ESTRUTURA DE DADOS

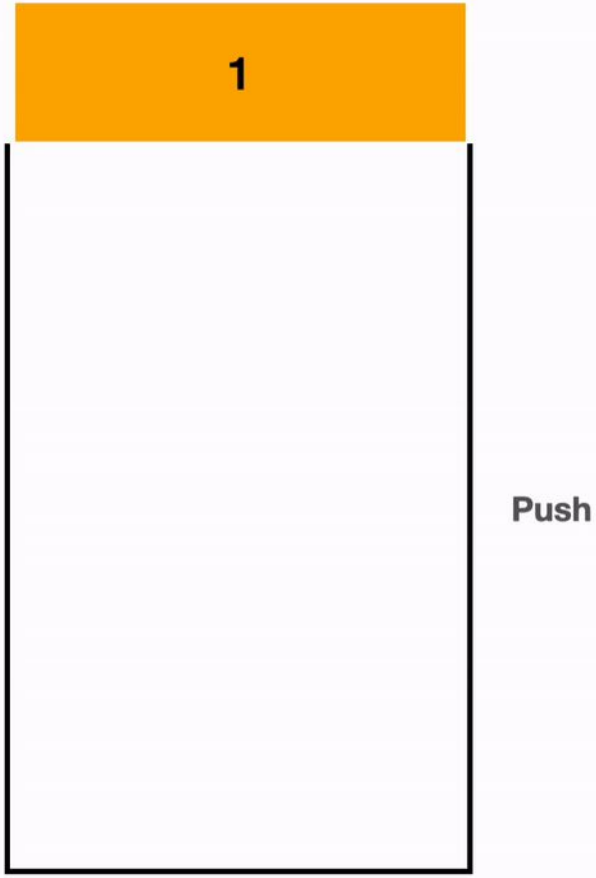
Huilson José Lorenzi

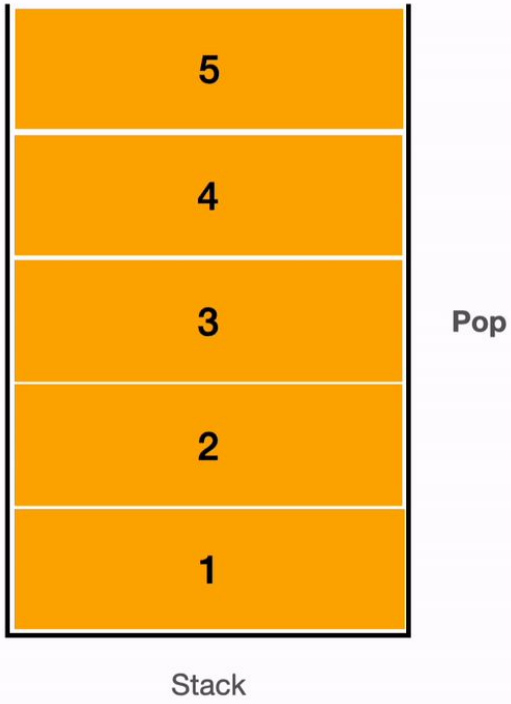
O QUE VAMOS ABORDAR

- Pilhas (First In, Last Out) - Lista Encadeada
- Filas (First In, First Out) - Lista Encadeada
- Mapas

PILHA

Uma pilha é uma estrutura de dados que segue o princípio "último a entrar, primeiro a sair" (LIFO). É semelhante a uma pilha de objetos físicos, onde você pode adicionar um novo objeto ao topo e remover o objeto mais alto quando necessário.


1
Push
Stack


5
4
3
2
1
Pop
Stack

PILHA

- **Ordem:** A ordem de inserção na pilha não é importante, a menos que seja desejada uma ordem reversa intencional, como no exemplo do refrigerador.
- **Capacidade Limitada:** As pilhas têm uma capacidade máxima. Se excedida, resulta em uma `stack overflow`. É essencial monitorar o tamanho da pilha para evitar esse problema.

PILHA

- **Matrizes:** Pilhas podem ser implementadas usando matrizes, onde os elementos são armazenados em locais de memória consecutivos. O topo da pilha é representado por um índice, e inserir ou remover elementos envolve atualizar esse índice adequadamente.

