Universidade Presbiteriana Mackenzie

Faculdade de Computação e Informática
Ciência da Computação
Estrutura de Dados I – 3ª etapa – 2023.1
Professores André Kishimoto e Ivan Carlos Alcântara de Oliveira

Atividade Apl2 – Operações com listas encadeadas

A *BR School Solutions* é uma empresa de tecnologia que oferece soluções para escolas de todos os níveis e portes, com clientes espalhados por todo o Brasil e em alguns países das Américas.

Para continuar oferecendo soluções de ponta aos seus clientes, a empresa está atualizando parte da sua tecnologia proprietária e, após reuniões com alguns clientes, foi identificado que alguns sistemas precisam ser modificados para atender aos novos requisitos e, nesse momento, a empresa decidiu classificar tais sistemas como sistemas obsoletos/legados (*legacy systems*).

Sua equipe foi contratada para atualizar um módulo pertencente a um sistema [legado] de notas de um dos clientes da empresa.

O ***sistema*** atual, agora considerado ***obsoleto***, trabalha com uma ***base de dados*** em ***formato texto*** que contém o seguinte formato:

ID da pessoa, nome da pessoa, parte inteira da nota, parte decimal da nota

Sendo:

- ID da pessoa: um número inteiro de 3 dígitos, no intervalo [000, 999].
- Nome da pessoa: um texto (*string*).
- Parte inteira da nota: um número inteiro de 1 ou 2 dígitos, no intervalo [0, 10].
- Parte decimal da nota: um número inteiro de 1 dígito, no intervalo [0, 9].

É importante observar que uma pessoa pode não ter uma nota atribuída (por exemplo, a pessoa não realizou uma prova). Nesse caso, o sistema identifica a *ausência de nota* quando a *parte inteira e/ou parte decimal da nota* tem o valor -1. A tabela a seguir resume como o sistema identifica a ausência de nota de uma pessoa.

Parte inteira da nota	Parte decimal da nota	Ausência de nota?
Número no intervalo [0,10]	Número no intervalo [0,9]	Não
Número no intervalo [0,10]	-1	Sim
-1	Número no intervalo [0,9]	Sim
-1	-1	Sim

Ou seja: independente da parte inteira e da parte decimal da nota, se uma das partes tiver o valor -1, o sistema entende que a nota da pessoa não foi atribuída (ausência de nota).

Quanto ao conteúdo presente na base de dados (arquivo texto), os dados de uma pessoa são separados pelo símbolo cerquilha (#) e cada linha do arquivo texto representa os dados de uma pessoa.

Por exemplo, uma linha pertencente a uma base de dados desse sistema pode ser definida como:

```
999#Jane#8#7
```

Sendo:

- ID da pessoa: **999**
- Nome da pessoa: **Jane**
- Parte inteira da nota: **8**
- Parte decimal da nota: **7**

Assim, podemos indicar que **Jane** tem **ID 999** e nota **8,7**.

Um exemplo de ausência de nota pode ser visto na linha a seguir:

```
123#John#4#-1
```

Nesse caso,

- ID da pessoa: **123**
- Nome da pessoa: **John**
- Parte inteira da nota: **4**
- Parte decimal da nota: **-1**

Com base nas regras de negócios, como a parte decimal da nota tem o valor **-1**, concluímos que **John**, que tem o **ID 123**, **não possui nota**, mesmo que a parte inteira da nota tenha o valor 4.

É possível observar que o sistema atual usa uma representação de dados que pode gerar confusão: Será que o -1 da parte decimal da nota de John foi um erro de digitação? Será que John realmente não tem uma nota ou sua nota deveria ser 4,1?

Para evitar esse tipo de problema, a empresa e o cliente decidiram que é importante atualizar a representação dos dados. Na época que o sistema original foi implantado, talvez fizesse sentido separar a parte inteira e decimal da nota de uma pessoa, usando um número inteiro para cada parte. No entanto, empresa e cliente optaram por substituir esses dois números inteiros por um único número real (*float*).

Para o caso “ausência de nota”, o cliente pediu para usar o valor `99.9`.

