Try-Catch, Sets e Queues

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Try and Catch
- Throw
- Set
- FIFO
- Queue
- LIFO
- Dequeue

Try and Catch

Try and Catch são blocos de código, o Try é o bloco de execução normal e o Catch o é o bloco de execução que acontece caso haja uma falha na execução do bloco Try.

No Java os erros são chamados de Exceptions, cada Exception é uma classe com diversos métodos que nos permitem saber qual foi o erro que tivemos, onde ele se encontra e nos levam a pensar nas soluções para esses erros

Podemos mais de um Catch por Try, sendo assim podemos ter vários tipos de erro por Try, mas cada tipo de erro pode ter um tratamento diferente.

Try and Catch

Vamos ver alguns casos do Try-Catch.

O exemplo abaixo não vai gerar uma Exception, pois não há erro no código:

```
try {
    System.out.println("código normal");
}catch (Exception exception){
    throw exception;
}
```

O exemplo abaixo vai gerar uma RuntimeException, pois estamos forçando a existência de uma Exception:

```
try {
    System.out.println("código normal");
    throw new Exception();
}catch (Exception exception){
    System.out.println(exception.getMessage());
}
```

Try and Catch

No código abaixo temos um erro muito comum que é a divisão por 0, que vai dar erro pois a divisão por 0 tende ao infinito, e por isso não pode ser executado por um computador:

```
try {
  int a = 0;
  int b = a/a;
}catch (Exception exception){
   System.out.println(exception.getMessage());
   System.out.println(exception.getCause().toString());
}
```

Acima vemos que podemos realizar o print das informações que existem dentro da exception, podemos também realizar um throw dentro do bloco Catch.

Throw

O throw para a execução do código e retorna um valor, com a diferença que o Java sempre trata exceções a parte do código.

Sempre que temos um erro ele é demonstrado no terminal e o código não, sendo assim é algo que nos fala sobre o código, onde temos um erro e que tipo de erro é esse.

Abaixo temos um exemplo:

```
throw new Exception();
```

```
Exception in thread "main" java.lang.Exception
        at aula2.exemplo3.Main.main(Main.java:17)
```

Set

O Set, ou Conjuntos, no Java é uma interface que representa os conjuntos que temos na dentro da matemática, sendo assim eles não tem repetição de valores, também são desordenados.

Então podemos pensar no conjunto dos números inteiros, esse tem todos os número que existem, mas não tem nenhum número que tenha casas decimais.

No caso do Java podemos ter um conjunto de Strings, ou de Inteiros, ou de qualquer outro tio. Mas a diferença para um Array é que um Set não tem itens repetidos, pois o Set Utiliza Hash e Árvore para armazenar seus valores.

Um Set pode ter várias forma de ser implementado, mas vamos focar no HashSet e no TreeSet

HashSet

HashSet é uma implementação de Sets baseada em Hash Functions, que organizam informações através de fórmulas matemáticas, sendo assim não podemos ter 2 informações idênticas pois elas ficariam no mesmo “lugar”.

```
Set<String> stringSet = new HashSet<>();

stringSet.add("a");
stringSet.add("a");
stringSet.add("b");
stringSet.add("c");
stringSet.add("c");
stringSet.forEach(System.out::println);
System.out.println(stringSet.contains("c"));
```

Sorted Set

A interface SortedSet contém todos os métodos de Set, adicionando que os dados podem ser ordenados. Sendo assim a maior diferença é a presença de funções como a first() e a last(), que respectivamente retornam o primeiro e o último elemento de um set.

```
SortedSet<String> stringSet = new TreeSet<>();

stringSet.add("a");
stringSet.add("a");
stringSet.add("b");
stringSet.add("c");
stringSet.add("c");
stringSet.forEach(System.out::println);
System.out.println(stringSet.contains("c"));
System.out.println(stringSet.first());//a
System.out.println(stringSet.last());//c
```

Navegable Set

A interface NavigableSet herda da interface SortedSet, em adição ao mecanismo de organização o NavigableSet também adiciona métodos de navegação, sendo assim podemos acessar o set de forma ascendente ou descendente(invertendo a ordem do set).

```
NavigableSet<Integer> integerNavigableSet = new TreeSet<>();
        integerNavigableSet.add(0);
        integerNavigableSet.add(1);
        integerNavigableSet.add(2);
        integerNavigableSet.add(3);
        integerNavigableSet.add(4);
        integerNavigableSet.add(5);
        integerNavigableSet.add(6);

        // Criando um set na ordem reversa do integerNavigableSet
        NavigableSet<Integer> reverseIntegerNavigableSet = integerNavigableSet.descendingSet();

        // Print the normal and reverse views
        System.out.println("Ordem ascendente: " + integerNavigableSet);
        System.out.println("Ordem descendente: " + reverseIntegerNavigableSet);
```

