Collections Framewok

O êxito não se consegue só com qualidades especiais. É sobretudo um trabalho de constância, de método e de organização

LISTA

LISTA

Conjunto de objetos que são acessiveis pela sua posição em uma listagem.

FIFO

PEPS

FCFS

FILA

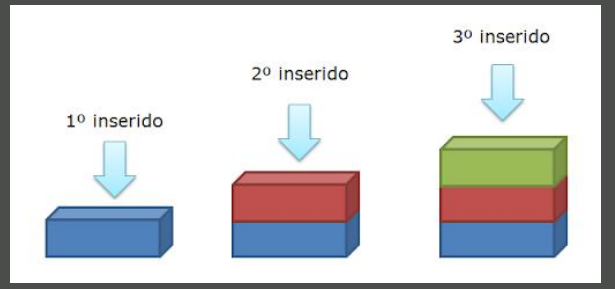
Em Ciência da Computação, algoritmo de fila simples:

FIFO (do inglês: first in, first out , "primeiro a entrar, primeiro a sair", "PEPS") ou FCFS (do inglês: first come, first served , "primeiro a chegar, primeiro a ser servido") é um algoritmo de escalonamento para estruturas de dados do tipo fila.

Apresenta o seguinte critério: o primeiro elemento a ser retirado é o primeiro que tiver sido inserido, é um algoritmo de escalonamento não preemptivo que entrega a CPU os processos pela ordem.

LIFO

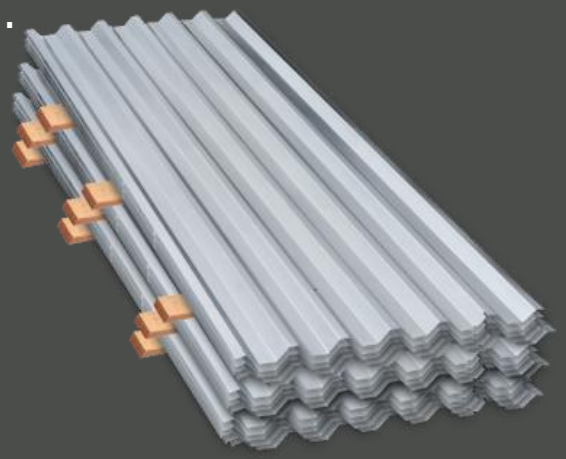
PILHA



Em ciência da computação:

LIFO (acrônimo para a expressão inglesa Last In, First Out que, em português significa último a entrar, primeiro a sair)

refere-se a estruturas de dados do tipo pilha.
É equivalente a FILO, que significa First In, Last Out .



O conceito de pilha é amplamente utilizado na informática, como, por exemplo, durante a execução de um programa, para o armazenamento de valores de variável local a um bloco e também para conter o endereço de retorno do trecho de programa que chamo.

LISTA

ENCADEADA

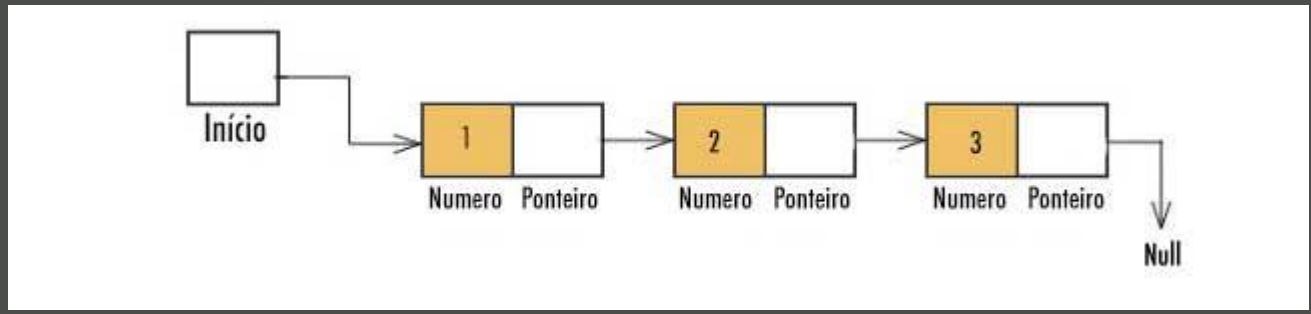
LISTA ENCADEADA OU LIGADA

Uma lista ligada ou lista encadeada é uma estrutura de dados linear e dinâmica.

Ela é composta por células que apontam para o próximo elemento da lista.

Para "ter" uma lista ligada/encadeada, basta guardar seu primeiro elemento, e seu último elemento aponta para uma célula nula.

