ESTRUTURAS

SENAI <LAB365>

SUMÁRIO

OS VETORES EM JAVA

Vetores, também conhecidos por array, são elementos que armazenam um determinado tipo de dado sendo referenciado por um índice de posições. Esses elementos são agrupados dentro de uma variável, e cada elemento também pode ser considerado uma variável.

Todos os elementos dentro de um array são do mesmo tipo, seja esse tipo um inteiro (int, ou Integer), seja ele uma classe criada no programa.

Confira a seguir o que são arrays primitivos.

Arrays de tipos primitivos

No Java 8, podem ser criados arrays de tipos primitivos, que incluem tipos como *byte*, *short*, *int*, *long*, *boolean*, *char*, *float* e *double*. Vale ressaltar que estes arrays não podem ter seu tamanho alterado uma vez que setado.

A seguir, você confere alguns exemplos, utilizando vetores na linguagem Java.

```
int[] arrayNumeros = new int[5]; //array de tipo primitivo int que
contém 5 posições
arrayNumeros[0] = 10; // todo índice de Arrays começa a contar a
partir do 0
arrayNumeros[1] = 11;
arrayNumeros[2] = 12;
arrayNumeros[3] = 13;
arrayNumeros[4] = 14;// como temos 5 posições e o programa começa
em 0 temos o ultimo elemento com o índice 4

System.out.println("%d",arrayNumeros[1]);
System.out.println("%d",arrayNumeros[2]);
System.out.println("%d",arrayNumeros[3]);
```

```
int[] arrayNumeros2 = {10,20,30,40,50}; // criarmos um array já com
os valores e com as posições preenchidas

System.out.println("%d",arrayNumeros2[4]);

for (int i=0; i < 5 ;i++){
    System.out.println("%d",arrayNumeros2[i]);
}

for (int i=0; i < 10 ;i++){ //vamos ter um erro de quebra de
                            //borda, out bound
    System.out.println("%d",arrayNumeros2[i]);
}

for (int i=0; i < arrayNumeros2.length ;i++){
    System.out.println("%d",arrayNumeros2[i]);
}

int[] arrayNumeros3 = arrayNumeros2.clone();
```

Em seguida, aprenda mais sobre operações de arrays em Java.

Operações

Há algumas operações para manipulação de arrays no Java, confira a seguir.

- Como podemos ver, os arrays em Java nos permitem armazenar valores como se estivessem em linha. Podemos também descobrir quantos registros um array tem, através da variável array.lenght, essa está armazenada como parte do array.
- Também podemos realizar a duplicação de um array, através do método array.clone().
- Para iteramos através de um array, precisamos utilizar um método de *loop*, no caso estamos utilizando o For loop, que nos permite iniciar um índice que vai aumentar cada vez que o código do *loop* é executado.

As matrizes são outro item importante. Entenda mais, em seguida.

MATRIZES EM JAVA

Matrizes são vetores direcionais, ou seja, há mais de um índice, dessa forma podemos achar valores através da combinação desses índices. Matrizes seguem as mesmas regras que os arrays e podem ser criadas de maneira parecida, sendo que agora temos a adição de um índice para as colunas.

> Pode-se dizer que a matriz é um vetor de vetores, onde para cada elemento do primeiro vetor temos um segundo vetor; assim, precisamos dos índices de ambos para chegar ao valor desejado.

Confira a seguir um exemplo. Se pensarmos em termos de colunas (representada por j) e linhas (representada por i), podemos ter a seguinte representação:

	j = 0	j = 1
i = 0	1	2
i = 1	3	4

Quadro 1 - exemplo de matriz

Fonte: Do Autor (2022)

Sendo assim, temos a linha i e a coluna j, ou seja, j é o índice do vetor de vetores e i é o índice dos vetores dentro de j. No exemplo anterior, se quisermos pegar o número 4, vamos ir para a linha i = 1 e coluna j = 1.

```java
class Main{
  public static void main(String[] args){

    int[][] matrizNumeros = new int[2][2];


    matrizNumeros[0][0] = 1;
    matrizNumeros[0][1] = 2;
    matrizNumeros[1][0] = 3;
    matrizNumeros[1][1] = 4;

    for(int j = 0; j< matrizNumeros.length;j++){
    //podemos pegar o tamanho do vetor correspondente a "linha" 0
      for (int i = 0; i < matrizNumeros[0].length; i++){
        System.out.println(matrizNumeros[j][i]);
      }
    }

    int[][] matrizNumeros2 = matrizNumeros.clone();

    for(int j = 0; j< matrizNumeros2.length;j++){
    //podemos pegar o tamanho do vetor correspondente a "linha" 0
      for (int i = 0; i < matrizNumeros2[0].length; i++){
        System.out.println(matrizNumeros2[j][i]);
      }
    }
  }
}
```

Operações

Como você já conferiu, há o mesmo método clone e o mesmo atributo *length* para ambos os vetores, tanto o vetor de vetores quanto para os vetores internos.