Além da nota, o cliente pediu para a empresa atualizar a representação do ID da pessoa: todo ID deve ter um prefixo no formato `{YY}.S{N}-`, sendo:

- `{YY}`: Ano atual, com dois dígitos (por exemplo, `23`).
- `{N}`: `1` para primeiro semestre ou `2` para segundo semestre do ano.
- Os outros três caracteres (`.S-`) são sempre fixos.

E, ao invés de usar a cerquilha (`#`) como separador de dados, a empresa optou por usar o símbolo ponto-e-vírgula (`;`) como separador de dados, seguindo um dos formatos de arquivo CSV.

Cada linha do arquivo texto continua representando os dados de uma pessoa.

Usando Jane e John como exemplo, os dados dessas pessoas, na nova base de dados, seriam descritos da seguinte maneira, respectivamente:

```
23.S1-999;Jane;8.7
23.S1-123;John;99.9
```

Portanto, o ***novo sistema*** deve trabalhar com uma ***base de dados*** em ***formato texto*** que contém o seguinte formato:

```
ID da pessoa, nome da pessoa, nota da pessoa
```

Sendo:

- `ID da pessoa`: um texto (*string*) que começa com `23.S1-` seguido de 3 dígitos, no intervalo [`000`, `999`] (antigo ID da pessoa).
- `Nome da pessoa`: um texto (*string*).
- `Nota da pessoa`: um número real (*float*), no intervalo [`0.0`, `10.0`] ou `99.9` no caso de ausência de nota. O número é a combinação das partes inteira e decimal da nota da pessoa.
- Cada dado é separado pelo símbolo ponto-e-vírgula (`;`).
- Cada linha do arquivo texto representa uma pessoa.

Implementação

A empresa *BR School Solutions*, como contratante dos serviços de desenvolvimento da sua equipe, pediu para que a atualização do sistema legado descrito anteriormente seja feita da seguinte maneira:

1) Sua equipe está recebendo um arquivo ZIP (**Apl2.zip**) que contém os seguintes arquivos:

- **dados.txt**: Uma base de dados que você deve usar para testar e validar as funcionalidades do sistema.
- ***src/MainApl2.java****: Uma classe Java com uma sequência de instruções para testar as funcionalidades do sistema e que sua equipe deverá implementar parcialmente.*
- **src/apl2/Data.java**: Uma classe Java que possui um método para carregar o conteúdo de um arquivo texto em uma string e um método para salvar o conteúdo de uma string em um arquivo texto.
- ***src/apl2/DLinkedList.java****: Uma classe Java que sua equipe deverá implementar.*
- **src/apl2/LinkedListOriginal.java**: Uma classe Java que implementa uma lista simplesmente encadeada e usa a classe `NodeOriginal` para os nós da lista.
- ***src/apl2/Node.java****: Uma classe Java que sua equipe deverá implementar.*
- **src/apl2/NodeOriginal.java**: Uma classe Java que implementa um nó que é usado para representar os dados de uma pessoa do sistema de notas.
- ***src/apl2/Operation.java****: Uma classe Java que sua equipe deverá implementar.*

ATENÇÃO! Os arquivos **dados.txt**, **Data.java**, **LinkedListOriginal.java** e **NodeOriginal.java** **NÃO** devem ser alterados!

2) Conteúdo do arquivo **src/apl2/Node.java**:

- A classe Node (que pertence ao pacote apl2) deve conter os atributos que representam a nova versão dos dados de uma pessoa, conforme descrito na seção anterior.
- A classe deve conter os construtores apropriados, assim como os métodos *getters* e *setters*.
- A classe também representa um nó que é usado na implementação da lista duplamente encadeada (classe DLinkedList).
- A classe deve sobrescrever (*override*) o método public String toString() {...}, retornando uma string com os valores dos atributos da classe (caso queira, use o exemplo do método toString() da classe NodeOriginal).