O esquema a seguir representa uma lista ligada/encadeada com 5 elementos:



Para inserir dados ou remover dados é necessário ter um ponteiro que aponte para o 1º elemento e outro que aponte para o fim, porque se queremos

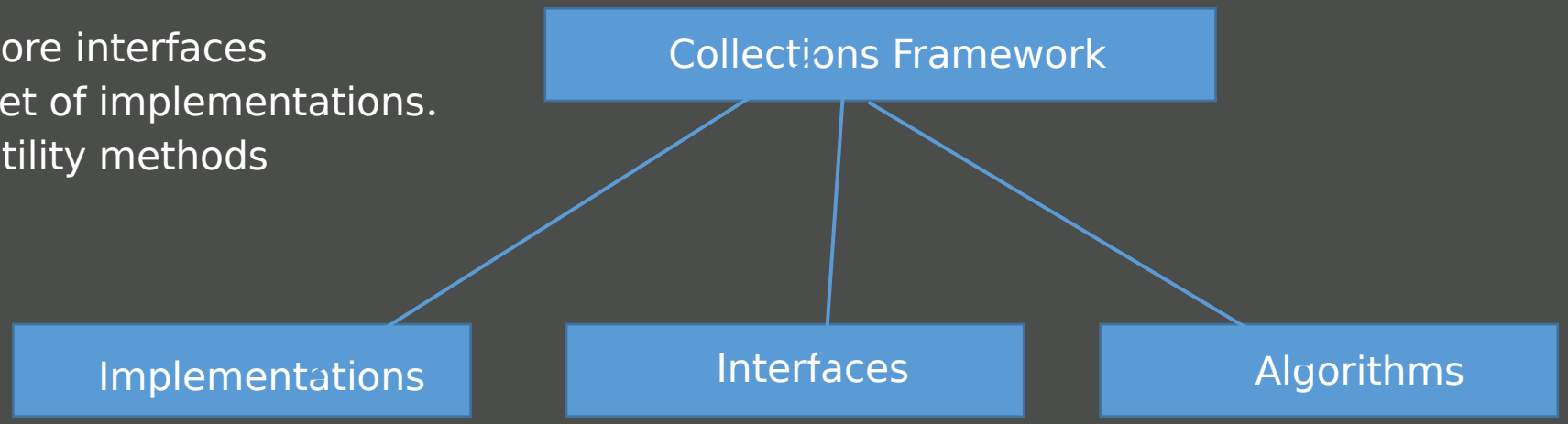
Definição

É um objeto onde podemos agrupar vários elementos (outros objetos)

Métodos implementados que realizam operações(sort, reverse, isEmpty, size ...) sobre as coleções

Framework é provido da java.util package sendo dividido em 3 partes:

