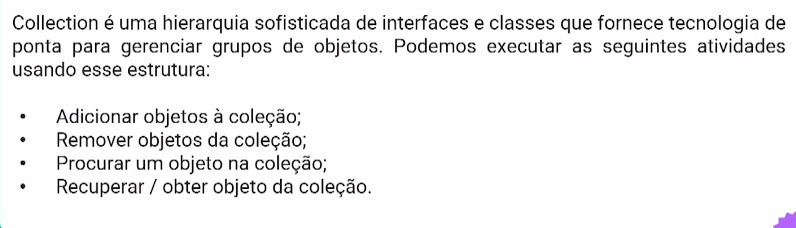
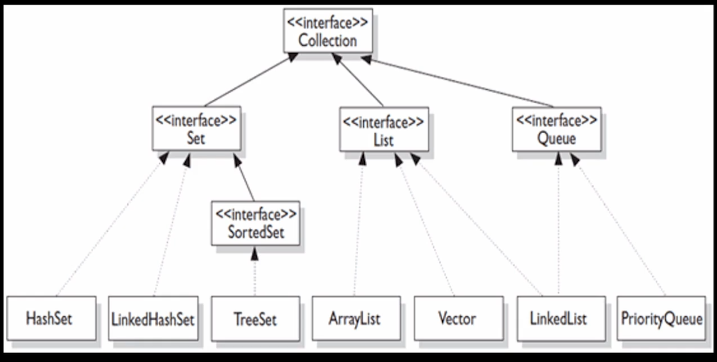
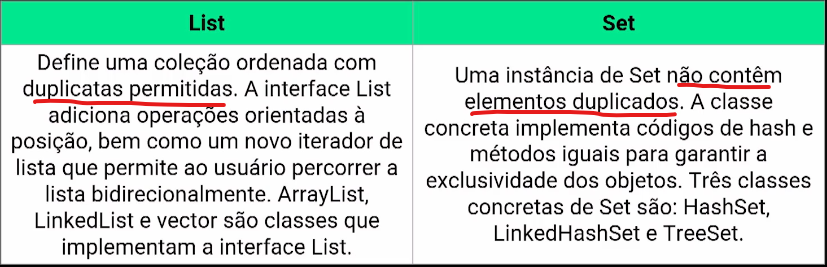
**Collection:**

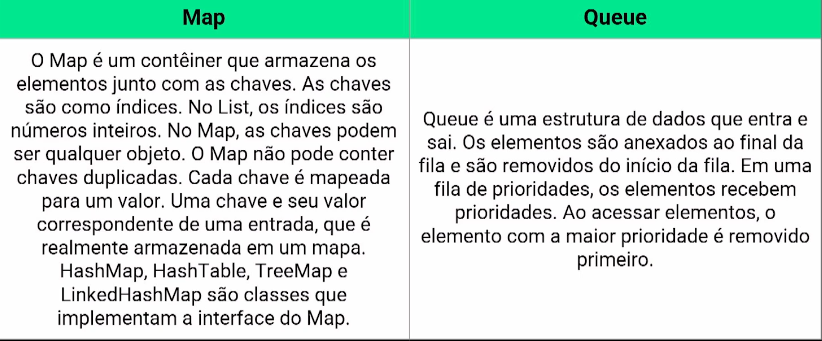
Uma estrutura de dados é uma coleção de dados organizados de alguma maneira. A estrutura não armazena dados, mas também suporta operações para acessar e manipular os dados. O pacote java.util contem um dos subsistemas mais poderosos do java: Collections.





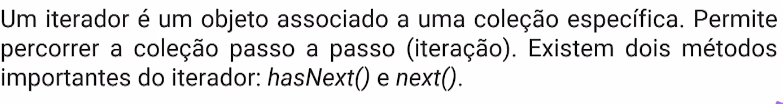
**listas (List), conjuntos (Set), filas (Queue), mapas (Map)**



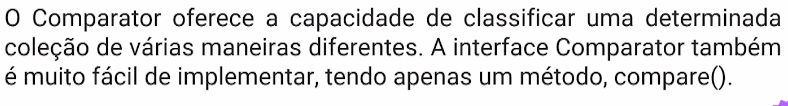


**Interface iterator e comparator:**

Iterator: para percorrer um vetor a gente não faz um for? Ele faz exatamente isso com uma coleção.



