Estrutura de dados – Java

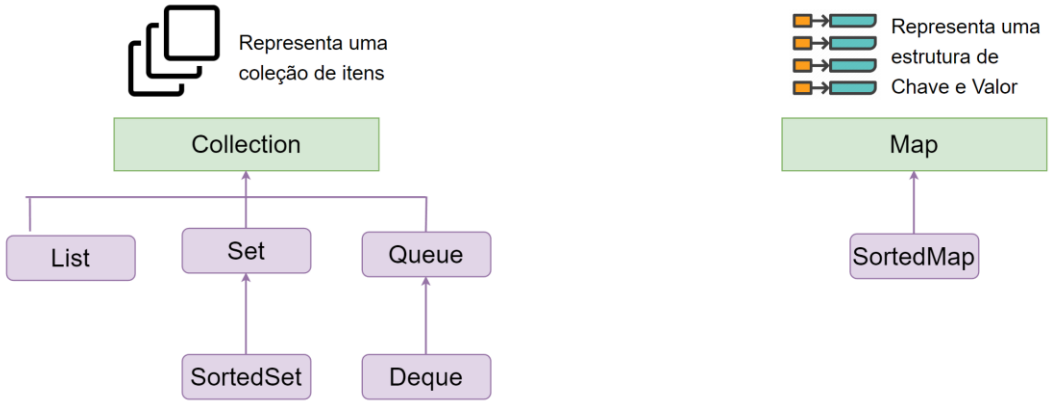
Arrays são muito poderosos e existem em qualquer linguagem, mas você não constrói software somente com arrays. Eles têm ***valores fixos***, a ***busca de elementos é cara(linear)*** e eles ***não são extensíveis (não se reorganizam automaticamente).***

Na programação precisamos de **requerimentos de mais alto nível, como por exemplo:**

-> ter uma estrutura que não saiba exatamente quantos elementos ela vai conter.
-> precisa ser capaz de se reorganizar automaticamente (abrir espaço e etc).
-> ser capaz de prover um acesso randômico de maneira muito rápida
-> preservar a ordem com que os elementos são inseridos.
-> conter somente valores únicos ou pode ter duplicados.
-> pode ter valor null ou não ter nenhum valor null
-> se ordenar automaticamente
-> conter valores em par com chaves de acesso a esses valores

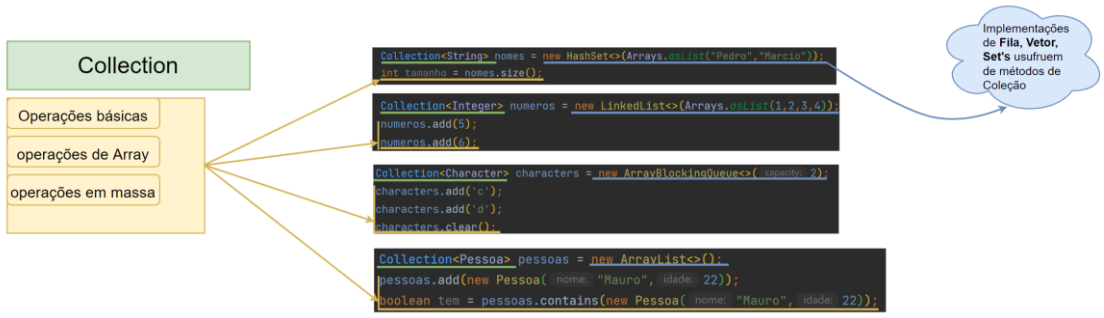
Todos **esses conceitos são provenientes de estrutura de dados**, e a API de **Collection** do Java serve justamente para isso. Todo tipo de estrutura de dados que existe, o Java tem uma **implementação disponível**.

A **API de Collection** tem **Interfaces que provêm assinaturas para o uso de certos algoritmos de estrutura de dados**:



Collection – Java

A **collection é o máximo de abstração** do **comportamento** que uma **coleção de dados de qualquer tipo deve ter**. Toda coleção de dados precisa saber seu ***tamanho, se contém um determinado objeto, adicionar um elemento etc***.... E tudo isso vem como contrato da **Interface Collection**:



List – Java

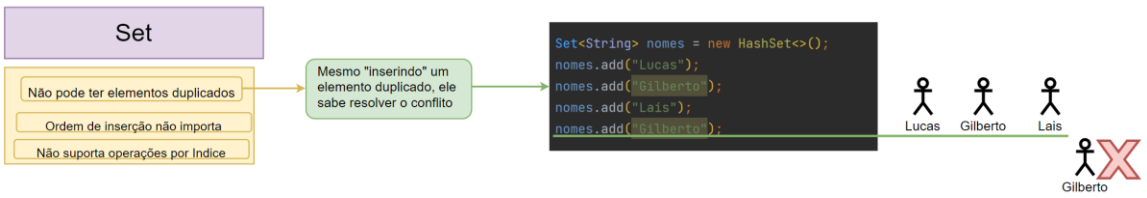
A **List** é uma especificação de coleção utilizada para **vetores,** em casos **onde a sequência *(ordem de inserção)* e posição de elementos da coleção** importam:

Beleza, tudo que for uma implementação de **List** tem essas características em comum, mas em relação as implementações concretas, porque usar qual?