1. Core interfaces
2. Set of implementations.
3. Utility methods

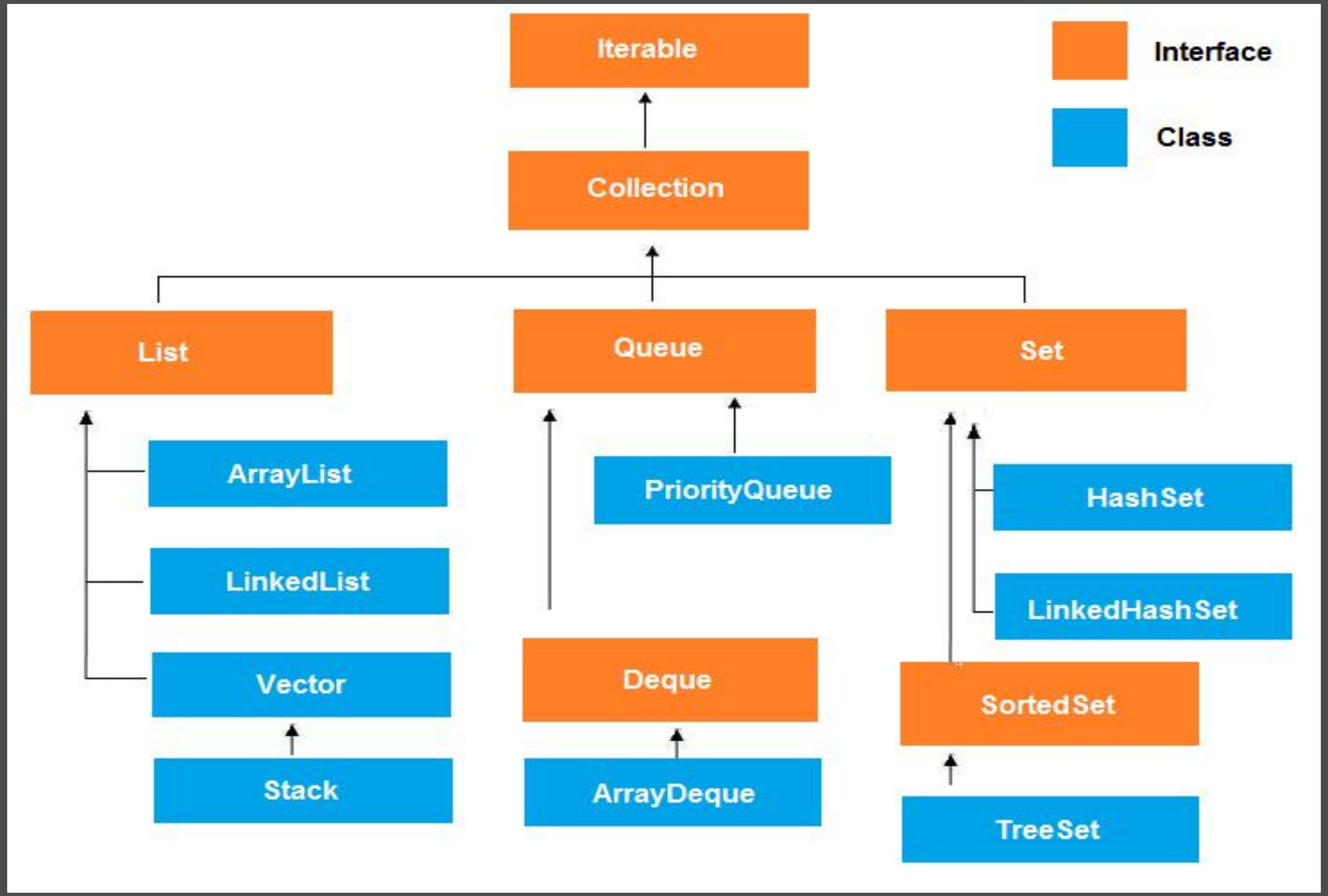


key

Word


Properties
WeakHashMap
TreeSet
SortedSet
Stack
LinkedHashSet
TreeMap
IdentityHashMap
List
Vector
Map
ArrayList
HashMap
SortedMap
Hashtable
LinkedHashMap
Set
HashSet
Collection
LinkedList

Hierarquia


Iterable
Interface
Class
Collection
List
Queue
Set
ArrayList
PriorityQueue
HashSet
LinkedList
LinkedHashSet
Vector
Deque
SortedSet
Stack
ArrayDeque
TreeSet

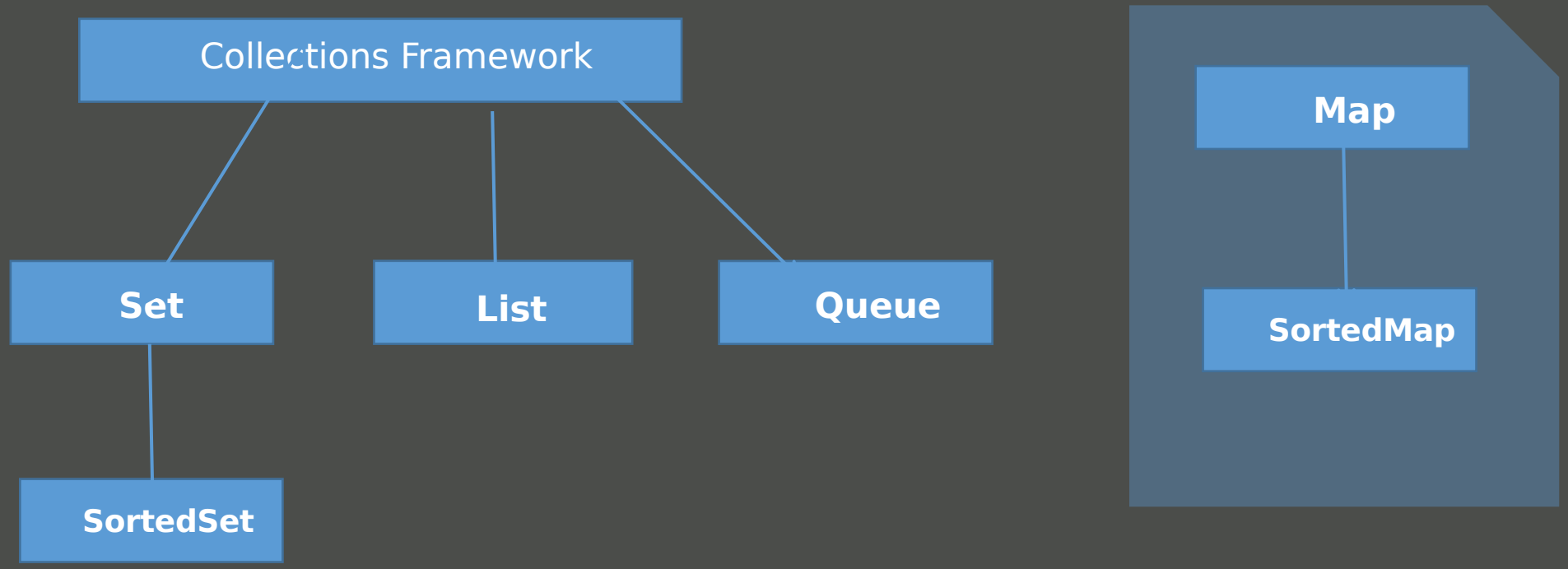
Framework

Collections

...is implemented as a series of hierarchies
with
interfaces at the top
abstract classes in the middle
and fully defined classes at the bottom

More than 200 methods in all!

