Loops, Arrays e Lists

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- Loops
- Arrays
- Matrizes
- Collections
- List
- ArrayList
- LinkedList

Loops

No Java temos algumas formas de realizar Loops de código. Temos o While, o Do While e o For.

While: é o Loop que realiza uma validação, se essa validação for False o Loop irá parar de executar, sendo assim podes fazer diversos tipos de validação dentro desse Loop através das expressões Booleanas.

```
int j = 0;
while(j<10){
    j++;
    //executa 10 vezes por causa do valor de j
    // j existe antes do while
    // e é usado para controle da execução do loop
}
```

Loops

Do While: realiza uma validação parecida com o While, porém ele irá executar o código primeiro e depois irá validar se o Loop será repetido.

```
int k = 0;
do{
    // vai executar 9 vezes
    // pois o do while sempre executa o código do loop
    // uma vez sem fazer a validação do loop
    k++;
    System.out.println(k);
}while(k<10);
```

Loops

For: executa um Loop de forma limitada. No For temos 3 parâmetros, uma variável com um valor inicial, a validação do valor dessa variável, e por fim o passou ou incremento dessa variável.

```
for(int i = 0; i < 10; i++){
            //código que será executado 10 vezes
            // a variável i existe apenas dentro do for
        }
//          i //não existe nesse ponto
```

Arrays

```
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
```

Matrizes

Matrizes são vetores bidirecionais, ou seja, temos 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.

Podemos 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 chegarmos ao valor desejado.

Matrizes

```
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]);
            }
        }
```

Collections

Collections, ou Coleções, são objetos que agrupam outros objetos, e esses elementos podem ser de qualquer tipo dentro de um programa Java.

Dentro do Java temos o pacote de temos a interface Collection que é pai de outras 3 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.

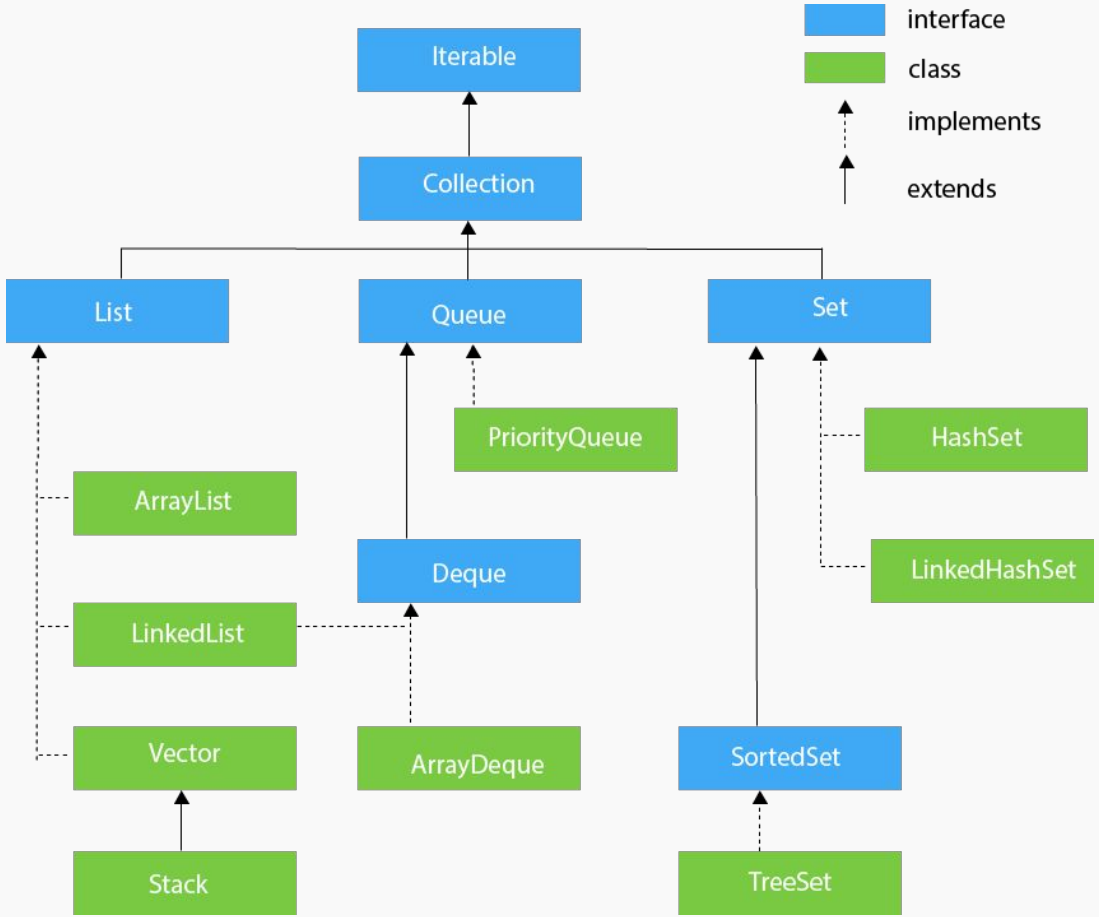
A Collection tem um número de métodos que podem ser utilizados com ela, para acessarmos esses métodos precisamos criar uma Collection e um Implementação de um de seus descendentes.