PILHA

- **Listas Encadeadas:** Pilhas também podem ser implementadas usando listas encadeadas, onde cada elemento é armazenado em um nó contendo tanto o valor quanto uma referência ao próximo nó. O topo da pilha é representado pela cabeça da lista encadeada, e empurrar ou remover elementos envolve adicionar ou remover nós na cabeça.

PILHA

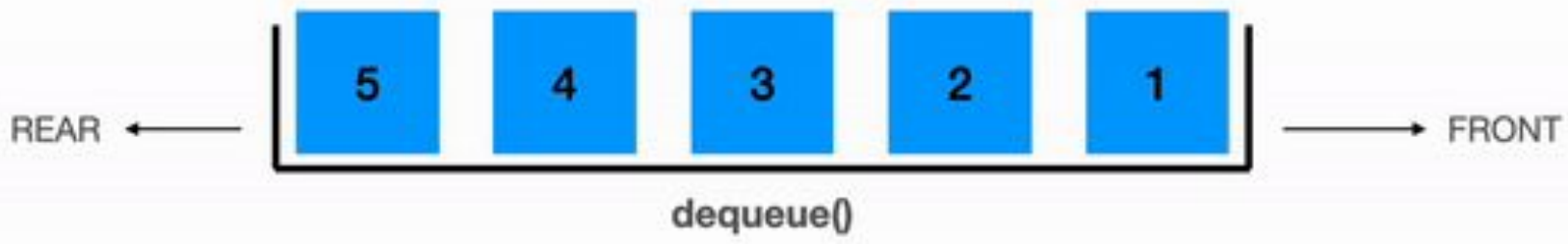
- **Exemplo prático:** Um supermercado precisa repor as prateleiras movendo os produtos do estoque. Tanto no estoque como nas prateleiras, geralmente os últimos itens a serem colocados são os primeiros a serem comprados/consumidos ou movidos.

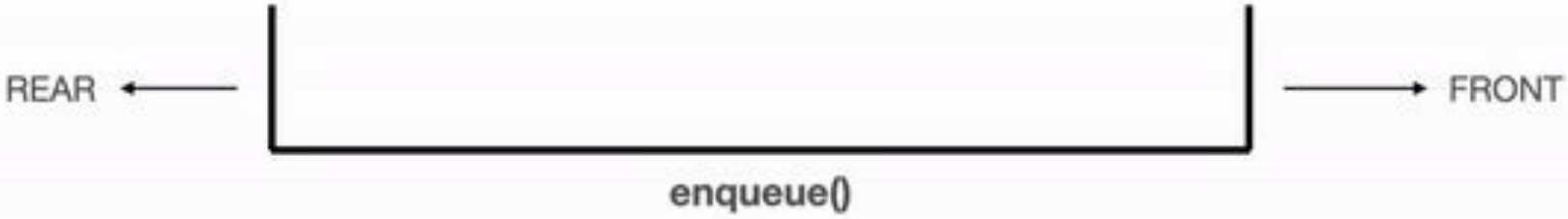
FILA

A estrutura de dados da fila é uma coleção ordenada de elementos onde o primeiro elemento inserido é o primeiro a ser removido, seguindo o princípio Primeiro a Entrar, Primeiro a Sair (FIFO).

FILA

- **Frente:** Representa o elemento que está na fila há mais tempo e será o próximo a ser removido.
- **Traseiro:** Também conhecido como “cauda”, representa o elemento mais recente adicionado à fila.


5
4
3
2
1
REAR
FRONT
dequeue()


REAR
FRONT
enqueue()

FILA

- **Ordem:** A ordem de inserção na fila é importante e mantida.
- **Implementações baseadas em arrays:** Usando arrays, um buffer circular pode ser empregado para implementar uma fila com eficiência. Isso permite complexidade de tempo constante para operações de enfileiramento e desenfileiramento.

FILA

- **Implementações baseadas em listas encadeadas:** Usando uma lista encadeada, a Fila pode ser implementada mantendo uma referência aos elementos da frente e de trás. Isso permite inserção e remoção eficientes em ambas as extremidades da lista encadeada.