1. Core interfaces are:



Review

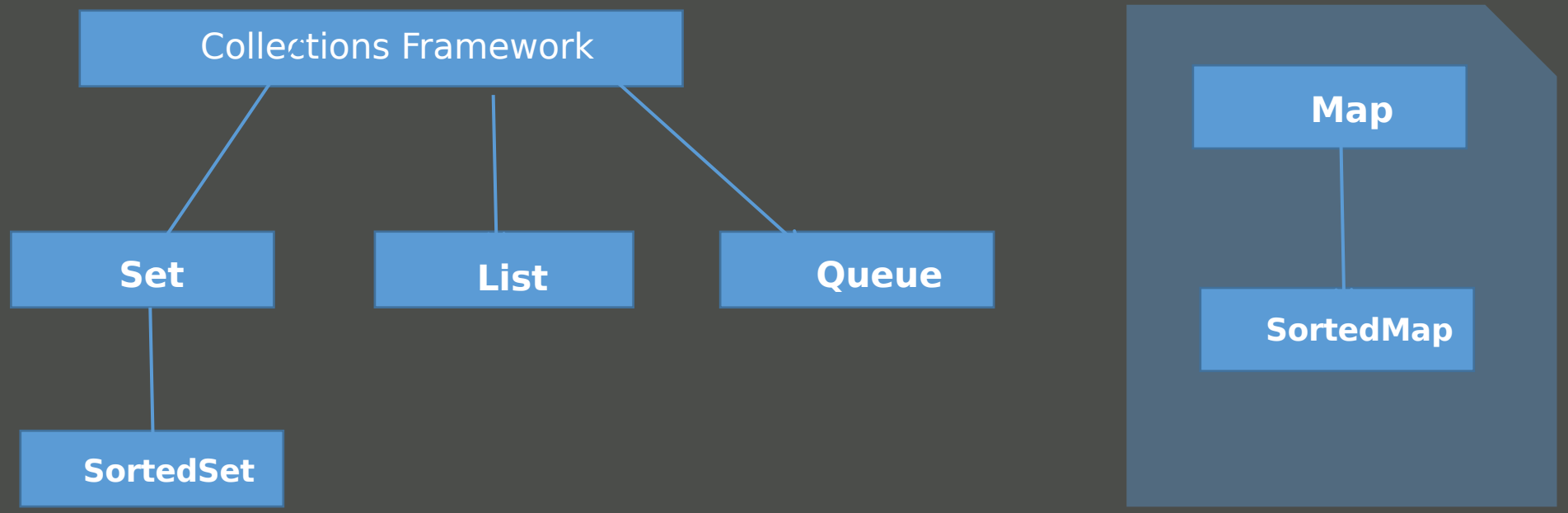
What is an interface?

Uma interface na linguagem de programação Java é um tipo abstrato que é usado para especificar um comportamento que as classes devem implementar. Eles são semelhantes aos protocolos.

As interfaces são declaradas usando a palavra-chave da interface e podem conter apenas assinatura de método e declarações constantes.