-> ArrayList: Busca de elemento rápida (pois conhece o índice e preserva ordem), inserção e deleção não (precisa percorrer sequencialmente a lista atrás da posição requerida).
-> LinkedList: Inserção e deleção performáticas (isso porque cada elemento conhece seu anterior e próximo), busca de elemento não.
-> CopyOnWriteArrayList: Uma implementação de ArrayList que é thread safe.

Set – Java

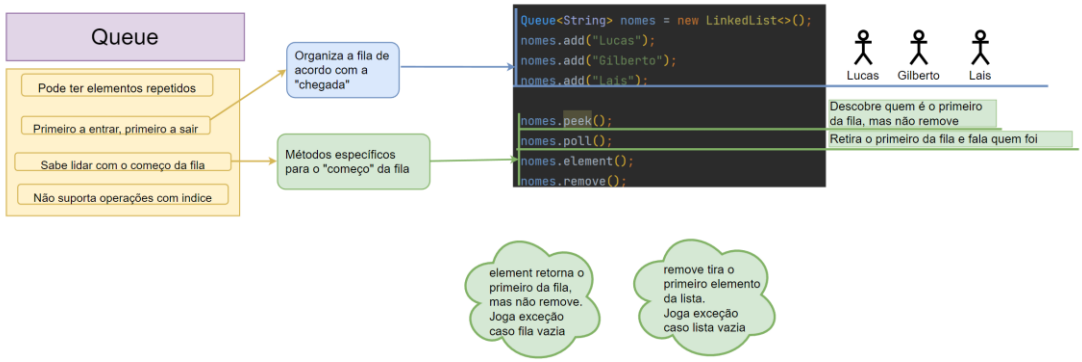
O set é a especificação da coleção que lida com conjuntos que não podem ter elementos repetidos:



-> HashSet: Rápido para consultar, inserir e remover. Mas não garante ordenação dos elementos.
-> TreeSet: Acesso aos elementos mais custoso, mas mantém a ordem natural dos elementos (de acordo com o Comparable implementado).
-> LinkedHashSet: Seus elementos conhecem o próximo e anterior a eles(não pensei em um uso).

Queue – Java

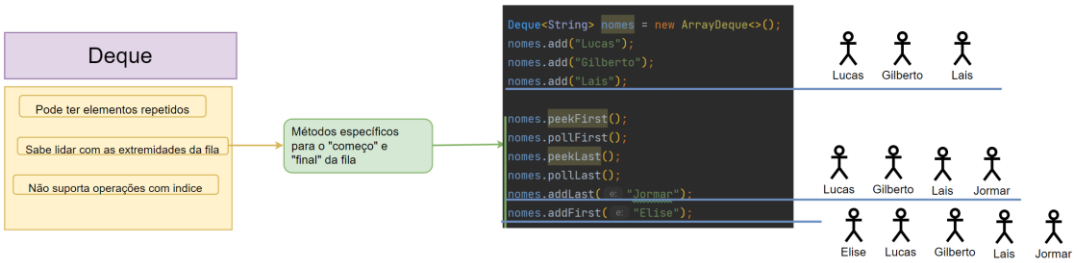
A **queue** é a especificação da coleção que vai lidar com “filas”, pensando puramente no início da fila, facilitando manipulação do começo da fila. Fila segue o conceito **FIFO *(first in, first out):***



-> LinkedList: É uma implementação concreta comum para Queues, embora não seja a melhor.
-> ArrayDeque: Essa é específica para filas e mais performática, priozar.

Deque – Java

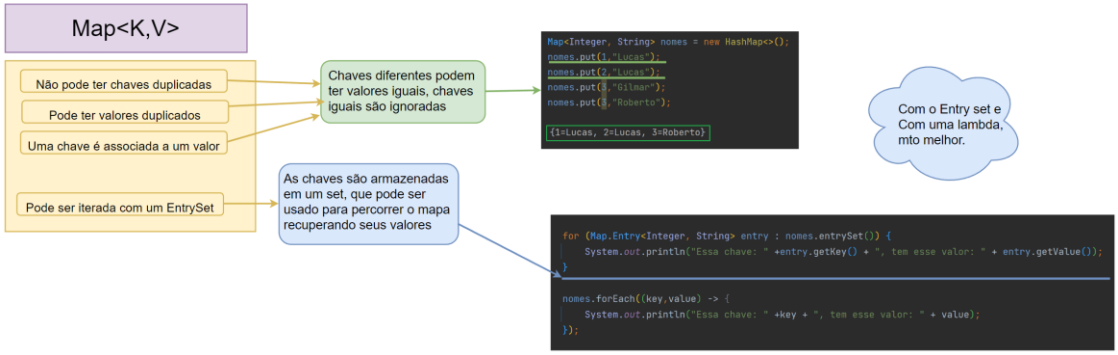
A **Deque** é a especificação da coleção que vai lidar com “filas”, pensando tanto no início da fila, quanto no final, facilitando manipulação das extremidades da fila:



-> ArrayDeque: Essa é específica para filas e mais performática, priozar.

Map – Java

Map não é uma coleção, mas ta ali do lado. É a estrutura para se trabalhar com chave e valor:



-> HashMap: Elementos não são ordenados, mas é rápida na busca/Inserção de dados.
-> TreeMap: Elementos mantém ordem natural (de acordo com o Comparable implementado), menos performático que o HashMap.
-> LinkedHashMap: Mantém a ordem de inserção dos elementos, ou seja, itera o mapa seguindo a ordem de inserçã
-> HashTable: Um mapa Thread Safe, não aceita valores nulos, pesquisa de elementos eficiente.