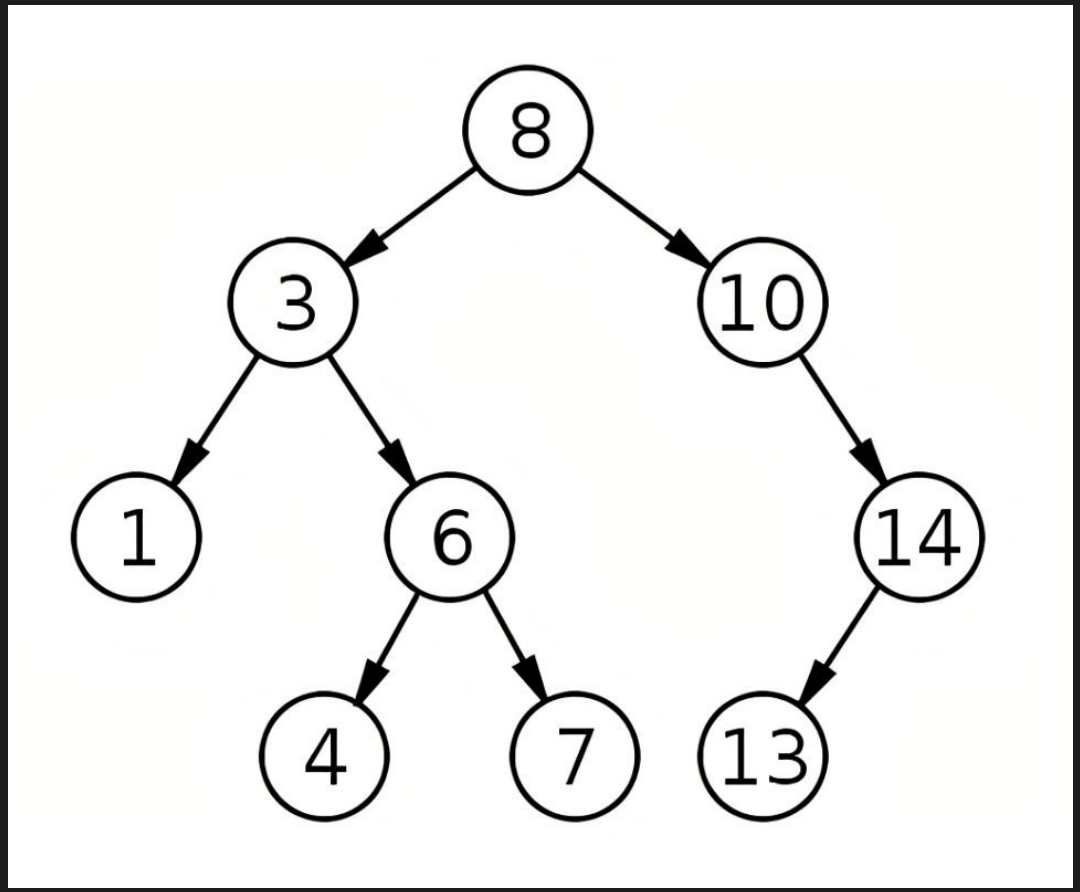
TreeSet

O TreeSet é uma classe que herda da interface NavigableSet, sua implementação é baseada na estrutura em árvore, onde cada elemento é uma das folhas das árvores.