Framework

Collections



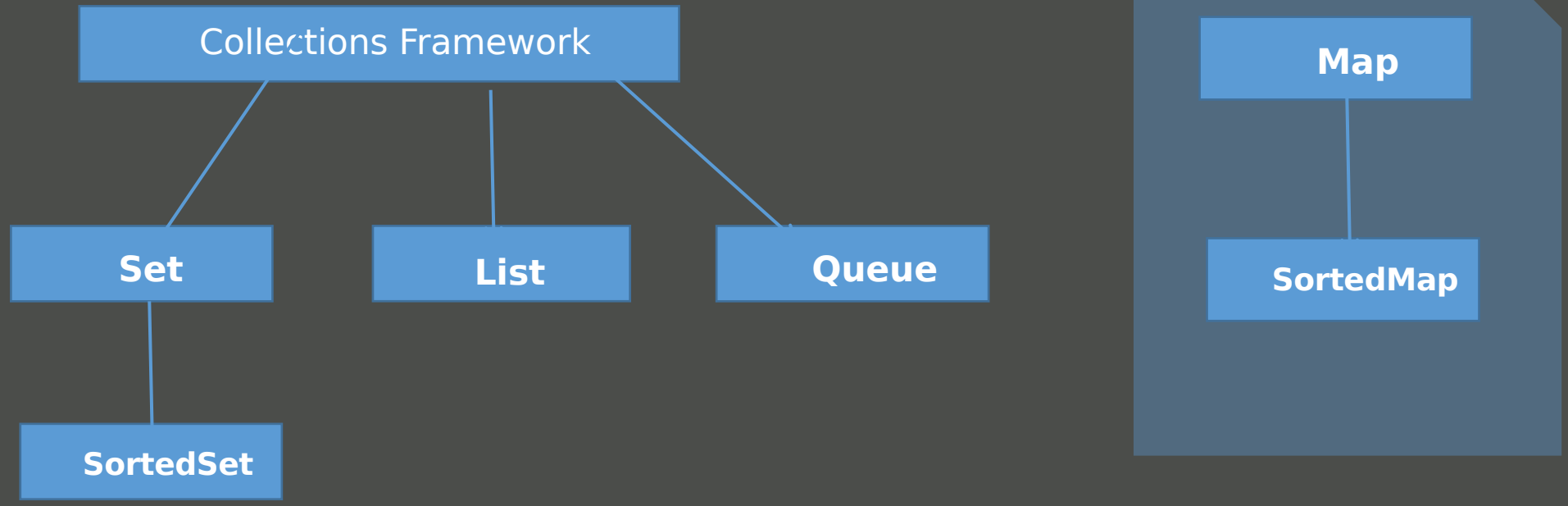
A interface Collection define certos comportamentos básicos, como (para citar apenas alguns):

- **add**(e) – allows an element to be added to the collection
- **clear**() – removes all elements from the collection
- **contains**(e) – determines if an element is in the collection
- **isEmpty**() – determines if the collection is empty
- **remove**(e) – removes a specific element from the collection
- **size**() – gets the number of elements in the collection

Nota: A coleção não define uma maneira de recuperar um elemento de uma coleção

Hierarquia

The List interface declares methods for inserting, removing and retrieving an element at a certain location in a collection



List – an ordered collection, or a sequence that defines specific control over **where** elements are placed in the collection

Set – a collection that *does not allow duplicate* items
SortedSet – self explanatory

Queue – an ordered collection supporting a specific way of adding and removing elements from a collection

List

A interface List define alguns métodos orientados a índices, como:

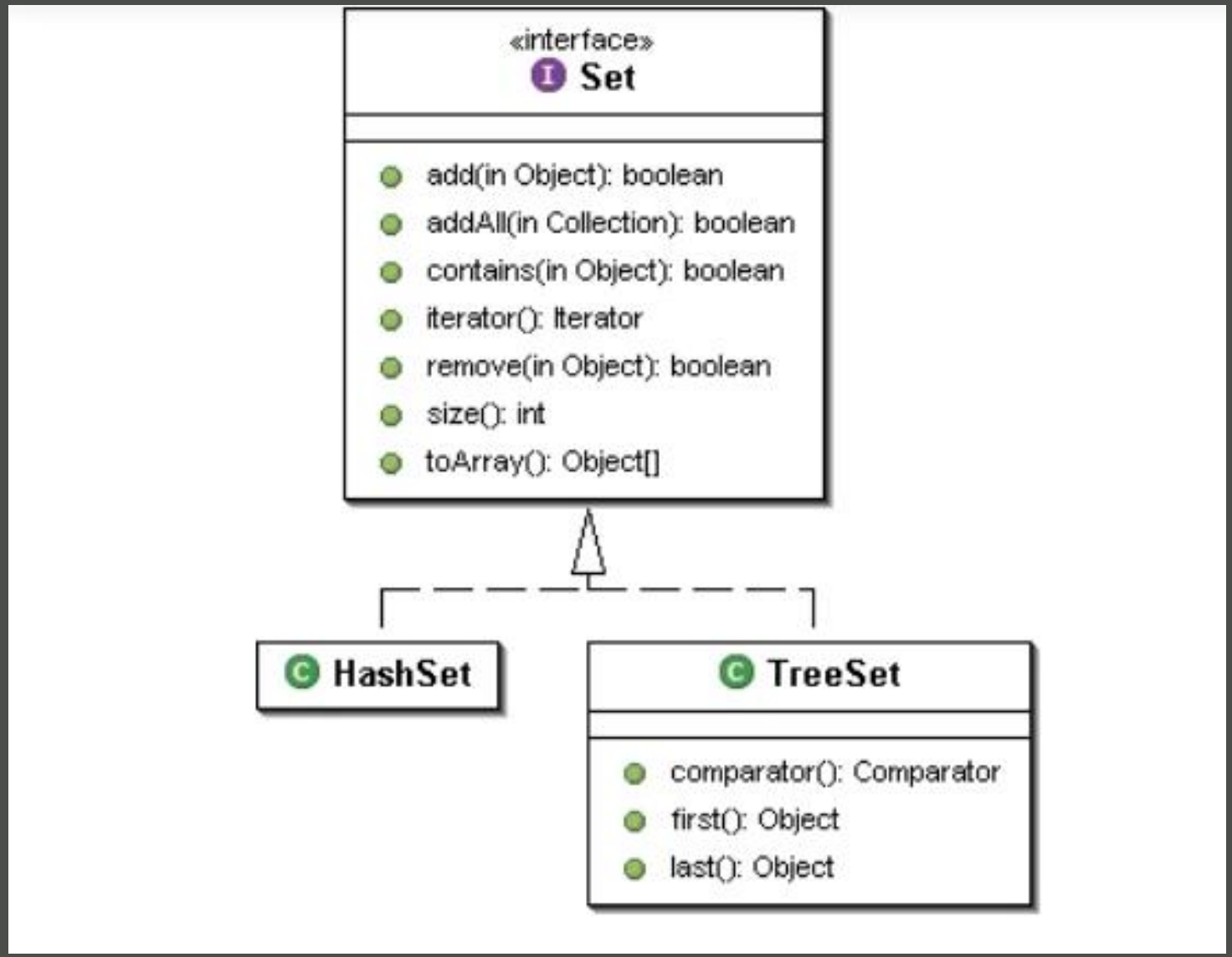
add (index, e) - adiciona um elemento em um local / índice específico na lista

