



Algoritmos e Estruturas de Dados

Tipos Abstractos de Dados
2019/20

Tipo Abstracto de Dados (TAD)

- Um Tipo Abstracto de Dados (**TAD**) representa um tipo de dados, e é caracterizado pelas operações que se podem efectuar sobre os seus valores.
- Um TAD não especifica como se fazem essas operações.
- Em Java, um TAD deve ser especificado como uma **Interface**.
- Exemplos: *List*, *Map*, *SortedMap*, *Queue* ...



Programa solução

- Nas soluções que vamos apresentar/implementar para problemas concretos, vão existir dois tipos de TADs:
 - TADs diretamente relacionados com o domínio do problema que pretendemos resolver. Por exemplo, ShowPedia, Show, ...
 - TADs genéricos usados em Programação para resolver problemas com determinadas características (nomeadamente coleções). Por exemplo, List, Map, ...
- Frequentemente, o programa tem a classe Main, um package com as interfaces e classes do domínio do problema, e um package (regularmente designado por *dataStructures*) com as interfaces e classes do domínio das estruturas de dados.

Package dataStructures: interfaces (TADs)

- **Fila** (interface *Queue*)
 - Coleção de elementos com disciplina FIFO (first-in-first-out)



TAD *Queue*

```
package dataStructures;
```

```
public interface Queue<E> {
```

```
    // Returns true iff the queue contains no elements.
```

```
    boolean isEmpty( );
```

```
    // Returns the number of elements in the queue.
```

```
    int size( );
```

```
    // Inserts the specified element at the rear of the queue.
```

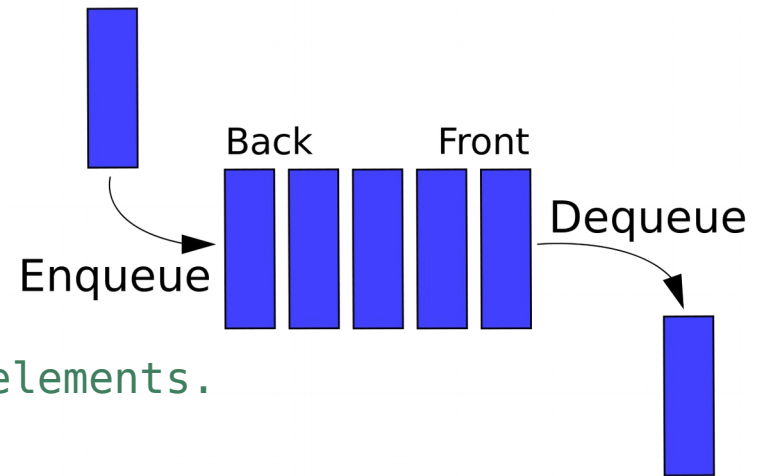
```
    void enqueue( E element );
```

```
    // Removes and returns the element at the front of the queue.
```

```
    // @throws NoSuchElementException if isEmpty()
```

```
    E dequeue( ) throws NoSuchElementException;
```

```
}
```



Classe *NoElementException*

```
package dataStructures;

public class NoElementException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public NoElementException( ){
        super();
    }

    public NoElementException( String msg ){
        super(msg);
    }

}
```

Package dataStructures: interfaces (TADs)

- **Pilha** (interface **Stack**)
 - Coleção de elementos com disciplina LIFO (last-in-first-out)



TAD *Stack*

```
package dataStructures;
```

```
public interface Stack<E> {
```

```
    // Returns true iff the stack contains no elements.  
    boolean isEmpty( );
```