3) Conteúdo do arquivo **src/apl2/DLinkedList.java**:

- A classe DLinkedList (que pertence ao pacote apl2) deve implementar uma lista duplamente encadeada. Os nós dessa lista são do tipo Node.
- A classe deve possuir dois nós especiais, head e tail, que são referências para o primeiro e último nó da lista, respectivamente.
- A classe deve possuir um contador de quantos nós existem na lista.
- A classe deve sobrescrever (*override*) o método public String toString() {...}, retornando uma string com o conteúdo da lista (caso queira, use o exemplo do método toString() da classe LinkedListOriginal).
- A classe deve implementar as operações da tabela a seguir, respeitando o comportamento descrito em cada operação.

OPERAÇÃO	COMPORTAMENTO
Método construtor	Cria uma lista vazia.
insert(*<dados da pessoa>*)	Aloca um Node que contém os *<dados da pessoa>* e insere o novo nó no início da lista.
append(*<dados da pessoa>*)	Aloca um Node que contém os *<dados da pessoa>* e insere o novo nó no final da lista.
removeHead()	Remove o nó do início da lista e retorna a referência do nó removido. Ou retorna null caso a lista esteja vazia.
removeTail()	Remove o nó do final da lista e retorna a referência do nó removido. Ou retorna null caso a lista esteja vazia.
removeNode(*<ID da pessoa>*)	Remove o nó que contém o *<ID da pessoa>* da lista e retorna a referência do nó removido. Ou retorna null caso não exista um nó com *<ID da pessoa>*.
getHead()	Retorna uma referência para o nó do início da lista. Ou retorna null caso a lista esteja vazia.
getTail()	Retorna uma referência para o nó do final da lista. Ou retorna null caso a lista esteja vazia.
getNode(*<ID da pessoa>*)	Retorna uma referência para o nó que contém o *<ID da pessoa>* da lista. Ou retorna null caso não exista um nó com *<ID da pessoa>*.
count()	Retorna a quantidade de nós da lista.
isEmpty()	Retorna true se a lista estiver vazia ou false, caso contrário.
clear()	Esvazia a lista, liberando a memória de todos os nós da lista.
toString()	Retorna uma string com o conteúdo da lista (caso queira, use o exemplo do método toString() da classe LinkedListOriginal).

4) Conteúdo do arquivo **src/apl2/Operation.java**:

- A classe Operation (que pertence ao pacote apl2) deve implementar métodos estáticos que realizam operações na base de dados, sendo que a base de dados pode estar representada na memória do computador por uma LinkedListOriginal ou por uma DLinkedList (ver tabela a seguir).
- A classe deve implementar as operações da tabela a seguir, respeitando o comportamento descrito em cada operação.

ATENÇÃO! As assinaturas dos métodos da classe Operation **NÃO** devem ser alteradas!

OPERAÇÃO	COMPORTAMENTO
map(*LinkedListOriginal*)	Recebe como parâmetro uma lista encadeada do tipo LinkedListOriginal, sendo que os nós da lista estão populados com o conteúdo da base de dados original (conteúdo do arquivo **dados.txt**). A operação map() deve mapear os dados originais para uma lista encadeada do tipo DLinkedList e retornar a referência da DLinkedList que possui os dados mapeados para a nova estrutura usada pelo sistema de notas.
filterRemoveNonGraded (*DLinkedList*)	Recebe como parâmetro uma lista duplamente encadeada do tipo DLinkedList, sendo que os nós da lista estão populados com o resultado da operação map(). A operação filterRemoveNonGraded() deve filtrar os nós que não possuem notas válidas (caso de "ausência de nota") e retornar uma nova lista do tipo DLinkedList contendo apenas os nós com notas válidas.
filterRemoveGraded (*DLinkedList*)	Recebe como parâmetro uma lista duplamente encadeada do tipo DLinkedList, sendo que os nós da lista estão populados com o resultado da operação map(). A operação filterRemoveGraded() deve filtrar os nós que possuem notas válidas e retornar uma nova lista do tipo DLinkedList contendo apenas os nós com notas inválidas (caso de "ausência de nota").
filterRemoveBelowAverage (*DLinkedList*, *float*)	Recebe como parâmetro uma lista duplamente encadeada do tipo DLinkedList, sendo que os nós da lista estão populados com o resultado da operação filterRemoveNonGraded(), e a média de notas válidas, calculadas com a operação reduce() (ver a seguir). A operação filterRemoveBelowAverage() deve filtrar os nós que possuem notas abaixo da média e retornar uma nova lista do tipo DLinkedList contendo apenas os nós com notas acima da média.
reduce(*DLinkedList*)	Recebe como parâmetro uma lista duplamente encadeada do tipo DLinkedList, sendo que os nós da lista estão populados com o resultado da operação filterRemoveNonGraded(). A operação reduce() deve calcular a média das notas contidas na coleção de dados passada como parâmetro e retornar a média calculada.
mapToString(*DLinkedList*)	Recebe como parâmetro uma lista duplamente encadeada do tipo DLinkedList, sendo que os nós da lista estão populados com o resultado da operação map(). A operação mapToString() deve mapear todos os nós da coleção de dados passada como parâmetro para uma única String, sendo que