get (index) - recupera o elemento no local / índice específico

remove (index) - remove o elemento no local / índice específico

set (index, e) - substitua o elemento no local / índice específico

Set


«interface»
Set
add(in Object): boolean
addAll(in Collection): boolean
contains(in Object): boolean
iterator(): Iterator
remove(in Object): boolean
size(): int
toArray(): Object[]
HashSet
TreeSet
comparator(): Comparator
first(): Object
last(): Object


Java

Queue

Métodos de interface da fila

add (E e): boolean esse método adiciona o elemento especificado no final da fila. Retorna true se o elemento for adicionado com êxito ou false se o elemento não for adicionado. Isso acontece basicamente quando a Fila está em sua capacidade máxima e não pode receber mais elementos.

E Elemento (): esse método retorna a cabeça (o primeiro elemento) da fila.

boolean offer (E e): é o mesmo que o método add ().

E remove (): este método remove a cabeça (primeiro elemento) da Fila e retorna seu valor.

E poll (): esse método é quase o mesmo que o método remove (). A única diferença entre poll () e remove () é que o método poll () retorna nulo se a fila estiver vazia.

E peek (): esse método é quase o mesmo que o método element (). A única diferença entre peek () e element () é que o método peek () retorna nulo se a fila estiver vazia.

Stack

O Java Stack estende a classe Vector apenas com as cinco operações a seguir.

boolean empty (): testa se esta pilha está vazia.

