Algoritmos e Estruturas de Dados

- **Tipos Abstratos de Dados (TAD)**

LEI - Licenciatura em Eng. Informática

2025/26

FACULDADE DE
CIÊNCIAS E TECNOLOGIA

Tipo Abstrato de Dados

- Os tipos de dados podem ser **abstratos**.

- Neste caso definimos quais as operações que se podem efectuar, sem definirmos como se guardam os valores em memória.

- Ex. Pilha de livros
  Algumas operações possíveis:
  - pôr livros no topo da pilha,
  - retirar livros do topo da pilha.

  Neste caso não definimos como se guarda a coleção de objetos.

Tipo Abstrato de Dados

- Um TAD:
  - Representa um tipo de dados
  - É caracterizado pelas operações que se podem efectuar sobre os diferentes valores/instâncias desse tipo de dados.

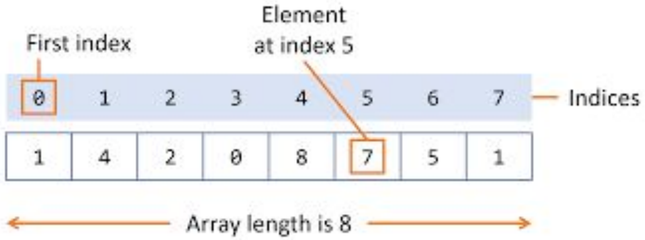
- Num TAD são propositadamente excluídos os detalhes relativos à maneira como os métodos executam as suas tarefas

- À especificação de um TAD chamamos *interface*.

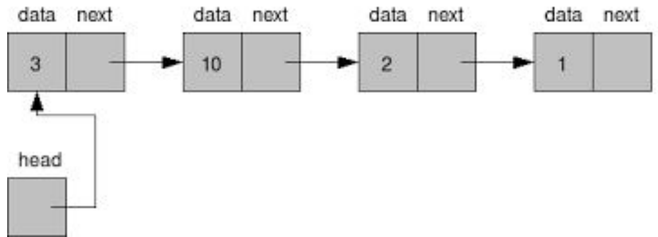
Estruturas de Dados

Como guardamos a coleção de livros em memória?

Ex.: em vetor:



Ex.: em lista ligada:



TAD vs ED

Um **Tipo Abstracto de Dados (TAD)** é caracterizado pelas operações que podem ser efetuadas sobre os seus valores.

Exemplo: Pilha, Fila.

em Java: Interface

Uma **Estrutura de Dados (ED)** é uma forma concreta de organizar informação na memória de um computador.

Exemplo: Vetor, Lista ligada.

em Java: Classe

Tipo Abstrato de Dados em OO

- Um Tipo Abstrato de Dados pode ser associado a várias implementações, instanciadas em classes.
- Cada classe poderá encapsular a utilização de uma determinada estrutura de dados, para implementar o TAD, com diferentes performances:
  - Em termos do tempo necessário para executar as operações sobre a Estrutura de Dados
  - Em termos do espaço (em memória) necessário para guardar a informação associada ao Tipo de Dados

TADs em AED

- Os exemplos e exercícios que vamos resolver em AED vão implicar a especificação de dois tipos de TADs
  - **TADs do domínio do problema** - TADs diretamente relacionados com o domínio do problema que pretendemos resolver. Por exemplo, Service, Student, HomeWayApp ...
  - **TADs genéricos** - usados em Programação para resolver problemas com determinadas características. Por exemplo, Map, List, Stack, Queue…

- As implementações de TADs genéricos serão auxiliadas pela escolha de estruturas de dados adaptadas às características subjacentes aos tipos
  - Por exemplo, o TAD List pode ser implementada em Vetor ou em Lista Ligada Simples ou Dupla

- As classes resultantes de implementações de TADs genéricos podem contribuir para as implementações de TADs associados ao domínio do problema
  - Uma fila de espera num Supermercado pode ser instanciada através de uma implementação do TAD Queue

Uma sequência
TAD List

- O TAD *List* é uma coleção de elementos, em que cada elemento está associado a uma dada posição (e.g. lista de actividades para realizar, uma playlist,...)

- Representa uma sequência de elementos a que podemos aceder através:
  - da posição de um elemento;
  - de um elemento igual.

- As operações incluem:
  - aceder a um elemento por posição;
  - inserir um elemento numa posição;
  - remover o elemento numa posição;
  - devolver a posição de um dado elemento..