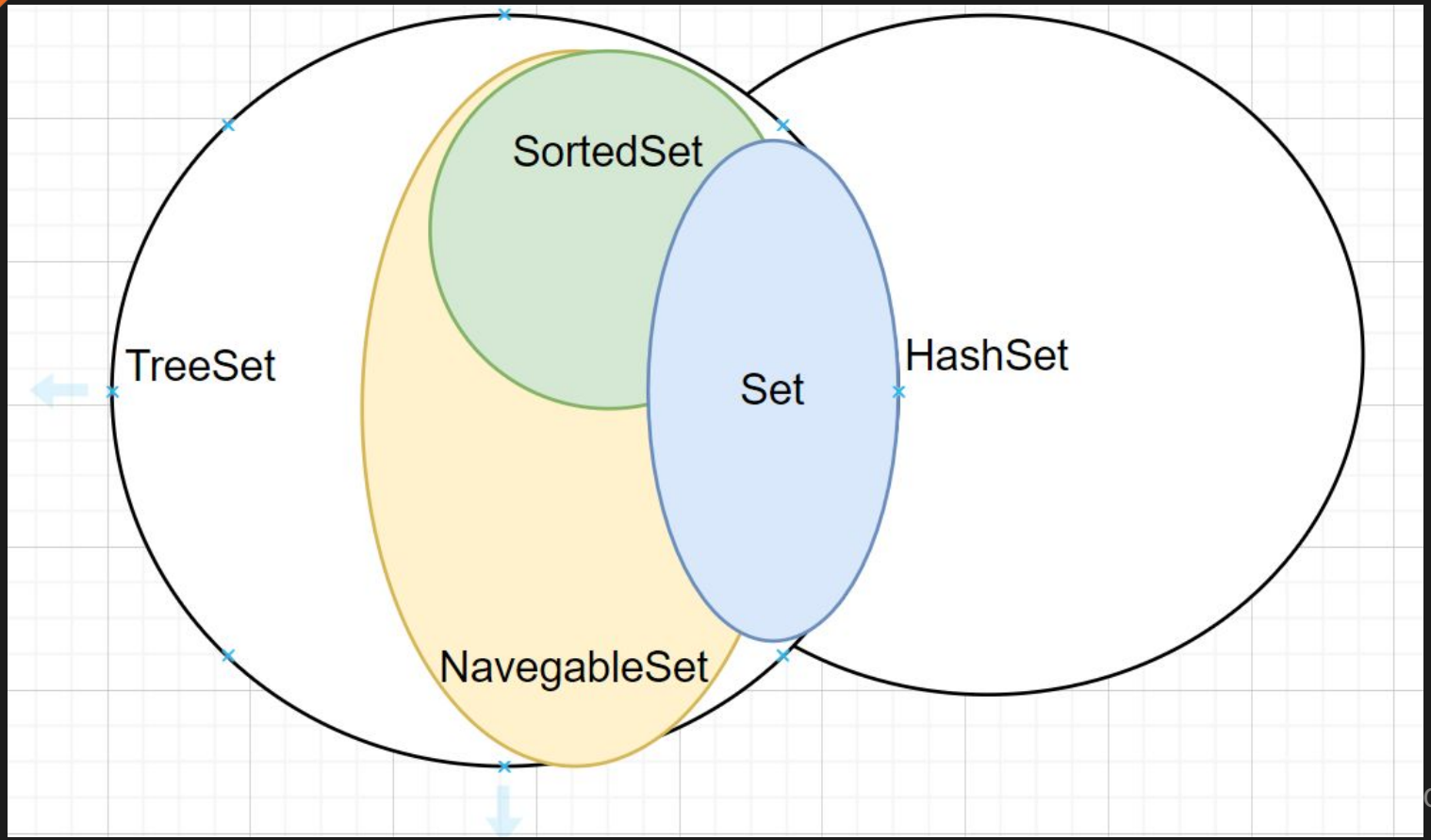
A ordenação é feita utilizando a ordenação natural (ascendente ou alfabética dependendo do tipo).

```
TreeSet<String> stringTreeSet = new TreeSet<>();

 stringTreeSet.add("A");
 stringTreeSet.add("B");
 stringTreeSet.add("C");
 stringTreeSet.add("C");

 // Ordem do Set Ascendente(Ascending)
 System.out.println(stringTreeSet);
```