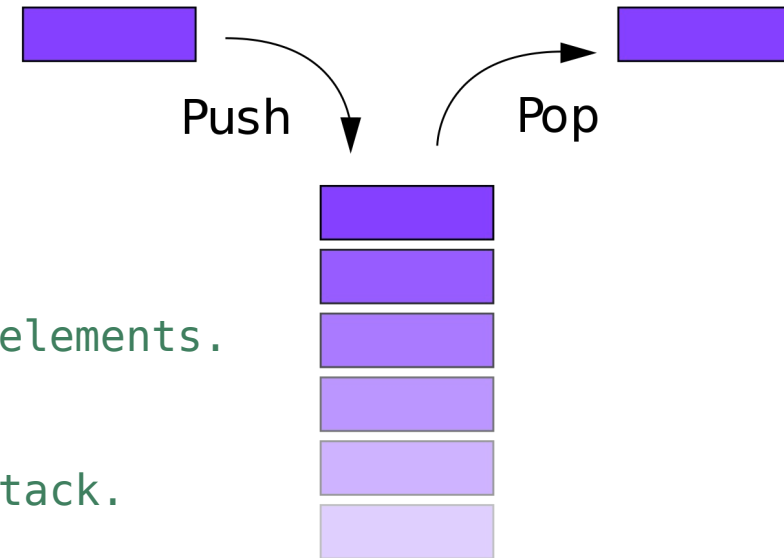
```
    // Returns the number of elements in the stack.  
    int size( );
```

```
    // Returns the element at the top of the stack.  
    // @throws NoSuchElementException if isEmpty()  
    E top( ) throws NoSuchElementException;
```

```
    // Inserts the specified element onto the top of the stack.  
    void push( E element );
```

```
    // Removes and returns the element at the top of the stack.  
    // @throws NoSuchElementException if isEmpty()  
    E pop( ) throws NoSuchElementException;
```

```
}
```



Package dataStructures: interfaces (TADs)

- **Lista** (interface *List*)
 - Coleção de elementos, em que cada elemento está associado a uma dada posição



TAD *List* (1)

```
package dataStructures;
```

```
public interface List<E> {
```

```
    // Returns true iff the list contains no elements.
```

```
    boolean isEmpty( );
```

```
    // Returns the number of elements in the list.
```

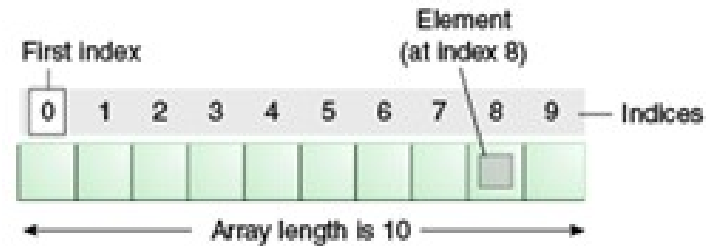
```
    int size( );
```

```
    // Returns position of first occurrence of specified element in the list,  
    // if the list contains the element. Otherwise, returns -1.
```

```
    int find( E element );
```

```
    ...
```

```
}
```



TAD *List* (2)

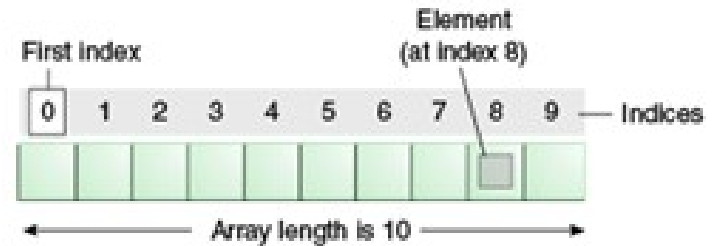
```
package dataStructures;
```

```
public interface List<E> {
```

```
...
// Returns the first element of the list.
E getFirst( ) throws NoSuchElementException;

// Returns the last element of the list.
E getLast( ) throws NoSuchElementException;

// 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.
// @throws InvalidPositionException if position<0 || position >=size()
E get( int position ) throws InvalidPositionException;
...
}
```



TAD *List* (3)

```
package dataStructures;
```

```
public interface List<E> {
```

```
    ...
```

```
    // Inserts the specified element at the first position in the list.
```

```
    void addFirst( E element );
```

```
    // Inserts the specified element at the last position in the list.
```

```
    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.
```