TAD List (1)

```java
package dataStructures;
import dataStructures.exceptions.*;
import java.io.Serializable;
/**
 * List (sequence) Abstract Data Type
 * Includes description of general methods to be implemented by lists.
 * @author AED  Team
 * @version 1.0
 * @param <E> Generic Element
 */
public interface List<E> extends Serializable {
    int NOT_FOUND=-1;
  /**
     * Returns true iff the list contains no elements.
     * @return true if list is empty
     */
    boolean isEmpty();
    /**
     * Returns the number of elements in the list.
     * @return number of elements in the list
     */
    int size();
/**
 * Returns an iterator of the elements in the list (in proper sequence).
 * @return Iterator of the elements in the list
 */
Iterator<E> iterator();
```

TAD List (2)

```
/**
 * Returns the first element of the list.
 * @return first element in the list
 * @throws NoSuchElementException - if size() == 0
 */
E getFirst();

/**
 * Returns the last element of the list.
 * @return last element in the list
 * @throws NoSuchElementException - if size() == 0
 */
E getLast();

/**
 * Returns the element at the specified position in the list.
 * Range of valid positions: 0, ..., size()-1.
 * If the specified position is 0, get corresponds to getFirst.
 * If the specified position is size()-1, get corresponds to getLast.
 *
 * @param position - position of element to be returned
 * @return element at position
 * @throws InvalidPositionException if position is not valid in the list
 */
E get(int position);
```

TAD List (3)

```java
/**
 * Inserts the specified element at the first position in the list.
 *
 * @param element to be inserted
 */
void addFirst(E element);

/**
 * Inserts the specified element at the last position in the list.
 *
 * @param element to be inserted
 */
void addLast(E element);

/**
 * Inserts the specified element at the specified position in the list.
 * Range of valid positions: 0, ..., size().
 * If the specified position is 0, add corresponds to addFirst.
 * If the specified position is size(), add corresponds to addLast.
 *
 * @param position - position where to insert element
 * @param element  - element to be inserted
 * @throws InvalidPositionException - if position is not valid in the list
 */
void add(int position, E element);
```

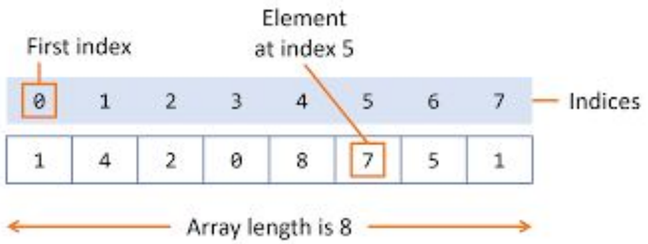
TAD List (4)

```java
/**
 * Removes and returns the element at the first position in the list.
 * @return element removed from the first position of the list
 * @throws NoSuchElementException - if size() == 0
 */
E removeFirst();

/**
 * Removes and returns the element at the last position in the list.
 * @return element removed from the last position of the list
 * @throws NoSuchElementException - if size() == 0
 */
E removeLast();

/**
 * Removes and returns the element at the specified position in the list.
 * Range of valid positions: 0, ..., size()-1.
 * If the specified position is 0, remove corresponds to removeFirst.
 * If the specified position is size()-1, remove corresponds to removeLast.
 *
 * @param position - position of element to be removed
 * @return element removed at position
 * @throws InvalidPositionException - if position is not valid in the list
 */
E remove(int position);
```

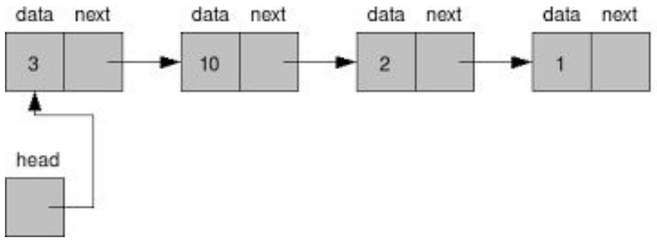
TAD List (5)

```
    /**
     * Returns the position of the first occurrence of the specified element
     * in the list, if the list contains the element.
     * Otherwise, returns NOT_FOUND.
     *
     * @param element - element to be searched in list
     * @return position of the first occurrence of the element in the list (or -1)
     */
    int indexOf(E element);
}
```

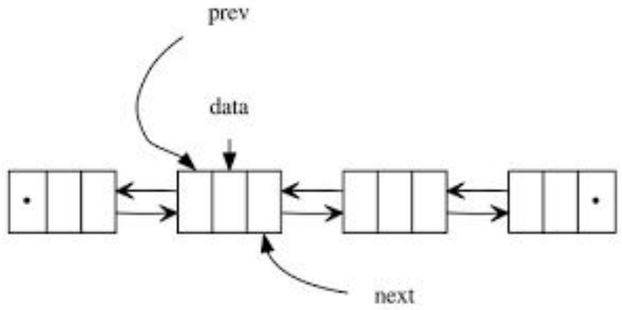
Possíveis estruturas de dados



Vetores



Lista simplesmente ligada



Lista duplamente ligada

Uma sequência
TAD TwoWayList

- O TAD *TwoWayList* é uma coleção de elementos, em que cada elemento está associado a uma dada posição (e.g. uma playlist,...)

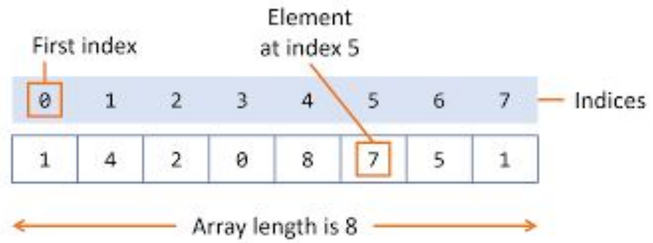
- Representa uma sequência de elementos a que podemos aceder através da posição, como a lista.

- As operações incluem todas as da lista, mais:
  - “twoWayIterator”, retorna um iterador que permite percorrer a sequência nos dois sentidos.

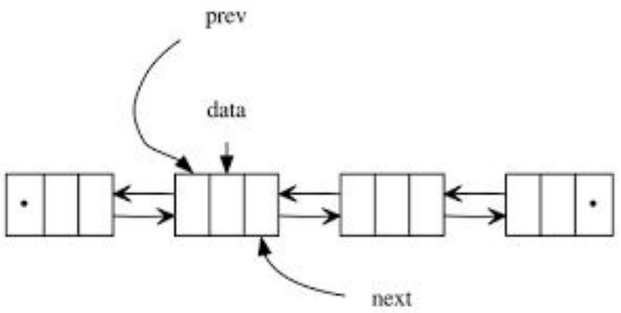
TAD TwoWayList (1)

```java
package dataStructures;
/**
 * Two-Way List
 *
 * @author AED team
 * @version 1.0
 *
 * @param <E> Generic Element
 */
public interface TwoWayList<E> extends List<E> {
    /**
     * Returns a two-way iterator of the elements in the list.
     *
     * @return Itwo-Way terator of the elements in the list
     */
    TwoWayIterator<E> twoWayIterator();
}
```

Possíveis estruturas de dados



Vetores



Lista duplamente ligada

Um dicionário
TAD Map

- O TAD *Map* é uma coleção de elementos não repetidos, identificados por uma chave única - identificador (e.g. os alunos duma turma – chave: número aluno, os carros dum país – chave: matrícula,...)

- Representa um coleção de elementos a que podemos aceder através:
  - da chave do elemento.

- As operações incluem:
  - aceder a um elemento por chave;
  - inserir um elemento com uma chave;
  - remover o elemento associado a uma chave.

TAD Map (1)

```java
package dataStructures;
import java.io.Serializable;

/**
 * Dictionary Abstract Data Type
 * Includes description of general methods to be implemented by dictionaries.
 * @author AED  Team
 * @version 1.0
 * @param <K> Generic Key
 * @param <V> Generic Value
 */
public interface Map<K,V> extends Serializable {

    record Entry<K,V>(K key,V value) implements Serializable{
        @Override
        public boolean equals(Object obj) {...}
    }
    /**
     * Returns true iff the dictionary contains no entries.
     * @return true if dictionary is empty
     */
    boolean isEmpty( );
    /**
     * Returns the number of entries in the dictionary.
     * @return number of elements in the dictionary
     */
    int size( );
```