	cada dado de uma pessoa é separado por ponto-e-vírgula (;) e cada pessoa é separada por uma quebra de linha.

5) Conteúdo do arquivo **src/MainApl2.java**:

- A classe MainApl2 (que **não** pertence ao pacote apl2) já contém instruções que chamam as operações das classes Operation e DLinkedList. Assumindo que essas classes foram implementadas corretamente, o código deve compilar e executar sem problemas/erros.
- Há vários comentários TODO ("a fazer") no código da classe MainApl2 que descrevem o que sua equipe deve implementar no método main() para que o projeto funcione corretamente.

ATENÇÃO! As instruções que já existem no método main() da classe MainApl2 **NÃO** devem ser alteradas! Apenas os comentários TODO que devem ser substituídos por código que realizam corretamente o que está descrito em cada comentário TODO.

Observações:

- O ***Apêndice B: Saída do projeto (terminal)*** apresenta a saída gerada pelo programa, no terminal do sistema, após a execução do projeto, assumindo que todas as tarefas foram realizadas e implementadas corretamente pela equipe.
- O ***Apêndice C: Saída do projeto (arquivo dados.csv)*** apresenta o conteúdo do arquivo dados.csv gerado pelo programa, após a execução do projeto, assumindo que todas as tarefas foram realizadas e implementadas corretamente pela equipe.

No arquivo **src/MainApl2.java** também implementar um menu de opções encarregado de executar algumas operações individualmente para a Aplicação 2, como a seguir:

Sistema Conversor de Notas

1) Dados originais: lê arquivo dados.txt e apresenta todos os dados do Sistema de Notas Legado;
2) Dados convertidos: gera arquivo dados.csv e apresenta todos os dados do Sistema de Notas Atualizado;
3) Lista notas filtradas válidas: apresenta os dados somente das notas válidas filtradas;
4) Lista notas filtradas inválidas: apresenta os dados somente das notas filtradas pela "ausência de notas";
5) Média de notas válidas: apresenta a média das notas válidas filtradas;
6) Notas acima da média: apresenta os dados para as notas acima da média;
7) Lista mapeada para uma única string: apresenta a String contendo os dados mapeados;
8) Finaliza sistema.

Obs.: Ao implementar o menu acima e para fazer os testes com ele, deixar em comentário os códigos internos que constam no arquivo **src/MainApl2.java,** que apresenta o resultado encontrado no **Apêndice B: Saída do projeto (terminal).**

Desenvolvimento e explicativo

Grupo:
A atividade deve ser realizada em **grupo de, no máximo, 5 pessoas**.

Um único aluno do grupo deverá publicar o trabalho no Moodle.

Código:

- A solução deve ser implementada em linguagem Java, seguindo o texto apresentado neste documento.
- A solução não deve usar estruturas de dados oferecidas pela linguagem Java (projetos que usarem estruturas de dados oferecidas pela linguagem Java serão desconsiderados – zero).
- Lembre-se de incluir a identificação do grupo (nome completo e TIA de cada integrante) nos locais indicados nos arquivos do projeto.
- Inclua as referências (livros, artigos, sites, entre outros) consultadas para solucionar a atividade, como comentário no arquivo MainApl2.java.