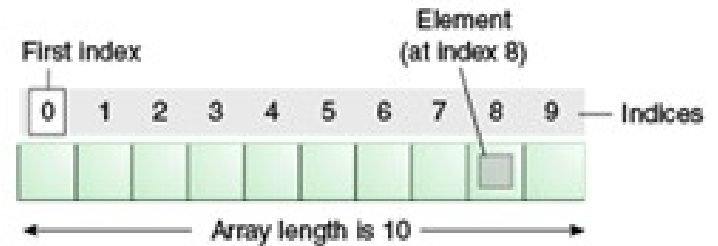
```
    // If the specified position is size(), add corresponds to addLast.
```

```
    // @throws InvalidPositionException if position<0 || position >size()
```

```
    void add( int position, E element ) throws InvalidPositionException;
```

```
    ...
```

```
}
```



TAD *List* (4)

```
package dataStructures;
```

```
public interface List<E> {
```

```
...
```

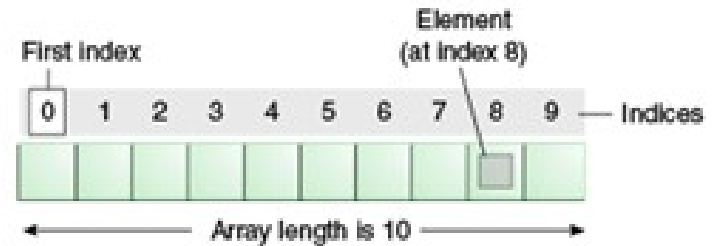
```
// Removes and returns the element at the first position in the list.  
E removeFirst( ) throws NoSuchElementException;
```

```
// Removes and returns the element at the last position in the list.  
E removeLast( ) throws NoSuchElementException;
```

```
// 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.  
// @throws InvalidPositionException if position<0 || position >=size()  
E remove( int position ) throws InvalidPositionException;
```

```
...
```

```
}
```





Classe **InvalidPositionException**

```
package dataStructures;
```

```
public class InvalidPositionException extends RuntimeException {
```

```
    private static final long serialVersionUID = 1L;
```

```
    public InvalidPositionException( ){  
        super();  
    }
```

```
    public InvalidPositionException( String msg ){  
        super(msg);  
    }
```

```
}
```

TAD *List* (5)

```
package dataStructures;
```

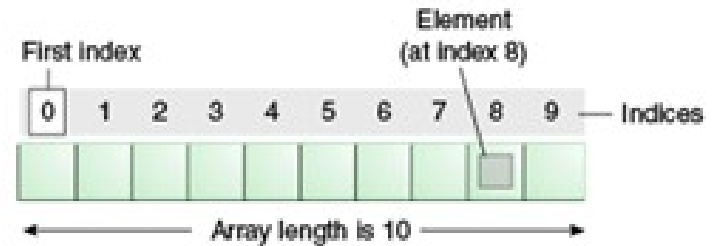
```
public interface List<E> {
```


```
...
```

```
// Returns an iterator of the elements in the list (in proper sequence).
```

```
Iterator<E> iterator( ) throws NoSuchElementException;
```

```
}
```

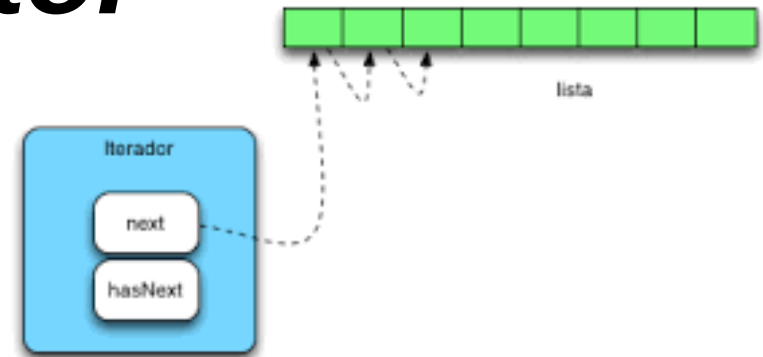




Package dataStructures: interfaces (TADs)