Comparator: comparar elementos.

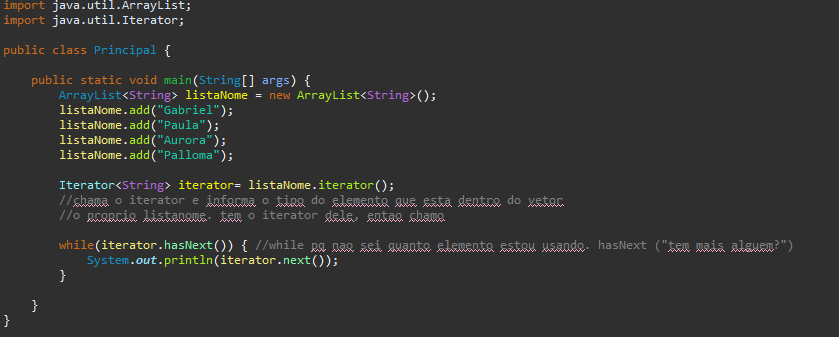




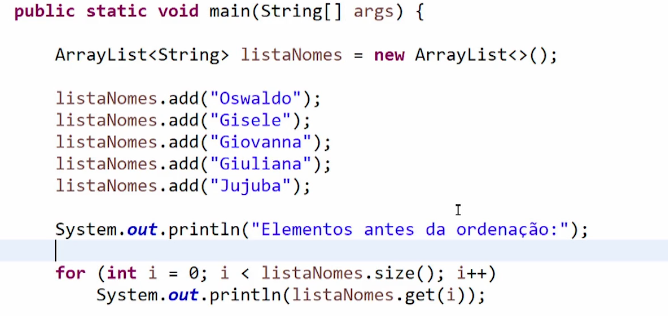
Ex pratico:

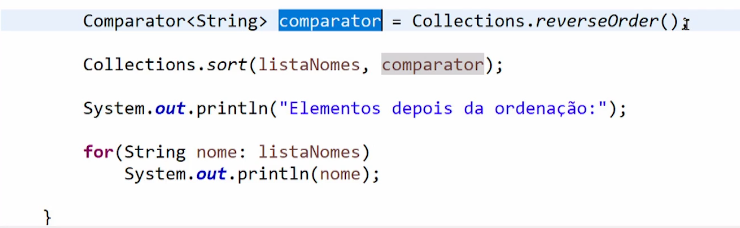


Embora funcione com o for, não é a melhor forma.

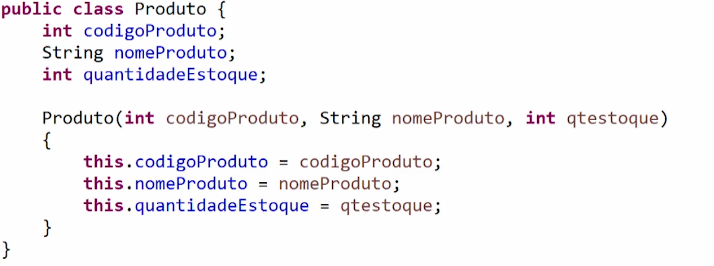


E o comparato?



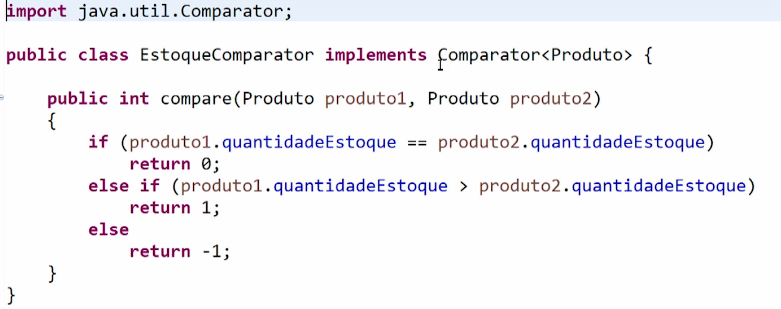


Outro exemplo:





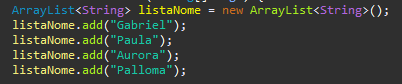


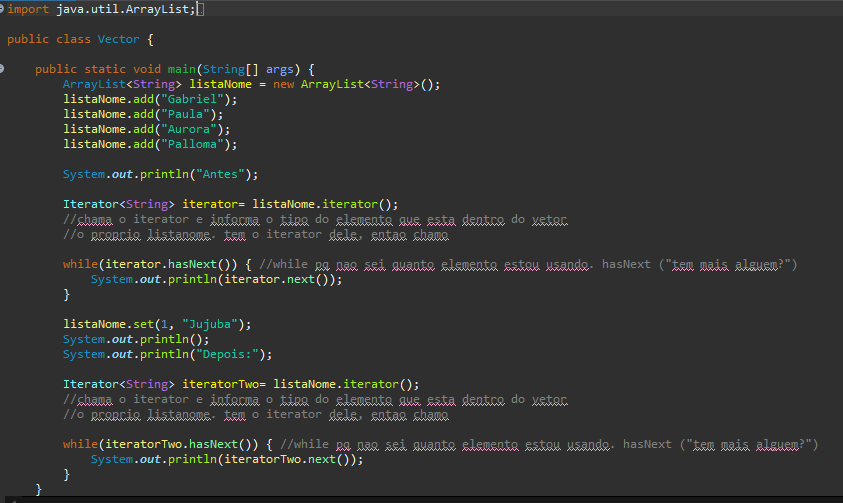


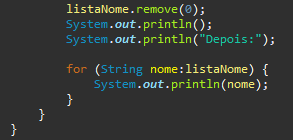
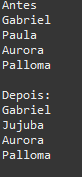
**ArrayList e Vector:**

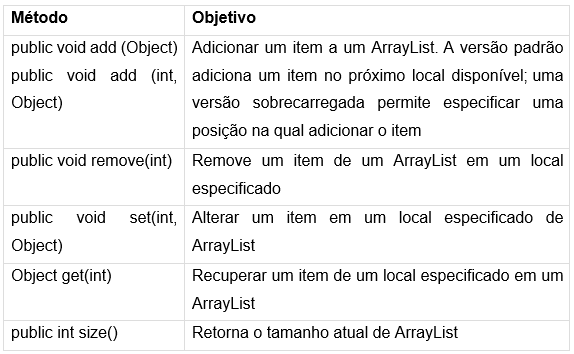
**Array:** (quando vou usar uma interface, só coloco List) mas a estrutura continua igual.

Usada para criar contêineres que armazenam listas de objetos. ArrayList pode ser considerado como uma matriz expansível. Oferece iteração rápida e acesso aleatório rápido.