TreeSet



TreeSet



INTERVALO DE AULA

DEV!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

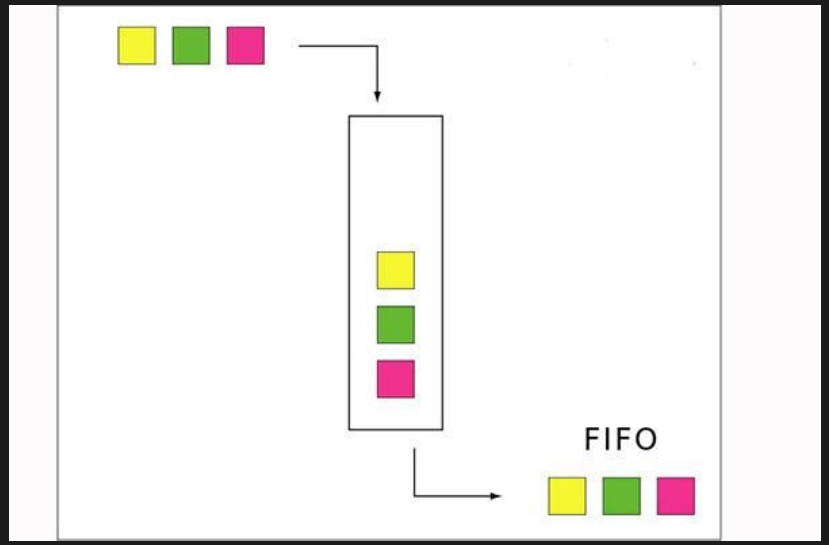
Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40

FIFO

FIFO quer dizer “First In First Out”, primeiro a entrar é o primeiro a sair. Essa é uma das formas de representarmos filas na programação, então todo elemento que entrar em uma fila deve sair na ordem em que entrou.

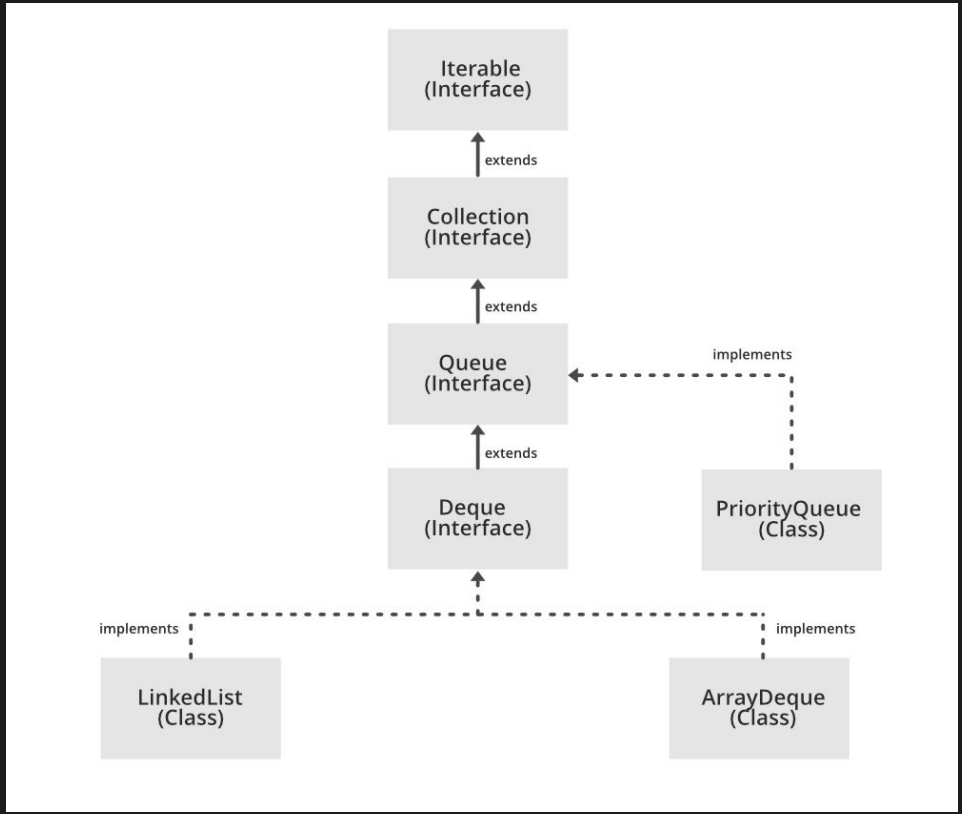
Podemos utilizar filas para tráfego de dados, assim os dados que saem de um lado do programa chegam em outro na ordem que desejamos.



QUEUE

No java a Queue é uma interface, sendo assim precisamos de uma classe para declará-la, essa classe pode ser uma PriorityQueue ou uma LinkedList.

Podemos utilizar LinkedLists devido a natureza dele de ser fácil de adicionar itens no início e no fim dele, e também ambos as Filas e as LinkedLists são “parentes” dentro do Java e podem ser utilizados juntos.



Queue

A PriorityQueue adiciona prioridade aos elementos de uma lista, sendo assim elementos mais prioritários podem “furar a fila” e serem expostos primeiro para serem lidos ou removidos. Por padrão a ordem e a ordem natural, ou seja ascendente ou alfabética dependendo do tipo dos elementos da fila.

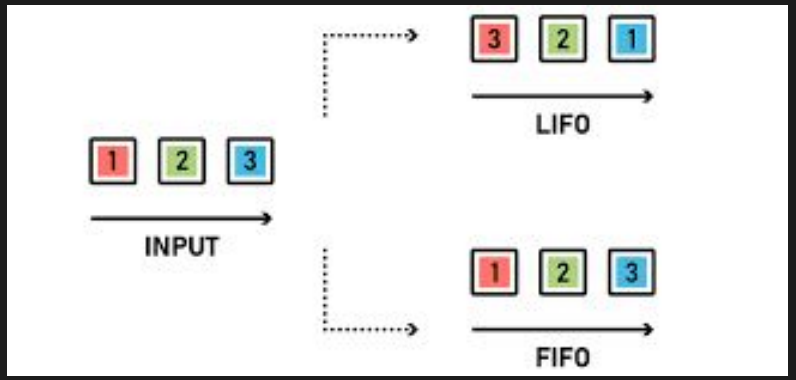
```
PriorityQueue<Integer> integerQueue = new PriorityQueue<>();

integerQueue.add(9);
integerQueue.add(2);
integerQueue.add(4);

int first = integerQueue.poll();
int second = integerQueue.poll();
int third = integerQueue.poll();

System.out.println("first:" + first);
System.out.println("second:" + second);
System.out.println("third:" + third);
```

Queue

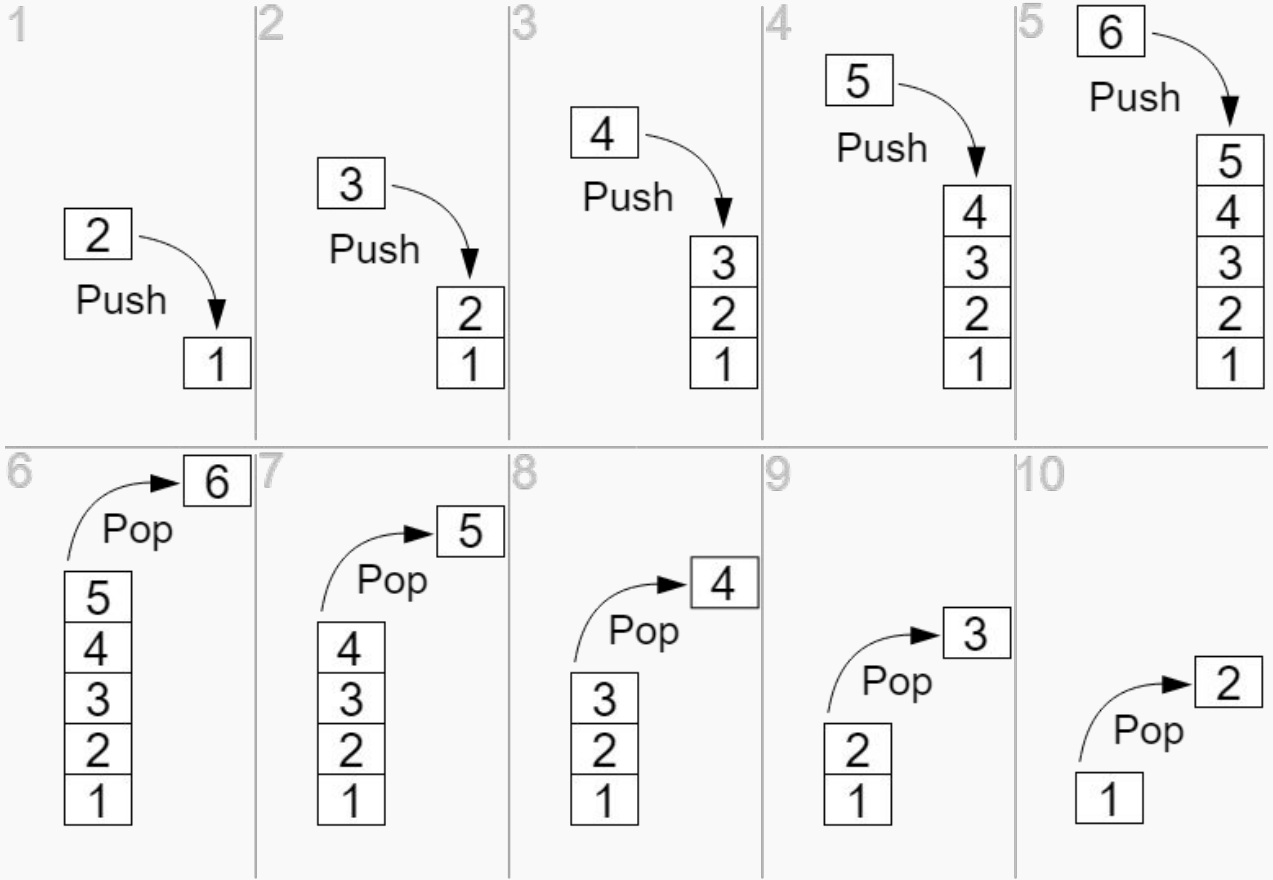
```
Queue<Integer> integerQueue = new LinkedList<>();
 // adiciona elementos de 0 a 4 na fila
 for (int i = 0; i < 5; i++)
     integerQueue.add(i);
 System.out.println("Elements of queue " + integerQueue);
 // removendo o primeiro intem da fila
 int removedItem = integerQueue.remove();
 System.out.println("Elemento Removido: " + removedItem);
 System.out.println(integerQueue);
 // Visualizando o primeiro elemento da fila
 int head = integerQueue.peek();
 System.out.println("Primeiro elemento da fila: "
         + head);

 integerQueue.add(5);//item adicionado ao fim da lista
 System.out.println(integerQueue);
```