Aqui você utilizará o tamanho do primeiro vetor, matrizNumeros2.length, e o tamanho do primeiro vetor interno (que tem o mesmo tamanho que os demais), matrizNumeros2[0].length.

Bacana esse exemplo, não é mesmo? Que tal você aprofundar mais seus conhecimentos sobre coleções em Java? Continue e a seguir entenda tudo sobre o assunto.

COLEÇÕES

Coleções são objetos que agrupam outros objetos, e esses elementos podem ser de qualquer tipo dentro de um programa Java. Compreenda mais sobre suas características a seguir.

Características de uma coleção

Dentro do Java temos a interface Collection que é pai de outras três interfaces que são: Set, List e Queue. A Collection em si não tem implementação, é apenas uma interface, o que permite que qualquer implementação de List, Set ou de Queue seja usada com ela.

A Collection tem um número de métodos que podem ser utilizados com ela. Para acessarmos esses métodos, precisamos criar uma Collection e uma implementação de um de seus descendentes.

Confira um exemplo a seguir:

```java
Collection<String> stringCollection = new ArrayList<>();

stringCollection.add("1");
stringCollection.add("2");
stringCollection.add("3");
stringCollection.add("4");
stringCollection.add("");

stringCollection.forEach(itemCollection ->
System.out.println(itemCollection));
System.out.println("Fim da execução");
```

No exemplo, estamos utilizando um ArrayLis*t* que é uma implementação de List, que por sua vez herda as operações de Collection. Quando realizamos essa instância de ArrayList como Collection ficamos limitados aos métodos que existem dentro de Collection.

Podemos realizar o.add() para adicionarmos novos objetos, nesse caso Strings, como está definido na instância da stringCollection, e também podemos utilizar o .forEach(), para executarmos uma lambda function, que é uma função que é executada sobre todos os elementos da Collection.

Operações

Além dos métodos do exemplo anterior, você pode utilizar também:

```java
stringCollection.addAll(new ArrayList<String>());
System.out.println(stringCollection.contains("1"));
System.out.println(stringCollection.size());
System.out.println(stringCollection.isEmpty());
System.out.println(stringCollection.remove("1"));
stringCollection.removeAll(new ArrayList<String>());
stringCollection.removeIf(s -> s.isBlank());
stringCollection.iterator();
stringCollection.spliterator();
stringCollection.clear();
```

Em seguida, confira mais sobre heranças e como elas se relacionam com Collections.

HERANÇA

Analise o esquema apresentado a seguir em que você tem um desenho das heranças que pode ocorrer a partir de uma Collections. Ali, você também pode perceber que a Collection herda de uma interface chamada Iterable, essa apenas nos permite ver o index(iterator) de uma Coleção.

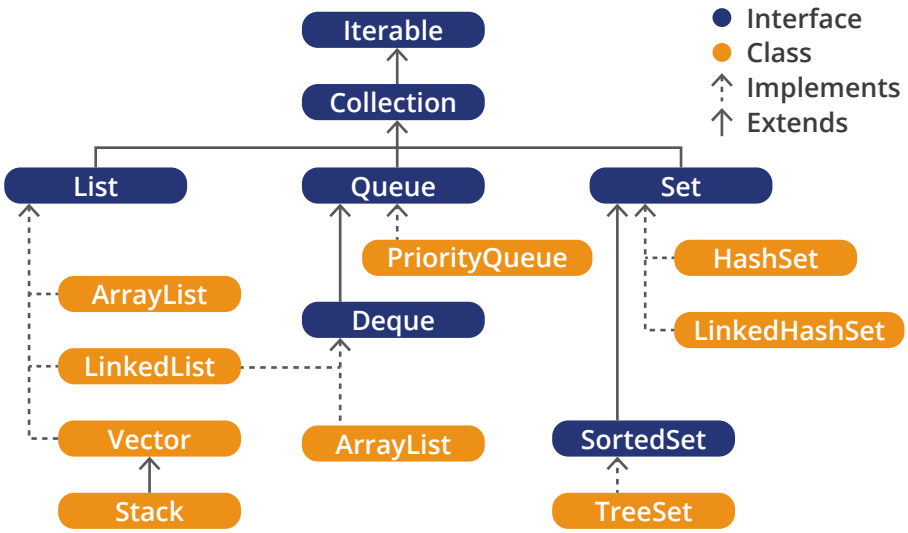


Figura 1 - Exemplo de herança

Fonte: do Autor (2022)