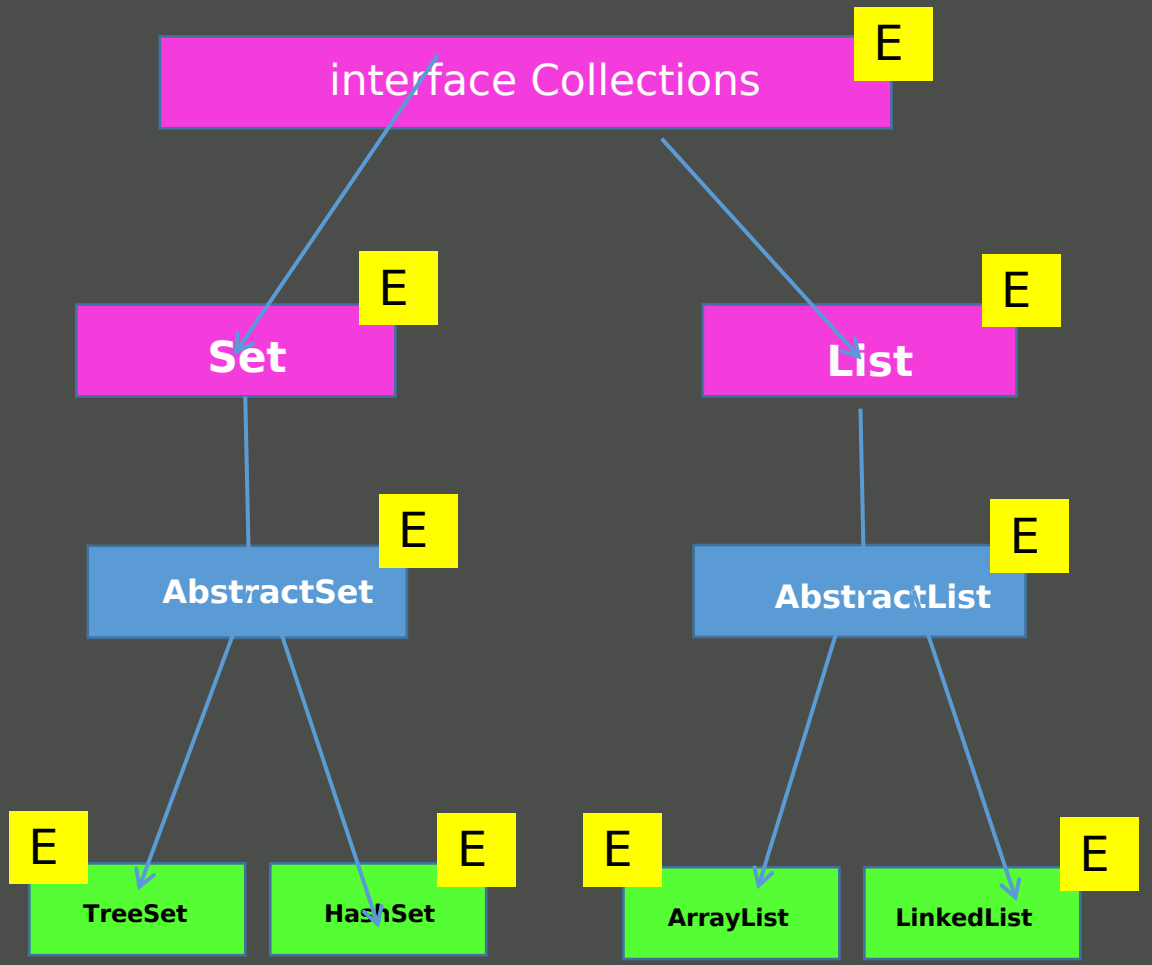
E peek (): examina o objeto no topo desta pilha sem removê-lo da pilha.

E pop (): remove o objeto no topo desta pilha e retorna esse objeto como o valor dessa função.

E push (item E): Empurra um item para o topo desta pilha.

int search (Object o): Retorna a posição baseada em 1 em que um objeto está nesta pilha.

Hierarquia


interface Collections
E
Set
E
List
E
AbstractSet
E
AbstractList
E
TreeSet
E
HashSet
E
ArrayList
E
LinkedList
E

Hierarquia

As Classes Abstratas JCF residem entre interfaces e classes de coleção

- Tem métodos indefinidos (como uma interface)
- bem como métodos definidos (como uma classe regular).
- O que uma classe abstrata fornece que uma interface não?
- Definições simples de métodos comuns que não precisam ser substituídas nas subclasses totalmente definidas.
- Um exemplo é que uma subclasse de AbstractList não precisa substituir os métodos isEmpty () ou size ().

Hierarquia

- Na parte inferior da hierarquia JCF estão as classes de coleção totalmente definidas (ou contêineres)
- Um contêiner é outro nome para uma classe cujas instâncias são coleções (de elementos)
- As classes de coleção implementam as interfaces no nível mais baixo
- O JCF fornece (a maioria) contêineres comuns com iteradores e uma variedade de algoritmos.

Interface	General-purpose Implementations				
	Hash Table	**Resizable array**	Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

ArrayList

JCF Collection class: ArrayList

```
List<Double> salaryList = new
ArrayList<Double>();
salaryList.add(1000);
salaryList.add(0,2000); // add before 1000
salaryList.remove(1); // remove 2nd element
salaryList.clear(1000); // clear list
```

- Isso cria uma instância, employeeList, da classe de coleção ArrayList <>. Os elementos em employeeList devem ser (referências a) Objetos duplos.
- O tamanho não precisa ser especificado

Hierarquia

JCF Collection class: LinkedList

```
List<Double> salaryList = new
LinkedList<Double>();
salaryList.add(1000);
salaryList.add(0,2000); // add before 1000
salaryList.remove(1); // remove 2nd element
salaryList.clear(1000); // clear list
```

- Isso cria uma instância, employeeList, da classe de coleção LinkedList <>. Os elementos em employeeList devem ser (referências a) Objetos duplos.

List

Então, qual é a diferença entre ArrayList e LinkedList?

LinkedList é mais rápido para inserir e retirar e sua iteração é mais lenta.
LinkedList é bom pra implementar stack ou queue.

Pelo próprio algoritmo de uma LinkedList você vê que ela é mais lenta pra percor
imagina você tá no meio da lista e quer ir pro inicio ou fim.