FILA

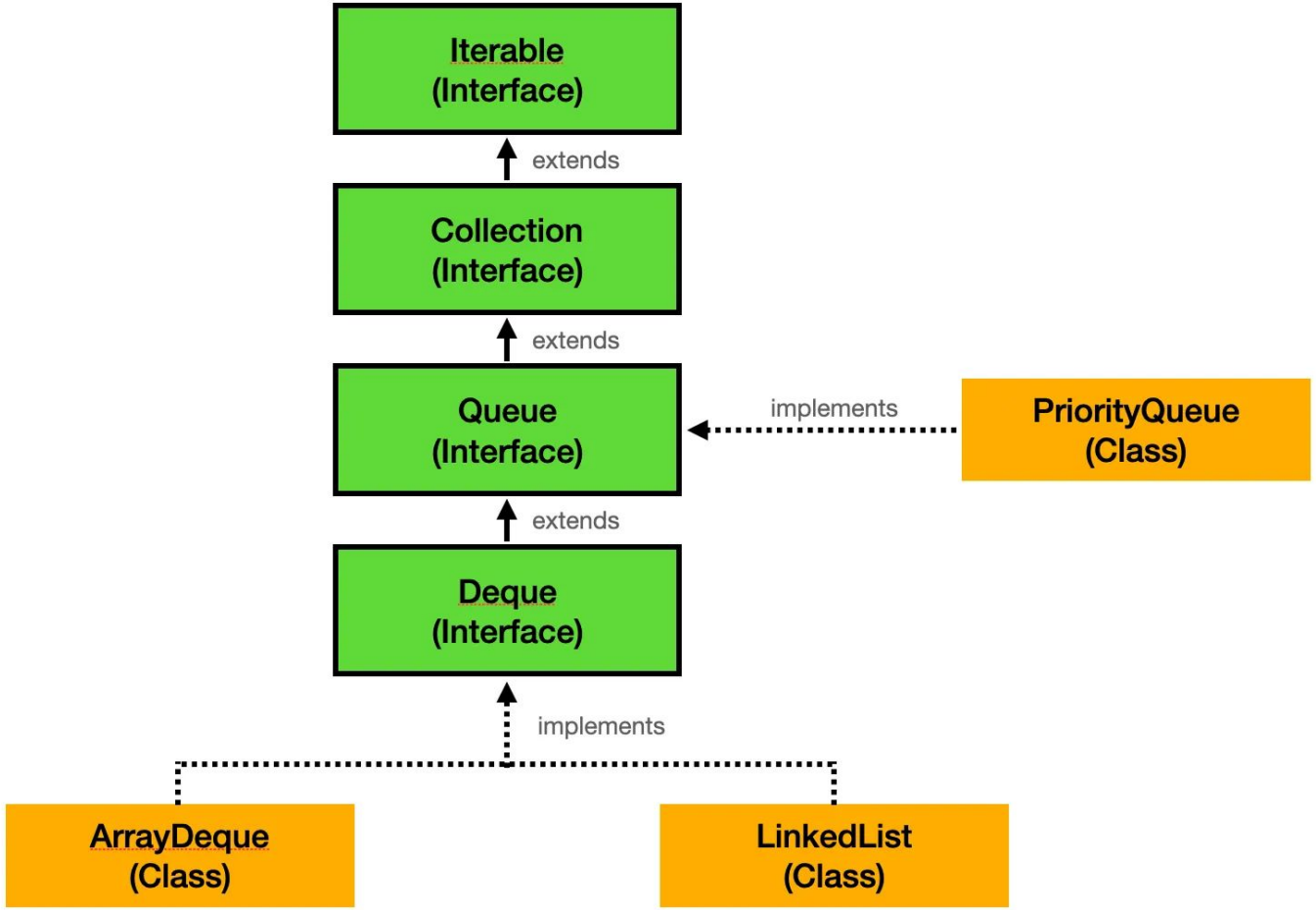
- **Capacidade:** A capacidade de uma fila refere-se ao número máximo de elementos que ela pode conter. Em muitas implementações de filas, não há um limite de tamanho fixo, permitindo que a fila cresça dinamicamente à medida que os elementos são adicionados. Isso significa que a capacidade da fila pode aumentar ou diminuir conforme necessário. No entanto, algumas implementações podem oferecer filas com capacidade restrita, onde o número máximo de elementos é predefinido. Nesses casos, é importante gerenciar a capacidade cuidadosamente para evitar exceder o limite e encontrar problemas como estouro.