Percebeu no desenho um item chamado List? Confira o que ele significa a seguir.

List (Listas)

List é uma interface do Java que herda Collection e nos permite realizar mais operações do que Collections. Temos algumas implementações de List dentro do Java.Util, entre elas pode-se destacar: ArrayList, LinkedList, Vector e Stack.

No geral, Vector e Stack são tipos depreciados e lentos, sendo assim você pode focar nos demais. Stack pode ser substituído pelo Deque, que faz parte da família das Queue (Filas).

Em seguida aprenda mais sobre os ArraysList.

ArrayList

As ArrayList são arrays que podem ter seu tamanho alterado. São baseados nos arrays que você já estudou anteriormente e permitem criar listas de item que tem os mesmos métodos de Collection, e ainda métodos que nos permitem acessar itens específicos através de índices.

Elas herdam de List, sendo assim podemos instanciar uma List como uma ArrayList, ou podemos apenas instanciar uma ArrayList como qualquer outra Classe, confira:

```
public static void main(String[] args){

  List<String> stringList = new ArrayList<>();
  ArrayList<String> stringArrayList = new ArrayList<>();

}
```

Entenda que ambas as formas criam ArrayLists, porém quando criamos um tipo List podemos utilizar qualquer Classe que herda de List nessa variável, sendo assim é mais indicado para criar retorno de métodos e parâmetros também.

Você pode perceber também que como já temos o tipo da List<String>, nesse caso String, sendo assim não precisamos colocar o tipo da lista na instanciação, ficamos apenas com o operador diamante, como o '<>' é conhecido. A mesma coisa acontece com o tipo ArrayList.

No caso acima tanto o stringList quanto o stringArrayList têm os mesmos métodos e funcionam de forma idêntica.

Agora, analise o caso a seguir:

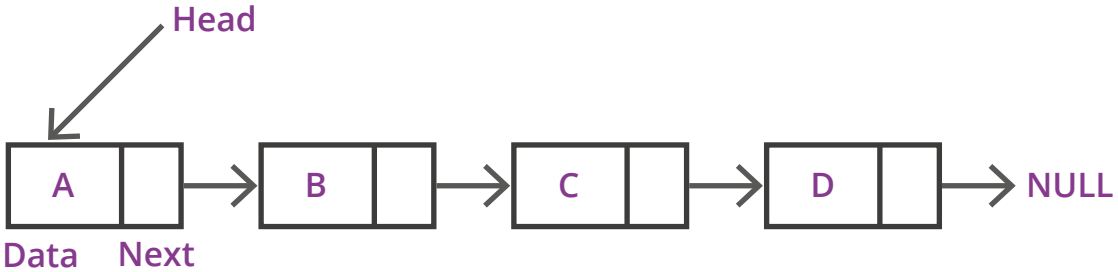
```java
public static void main(String[] args){
    List<String> stringList = new ArrayList<>();

    stringList.add("e");
    stringList.add("f");
    stringList.add("a");
    stringList.add("b");
    stringList.add("c");
    stringList.add("d");

    //método que nos pemite realizar um ordenação por meio
    //da Classe Comparator
    stringList.sort(Comparator.naturalOrder());

    stringList.forEach(System.out::println);

    //podemos acessar os itens por meio de um index
    System.out.println(stringList.get(2));
    System.out.println(stringList.set(2, "ab"));
    stringList.remove(0);
    System.out.println(stringList.size());
    stringList.clear();

}
```

Acima, podemos entender alguns dos métodos da List que demonstram as operações baseadas em índice. Temos, também, a operação de sort(), que nos permite realizar a ordenação de uma lista baseada em uma ordem.

Outra implementação da List é o LinkedList. É sobre ela que você estudará em seguida.