LIFO

LIFO quer dizer “Last In First Out”, ou o último a entrar é o primeiro a sair, essa estrutura gera pilhas, ou seja você apenas consegue remover o último item adicionado. Isso garante um recebimento dos dados mais recentes para os mais antigos. Dependendo do programa essa estrutura pode ser melhor do que uma FIFO.



AGENDA


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
Pop
6
5
4
3
2
1
7
Pop
5
4
3
2
1
8
Pop
4
3
2
1
9
Pop
3
2
1
10
Pop
2
1

DeQueue

A interface Deque(Double ended queue - fila com dois fins) herda da interface Queue, porém ela suporta adição e retirada de elementos das duas pontas. Sendo assim pode atuar tanto como fila quanto como Pilha, sendo que ela é mais recomendada do que a classe Stack, que tem o problema de ser muito lenta e pesada em relação a Deque.

O padrão de Pilha segue a ordem Last In First Out - o Primeiro a Entrar é o Primeiro a Sair.

Para isso adicionamos no início da pilha e depois removemos os elementos começando pelo primeiro da fila e seguindo até o último elemento. Nesse caso o último elemento é o que foi adicionado primeiro a pilha.

DeQueue

A ArrayDeque é uma classe que implementa a interface Deque e a interface Queue, sendo assim tem os mesmos métodos do Deque, e nos permite realizar tanto operações FIFO quanto LIFO.

```
    Deque<String> stringArrayDeque
                = new ArrayDeque<String>();


        stringArrayDeque.add("AA");
        stringArrayDeque.addFirst("BB");
        stringArrayDeque.addFirst("BA");
        stringArrayDeque.addFirst("BC");
        stringArrayDeque.addFirst("BD");
        stringArrayDeque.addLast("CC");


        System.out.println(stringArrayDeque);
        System.out.println(stringArrayDeque.pop()); // retorna e remove o primeiro elemento da Deque
        System.out.println(stringArrayDeque.poll());// retorna e remove o primeiro elemento da Deque
        System.out.println(stringArrayDeque.pollFirst()); // retorna e remove o primeiro elemento da
Deque
        System.out.println(stringArrayDeque.pollLast());
```

DeQueue

Para fazemos uma pilha, ou seja uma operação LIFO, podemos utilizar o seguinte código:

```
Deque<String> stringArrayDeque
              = new ArrayDeque<String>();
      stringArrayDeque.addFirst("AA");
      stringArrayDeque.addFirst("BB");
      stringArrayDeque.addFirst("BA");
      stringArrayDeque.addFirst("BC");
      stringArrayDeque.addFirst("BD");
      stringArrayDeque.addFirst("CC");

      System.out.println(stringArrayDeque);

      int tamanhoOriginal = stringArrayDeque.size();
      for (int i=0; i <= tamanhoOriginal;i++) {
          System.out.println(stringArrayDeque.pollFirst());
      }
```

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>