FILA

- **Exemplo prático:** Um servidor com um número limitado de acessos, para cada usuário que fica de fora do servidor cria-se uma fila, conforme o espaço do servidor vai sendo liberado, os primeiros da fila vão saindo tendo acesso ao servidor.

IMPLEMENTANDO

- Em Kotlin, essencialmente todas as classes que podemos usar para implementar **Stack** e **Queue** estendem duas interfaces: **Queue** e **Deque**.
- Mas hoje podemos usar algo muito melhor. **Lista Encadeadas**!

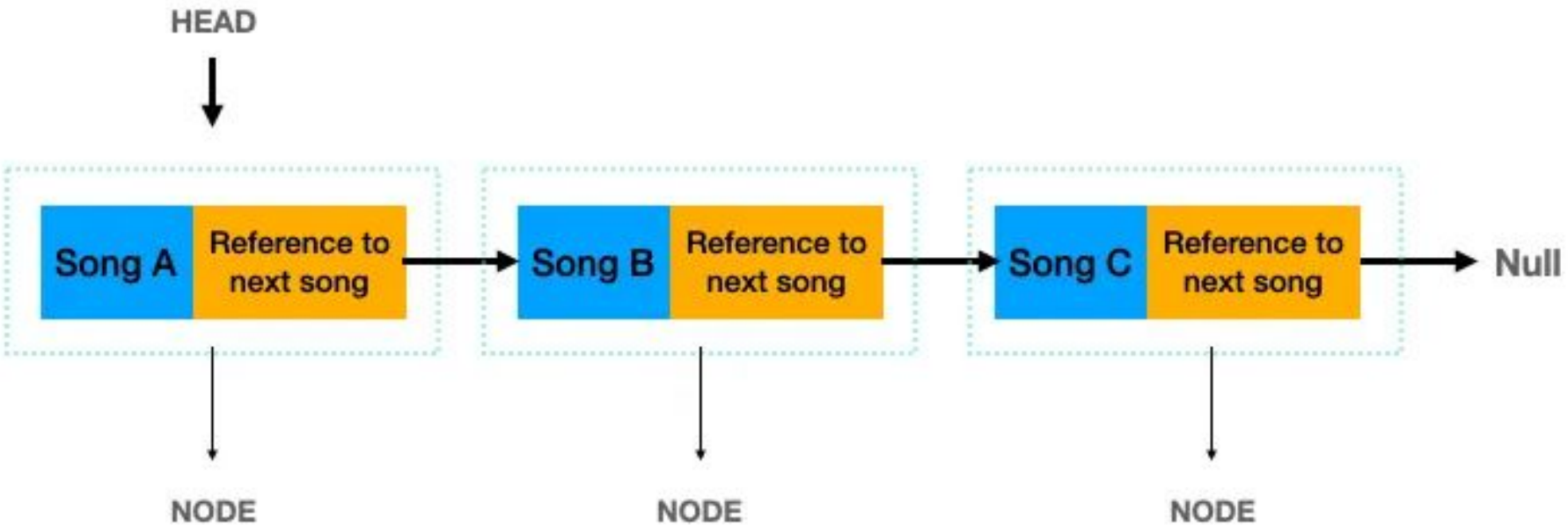

Iterable
(Interface)
extends
Collection
(Interface)
extends
Queue
(Interface)
implements
PriorityQueue
(Class)
extends
Deque
(Interface)
implements
ArrayDeque
(Class)
LinkedList
(Class)

LISTAS ENCADEADAS

- Imagine uma playlist de músicas que você gosta de ouvir. Depois que uma música termina de tocar, a próxima começa. Mas como a playlist sabe qual música vem a seguir? É como se cada música tivesse um "link" para a próxima. É basicamente assim que uma lista encadeada funciona.

LISTAS ENCADEADAS

- Um nó consiste em duas partes principais: os dados ou informações do item e uma referência ou link para o próximo item na lista.