- Iterador (interface ***Iterator***)
 - Percurso sequencial dos elementos numa coleção
- Iterador Bidireccional (interface ***TwoWayIterator***)
 - Percurso sequencial e bidireccional dos elementos numa coleção

TAD *Iterator*



```
package dataStructures;
```

```
public interface Iterator<E> {
```

```
    // Returns true iff the iteration has more elements.  
    // In other words, returns true if next would return an element.  
    boolean hasNext( );
```

```
    // Returns the next element in the iteration.  
    // @throws NoSuchElementException if !hasNext()  
    E next( ) throws NoSuchElementException;
```

```
    // Restarts the iteration.  
    // After rewind, if iteration is not empty,  
    // next will return the first element.  
    void rewind( );
```

```
}
```

TAD *TwoWayIterator*

```
package dataStructures;
```

```
public interface TwoWayIterator<E> extends Iterator<E> {
```

```
    // Returns true iff iteration has more elements in the reverse direction.  
    // In other words, returns true if previous would return an element.
```

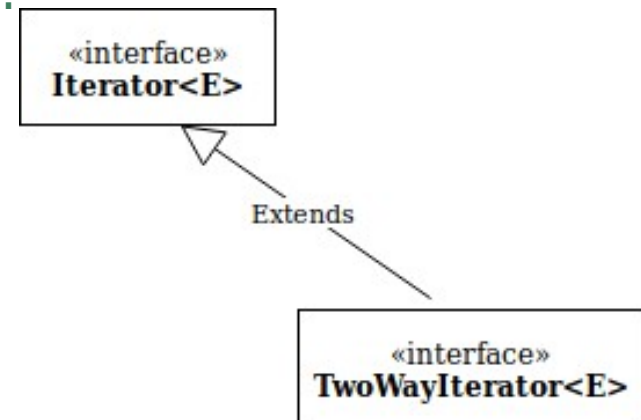
```
    boolean hasPrevious( );
```

```
    // Returns the previous element in the iteration.  
    // @throws NoSuchElementException if !hasPrevious()  
    E previous( ) throws NoSuchElementException;
```

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

```
    void fullForward( );
```

```
}
```



Classe NoSuchElementException

```
package dataStructures;

public class NoSuchElementException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public NoSuchElementException( ){
        super();
    }

    public NoSuchElementException( String msg ){
        super(msg);
    }

}
```

TAD *TwoWayList* (5)

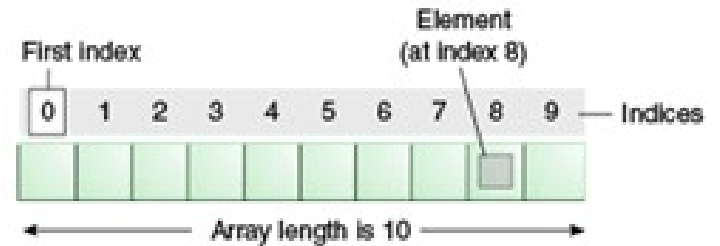
```
package dataStructures;
```

```
public interface TwoWayList<E> extends List<E> {
```

```
    // Returns a bidirectional iterator of the elements in the list.
```

```
    TwoWayIterator<E> iterator( ) throws NoSuchElementException;
```

```
}
```



Package dataStructures: interfaces (TADs)

- **Dicionário** (interface *Map*)
 - Coleção de elementos não repetidos, identificados por uma chave.
- **Dicionário Ordenado** (interface *SortedMap*)
 - Dicionário, em que é necessário ordenação nas chaves.



