UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA: Qualidade de Software
PROFESSOR: Samyr Beliche Vale
ALUNOS: Ingrid Coelho Carvalho, Renef Ricardo Costa da Silva e Thiago Augusto Pereira Amaral

Trabalho de Qualidade de Software

Link do projeto no GitHub

1.0 Sobre o sistema

1.1 Requisitos Funcionais

1. Registro de Salas
    - A aplicação deve permitir o registro de salas com nomes únicos.
    - Cada sala pode ter no máximo 05 (cinco) discentes.
2. Registro de Discentes
    - A aplicação deve permitir o registro de discentes.
    - Cada discente pode pertencer a uma única sala.
    - Os nomes dos discentes podem se repetir.
3. Cálculo da Média de Notas
    - A aplicação deve permitir a entrada de 03 (três) notas para cada discente, variando de 0,0 a 10,0, com uma casa decimal.
    - O programa deve calcular a média das 03 (três) notas usando a fórmula: $MÉDIA = \frac{NOTA1 + NOTA2 + NOTA3}{3}$
    - A média calculada deve ser exibida na tela.
4. Registro e Utilização da Quarta Nota
    - A aplicação deve possibilitar o registro de uma quarta nota (nota4) para o discente, caso a média das três primeiras notas seja inferior a 7,0.
    - A quarta nota (nota4) deve substituir a menor das três notas originais.

- Se a nota4 for inferior ou igual à menor das notas originais, ela não deve ser utilizada no cálculo da nova média.
- A nova média, se recalculada, deve ser exibida na tela.

5. Interface de Entrada de Dados
    - A aplicação deve possuir uma interface que permita ao usuário informar as notas dos discentes e o nome da sala.
6. Persistência de Dados
    - Os dados de salas, discentes e notas devem ser armazenados persistentemente (nesse caso em JSON).
7. Validação das Notas
    - A aplicação deve validar que as notas inseridas estão no intervalo de 0,0 a 10,0 e possuem uma casa decimal.

1.2 Regras de Negócio

1. Registro de Salas
    - O nome da sala deve ser único.
    - Cada sala pode conter no máximo 05 (cinco) discentes.
2. Cálculo da Média Inicial
    - A média das três primeiras notas deve ser calculada e exibida sempre que as notas forem registradas.
3. Substituição da Nota e Recálculo da Média
    - A quarta nota só deve ser registrada e utilizada se a média das três primeiras notas for inferior a 7,0.
    - A menor das três notas deve ser substituída pela quarta nota, se esta for maior que a menor nota original.
4. Validação da Quarta Nota
    - A quarta nota não deve ser considerada se for inferior ou igual à menor das três notas originais.

1.3 Regras de Validação e Mensagens do Menu

1. Nomes de Salas Repetidos
    - Mensagem: “O nome da sala já existe! Por favor, insira um nome único para a sala.”
2. Número de Discentes Excedido
    - Mensagem: “A sala não pode ter mais de 5 discentes.”
3. Notas Fora do Intervalo Válido
    - Mensagem: “Nota inválida! As notas devem estar entre 0,0 e 10,0.”
4. Média Calculada

- Mensagem: “A média das notas é: [média]”

5. Média Recalculada
    - Mensagem: “A nova média das notas é: [nova média]”

1.4 Persistência de Dados

1. Armazenamento de Salas
    - As informações das salas devem ser armazenadas persistentemente.
2. Armazenamento de Discentes
    - As informações dos discentes e suas notas devem ser armazenadas persistentemente.