Relatório:

- Além da solução escrita em Java, o grupo deverá entregar um relatório com Casos de teste da execução do seu programa e uma explicação para:
  - Como a base de dados original é mapeada para a nova representação dos dados do sistema de notas (operação map);
  - Como funcionam os filtros da classe Operation (operações filter*);
  - Como funciona o cálculo da média das notas válidas (operação reduce);
  - Como funciona o mapeamento do conteúdo da lista para string, de forma a preparar o conteúdo da lista para ser salvo em um arquivo CSV (operação mapToString).
- Inclua a identificação do grupo (nome completo e TIA de cada integrante) no relatório.
- Use a sua solução na explicação do relatório.
- Você pode usar quaisquer recursos que achar necessário para o relatório, tais como slides, captura do projeto rodando etc.
- A entrega pelo Moodle (relatório + fontes em um arquivo compactado no formato “zip”) da Aplicação 2 deve ser realizada no dia 22/05 (segunda-feira) ou 24/03 (quarta), dependendo do dia da sua turma de laboratório, até as 23h59min.
- Você deve apresentar a Apl2 (com o relatório/execução e/ou com uma Apresentação) no dia 23 de maio (turmas com aula de laboratório na terça-feira) e 25 de maio (turmas com aula de laboratório na quinta-feira) no tempo máximo de 7 minutos. O rigor no tempo de apresentação é ESSENCIAL, então, fazer prévias para não ultrapassar o limite de tempo. Outra coisa, deixar tudo preparado para apresentar para não atrasar por imprevistos.

Entrega

- **Código**: Compacte o código-fonte (somente arquivos *.java) no **formato zip**.
  No arquivo zip, inclua um **arquivo texto** (.txt) contendo a identificação do grupo e o link do vídeo no Youtube (veja o item a seguir).

- **Relatório**: Com as informações solicitadas anteriormente.

Atenção: O arquivo zip não deve conter arquivos intermediários e/ou pastas geradas pelo compilador/IDE (ex. arquivos *.class, etc.).

Critérios de avaliação

A nota da atividade é calculada de acordo com os critérios da tabela a seguir.

Item avaliado	Pontuação máxima
Relatório + execução da Apl2 + código fonte da Apl2	até 8,0 pontos
Apresentação.	até 2,0 pontos

Tabela 1 - Critérios de avaliação.

A tabela a seguir contém critérios de avaliação que podem **reduzir** a nota final da atividade.

Item indesejável	Redução de nota
O projeto é cópia de outro projeto.	Projeto é zerado
O projeto usa estruturas de dados oferecidas pela linguagem Java.	Projeto é zerado
Há erros de compilação e/ou o programa trava durante a execução.[1]	-1,0 ponto
O projeto foi alterado em partes que não deveria (ex. assinaturas dos métodos da classe Operation, partes da main() não relacionadas aos comentários TODO, etc).	-1,0 ponto
Não há identificação do grupo (código-fonte e vídeo). Não há indicação de referências (código-fonte). Arquivos enviados em formatos incorretos. Arquivos e/ou pastas intermediárias que são criadas no processo de compilação foram enviadas junto com o código-fonte. A apresentação que possui mais de 7 (sete) minutos.	-1,0 ponto

Tabela 2 - Critérios de avaliação (redução de nota).

Observação: O código-fonte será compilado e executado com o JDK 17.0.6 na plataforma Windows.

Apêndice A: Dicas gerais

1) Analise a saída do programa apresentada nos Apêndices B e C para ter uma ideia de como cada operação da classe Operation funciona.

2) Observe que map() converte o formato:
ID#Nome#Inteiro#Decimal

Para o formato:
23.S1-ID;Nome;Inteiro.Decimal

[1] Sobre erros de compilação: considere apenas erros. Não há problema se o projeto tiver *warnings* (embora *warnings* podem avisar sobre possíveis travamentos em tempo de execução, como loop infinito, divisão por zero etc.).

3) A lista encadeada que sua equipe deve implementar é do tipo lista duplamente encadeada, sendo possível navegar pelos nós da lista em duas direções (avançando do início ao fim e retrocedendo do fim ao início), enquanto a lista original do projeto é do tipo lista simplesmente encadeada.

