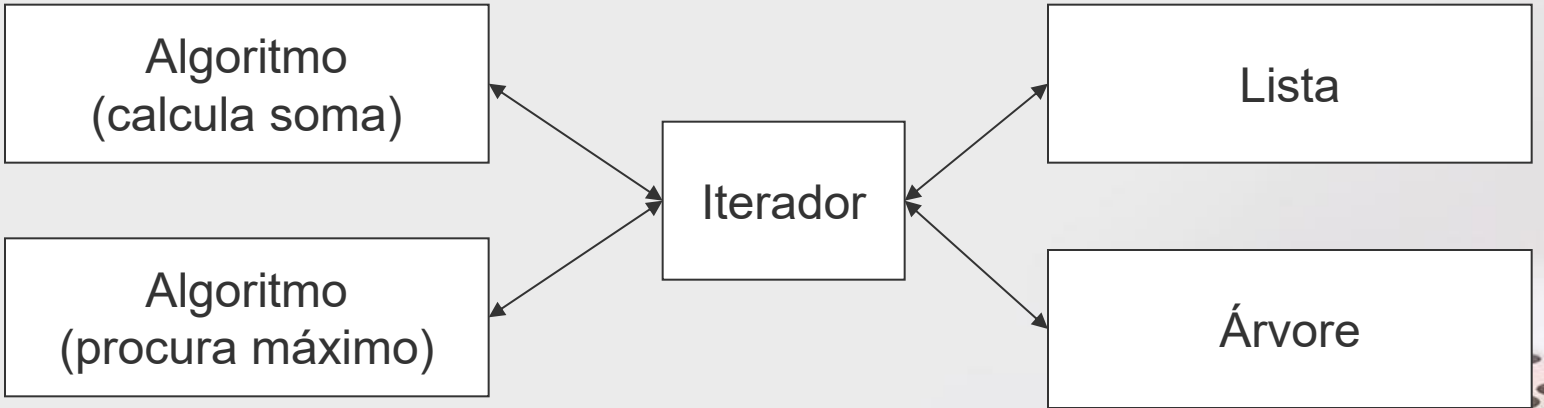
Estruturas de Dados

Iteradores

2025/2026

Iteradores

- Os iteradores são objectos utilizados para percorrer estruturas de dados.
  - Permitem preservar a independência entre os algoritmos e as estruturas de dados manipuladas.
  - Os algoritmos requerem um iterador, que é fornecido pela estrutura de dados.



Iteradores

- Em java, os iteradores devem implementar o interface seguinte:

```
public interface Iterator<E>{
   boolean hasNext();
   E next() ;   // gera excepção caso não exista
   default void remove();
   default void forEachRemaining(Consumer<? super E>
   action);  //Java 8+
}
```

Iteradores

- A criação de uma nova estrutura de dadas deve ser acompanhada pela criação de um iterador adequado.

- As estruturas de dados disponibilizam iteradores adequados através de um método *fábrica,* que devolve um iterador adequado.

```
Iterator<E> interator();
```

- O interface *Iterable<T>* indica a existência do método acima descrito.

Iteradores

- Exemplo: fábrica de Iterador

```
public class Par <T> implements Iterable<T>
{
T p1,p2;

Iterator<T> iterator(){
return new IteratorPar<T>(this);
}
…
};
```

Iteradores

```
public class IteratorPar<T> implements Iterator<T>{
int counter=0;
Par<T> par;
IteratorPar( Par<T> p){par=p;}

Boolean hasNext() {return counter!=2;}

T next(){
    switch(counter)
    {
          case 0: counter++; return par.p1;
          case 1: counter++; return par.p2;
          default: throw new NoSuchElementException();
    };
}

}
```

Algoritmos

- Os algoritmos podem ser tornados independentes dos contentores de dados através do uso de Iteradores

```
public <T> boolean procura(Iterable<T> m, T o){
Iterator<T> it=m.iterator();
boolean proximo=it.hasNext();
while(proximo){
    if(it.next()==o)    // compara referência, não conteúdo
        return true;
    proximo=it.hasNext();
   }
return false;
}
```

Iteradores e Excepções

- Os iteradores podem/devem gerar as seguintes excepções:
  - **UnsupportedOperationException** – A operação (p.ex: remoção) não é suportada.
  - **NoSuchElementException** – Tentativa de acesso a um elemento que não existe
  - **IllegalStateException** – Tentativa de remoção sem avançar para primeiro elemento ou tentativa de remover o mesmo elemento mais do que uma vez.
  - **ConcurrentModificationException** – Quando se tenta usar um iterador inválido. Um iterador é inválido quando a colecção foi alterada externamente (através de um outro iterador, por exemplo).

Iteradores e Excepções

- O suporte para geração da exceção **ConcurrentModificationException** pode ser realizado através de uma contagem de modificações (*p.ex., add, remove*) armazenada na estrutura e no iterador: