UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática
Prof. Dr. Ivan Carlos Alcântara de Oliveira
Estruturas de Dados I

Relatório da Aplicação 2
Operações com listas encadeadas

Nome do Integrante (ordem alfabética)	TIA
Enrico Cuono Alves Pereira	32258798
Erik Samuel Viana Hsu	32265921
Mateus Kenzo Iochimoto	32216289
Rodrigo Machado de Assis Oliveira de Lima	32234678
Thiago Shihan Cardoso Toma	32210744

Conteúdo do Relatório

Casos de teste da execução do programa:
opção 1 do menu

```
>>>>>>>>>> Dados originais (sistema legado) >>>>>>>>>>
(26)
(111 # Allana # 10 # 0) ->
(222 # Breno # 8 # 9) ->
(333 # Cida # 6 # 7) ->
(444 # Denis # -1 # 0) ->
(555 # Edna # 4 # 5) ->
(666 # Felix # 2 # 3) ->
(777 # Gertrudes # 0 # -1) ->
(888 # Hector # 0 # 1) ->
(999 # Ivone # -1 # -1) ->
(123 # Jorge # 7 # 5) ->
(456 # Karen # 9 # 9) ->
(789 # Leonidas # 5 # 2) ->
(321 # Matilda # 6 # 3) ->
(654 # Nivaldo # 2 # 1) ->
(987 # Odete # 4 # 7) ->
(191 # Plinio # 0 # 3) ->
(282 # Quiteria # 3 # 9) ->
(373 # Ronildo # -1 # -1) ->
(464 # Samanta # 8 # 3) ->
(505 # Thales # -1 # 0) ->
(135 # Ursula # 9 # 1) ->
(246 # Vanderlei # 10 # 0) ->
(357 # Wilma # 5 # -1) ->
(468 # Xavier # 8 # 2) ->
(579 # Yasmin # 9 # 3) ->
(680 # Zacarias # -1 # 0) ->
null.
<<<<<<<<<< Dados originais (sistema legado) <<<<<<<<<<
```

opção 2 do menu

```
>>>>>>>>>> Dados convertidos para a nova representação dos dados >>>>>>>>>>
(26)
null<-(23.S1-111;Allana;10.0)->23.S1-222
23.S1-111<-(23.S1-222;Breno;8.9)->23.S1-333
23.S1-222<-(23.S1-333;Cida;6.7)->23.S1-444
23.S1-333<-(23.S1-444;Denis;99.9)->23.S1-555
23.S1-444<-(23.S1-555;Edna;4.5)->23.S1-666
23.S1-555<-(23.S1-666;Felix;2.3)->23.S1-777
23.S1-666<-(23.S1-777;Gertrudes;99.9)->23.S1-888
23.S1-777<-(23.S1-888;Hector;0.1)->23.S1-999
23.S1-888<-(23.S1-999;Ivone;99.9)->23.S1-123
23.S1-999<-(23.S1-123;Jorge;7.5)->23.S1-456
23.S1-123<-(23.S1-456;Karen;9.9)->23.S1-789
23.S1-456<-(23.S1-789;Leonidas;5.2)->23.S1-321
23.S1-789<-(23.S1-321;Matilda;6.3)->23.S1-654
23.S1-321<-(23.S1-654;Nivaldo;2.1)->23.S1-987
23.S1-654<-(23.S1-987;Odete;4.7)->23.S1-191
23.S1-987<-(23.S1-191;Plinio;0.3)->23.S1-282
23.S1-191<-(23.S1-282;Quiteria;3.9)->23.S1-373
23.S1-282<-(23.S1-373;Ronildo;99.9)->23.S1-464
23.S1-373<-(23.S1-464;Samanta;8.3)->23.S1-505
23.S1-464<-(23.S1-505;Thales;99.9)->23.S1-135
23.S1-505<-(23.S1-135;Ursula;9.1)->23.S1-246
23.S1-135<-(23.S1-246;Vanderlei;10.0)->23.S1-357
23.S1-246<-(23.S1-357;Wilma;99.9)->23.S1-468
23.S1-357<-(23.S1-468;Xavier;8.2)->23.S1-579
23.S1-468<-(23.S1-579;Yasmin;9.3)->23.S1-680
23.S1-579<-(23.S1-680;Zacarias;99.9)->null

<<<<<<<<<< Dados convertidos para a nova representação dos dados <<<<<<<<<<
```

opção 3 do menu

```
>>>>>>>>>> Lista filtrada (somente notas válidas) >>>>>>>>>>
(19)
null<-(23.S1-111;Allana;10.0)->23.S1-222
23.S1-111<-(23.S1-222;Breno;8.9)->23.S1-333
23.S1-222<-(23.S1-333;Cida;6.7)->23.S1-555
23.S1-333<-(23.S1-555;Edna;4.5)->23.S1-666
23.S1-555<-(23.S1-666;Felix;2.3)->23.S1-888
23.S1-666<-(23.S1-888;Hector;0.1)->23.S1-123
23.S1-888<-(23.S1-123;Jorge;7.5)->23.S1-456
23.S1-123<-(23.S1-456;Karen;9.9)->23.S1-789
23.S1-456<-(23.S1-789;Leonidas;5.2)->23.S1-321
23.S1-789<-(23.S1-321;Matilda;6.3)->23.S1-654
23.S1-321<-(23.S1-654;Nivaldo;2.1)->23.S1-987
23.S1-654<-(23.S1-987;Odete;4.7)->23.S1-191
23.S1-987<-(23.S1-191;Plinio;0.3)->23.S1-282
23.S1-191<-(23.S1-282;Quiteria;3.9)->23.S1-464
23.S1-282<-(23.S1-464;Samanta;8.3)->23.S1-135
23.S1-464<-(23.S1-135;Ursula;9.1)->23.S1-246
23.S1-135<-(23.S1-246;Vanderlei;10.0)->23.S1-468
23.S1-246<-(23.S1-468;Xavier;8.2)->23.S1-579
23.S1-468<-(23.S1-579;Yasmin;9.3)->null

<<<<<<<<<< Lista filtrada (somente notas válidas) <<<<<<<<<<
```

opção 4 do menu

```
>>>>>>>>>> Lista filtrada (somente 'ausência de nota') >>>>>>>>>>
(7)
null<-(23.S1-444;Denis;99.9)->23.S1-777
23.S1-444<-(23.S1-777;Gertrudes;99.9)->23.S1-999
23.S1-777<-(23.S1-999;Ivone;99.9)->23.S1-373
23.S1-999<-(23.S1-373;Ronildo;99.9)->23.S1-505
23.S1-373<-(23.S1-505;Thales;99.9)->23.S1-357
23.S1-505<-(23.S1-357;Wilma;99.9)->23.S1-680
23.S1-357<-(23.S1-680;Zacarias;99.9)->null

<<<<<<<<<< Lista filtrada (somente 'ausência de nota') <<<<<<<<<<
```

opção 5 do menu

```
>>>>>>>>>> Média das notas válidas >>>>>>>>>>
6.173684
<<<<<<<<<< Média das notas válidas <<<<<<<<<<
```

opção 6 do menu

```
>>>>>>>>>> Lista com notas acima da média >>>>>>>>>>
(11)
null<-(23.S1-111;Allana;10.0)->23.S1-222
23.S1-111<-(23.S1-222;Breno;8.9)->23.S1-333
23.S1-222<-(23.S1-333;Cida;6.7)->23.S1-123
23.S1-333<-(23.S1-123;Jorge;7.5)->23.S1-456
23.S1-123<-(23.S1-456;Karen;9.9)->23.S1-321
23.S1-456<-(23.S1-321;Matilda;6.3)->23.S1-464
23.S1-321<-(23.S1-464;Samanta;8.3)->23.S1-135
23.S1-464<-(23.S1-135;Ursula;9.1)->23.S1-246
23.S1-135<-(23.S1-246;Vanderlei;10.0)->23.S1-468
23.S1-246<-(23.S1-468;Xavier;8.2)->23.S1-579
23.S1-468<-(23.S1-579;Yasmin;9.3)->null

<<<<<<<<<< Lista com notas acima da média <<<<<<<<<<
```

opção 7 do menu

```
>>>>>>>>>> Lista mapeada para uma única string >>>>>>>>>>
23.S1-111;Allana;10.0
23.S1-222;Breno;8.9
23.S1-333;Cida;6.7
23.S1-444;Denis;99.9
23.S1-555;Edna;4.5
23.S1-666;Felix;2.3
23.S1-777;Gertrudes;99.9
23.S1-888;Hector;0.1
23.S1-999;Ivone;99.9
23.S1-123;Jorge;7.5
23.S1-456;Karen;9.9
23.S1-789;Leonidas;5.2
23.S1-321;Matilda;6.3
23.S1-654;Nivaldo;2.1
23.S1-987;Odete;4.7
23.S1-191;Plinio;0.3
23.S1-282;Quiteria;3.9
23.S1-373;Ronildo;99.9
23.S1-464;Samanta;8.3
23.S1-505;Thales;99.9
23.S1-135;Ursula;9.1
23.S1-246;Vanderlei;10.0
23.S1-357;Wilma;99.9
23.S1-468;Xavier;8.2
23.S1-579;Yasmin;9.3
23.S1-680;Zacarias;99.9

<<<<<<<<<< Lista mapeada para uma única string <<<<<<<<<<
```

opção 8 do menu

```
 Opção:8
Encerra o programa
```

test1

```
Encerra o programa
>>>>>>>>>> test1 >>>>>>>>>>
23.S1-888<-(23.S1-999;Ivone;99.9)->23.S1-123
<<<<<<<<<< test1 <<<<<<<<<<
```

test2

```
>>>>>>>>>> test2 >>>>>>>>>>
null<-(23.S1-999;Ivone;99.9)->null
<<<<<<<<<< test2 <<<<<<<<<<
```

test3

```
>>>>>>>>>> test3 >>>>>>>>>>
null
<<<<<<<<<< test3 <<<<<<<<<<
```

aboveAverageList.clear()

```
>>>>>>>>>> aboveAverageList.clear() >>>>>>>>>>
(0)

<<<<<<<<<< aboveAverageList.clear() <<<<<<<<<<
```

testList

```
>>>>>>>>>> testList >>>>>>>>>>
(4)
null<-(321;Test;2.3)->ABC
321<-(ABC;John Doe;4.7)->XYZ
ABC<-(XYZ;Jane Doe;9.9)->Nothing
XYZ<-(Nothing;Yada yada yada;99.9)->null

<<<<<<<<<< testList <<<<<<<<<<
```

```
<<<<<<<<<< testList <<<<<<<<<<

testList.getHead(): null<-(321;Test;2.3)->ABC
testList.getTail(): XYZ<-(Nothing;Yada yada yada;99.9)->null
testList.removeHead(): null<-(321;Test;2.3)->null
testList.removeTail(): null<-(Nothing;Yada yada yada;99.9)->null

>>>>>>>>>> testList >>>>>>>>>>
```

```
>>>>>>>>>> testList >>>>>>>>>>
(2)
null<-(ABC;John Doe;4.7)->XYZ
ABC<-(XYZ;Jane Doe;9.9)->null

<<<<<<<<<< testList <<<<<<<<<<
```

```
<<<<<<<<<< testList <<<<<<<<<<

testList.getHead(): null<-(ABC;John Doe;4.7)->null
testList.getTail(): null<-(XYZ;Jane Doe;9.9)->null
testList.removeNode("ABC"): (ABC;John Doe;4.7)

>>>>>>>>>> testList >>>>>>>>>>
```

```
>>>>>>>>>> testList >>>>>>>>>>
(1)
null<-(XYZ;Jane Doe;9.9)->null

<<<<<<<<<< testList <<<<<<<<<<
```

```
<<<<<<<<<< testList <<<<<<<<<<

testList.getHead(): (XYZ;Jane Doe;9.9)
testList.getTail(): (XYZ;Jane Doe;9.9)

>>>>>>>>>> testList >>>>>>>>>>
```

```
>>>>>>>>>> testList >>>>>>>>>>
(5)
null<-(ijkl;IJKL;5.6)->qwerty
ijkl<-(qwerty;QWERTY;1.2)->XYZ
qwerty<-(XYZ;Jane Doe;9.9)->WASD
XYZ<-(WASD;wasd;3.4)->1234
WASD<-(1234;Um Dois Tres Quatro;7.8)->null

<<<<<<<<<< testList <<<<<<<<<<
```

testList.clear()

```
>>>>>>>>>> testList.clear() >>>>>>>>>>
(0)

<<<<<<<<<< testList.clear() <<<<<<<<<<
```

dados.csv

	A	B	C	D	E
1	23.S1-111	Allana	10.0		
2	23.S1-222	Breno	8.9		
3	23.S1-333	Cida	6.7		
4	23.S1-444	Denis	99.9		
5	23.S1-555	Edna	4.5		
6	23.S1-666	Felix	2.3		
7	23.S1-777	Gertrudes	99.9		
8	23.S1-888	Hector	0.1		
9	23.S1-999	Ivone	99.9		
10	23.S1-123	Jorge	7.5		
11	23.S1-456	Karen	9.9		
12	23.S1-789	Leonidas	5.2		
13	23.S1-321	Matilda	6.3		
14	23.S1-654	Nivaldo	2.1		
15	23.S1-987	Odete	4.7		
16	23.S1-191	Plinio	0.3		
17	23.S1-282	Quiteria	3.9		
18	23.S1-373	Ronildo	99.9		
19	23.S1-464	Samanta	8.3		
20	23.S1-505	Thales	99.9		
21	23.S1-135	Ursula	9.1		
22	23.S1-246	Vanderlei	10.0		
23	23.S1-357	Wilma	99.9		
24	23.S1-468	Xavier	8.2		
25	23.S1-579	Yasmin	9.3		
26	23.S1-680	Zacarias	99.9		
27					

Explicação do mapeamento dos dados da base de dados original para a nova representação dos dados do sistema de notas:

A base de dados original é mapeada para a nova representação dos dados por meio da operação map(), onde ela recebe uma lista simplesmente encadeada e no final retorna uma lista duplamente encadeada. Dentro dessa operação é criada uma lista duplamente encadeada chamada ListaMapeada onde serão armazenados os dados presentes nos nós da lista simplesmente encadeada original.

```
public static DLinkedList map(final LinkedListOriginal list) {
    DLinkedList ListaMapeada = new DLinkedList(); //Cria a DLinkedList "L
    NodeOriginal pAnda = list.getHead();
```

Dessa forma, para percorrer os nós da lista original, utilizamos um nó do tipo NodeOriginal (pois é o mesmo tipo de nó utilizado na lista original) chamado pAnda, que começa pelo primeiro nó da lista e posteriormente tem seu valor alterado para o valor do próximo nó. Assim, percorremos a lista até que chegue no final da lista utilizando um while (pAnda != null) e, dentro desse loop, pegamos os dados dos nós dessa lista e atribuímos às variáveis respectivas dos dados existentes (id, nome, inteiro, décimo e uma referência para o próximo nó chamado next). Na imagem a seguir a seta vermelha indica o loop utilizado e, em setas azuis, o recolhimento dos dados existentes para as novas variáveis.

```
while(pAnda != null) {
    //Pega os dados do nó da LinkedListOriginal
    int id = pAnda.getId();
    String nome = pAnda.getNome();
    int inteiro = pAnda.getInteiro();
    int decimo = pAnda.getDecimo();
    NodeOriginal next = pAnda.getNext();
```

Após isso é criado o “IdNovo”, um novo identificador para o ID da pessoa, gerado pela concatenação entre “23.S1-” e o id da pessoa, visto a solicitação da empresa contratante da solução.

```
String IdNovo = "23.S1-" + id;
Node noNovo = new Node();
noNovo.setId(IdNovo);
noNovo.setNome(nome);
```

Assim, para salvarmos esses dados dentro da nossa DLinkedList, é necessário criar um nó novo do tipo Node (implementado pela equipe) pois é com esse tipo de dado que armazenaremos os valores e que a DLinkedList trabalha. Logo após a criação do nó novo, chamado de “noNovo”, definimos o Id do nó como o IdNovo gerado pela concatenação comentada anteriormente, utilizando o método setId() da classe Node. Realizamos a mesma definição de valor para o nome da pessoa no noNovo.

```
String IdNovo = "23.S1-" + id;
Node noNovo = new Node();        Nó novo
noNovo.setId(IdNovo);            Definição dos valores para
noNovo.setNome(nome);            o ID e Nome.
```

Visto a alteração solicitada pela empresa contratante no quesito das notas dos alunos, tivemos que criar uma variável chamada “nota” do tipo float que, caso o valor do inteiro ou do décimo do nó da lista original for -1, receberá o valor 99.9 como nota, valor que evidencia a falta de nota desse aluno. Caso o décimo tenha valor 0, a nota nova (em float) é a parte inteira da nota + 0, porém na formatação de float 0.0f. Caso contrário a nota em formatação nova é a soma do tipo float da parte inteira da nota com o float da parte decimal da nota dividido por 10 (para que seja gerado um valor em ponto flutuante). Finalizada essa parte, temos a nota do aluno em formatação de ponto flutuante e inserimos essa nota no nosso noNovo por meio do setNota() da classe Node.

```
float nota;
if (inteiro == -1 || decimo == -1) {        <- Ausência de nota
    nota = 99.9f;
}

else if (decimo == 0) {                     <- Caso o décimo = 0
    nota = inteiro + 0.0f;                  Nota em float = parte inteira + 0.0
} else {                                    <- Converter para float
    nota = ((float) inteiro + ((float) decimo) / 10);
}                                           Atribuição da nota no noNovo.
noNovo.setNota(nota);
```

Seguidamente, é verificado se o próximo nó (next) não é nulo, ou seja, se existe um próximo nó na lista original. Caso a verificação seja verdadeira (não é nulo), é criado um nó novo na DLinkedList chamado nextNode contento os valores do nó next da lista original. Para termos acesso aos valores do nó next da lista original, utilizamos os métodos getters da classe NodeOriginal para cada dado do nó (id, nome, inteiro e décimo). Após isso atribuímos o nosso nextNode como próximo nó do noNovo da DLinkedList.

```
if (next != null) {
    Node nextNode = new Node();             <- Criação do próximo nó na DLinkedList
    nextNode.setId(String.valueOf(next.getId()));   <- Atribuições dos valores de next (lista
    nextNode.setNome(next.getNome());               original) para nextNode (DLinkedList)
    nextNode.setNota(((float) next.getInteiro() + (float) next.getDecimo()) / 10);
    noNovo.setNext(nextNode);
}
```

No final realizamos a adição do nosso nó atual noNovo na DLinkedList pelo método append, passando o novo ID, nome e nota como argumentos. O valor de pAnda é atualizado para o próximo nó da lista original, fazendo com que o loop se repita para o próximo nó. Terminado o percorrer pelos nós da lista, a operação map() retornará a DLinkedList mapeada chamada “ListaMapeada”.

```
            ListaMapeada.append(IdNovo, nome, nota);
            pAnda = next;
        }

        return ListaMapeada;
    }
```

Explicação do funcionamento dos filtros da classe Operation (operações filter*):

O funcionamento dos 3 filtros (*RemoveNonGraded, RemoveGraded e RemoveBelowAverage*) é bem similar, suas estruturas são idênticas, tendo como diferença apenas a condicional “*if*” a ser avaliada. Explicando brevemente, as funções percorrem a lista recebida por meio de um loop, dentro desse loop usamos um if para separar as notas desejadas das indesejadas, adicionando os nós dos alunos desejados a uma nova lista que será retornada no final da execução da função; e pulando para o próximo nó quando se detecta um nó indesejado.

As operações se iniciam pela criação de novas listas, essas serão retornadas ao final da execução do método contendo apenas os nós desejados.O próximo passo é a criação do ponteiro “pAnda” e do loop “*while pAnda!=null*”, para que a lista seja percorrida de nó em nó até o final (indicado por *pAnda == null*), passando por todos os alunos. Dentro do loop é criada a variável nota, essa variável recebe a nota de cada aluno utilizando-se do método *getNota()*. Comparamos então a nota de cada aluno com a condição utilizada para

filtrar as notas desejadas das indesejadas, essa condição varia de função para função, sendo a única diferença entre os três filtros.

Nesse código em específico optamos por "entrar" no bloco "*if*" (*condicao == true*) apenas quando detectarmos uma nota indesejada, porém essa lógica poderia ser invertida com algumas adaptações ao código. Sendo assim, o filtro "*RemoveNonGraded*" tem como condicao "*if nota == 99.9*"(99.9 indicando nota inválida); o filtro "*RemoveGraded*" tem como condicao "*if nota != 99.9*"; e o filtro "*RemoveBelowAverage*" tem como condicao "if nota < *average*"(*average* indicando a média das notas).

Condição *RemoveNonGraded:*

```
while(pAnda != null) {
    float nota = pAnda.getNota();
    if (nota == 99.9f)
        pAnda = pAnda.getNext();
```

Condição *RemoveGraded:*

```
while(pAnda != null) {
    float nota = pAnda.getNota();
    if (nota != 99.9f)
        pAnda = pAnda.getNext();
```

Condição *RemoveBelowAverage:*

```
while(pAnda != null) {
    float nota = pAnda.getNota();
    if (nota < average) {
        pAnda = pAnda.getNext();
        }
```

Dentro de cada *if* existe apenas o comando *pAnda = pAnda.getNext()* que faz com que o nó atual seja pulado, já que não queremos colocá-lo dentro da lista a ser retornada. Caso a nota seja desejada (o bloco *if* não é executado então o nó não é ignorado) a linha de código que adiciona (*append()*) dado nó a sua respectiva lista (a ser retornada) é executada , e aí então troca-se de nó usando *pAnda = pAnda.getNext().*

Quando o *while* parar de rodar a nova lista já estará pronta, então ela é retornada e os métodos acabam.

Função *RemoveNonGraded:*

```
public static DLinkedList filterRemoveNonGraded(final DLinkedList data) {
    DLinkedList ListaValida = new DLinkedList(); //Cria a DLinkedList "ListaValida"
    Node pAnda = data.getHead();

    while(pAnda != null) {
        float nota = pAnda.getNota();
        if (nota == 99.9f)
            pAnda = pAnda.getNext();
        else {
            ListaValida.append(pAnda.getId(),pAnda.getNome(),pAnda.getNota());
            pAnda = pAnda.getNext();
        }
    }

    return ListaValida;
}
```

Função *RemoveGraded:*

```
public static DLinkedList filterRemoveGraded(final DLinkedList data) {
    DLinkedList ListaInvalida = new DLinkedList(); //Cria a DLinkedList "Listainvalida"
    Node pAnda = data.getHead();

    while(pAnda != null) {
        float nota = pAnda.getNota();
        if (nota != 99.9f)
            pAnda = pAnda.getNext();
        else {
            ListaInvalida.append(pAnda.getId(),pAnda.getNome(),pAnda.getNota());
            pAnda = pAnda.getNext();
        }
    }

    return ListaInvalida;
}
```

Função *RemoveBelowAverage:*

```
public static DLinkedList filterRemoveBelowAverage(final DLinkedList data, float average) {
    Node pAnda = data.getHead();
    DLinkedList ListaAcimaMedia = new DLinkedList(); //Cria a DLinkedList "ListaAcimaMedia"

    while(pAnda != null) {
        float nota = pAnda.getNota();
        if (nota < average) {
            pAnda = pAnda.getNext();
            }
        else {
            ListaAcimaMedia.append(pAnda.getId(),pAnda.getNome(),pAnda.getNota());
            pAnda = pAnda.getNext();
        }
    }

    return ListaAcimaMedia;
}
```

Explicação do cálculo da média das notas válidas:

A função que realiza o cálculo das médias têm uma estrutura também bem similar a dos filtros, utilizando-se da criação de uma nova lista que será retornada e um nó pAnda que junto com um loop *while (pAnda!=null)* percorre a lista nó por nó.

A diferença é que além de não conter uma condicional para separar as notas (já que elas já vêm separadas considerando que a lista recebida como parâmetro decorre da função *RemoveNonGraded*) também são criadas inicialmente duas variáveis, *soma* e *count*, essas variáveis são incrementadas a cada nó, a *soma* incrementada pelo valor de cada nota válida, e a *count* incrementada de 1 em 1 indicando a quantidade de notas a serem levadas em conta.

Quando o loop chega ao final e ambas as variáveis já estão com seus valores totais (soma de todas as notas válidas; e soma de todos alunos com notas válidas) realiza-se o cálculo da média (*média = soma / count*) e retorna-se esse valor.

Função *Reduce:*

```
public static float reduce(final DLinkedList data) {
    Node pAnda = data.getHead();
    float soma = 0;
    int count = 0;

    while(pAnda != null) {
        float nota = pAnda.getNota();
        soma+=nota;
        count+=1;
        pAnda = pAnda.getNext();
    }
    float media = soma/count;
    return media;
}
```

Explicação do funcionamento da operação mapToString:

A operação mapToString recebe uma lista duplamente encadeada como entrada e retorna a representação formatada em String dos dados dessa lista. Primeiramente é criada uma String vazia chamada “StringFormatada”, onde serão concatenados os valores de cada nó da DLinkedList. Partindo do primeiro nó utilizando o getHead() e atribuindo ele à variável pAnda, realizamos um loop enquanto pAnda não for nulo (ou seja, percorrer até o último elemento da lista).

Dentro do loop é realizada a concatenação da StringFormatada com o ID, nome, e nota pelos métodos getId(), getNome() e getNota() respectivamente. Para separar os valores de ID, nome e nota, são realizadas (após a concatenação comenta anteriormente do ID e nome) as concatenações do símbolo ‘;’ (ponto e vírgula) e concatenação da quebra de linha “\n” após a concatenação de cada nota, visando a separação de dados de uma pessoa por linha.

Após isso é feita a atualização do pAnda para o próximo nó na DLinkedList, para que o loop seja feito para os próximos nós.

No final da operação é retornada uma String “StringFormatada”, contendo os dados da DLinkedList formatados como uma string, tendo cada linha como um nó da lista e os valores do ID, nome e nota separados por ponto e vírgula.