TAD Map (2)

```
/**
 * If there is an entry in the dictionary whose key is the specified key, returns its value; otherwise, returns
null.
 * @param key whose associated value is to be returned
 * @return value of entry in the dictionary whose key is the specified key,
 * or null if the dictionary does not have an entry with that key
 */
V get( K key );

/**
 * If there is an entry in the dictionary whose key is the specified key, replaces its value by the specified
value
 * and returns the old value; otherwise, inserts the entry (key, value) and returns null.
 * @param key with which the specified value is to be associated
 * @param value to be associated with the specified key
 * @return previous value associated with key, or null if the dictionary does not have an entry with that key
 */
V put( K key, V value );

/**
 * If there is an entry in the dictionary whose key is the specified key, removes it from the dictionary and
returns
 * its value; otherwise, returns null.
 * @param key whose entry is to be removed from the map
 * @return previous value associated with key,  or null if the dictionary does not an entry with that key
 */
V remove( K key );
```

TAD Map (3)

```
/**
     * Returns an iterator of the entries in the dictionary.
     * @return  iterator of the entries in the dictionary
     */
    Iterator<Entry<K,V>> iterator( );

    /**
     * Returns an iterator of the values in the dictionary.
     * @return  iterator of the values in the dictionary
     */
    Iterator<V> values( );

    /**
     * Returns an iterator of the keys in the dictionary.
     * @return  iterator of the keys in the dictionary
     */
    Iterator<K> keys( );

}
```

Possíveis estruturas de dados

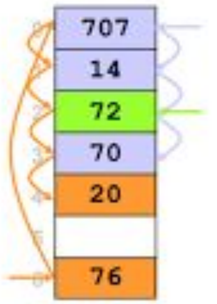
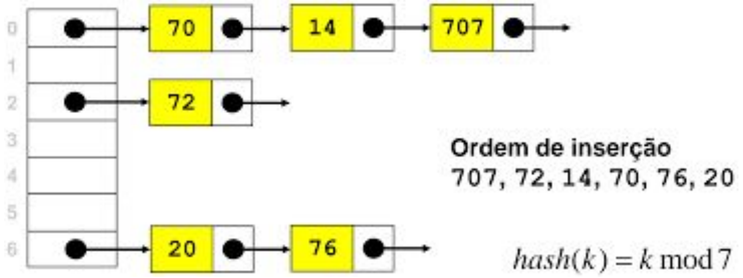


Tabela de dispersão fechada

Tabela de dispersão aberta



Um dicionário ordenado

TAD SortedMap

- O TAD *SortedMap* é uma coleção de elementos não repetidos, identificados por uma chave única – identificador, em que é necessário ordenação por chave (e.g. os alunos duma turma – chave: número aluno, os carros dum país – chave: matricula,...)

- Representa um conjunto de elementos a que podemos aceder através:
  - da chave do elemento;
  - da maior ou menor chave existente no conjunto .

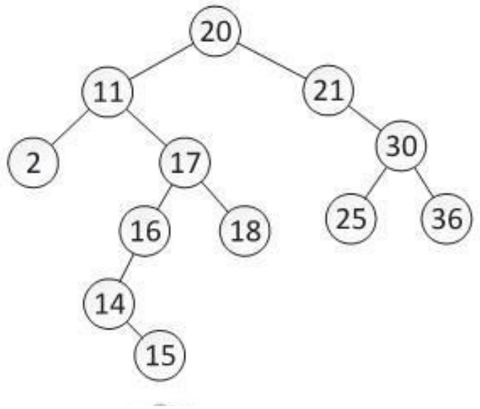
- As operações incluem:
  - aceder a um elemento por chave;
  - inserir um elemento com uma chave;
  - remover o elemento associado a uma chave;
  - aceder ao elemento com maior ou menor chave.

TAD SortedMap (1)

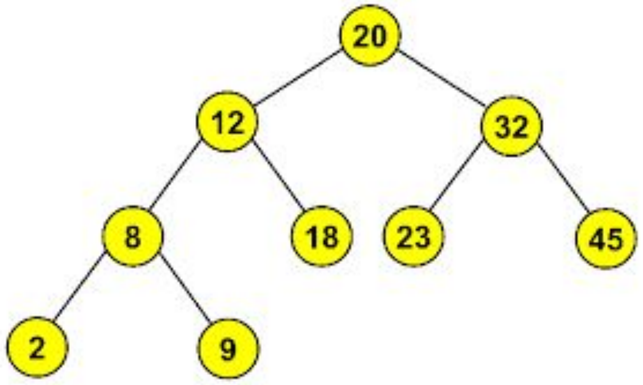
```
package dataStructures;
import dataStructures.exceptions.*;
/**
 * Ordered Dictionary interface
 * @author AED team
 * @version 1.0
 * @param <K> Generic type Key, must extend comparable
 * @param <V> Generic type Value
 */
public interface SortedMap<K extends Comparable<K>, V> extends Map<K,V> {

    /**
     * Returns the entry with the smallest key in the dictionary.
     * @return
     * @throws EmptyMapException
     */
    Entry<K,V> minEntry( );

    /**
     * Returns the entry with the largest key in the dictionary.
     * @return
     * @throws EmptyMapException
     */
    Entry<K,V> maxEntry( );
}
```