Já o ArrayList é mais rápido para iteração e acesso randomico.
Você fornece o index e já tem acesso ao valor.
Mas para adicionar um item ou remover, só a manipulação de memória j
á deixa mais lento.

Um Vector é igual um ArrayList, só que é sincronizado para uso com threads.

ArrayList

Este tipo de lista é implementado como um Array que é dimensionado d inamicamente, ou seja, sempre que é necessário o seu tamanho aumenta em 50% do tamanho da lista, significa que se você tiver uma lista de tamanho igual a 10 e ela “encher”, seu tamanho aumentará para 15 automaticamente.

Além disso a ArrayList permite que elementos sejam acessados

diretamente pelos métodos get() e set(), e adicionados através de add() e remove().

ArrayList

```
ArrayList al = new ArrayList();
        al.add(3);
        al.add(2);
        al.add(1);
        al.add(4);
        al.add(5);
        al.add(6);
        al.add(6);

        Iterator iter1 = al.iterator();
        while(iter1.hasNext()){
                System.out.println(iter1.next());
        }

        System.out.println(al.get(2));
```

O que você vai perceber no código acima é que o ArrayList não remove os elementos duplicados, e ainda podemos acessar qualquer elemento diretamente através do seu index, mas tudo tem um custo e veremos mais adiante.

O custo ArrayList

Todo ArrayList começa com um tamanho fixo, que vai aumentando conforme necessário, mas o custo deste aumento é alto, pois é feita uma cópia do array atual para um novo array com um novo tamanho, então imagine um array com 10mil elementos que será copiado para um novo array para criação de mais 5 mil elementos ?

De fato é um alto custo. Então é altamente aconselhável que você já inicie seu Array com uma quantidade de elementos que atenda ao seu objetivo atual, sem a necessidade de criação dinâmica de novos espaços, ou seja, se você saber que terá que armazenar de 300 a 400 objetos em um Array, defina 500,

pois é melhor sobrar espaço do que utilizar recurso do processador sem necessidade.

Note: Estes não são sincronizados, consequentemente não são thread-safe, ou seja, se sua aplicação precisa trabalhar com thread-safe em determinado ponto onde uma Lista é necessária, então descarte ArrayList

Vector

Do ponto de vista da API, ou seja, da forma como é utilizado, o Vector e o ArayList são muito similares, podemos arriscar até em dizer: iguais. Se você não conhece a fundo o conceito de Vector e ArrayList usará ambos como se fossem o mesmo, sem sentir nenhuma diferença

```
Vector al = new Vector();
        al.add(3);
        al.add(2);
        al.add(1);
        al.add(4);
        al.add(5);
        al.add(6);
        al.add(6);

        Iterator iter1 = al.iterator();
        while(iter1.hasNext()){
                System.out.println(iter1.next());
        }

        System.out.println(al.get(2));
```

Vector

Vector ser sincronizado e o ArrayList não. Significa dizer que se você possui uma aplicação que precisa ser thread-safe em determinado ponto, use Vector e você estará garantido.

Outro ponto importante é a alocação dinâmica do Vector, que é diferente do ArrayList. Lembra que falamos que o ArrayList aumenta 50% do seu tamanho quando a lista está cheia ? O Vector aumenta o dobro, ou seja, se você tem uma lista de 10 elementos cheia, essa lista aumentará para 20, com 10 posições vazias.

LinkedList

```
LinkedList ll = new LinkedList();
        ll.add(3);
        ll.add(2);
        ll.add(1);
        ll.add(4);
        ll.add(5);
        ll.add(6);
        ll.add(6);

        Iterator iter2 = ll.iterator();
        while(iter2.hasNext()){
            System.out.println(iter2.next());
        }
```

LinkedList

Este tipo de lista implementa uma “double linked list”, ou seja, uma lista duplamente “linkada”. A sua principal diferença entre o ArrayList é na performance entre os métodos add, remove, get e set.

Este tipo de lista possui melhor performance nos métodos add e remove, do que os métodos add e remove do ArrayList, em compensação seus métodos get e set possuem uma performance pior do que os do ArrayList. Vamos abaixo fazer uma comparação entre a complexidade apresentada de cada método do ArrayList e o da LinkedList.

get(int index): LinkedList possui O(n) e ArrayList possui O(1)

add(E element): LinkedList possui O(1) e ArrayList possui O(n) no pior caso, visto que o array será redimensionado e copiado para um novo array.

add(int index, E element): LinkedList possui O(n) e ArrayList possui O(n) no pior caso

remove(int index): LinkedList possui O(n) e ArrayList possui O(n-index), se remover o último elemento então fica O(1)

Prática

Comparação de Performance entre LinkedList e ArrayList

Thanks