HEAD
Song A
Reference to next song
Song B
Reference to next song
Song C
Reference to next song
Null
NODE
NODE
NODE

Exercícios

- Crie um programa que possua uma **PILHA** de objetos de uma classe chamada **Produto***(atributos: nome, validade e quantidade)*. A cada *X* segundos, *Y* objetos são removidos da PILHA, quando a pilha estiver vazia, carregue ela com mais objetos. Repita esse processo infinitamente.
  - Opcional: Quanto maior for a quantidade do objeto Produto, maior é o tempo para remover o próximo objeto da pilha.
  - Opcional *(nível hard)*: Faça a validade ser em minutos. Verifique se o produto já venceu com o passar do tempo. Cuidado ao inserir novos produtos!

Exercícios

- Crie um programa que possua uma **FILA** de objetos de uma classe chamada **Paciente** *(atributos: nome, idade, peso e cpf)*. Um determinado médico a cada *X* segundo irá chamar os pacientes da FILA, quando a FILA estiver vazia o médico deve ficar de prontidão para atender o paciente recém chegado. De tempo em tempo abasteça a fila com novos pacientes, infinitamente.
    - Opcional: calcule o IMC do paciente, quanto maior for o índice de massa corporal, maior será o tempo de atendimento do médico.

Dicas

- Use a função *random()* para criar valores, junto com intervalos. Há exemplos do uso desses conceitos no Github.
- Para fazer um loop infinito use um *while(true){ código aqui }*.
- Para gerar tempo (em milisegundos) use THREADs.
- Seja criativo!

MAPAS

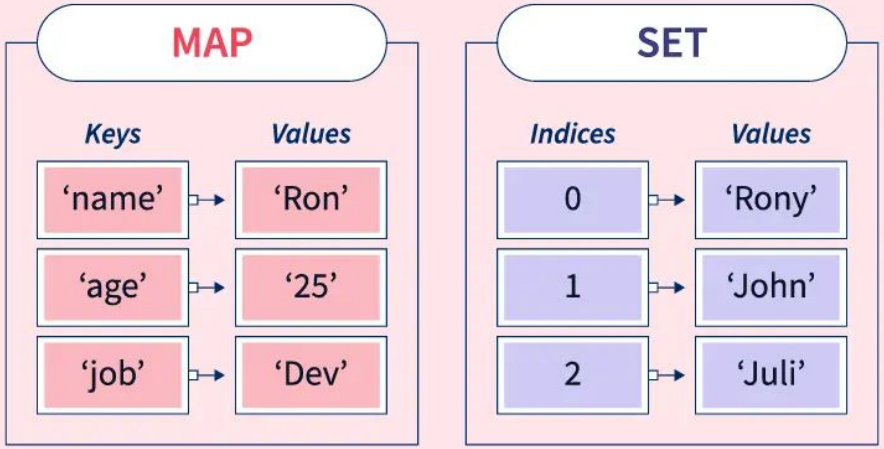
- Em computação, um Map é uma estrutura que guarda dados em pares de **chave** e **valor.**
- Um Map associa uma chave a um valor, assim encontrando elementos através de uma chave.

MAPAS

- Um Map não pode ter chaves duplicadas, cada chave se refere a pelo menos um valor.
- Em Kotlin um Map conecta uma chave a um valor que pode ser mapeado através dessa mesma chave. Você pode criar um Map incluindo um par de chave e valor para o método **mapOf( )**.

MAPAS

- Assim como nas Listas, temos Mapas Imutáveis e Mapas Mutáveis.
- Use **mapOf( )** quando for fazer um Mapa Imutável e **mutableMapOf()** para fazer um Mapa Mutável.


MAP
Keys
Values
'name'
'Ron'
'age'
'25'
'job'
'Dev'
SET
Indices
Values
0
'Rony'
1
'John'
2
'Juli'

Exercício

- Crie vários mapas de pokémons (151 no total). Veja a lista: https://bulbapedia.bulbagarden.net/wiki/List_of_Pok%C3%A9mon_by_Kanto_Pok%C3%A9dex_number
- A chave será o tipo do pokémon (ou tipos se houver 2), já o valor de cada chave será um array ou lista de Strings do qual o tipo é super efetivo (mais forte). https://bulbapedia.bulbagarden.net/wiki/Type

Dica

```
val nome_da_variavel = mapOf( chave : valor)

val bulbasaro = mapOf(
    “Grama” : listOf(“Água”, “Pedra”, “Terra”),
    “Veneno” : listOf(“Fada”, “Grama”)

)
```