LinkedList

LinkedLists herdam de List, porém têm uma diferença, que é a estrutura que elas representam. Confira o esquema:



Cada item adicionado à lista tem a referência para o próximo elemento, e o último deles aponta para Null, indicando o fim da lista. Sendo assim, é fácil identificarmos o primeiro e o último elemento de uma lista, também é fácil mudar ou adicionar elementos no final da lista; porém, o meio da lista é mais lento de ser alterado, e a lista é mais demorada para achar elementos que estejam no meio.

De forma geral, ArrayLists são muito bons para serem lidos, porém não são os melhores se tratando de adicionar novos elementos. Já as LinkedLists, são muito boas para adicionar itens, mas não muito boas para encontrar elementos da lista.

Confira um exemplo de código com LinkedList.

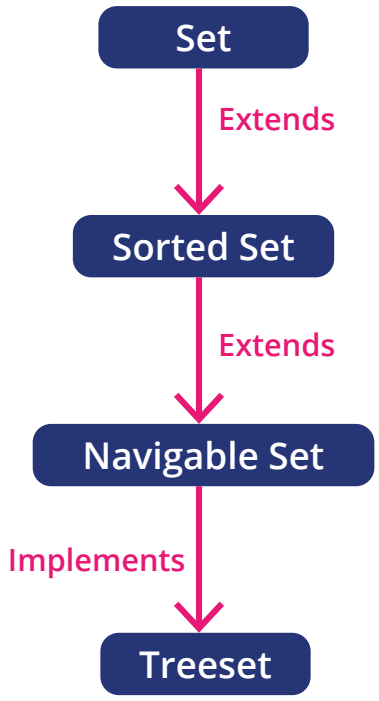
```
public static void main(String[] args){
  List<String> stringLinkedList = new LinkedList<>();

  stringLinkedList.add("a");
  stringLinkedList.add("b");
  stringLinkedList.add("c");
  stringLinkedList.add("d");

  stringLinkedList.contains("b");

  //pegando o ultimo item da lista
  System.out.println(stringLinkedList.get(stringLinkedList.size()-1));
}
```

A seguir aprenda mais sobre outro importante elemento em Java, o *Set*.

SET

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

Há duas interfaces que herdam de Set, SortedSet *e* NavigableSet. Confira o organograma a seguir.



Além das interfaces, temos uma Classes que herda diretamente de Set, HashSet. Aprenda mais sobre essa classe em seguida.

HashSet

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

Confira um exemplo de código com essa aplicação.

```java
public static void main(String[] args){

      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"));

}
```

Outra interface importante em Java é a SortedSet. Entenda mais sobre ela, a seguir.

SortedSet

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. Confira um exemplo.

```java
public static void main(String[] args){
      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

}
```

No caso de Strings o SortedSet ordena-se na ordem alfabética. Aprenda mais sobre NavigableSet em seguida.

NAVIGABLESET

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). Analise os códigos a seguir:

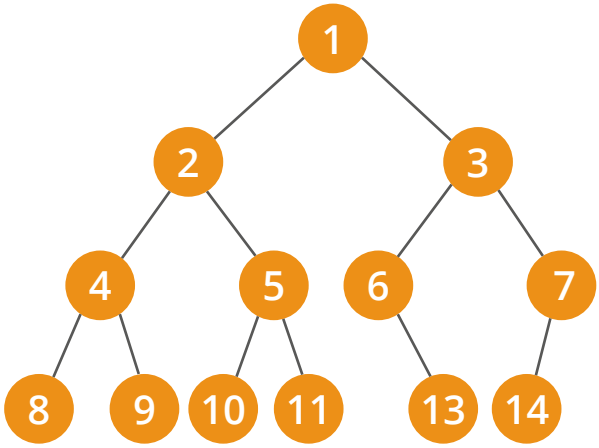
```
public static void main(String[] args){
      NavigableSet<Integer> integerNavigableSet = new TreeSet<>();
      integerNavigableSet.add(0);
      integerNavigableSet.add(1);
      integerNavigableSet.add(2);
      integerNavigableSet.add(3);
      integerNavigableSet.add(4);
      integerNavigableSet.add(5);
      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);


}
```

Você pode perceber que temos o método integerNavigableSet.descendingSet(), que gera uma cópia do primeiro Set, porém invertido na ordem dos seus elementos.

A partir de uma NavigableSet, pode surgir uma TreeSet, portanto, estude mais sobre essa classe no próximo item.