Collections

Podemos utilizar um tipo de lista chamada ArrayList, 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 executadas sobre todos os elementos da Collection.

Collections



Collections

```
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");
```

Operações de uma Collection

```
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();
```

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

List

As ArrayList, ou Listas de Vetor, são arrays que podem ter seu tamanho alterado. São baseados nos Arrays que vimos anteriormente e nos permitem criar listas de item que tem os mesmos métodos de Collection, mais 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.

List

```
List<Integer> list = new ArrayList<>(); //list2 sendo criada utilizando a arrayList
        list.add(11);
        list.add(12);
        list.add(13);
        int listLength = list.size();
        list.addAll(list2);
        list.remove(1);
        list.removeAll(list2);
        list.contains(2);
        list.containsAll(list2);
        list.clear();
        list.isEmpty();
        list.add(12);
        list.add(0, 13); // adiciona um item no índice especificado
```

Array List

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 podemos destacar: ArrayList, LinkedList, Vector e Stack.

Vector e Stack são tipos depreciados e lentos, sendo assim podemos focar nos demais. Stack pode ser substituído pelo Deque, que faz parte da família das Queue(Filas).

Array List

```
        ArrayList<Integer> arrayList = new ArrayList<>(); // criando um novo arrayList, funciona de forma identica a List
        arrayList.add(1);
        arrayList.add(1);
        arrayList.add(1);
        int arrayListLength = list.size();
        arrayList.addAll(list2);
        arrayList.remove(1);
        arrayList.removeAll(list2);
        arrayList.isEmpty();
        arrayList.contains(2);
        arrayList.containsAll(list2);
        arrayList.get(1);
        arrayList.clear();
```

Linked List

Linked Lists, ou Listas conectadas, herdam de List, porém tem uma diferença que é a estrutura que elas representam.

Cada item adicionado a 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 Linked Lists são muito boas para adicionar itens, mas não muito boas para encontrar elementos da lista.

Linked List

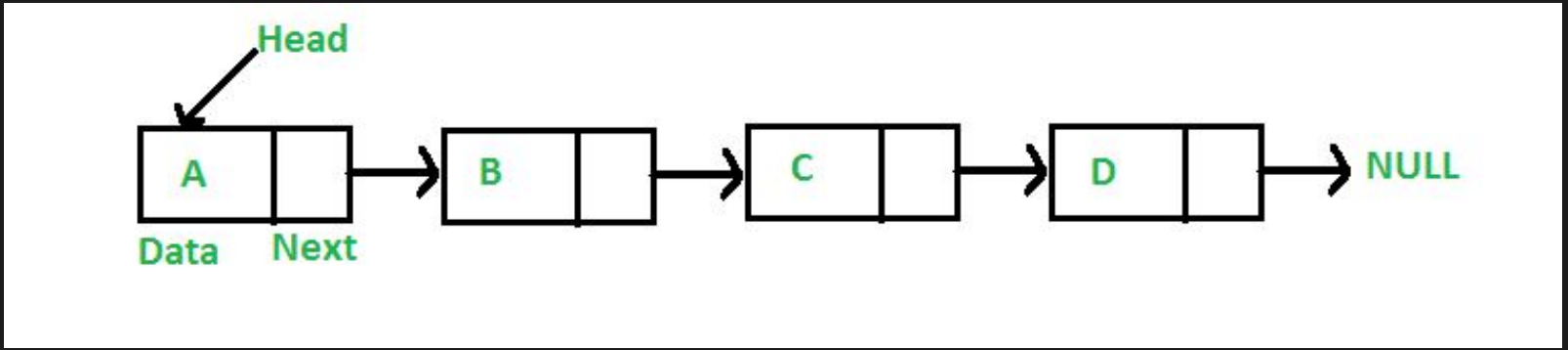
```
 LinkedList<Integer> linkedList = new LinkedList<>(); // cria uma nova
LinkedList
        linkedList.add(21); // adiciona no fim da linked list
        linkedList.add(1,14); // add por índice
        int linkedListLength = list.size();

        linkedList.addAll(list2);
        linkedList.remove(1);
        linkedList.removeAll(list2);
        linkedList.isEmpty();
        linkedList.contains(2);
        linkedList.containsAll(list2);
```

Linked List

```
    linkedList.get(0);
        linkedList.getLast(); // retorna o ultimo item da lista
        linkedList.getFirst(); // retorna o primeiro item da lista
        linkedList.removeLast(); // remove o ultimo item da lista
        linkedList.removeFirst(); // remove o primerio item da lista
        linkedList.clear();
        linkedList.removeFirstOccurrence(14); // remove a primeiro ocorrencia
de um item da lista
        linkedList.removeLastOccurrence(13); // remove a ultima ocorrencia de
um item da lista
//        System.out.println(linkedList);
        linkedList.toArray();
```

Linked List

As LinkedLists tem uma vantagem que é podemos ler o primeiro e o último item com facilidade.



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>