrays.

EnumMap

Constructor Summary

Constructors

Constructor	Description
`EnumMap(Class<K> keyType)`	Creates an empty enum map with the specified key type.
`EnumMap(EnumMap<K,? extends V> m)`	Creates an enum map with the same key type as the specified enum map, initially containing the same mappings (if any).
`EnumMap(Map<K,? extends V> m)`	Creates an enum map initialized from the specified map.

HHashMap X IdentityHashMap X LinkedHashMap

Constructor Summary

Constructors

Constructor	Description
`HashMap()`	Constructs an empty `HashMap` with the default initial capacity (16) and the default load factor (0.75).
`HashMap(int initialCapacity)`	Constructs an empty `HashMap` with the specified initial capacity and the default load factor (0.75).
`HashMap(int initialCapacity, float loadFactor)`	Constructs an empty `HashMap` with the specified initial capacity and load factor.
`HashMap(Map<? extends K,? extends V> m)`	Constructs a new `HashMap` with the same mappings as the specified `Map`.

Construtor HashMap

HHashMap X IdentityHashMap X LinkedHashMap

Constructor Summary

Constructors

Constructor	Description
`IdentityHashMap()`	Constructs a new, empty identity hash map with a default expected maximum size (21).
`IdentityHashMap(int expectedMaxSize)`	Constructs a new, empty map with the specified expected maximum size.
`IdentityHashMap(Map<? extends K,? extends V> m)`	Constructs a new identity hash map containing the keys-value mappings in the specified map.

Construtor IdentityHashMap

LiHashMap X IdentityHashMap X LinkedHashMap

Constructor Summary

Constructors

Constructor	Description
`LinkedHashMap()`	Constructs an empty insertion-ordered `LinkedHashMap` instance with the default initial capacity (16) and load factor (0.75).
`LinkedHashMap(int initialCapacity)`	Constructs an empty insertion-ordered `LinkedHashMap` instance with the specified initial capacity and a default load factor (0.75).
`LinkedHashMap(int initialCapacity, float loadFactor)`	Constructs an empty insertion-ordered `LinkedHashMap` instance with the specified initial capacity and load factor.
`LinkedHashMap(int initialCapacity, float loadFactor, boolean accessOrder)`	Constructs an empty `LinkedHashMap` instance with the specified initial capacity, load factor and ordering mode.
`LinkedHashMap(Map<? extends K,? extends V> m)`	Constructs an insertion-ordered `LinkedHashMap` instance with the same mappings as the specified map.

Construtor LinkedHashMap

HashMap X IdentityHashMap X LinkedHashMap

Main.java

```java
import java.util.HashMap;

class Main {

  public static void main(String[] args) {

    HashMap <String, Integer> hm = new HashMap(12, 0.8f);

    hm.put("Alberto", 12);
    hm.put("JOnas", 16);
    hm.put("abacate", 22);
    hm.put("16", 33);
    hm.put("Feijao", 47);
    hm.put("cacildes", 90);

    System.out.println(hm);
    // {cacildes=90, JOnas=16, abacate=22, 16=33, Alberto=12, Feijao=47}

    System.out.println(hm.get("16"));

    hm.clear();
    System.out.println(hm);
  }
}
```

```java
import java.util.IdentityHashMap;

class Main {

  public static void main(String[] args) {

    IdentityHashMap <String, Integer> ihm = new IdentityHashMap(10);

    ihm.put("Alberto", 12);
    ihm.put("JOnas", 16);
    ihm.put("abacate", 22);
    ihm.put("16", 33);
    ihm.put("Feijao", 47);
    ihm.put("cacildes", 90);

    System.out.println(ihm);
    // {cacildes=90, JOnas=16, abacate=22, 16=33, Alberto=12, Feijao=47}

    System.out.println(ihm.get("16"));

    ihm.clear();
    System.out.println(ihm);
  }
}
```

LinkedHashMap

Alguns recursos importantes de uma classe LinkedHashMap são:

- Um LinkedHashMap contém valores baseados na chave.
- pode ter uma chave nula e vários valores nulos;
- não é sincronizada;
- é como o HashMap porém pode manter o pedido de inserção.

TreeMap

- TreeMap em Java não permite chaves nulas (como Map) e, portanto;
- os pares de entradas retornados pelos métodos nesta classe e suas visualizações representam instantâneos do mapeamento no momento em que foram produzidos.

TreeMap

Main.java

```java
import java.util.TreeMap;

class Main {

  public static void main(String[] args) {

    TreeMap <String, Integer> tm = new TreeMap();

    tm.put("Alberto", 12);
    tm.put("JOnas", 16);
    tm.put("abacate", 22);
    tm.put("16", 33);
    tm.put("Feijao", 47);
    tm.put("cacildes", 90);

    System.out.println(tm);
    //{16=33, Alberto=12, Feijao=47, JOnas=16, abacate=22, cacildes=90}

    System.out.println(tm.get("16"));

    tm.clear();
    System.out.println(tm);
  }
}
```

WeakHashMap

Alguns recursos importantes de uma classe WeakHashMap são:

- Valores nulos e chaves nulas são suportados no WeakHashMap;
- não está sincronizado;
- A classe destina-se principalmente ao uso com objetos-chave cujos métodos equals testam a identidade do objeto usando o operador .

Bibliografia

PARVEZ, Faizan. Data Structures in Java – A Beginners Guide 2023
Great Learning, 14 de agosto de 2023.

Oracle. PriorityQueue(Java Platform SE 7). Oracle. 2020.

Oracle. ArrayDeque(Java Platform SE 7). Oracle. 2020.

Silva, Pepson. Estrutura de Dados em Java, site: mio.me. 03 de Abril de 2023.

Silveira, Paulo. Cosentino, Rafael. Algoritmos e Estrutura de Dados em Java.

Carlos. Java Collection- Como utilizar Collection. 2010.

Avelino. Paralelismo- O que é programação paralela. Site: StackOverFlow. 2015

Bibliografia

Avramovic, Slaviša . Guia para a palavra-chave: sincronizada em Java. 10 de agosto de 2023;

Srishilesh,PS. Introdução às estruturas de dados integradas em Java. 3 de fevereiro de 2021.

https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450

https://pt.stackoverflow.com/questions/86484/programar-voltado-para-interface-e-n%C3%A3o-para-a-implementa%C3%A7%C3%A3o-por-qu%C3%AA

https://pt.stackoverflow.com/questions/55798/o-que-%C3%A9-programa%C3%A7%C3%A3o-paralela

https://www.techtarget.com/whatis/definition/multithreading

Bibliografia

https://pt.stackoverflow.com/questions/202100/qual-a-diferen%c3%a7a-entre-abstra%c3%a7%c3%a3o-e-implementa%c3%a7%c3%a3o?noredirect=1&lq=1

https://pt.stackoverflow.com/questions/86484/programar-voltado-para-interface-e-n%C3%A3o-para-a-implementa%C3%A7%C3%A3o-por-qu%C3%AA

https://www.section.io/engineering-education/introduction-to-built-in-data-structures-in-java/#further-reading

https://www.devmedia.com.br/java-interface-aprenda-a-usar-corretamente/28798

https://www.baeldung.com/java-synchronized

https://www.mygreatlearning.com/blog/data-structures-using-java/#linked-list

Fim!