4) O método estático `loadTextFileToString()` da classe `Data` está implementado de tal forma que, para ler um arquivo texto chamado `exemplo.txt` e salvar o conteúdo do arquivo em uma `String conteudo`, podemos escrever um código similar ao abaixo:

```
try {
  String conteudo = Data.loadTextFileToString("exemplo.txt");
} catch (IOException e) {
  System.err.println("Arquivo não encontrado!");
  e.printStackTrace();
  System.exit(-1); // Caso queira encerrar o programa.
}
```

Usando o Eclipse IDE, o arquivo `exemplo.txt` deve estar localizado na raiz do projeto para que o Eclipse reconheça o arquivo indicado como parâmetro do método `Data.loadTextFileToString()`.

5) O método estático `saveStringToTextFile()` da classe `Data` está implementado de tal forma que, para salvar o conteúdo de uma `String conteudo` em um arquivo texto chamado `exemplo.txt`, podemos escrever um código similar ao abaixo:

```
String conteudo = "ABC\n123\nXYZ";
try {
  Data.saveStringToTextFile("exemplo.txt", conteudo);
} catch (IOException e) {
  System.err.println("Erro ao gravar arquivo!");
  e.printStackTrace();
}
```

Usando o Eclipse IDE, o arquivo `exemplo.txt` será salvo na raiz do projeto com o conteúdo:

```
ABC
123
XYZ
```

6) É possível separar uma `String` em substrings usando o método `split()`, que recebe como parâmetro uma `String` descrevendo uma expressão regular (*regex*). Analise os seguintes exemplos, que usam a mesma `String teste`.

```
String teste = "Hello_World_123\nTest_Bla bla bla";
```

Exemplo 1: `split("_")` Separa string pelo símbolo underscore _.

```
String s1[] = teste.split("_");
for (int i = 0; i < s1.length; ++i) {
  System.out.println("s1[" + i + "] = " + s1[i]);
}
```

Saída do exemplo 1:

```
s1[0] = Hello
s1[1] = World
s1[2] = 123
Test
s1[3] = Bla bla bla
```

Exemplo 2: `split("\\r?\\n|\\r")` Separa string pela quebra de linha `\r\n` ou `\n` ou `\r` (depende do sistema operacional/formato do arquivo texto).

```
String s2[] = teste.split("\\r?\\n|\\r");
for (int i = 0; i < s2.length; ++i) {
  System.out.println("s2[" + i + "] = " + s2[i]);
}
```

Saída do exemplo 2:

```
s2[0] = Hello_World_123
s2[1] = Test_Bla bla bla
```

Exemplo 3: `split("_|\\r?\\n|\\r")` Separa string pelo underscore ou pela quebra de linha.

```
String s3[] = teste.split("_|\\r?\\n|\\r");
for (int i = 0; i < s3.length; ++i) {
  System.out.println("s3[" + i + "] = " + s3[i]);
}
```

Saída do exemplo 3:

```
s3[0] = Hello
s3[1] = World
s3[2] = 123
s3[3] = Test
s3[4] = Bla bla bla
```

Apêndice B: Saída do projeto (terminal)

>>>>>>>>>> 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) <<<<<<<<<<

>>>>>>>>>> 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 <<<<<<<<<<

>>>>>>>>>> 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) <<<<<<<<<<

>>>>>>>>>> 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') <<<<<<<<<<

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

>>>>>>>>>> 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 <<<<<<<<<<

>>>>>>>>>> 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 <<<<<<<<<<

>>>>>>>>>> test1 >>>>>>>>>>
23.S1-888 <- (23.S1-999; Ivone; 99.9) -> 23.S1-123
<<<<<<<<<< test1 <<<<<<<<<<

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

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

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

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

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

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

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

testList.getHead(): null <- (XYZ; Jane Doe; 9.9) -> null
testList.getTail(): null <- (XYZ; Jane Doe; 9.9) -> null

>>>>>>>>>> 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() >>>>>>>>>>
(0)

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

Apêndice C: Saída do projeto (arquivo dados.csv)

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