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.



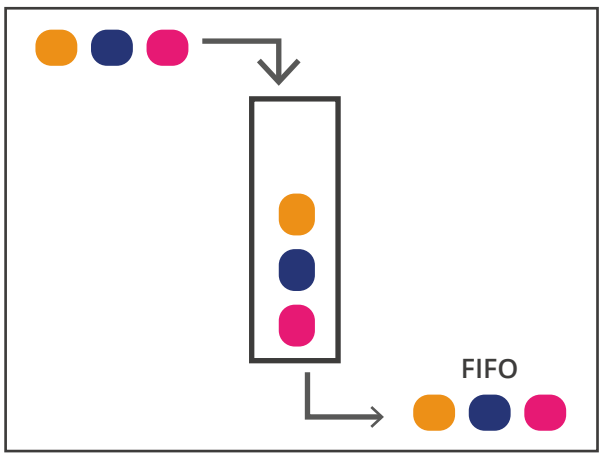
A ordenação é feita utilizando a ordenação natural (ascendente ou alfabética, dependendo do tipo). Confira o exemplo:

```
public static void main(String[] args){

      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);

}
```

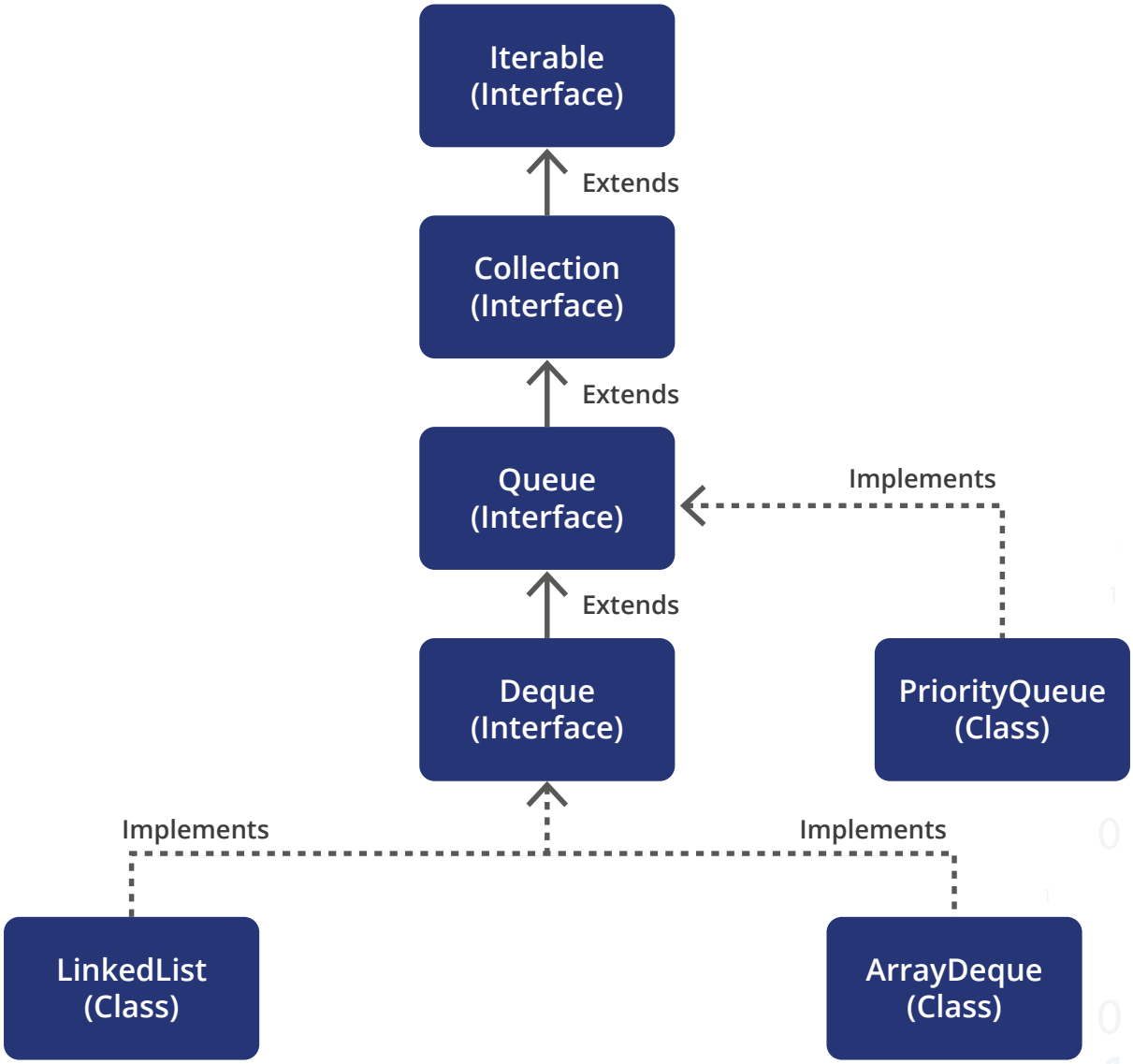
A seguir, aprenda mais sobre a Queue e a sua importância na linguagem Java.