TAD *Map* (1)

```
package dataStructures;
```

```
public interface Map<K, V> {
```

```
    // Returns true iff the map contains no entries.
```

```
    boolean isEmpty( );
```

```
    // Returns the number of entries in the map.
```

```
    int size( );
```

```
    // Returns an iterator of the keys in the map.
```

```
    Iterator<K> keys( ) throws NoSuchElementException;
```

```
    // Returns an iterator of the values in the map.
```

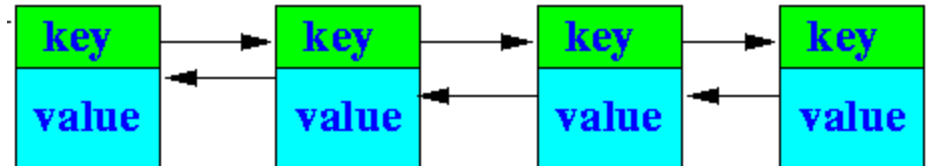
```
    Iterator<V> values( ) throws NoSuchElementException;
```

```
    // Returns an iterator of the entries in the map.
```

```
    Iterator<Entry<K,V>> iterator() throws NoSuchElementException;
```

```
    ...
```

```
}
```



TAD *Map* (2)

```
package dataStructures;
```

```
public interface Map<K, V> {
```

```
...
```

```
// If there is an entry in the map whose key is the specified key,  
// returns its value; otherwise, returns null.
```

```
V find( K key );
```

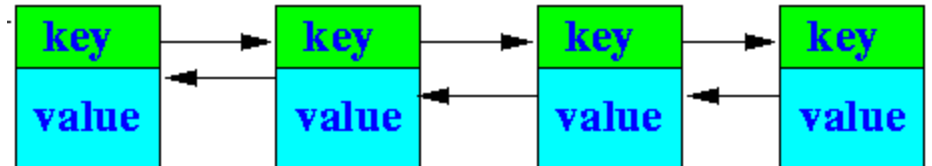
```
// If there is an entry in the map whose key is the specified key,  
// replaces its value by the specified value and returns the old value;  
// otherwise, inserts the entry (key, value) and returns null.
```

```
V insert( K key, V value );
```

```
// If there is an entry in the map whose key is the specified key,  
// removes it from the map and returns its value; otherwise, returns null.
```

```
V remove( K key );
```

```
}
```



Interface Entry

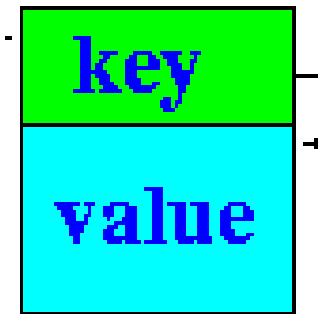
```
package dataStructures;

public interface Entry<K, V> {

    // Returns the key in the entry.
    K getKey( );

    // Returns the value in the entry.
    V getValue( );

}
```



TAD *SortedMap*

```
package dataStructures;
```

```
public interface SortedMap<K, V> extends Map<K,V>{
```

```
    // Returns the entry with the smallest key in the SortedMap.
```

```
    // @throws NoSuchElementException if isEmpty()
```

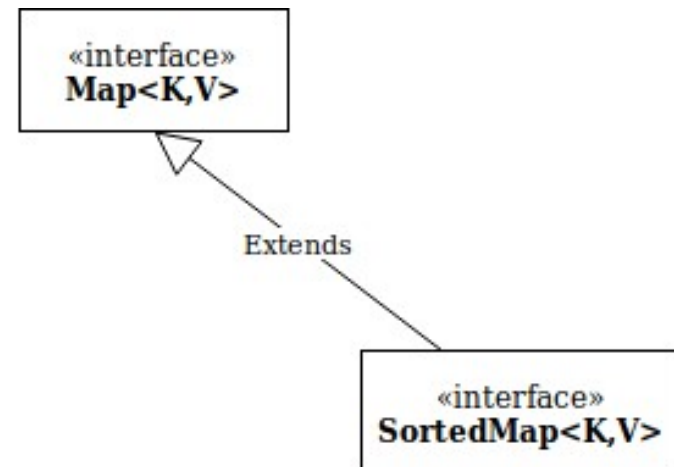
```
    Entry<K,V> minEntry( ) throws NoSuchElementException;
```

```
    // Returns the entry with the largest key in the SortedMap.
```

```
    // @throws NoSuchElementException if isEmpty()
```

```
    Entry<K,V> maxEntry( ) throws NoSuchElementException;
```

```
}
```



