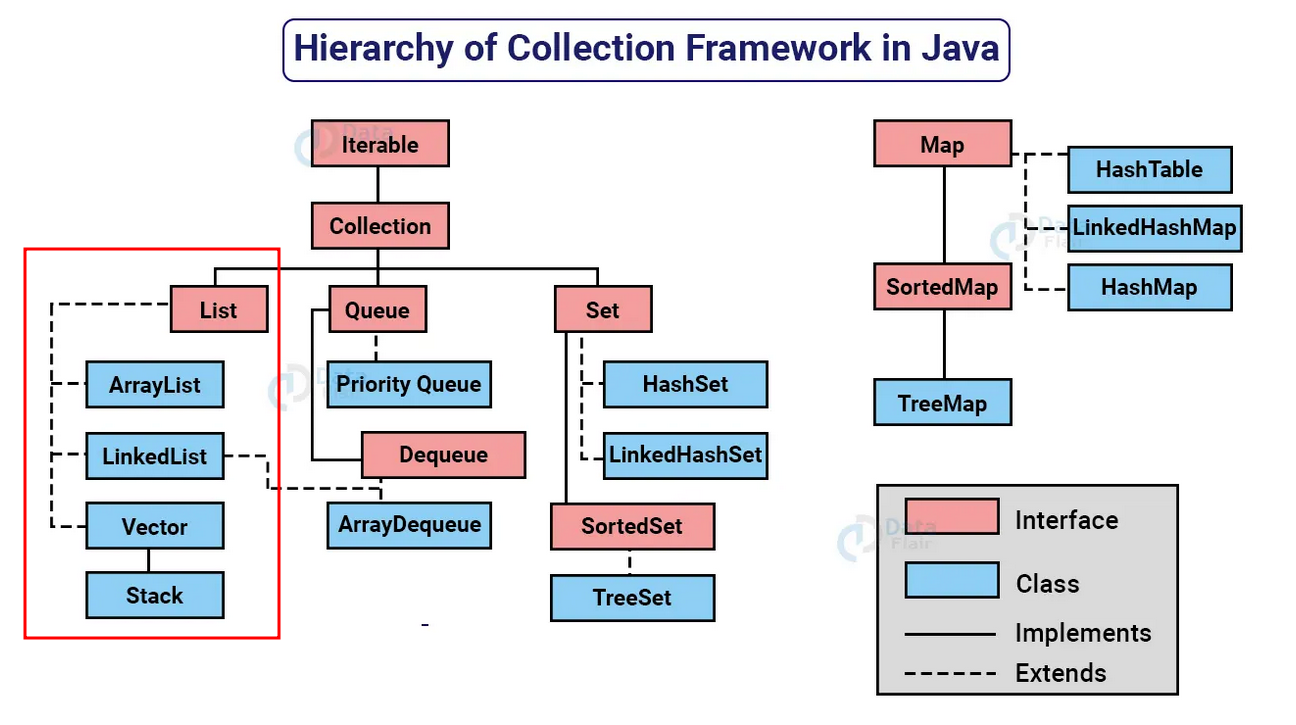
**List Interface**

**[](https://github.com/JacksonvBarbosa/collections-java-api-2023/blob/master/assets/image/list-interface-hierarchy.png)**[**Hierarchy of Collection Framework in Java**](https://data-flair.training/blogs/collection-framework-in-java/)

* A interface List é uma coleção ordenada que permite a inclusão de elementos duplicados.
* É um dos tipos de coleção mais utilizados em Java, e as classes de implementação comuns são ArrayList e LinkedList.
* A List se assemelha a uma matriz com comprimento dinâmico, permitindo adicionar ou remover elementos.
* A interface List fornece métodos úteis para adicionar elementos em posições específicas, remover ou substituir elementos com base no índice e obter sublistas usando índices.
* A classe Collections fornece algoritmos úteis para manipulação de List, como ordenação (sort), embaralhamento (shuffle), reversão (reverse) e busca binária (binarySearch).

**ArrayList:** O ArrayList é uma implementação da interface List que armazena os elementos em uma estrutura de array redimensionável. Isso significa que ele pode crescer automaticamente à medida que novos elementos são adicionados. A principal vantagem do ArrayList é o acesso rápido aos elementos por meio de índices, o que permite recuperar um elemento específico de forma eficiente. No entanto, adicionar ou remover elementos no meio da lista pode ser mais lento, pois requer a realocação de elementos.

**LinkedList:** O LinkedList é uma implementação da interface List que armazena os elementos em uma lista duplamente vinculada. Cada elemento contém referências para o elemento anterior e próximo na lista. A principal vantagem do LinkedList é a eficiência na adição ou remoção de elementos no início ou no final da lista, pois não é necessário realocar elementos. No entanto, o acesso aos elementos por meio de índices é mais lento, pois requer percorrer a lista até o elemento desejado.

**Vector:** O Vector é uma implementação antiga da interface List que é semelhante ao ArrayList, mas é sincronizada, ou seja, é thread-safe. Isso significa que várias threads podem manipular um objeto Vector ao mesmo tempo sem causar problemas de concorrência. No entanto, essa sincronização adiciona uma sobrecarga de desempenho, tornando o Vector menos eficiente do que o ArrayList em cenários em que a concorrência não é um problema. Por esse motivo, o uso do Vector é menos comum em aplicações modernas.