QUEUE

As filas agregam elementos em uma ordem específica, sendo na ordem First In First Out (FIFO) - Primeiro a Entrar Primeiro a Sair. Sendo assim, todos os elementos que entram na fila saem na mesma ordem em que entraram, e cada elemento novo entra no fim da fila, entenda mais com o esquema:



No Java, a **Queue** é uma interface, sendo assim você precisa de uma Classe para declará-la. Essa Classe pode ser uma PriorityQueue ou uma LinkedList. Há, também, a PriorityBlockingQueue, que funciona da mesma forma que a PriorityQueue mas para processos paralelos no Java, confira no organograma.



Em seguida, entenda mais sobre a Queue, a partir do código.

```
public static void main(String[] args){

      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);

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

Aprenda a adicionar prioridades, a seguir.

PriorityQueue

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, confira a seguir:

```java
public static void main(String[] args){

      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);

}
```

Em seguida, aprenda mais sobre a fila com dois fins (Deque).

Deque

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 à Deque.

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

```
public static void main(String[] args){

      Deque<String> stringDeque = new LinkedList<String>();

      // adiciona elemento ao final da deque
      stringDeque.add("Elemento 1 (Tail)");
      // adiciona elemento no inicio da deque
      stringDeque.addFirst("Elemento 2 (Head)");
      // adiciona elemento ao final da deque
      stringDeque.addLast("Elemento 3 (Tail)");
      // adiciona elemento no inicio da deque
      stringDeque.push("Elemento 4 (Head)");
      // adiciona elemento ao final da deque
      stringDeque.offer("Elemento 5 (Tail)");
      // adiciona elemento no inicio da deque
      stringDeque.offerFirst("Elemento 6 (Head)");

      System.out.println(stringDeque + "\n");

      // podemos remover o primeiro ou o último elemento também
      stringDeque.removeFirst();41
      stringDeque.removeLast();
      System.out.println("Deque after removing " + "first and last: "
      + stringDeque);
}
```

Estude agora um implemento à ArrayDeque.

ArrayDeque

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. Entenda, com o código a seguir:

```java
public static void main(String[] args){

      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);

      // retorna e remove o primeiro elemento da Deque
      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());

      System.out.println(stringArrayDeque.pollLast());
}
```

Confira em seguida, um LIFO com DEQUE.

LIFO com DEQUE

Para fazer uma Pilha, ou seja, uma operação LIFO, pode-se utilizar o seguinte código:

```java
public static void main(String[] args){
      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());
      }
}
```

Como você pode analisar pelo exemplo anterior, sempre que adicionarmos uma mensagem devemos colocá-la no início da Pilha e, depois, remover os elementos, começando pelo primeiro da fila e seguindo até o último. Nesse caso, o último elemento é o que foi adicionado primeiro à Pilha.

E então, gostou de conhecer mais sobre a linguagem Java? Sobre as diversas possiblidades de carreira nessa área tão promissora? Você pôde estudar diversos implementos e ainda analisar diversas operações que serão muito importantes para a continuidade de seus estudos! Continue seus aprendizados e aprofunde ainda mais os seus conhecimentos.

REFERÊNCIAS

Matriz e Vetores

CFB CURSOS. **Como criar Array/Vetor em Java:** Curso de Java. 2021. Disponível em: https://www.youtube.com/watch?v=-Hmzi-mhThc. Acesso em: 02 jun. 2022.

DEVMEDIA. **Matrizes:** aprenda a trabalhar com vetores bidimensionais. Disponível em: https://www.devmedia.com.br/matrizes-aprenda-a-trabalhar-com-vetores-bidimensionais-revista-easy-java-magazine-22/25766#modulo-mvp. Acesso em: 03 jun. 2022.

VANINI, F. **Curso de Java:** vetores e matrizes. 2008. Disponível em: https://www.ic.unicamp.br/~vanini/mc202/apresentacoes/Vetores%20e%20Matrizes.pdf. Acesso em: 06 jun. 2022.

Collections

DEVMEDIA. **Coleções Java:** Como usar Coleções. Disponível em: https://www.devmedia.com.br/java-collections-como-utilizar-collections/18450. Acesso em: 02 jun. 2022.

JAVATPOINT. **Coleções em Java.** 2021. Disponível em: https://www.javatpoint.com/collections-in-java#:~:text=The%20Collection%20in%20Java%20is,a%20single%20unit%20of%20objects. Acesso em: 03 jun. 2022.

NOLETO, C. **Java List, Set e ListIterator:** dominando as coleções Java! 2022. Disponível em: https://blog.betrybe.com/java/java-list-set-listiterator/. Acesso em: 02 jun. 2022.

VISUAL COMPUTER SCIENCE. **Java Collections Explained.** 2022. Disponível em: https://www.youtube.com/watch?v=viTHc_4XfCA. Acesso em: 06 jun. 2022.

List

DEVMEDIA. **Diferença entre ArrayList, Vector e LinkedList em Java.** 2022. Disponível em: https://www.devmedia.com.br/diferenca-entre-arraylist-vector-e-linkedlist-em-java/29162. Acesso em: 06 jun. 2022.

GEEKSFORGEEKS. **Estrutura de dados de lista vinculada.** 2022. Disponível em: https://www.geeksforgeeks.org/data-structures/linked-list/. Acesso em: 06 jun. 2022.

PANKAJ. **Lista Java:** lista em Java. 2022. Disponível em: https://www.journaldev.com/11444/java-list. Acesso em: 03 jun. 2022.

W3SCHOOLS. **Java ArrayList.** 2022. Disponível em: https://www.w3schools.com/java/java_arraylist.asp. Acesso em: 02 jun. 2022.

Set

GEEKSFORGEEKS. **Definir em Java.** 2022. Disponível em: https://www.geeksforgeeks.org/set-in-java/. Acesso em: 02 jun. 2022.

GEEKSFORGEEKS. **Interface SortedSet em Java com exemplos.** 2020. Disponível em: https://www.geeksforgeeks.org/sortedset-java-examples/?ref=lbp. Acesso em: 03 jun. 2022.

GEEKSFORGEEKS. **NavigableSet em Java com exemplos.** 2021. Disponível em: https://www.geeksforgeeks.org/navigableset-java-examples/?ref=lbp. Acesso em: 02 jun. 2022.

GEEKSFORGEEKS. **TreeSet em Java.** 2022. Disponível em: https://www.geeksforgeeks.org/treeset-in-java-with-examples/?ref=lbp. Acesso em: 06 jun. 2022.

PALMEIRA, T. V. **Trabalhando com a Interface Set no Java.** 2022. Disponível em: http://www.linhadecodigo.com.br/artigo/3669/trabalhando-com-a-interface-set-no-java.aspx. Acesso em: 06 jun. 2022.

Queue

GEEKSFORGEEKS. **Interface de fila em Java.** 2022. Disponível em: https://www.geeksforgeeks.org/queue-interface-java/. Acesso em: 02 jun. 2022.

GEEKSFORGEEKS. **Interface de fila em Java.** 2022. Disponível em: https://www.geeksforgeeks.org/queue-interface-java/?ref=lbp. Acesso em: 03 jun. 2022.

MILLINGTON, S. **Guia para a interface de filas Java.** 2021. Disponível em: https://www.baeldung.com/java-queue. Acesso em: 06 jun. 2022.

Dequeue

GEEKSFORGEEKS. **Interface Deque em Java com Exemplo.** 2022. Disponível em: https://www.geeksforgeeks.org/deque-interface-java-example/?ref=lbp. Acesso em: 02 jun. 2022.

Andre Santana Nunes

Experiência em desenvolvimento de software, especialmente em Java e outras tecnologias backend, como C# e Kotlin. Atuo com Análise de Engenharia no Itaú Unibanco, principalmente atuando com a linguagem Java. Também sou Mentor educacional no LAB365 do SENAI/SC.

SENAI <LAB365>