100

-



TAD *MinPriorityQueue*

```
package dataStructures;
```

```
public interface MinPriorityQueue<K, E> {
```

```
    // Returns true if the priority queue contains no elements.
```

```
    boolean isEmpty( );
```

```
    // Returns the number of elements in the priority queue.
```

```
    int size( );
```

```
    // Returns an entry with the smallest key in the priority queue.
```

```
    // @throws NoSuchElementException if isEmpty()
```

```
    Entry<K,E> minEntry( ) throws NoSuchElementException;
```

```
    // Inserts the entry (key, value) in the priority queue.
```

```
    void insert( K key, E value );
```

```
    // Removes an entry with smallest key from priority queue and returns it.
```

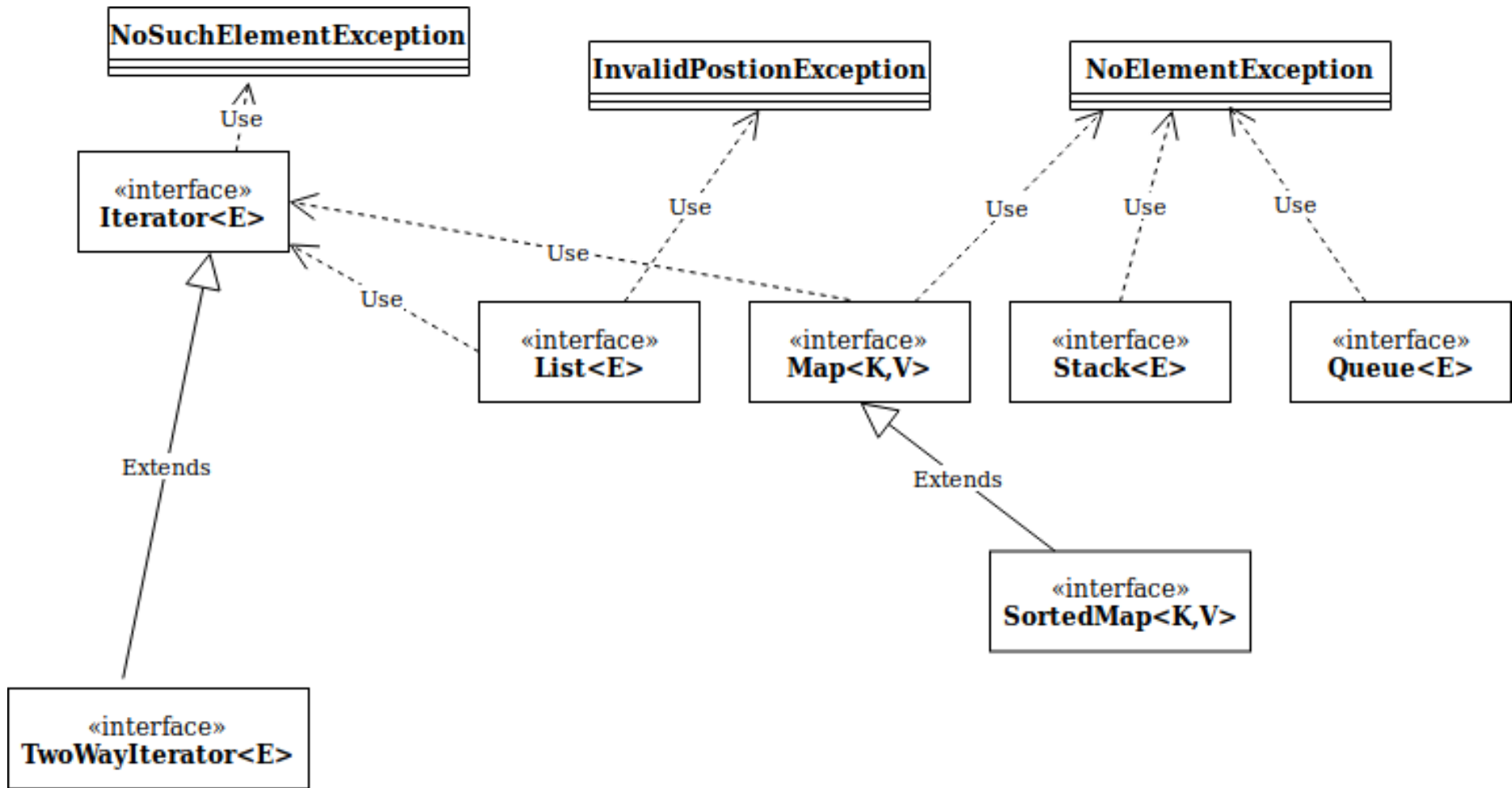
```
    // @throws NoSuchElementException if isEmpty()
```

```
    Entry<K,E> removeMin( ) throws NoSuchElementException;
```

```
}
```