TAD SortedMap (2)

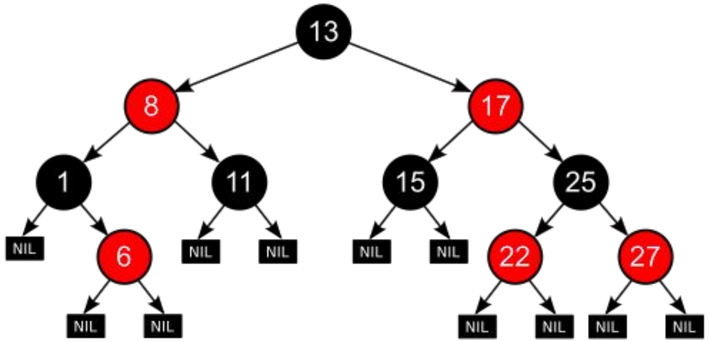
Possíveis estruturas de dados



Árvores Binárias de Pesquisa



Árvores AVL



Árvores Vermelha e Preta

Iterar os elementos duma coleção

TAD Iterator

- O TAD *Iterator* permite percorrer os elementos duma coleção de elementos

- Exemplos de percursos:
  - As actividades diárias (iterador de lista);
  - Os alunos da turma (iterador dos valores do dicionário);
  - Os números de alunos da turma (iterador das chaves do dicionário).

- As operações incluem:
  - aceder ao próximo elemento;
  - verificar se existem mais elementos.

TAD Iterator (1)

```java
package dataStructures;
import dataStructures.exceptions.*;

/**
 * Iterator Abstract Data Type
 * Includes description of general methods for one way iterator.
 * @author AED  Team
 * @version 1.0
 * @param <E> Generic Element
 */
public interface Iterator<E> {

    /**
     * Returns true if next would return an element.
     * @return true iff the iteration has more elements
     */
  boolean hasNext( );
    /**
     * Returns the next element in the iteration.
     * @return the next element in the iteration
     * @throws NoSuchElementException - if call is made without verifying pre-condition
     */
    E next( );
  /**
  * Restarts the iteration.
  * After rewind, if the iteration is not empty, next will return the first element.
   */
  void rewind();
}
```

Iterar os elementos duma coleção

TAD TwoWayIterator

- O TAD *TwoWayIterator* permite percorrer os elementos duma coleção de elementos em ambos os sentidos

- Exemplos de percursos:
  - As músicas duma playlist (iterador de lista nos dois sentidos);

- As operações incluem:
  - aceder ao próximo elemento;
  - aceder ao elemento anterior;
  - verificar se existem mais elementos.