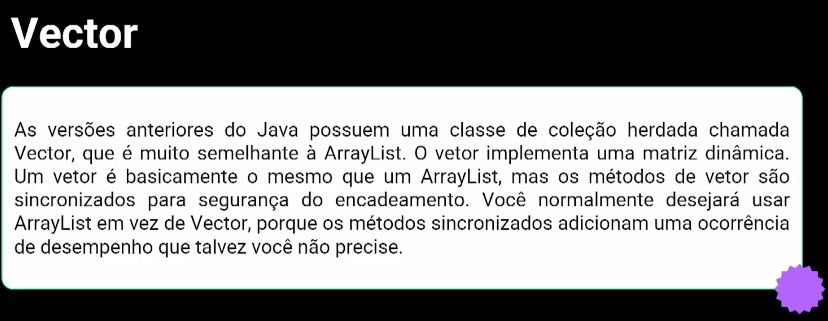


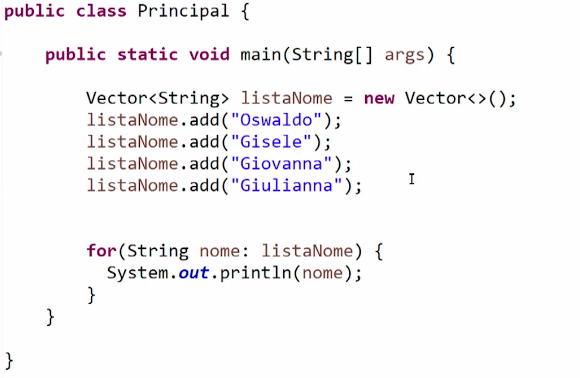






**Vector:**

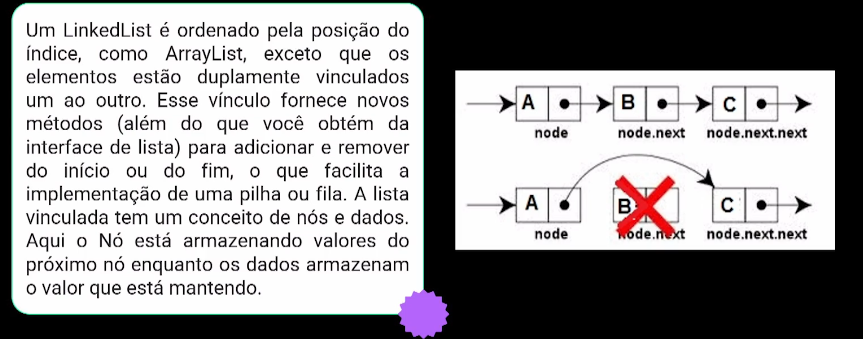




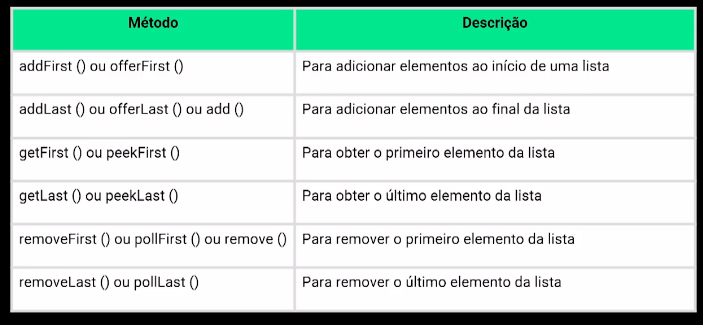
Igual ao arraylist, contudo ele é protegido enquanto o arraylist não.

**LinkedList:**

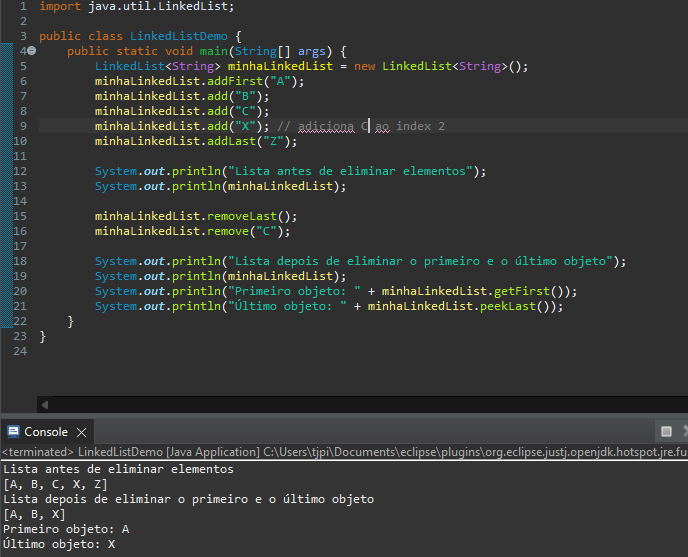
Parecido com o Arraylist, trabalha com referencia a próximos elementos.



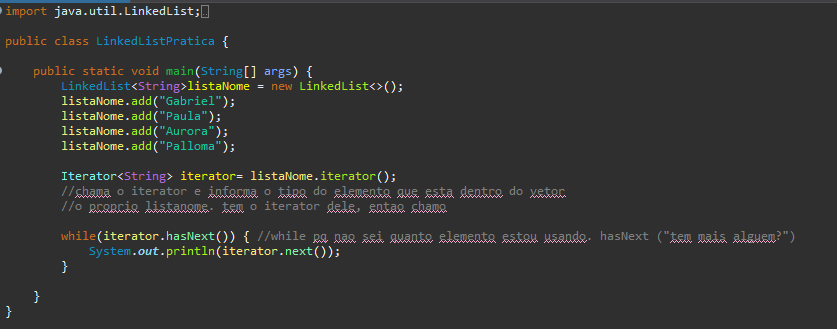
Métodos:



Ex:

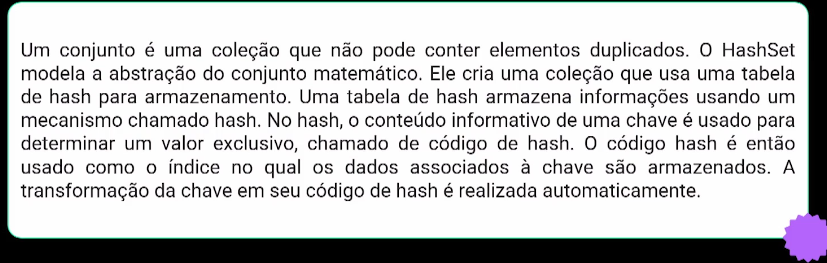


Na pratica:

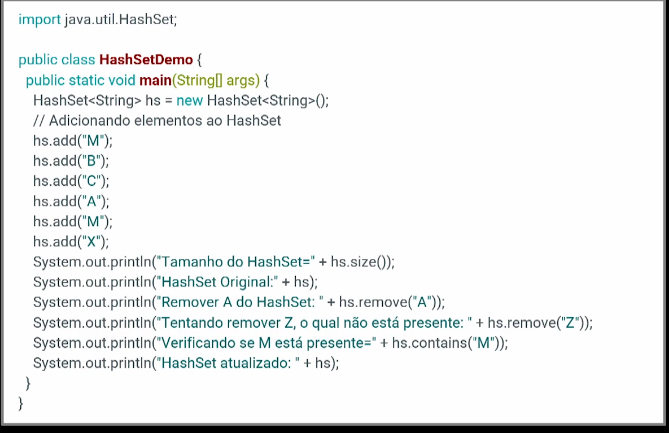


Agora métodos que não permitem duplicar: (**aqueles que terminam com set**):

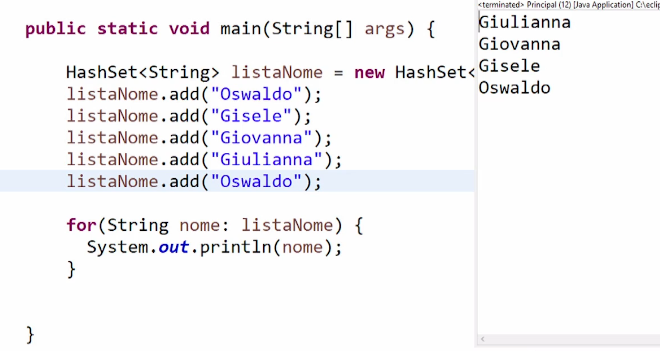
**HashSet:(calculado)**



Ex:

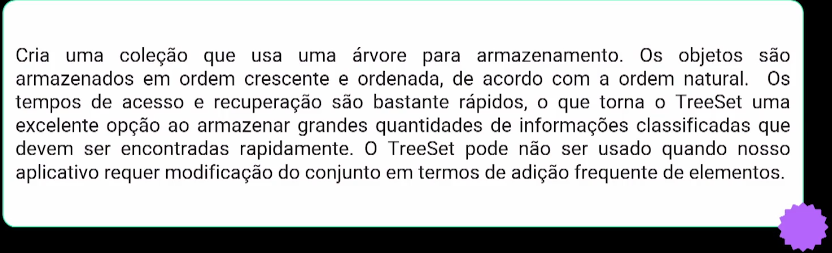


Na pratica:



Não repetiu oswaldo e também não tenho controle na ordenação que os elementos serão exibidos.

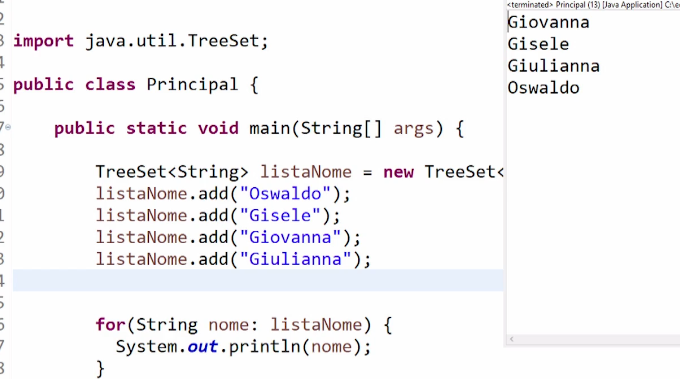
**TreeSet:(Arvore)**



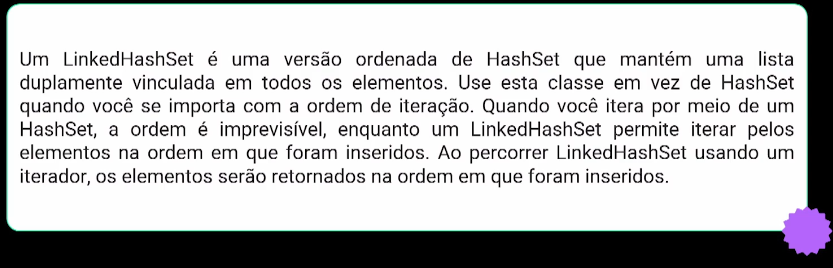
Ex:



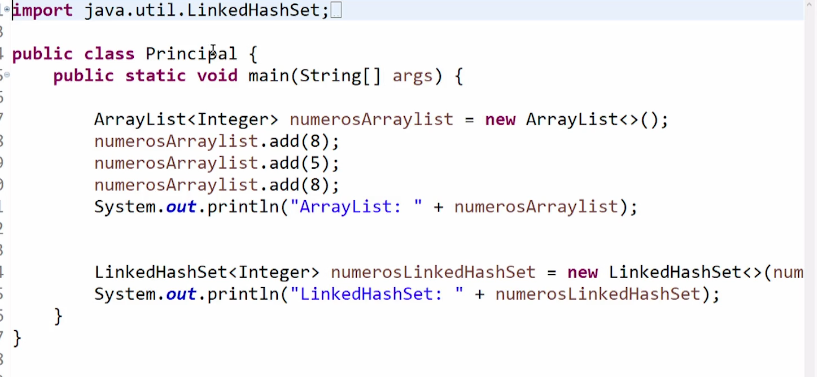
Na pratica:



**LinkedHashSet:**



Na pratica:



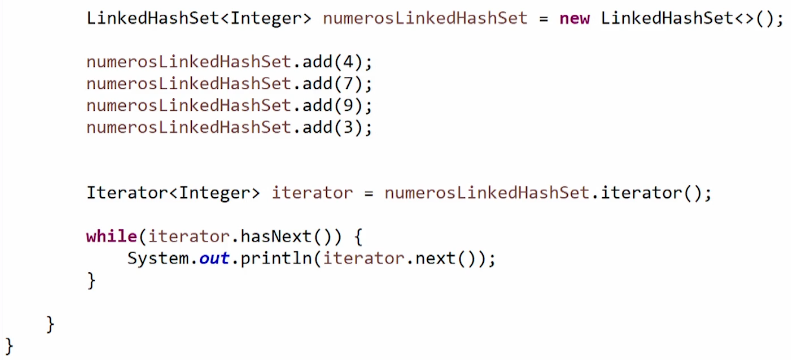


EX2:





Ex3:



1. **HashSet**:
   * **Armazenamento**: Utiliza uma tabela hash para armazenar os elementos.
   * **Ordenação**: Não garante a ordem dos elementos. A ordem pode mudar ao adicionar ou remover elementos.
   * **Desempenho**: Geralmente é a mais rápida das três em termos de operações básicas (inserção, remoção e busca), desde que a tabela hash esteja bem dimensionada.
2. **TreeSet**:
   * **Armazenamento**: Utiliza uma árvore binária de busca (Red-Black Tree) para armazenar os elementos.
   * **Ordenação**: Garante que os elementos sejam armazenados em ordem natural (ordem crescente) ou com base em um comparador fornecido.
   * **Desempenho**: Geralmente mais lenta que o HashSet para operações básicas, mas é útil quando se precisa manter os elementos ordenados.
3. **LinkedHashSet**:
   * **Armazenamento**: É semelhante ao HashSet, mas mantém uma lista duplamente ligada que vincula todos os elementos em uma sequência de inserção.
   * **Ordenação**: Mantém a ordem de inserção dos elementos. Se um elemento já estiver presente e for inserido novamente, a ordem de inserção original é mantida.
   * **Desempenho**: Um pouco mais lenta que HashSet devido à manutenção da lista de inserção, mas ainda geralmente mais rápida que TreeSet.

**Resumo**:

* Use HashSet quando não precisar de nenhuma ordem específica e desejar o melhor desempenho.
* Use TreeSet quando precisar que os elementos sejam mantidos em ordem natural ou personalizada.
* Use LinkedHashSet quando precisar preservar a ordem de inserção dos elementos.