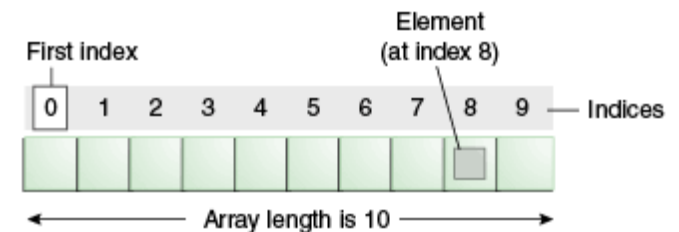
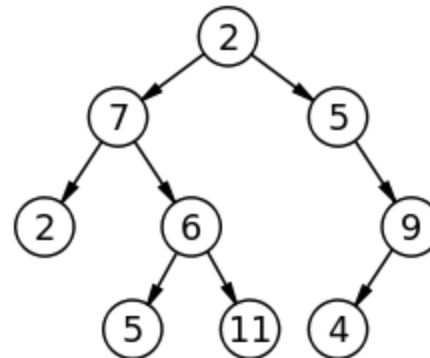
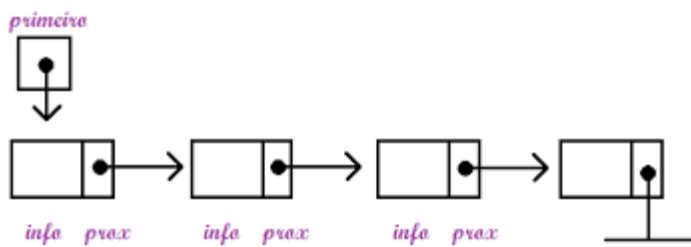
Diagrama de classes (interfaces)

package dataStructures



TAD vs Estrutura de Dados

- Um TAD pode ser implementado de diferentes formas.
- Uma **Estrutura de Dados** (ED) é uma forma concreta de organizar informação na memória dum computador.
- Em Java, uma implementação dum TAD é realizada numa **Classe**.



Implementação dos TADs

- Podemos implementar os nossos TADs com algumas classes existentes na API do Java:
 - TAD **List**: *ArrayList, LinkedList, ...*
 - TAD **Stack**: *ArrayList, LinkedList, ...*
 - TAD **Queue** : *ArrayList, LinkedList, ...*
 - TAD **Map**: *HashMap, LinkedHashMap, ...*
 - TAD **SortedMap**: *TreeMap, ...*
 - TAD **PriorityQueue**: *PriorityQueue, ...*
- A escolha da melhor alternativa deve tornar o ***programa mais eficiente***, como iremos ver!

Exercícios propostos

- Realizar ambos os exercícios (consultório e visita a parque) com os TADs dados.

Apresentar o diagrama de classes e submeter o programa no mooshak.

A implementação de cada TAD do package dataStructures é realizada com as classes da API do Java. Por exemplo, pode ter uma classe `QueueWithJavaClass` que implementa a interface (TAD) `Queue`. Esta classe tem como variável de instância um `ArrayList` ou `LinkedList` da API do Java.

- ✓ 1ª aula prática → consultório médico
- ✓ TPC 1ª semana → visita a parque de diversões versão A