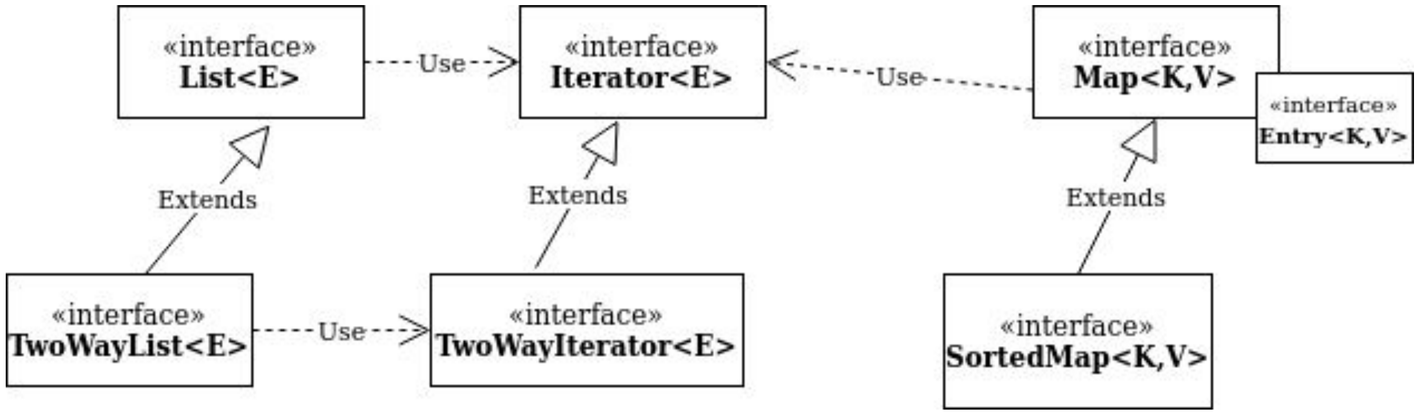
TAD TwoWayIterator (1)

```java
package dataStructures;
import dataStructures.exceptions.*;

/**
 * TwoWayIterator Abstract Data Type
 * Includes description of general methods for two-way iterator.
 * @author AED  Team
 * @version 1.0
 * @param <E> Generic Element
 *
 */
public interface TwoWayIterator<E> extends Iterator<E> {

    /**
     * Returns true if previous would return an element.
     * @return true iff the iteration has more elements in the reverse direction
     */
    boolean hasPrevious( );

    /**
     * Returns the previous element in the iteration.
     * @return previous element in the iteration
     * @throws NoSuchElementException - if call is made without verifying pre-condition
     */
    E previous( );
```

TAD TwoWayIterator (2)

```
/**
     * Restarts the iteration in the reverse direction.
     * After fullForward, if iteration is not empty,
     * previous will return the last element
     */
    void fullForward( );

}
```

Diagrama UML
Coleções iteráveis



Uma pilha
TAD Stack

- O TAD ***Stack*** é uma coleção de elementos com disciplina LIFO (Last – In – First - Out).

- Representa um conjunto de elementos a que podemos aceder unicamente:
  - Ao elemento que foi adicionado à menos tempo à coleção .

- As operações incluem:
  - aceder ao elemento mais recente na coleção;
  - colocar um novo elemento.

TAD Stack (1)

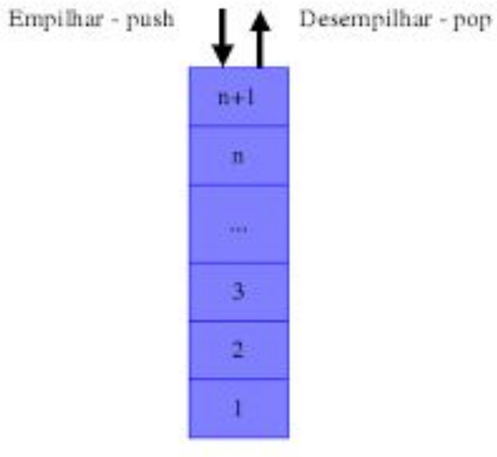
```java
package dataStructures;
import dataStructures.exceptions.*;
/**
 * Stack Abstract Data Type
 * Includes description of general methods for the Stack with the LIFO discipline.
 * @author AED  Team
 * @version 1.0
 * @param <E> Generic Element
 *
 */
public interface Stack<E> {

    /**
     *  Returns true iff the stack contains no
     *  elements.
     * @return true iff the stack contains no
     *         elements, false otherwise
     */
    boolean isEmpty( );

    /**
     *  Returns the number of elements in the stack.
     * @return number of elements in the stack
     */
    int size( );
```

TAD Stack (2)

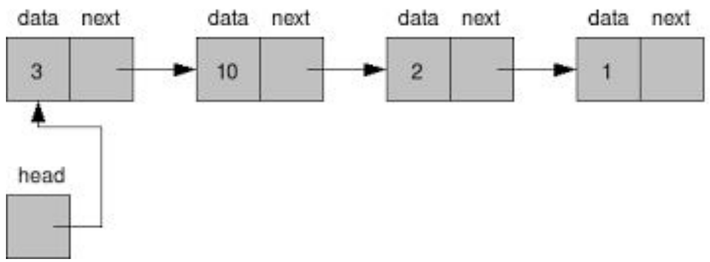
```
/**
     *  Returns the element at the top of the stack.
     *  Requires
     * @return element at top of stack
     * @throws EmptyStackException when size = 0
     */
    E top( );

    /**
     *  Inserts the specified <code>element</code> onto
     *  the top of the stack.
     * @param element element to be inserted onto the stack
     */
    void push( E element );

    /**
     *  Removes and returns the element at the top of the
     *  stack.
     * @return element removed from top of stack
     * @throws EmptyStackException when size = 0
     */
    E pop( );
}
```

TAD Stack (3)

Possíveis estruturas de dados



Vetores



Listas ligadas

Uma fila
TAD Queue

- O TAD ***Queue*** é uma coleção de elementos com disciplina FIFO (First – In – First - Out).

- Representa um conjunto de elementos a que podemos aceder unicamente:
  - Ao elemento que foi adicionado há mais tempo à coleção .

- As operações incluem:
  - aceder ao elemento mais antigo na coleção;
  - colocar um novo elemento.

TAD Queue (1)

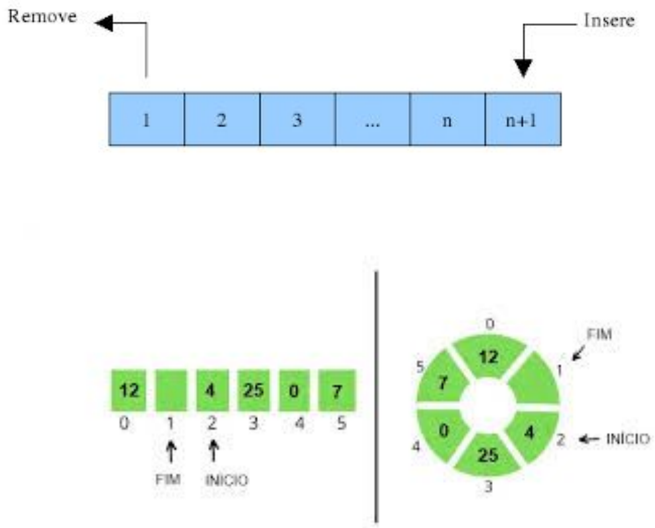
```
package dataStructures;
import dataStructures.exceptions.*;
/**
 * Queue Abstract Data Type
 * Includes description of general methods for the Queue with the FIFO
discipline.
 * @author AED  Team
 * @version 1.0
 * @param <E> Generic Element
 *
 */
public interface Queue <E>{
    /**
     * Returns true iff the queue contains no elements.
     * @return
     */
    boolean isEmpty( );

    /**
     * Returns the number of elements in the queue.
     * @return
     */
    int size( );
```

TAD Queue (2)

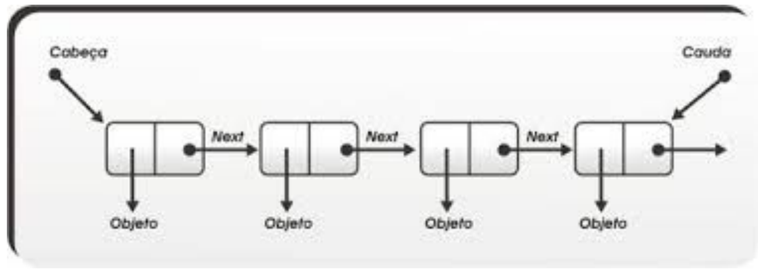
```
    /**
     * Inserts the specified element at the rear of the queue.
     * @param element
     */
    void enqueue( E element );

    /**
     * Removes and returns the element at the front of the queue.
     * @return
     * @throws EmptyQueueException
     */
    E dequeue( );
}
```

TAD Queue (3)

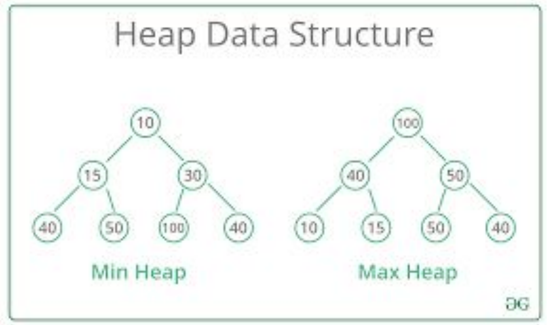
Possíveis estruturas de dados



Vetores



Listas ligadas



Árvores – caso PriorityQueue