1.5 Casos de Teste Funcional de Unidade

1. Teste de Cálculo da Média Simples
    - Objetivo: Verificar se a aplicação calcula corretamente a média das três primeiras notas.
    - Script de Teste: Inserir três notas válidas e verificar a média calculada.
    - Resultado Esperado: A média deve ser exibida corretamente na tela.
2. Teste de Registro da Quarta Nota e Recálculo da Média
    - Objetivo: Verificar se a aplicação registra a quarta nota corretamente e recalcula a média quando a média das três primeiras notas é inferior a 7,0.
    - Script de Teste: Inserir três notas que resultem em uma média inferior a 7,0, inserir uma quarta nota maior que a menor das três primeiras notas, e verificar a nova média calculada.
    - Resultado Esperado: A nova média deve ser calculada corretamente, substituindo a menor das três notas originais.
3. Teste de Validação das Notas
    - Objetivo: Verificar se a aplicação valida corretamente as notas inseridas.
    - Script de Teste: Tentar inserir notas fora do intervalo válido (menor que 0,0 ou maior que 10,0) e verificar a mensagem de erro.
    - Resultado Esperado: A mensagem de erro “Nota inválida! As notas devem estar entre 0,0 e 10,0.” deve ser exibida.
4. Teste de Limite de Discentes na Sala
    - Objetivo: Verificar se a aplicação impede o registro de mais de cinco discentes em uma sala.

- Script de Teste: Tentar registrar seis discentes em uma sala e verificar a mensagem de erro.
- Resultado Esperado: A mensagem de erro “A sala não pode ter mais de 5 discentes.” deve ser exibida.

2.0 Código da Aplicação

2.1.0 Aluno Controller

2.1.1 Registro e exclusão dos discentes

```
16
17      public AlunoModel criarAluno(String nome) {  5 usages  Renef13
18          List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
19          int idAluno = alunos != null ? alunos.size() + 1 : 1;
20          AlunoModel novoAluno = new AlunoModel(idAluno, nome);
21          alunos.add(novoAluno);
22          jsonManager.salvarDadosAlunos(alunos);
23          return novoAluno;
24      }
25
26      public int excluirAluno(int idAluno) {  no usages  Renef13
27          int removed = 0;
28          List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
29          if (alunos != null) {
30              if(alunos.removeIf(aluno -> aluno.getIdAluno() == idAluno)){
31                  removed = 1;
32              }
33              jsonManager.salvarDadosAlunos(alunos);
34
35              List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();
36              if (turmas != null) {
37                  for (TurmaModel turma : turmas) {
38                      turma.removerAluno(idAluno);
39                  }
40                  jsonManager.salvarDadosTurmas(turmas);
41              }
42          }
43          return removed;
44      }
```

Figura 1 - Código para registro e exclusão dos discentes

1. public AlunoModel criarAluno(String nome):
    - Este método cria um novo aluno com o nome fornecido.
    - Calcula o próximo ID de aluno com base no tamanho atual da lista de alunos.
    - Cria e adiciona o novo aluno à lista de alunos.
    - Retorna um novo aluno.
2. public int excluirAluno(int idAluno):
    - Este método exclui um aluno com o ID fornecido.
    - Se encontrar um aluno com o ID fornecido, remove-o da lista.
    - Retorna 1 se o aluno foi removido e 0 caso não tenha sido.

2.1.2 Atribuição de nota ao discente

```java
public void adicionarNotaAoAluno(int idAluno, float nota) {  11 usages   Renef13
    List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
    if (alunos != null) {
        for (AlunoModel aluno : alunos) {
            if (aluno.getIdAluno() == idAluno) {
                if (aluno.getListaNotas() != null && aluno.getListaNotas().size() >= aluno.getNUMAXNOTAS()) {
                    float mediaAtual = aluno.calcularMedia();
                    if (mediaAtual < 7) {
                        float menorNota = Collections.min(aluno.getListaNotas());
                        if (nota > menorNota) {
                            aluno.getListaNotas().remove(menorNota);
                            aluno.getListaNotas().add(nota);
                            aluno.setMedia(aluno.calcularMedia());
                            jsonManager.salvarDadosAlunos(alunos);
                        } else {
                            System.out.println("Nota não adicionada.");
                        }
                    } else {
                        System.out.println("O aluno já possui três notas e sua média é maior que 7.0. Não é possível adicionar mais notas.");
                        return;
                    }
                } else {
                    aluno.adicionarNota(nota);
                    if (aluno.getListaNotas().size() == aluno.getNUMAXNOTAS()) {
                        float novaMedia = aluno.calcularMedia();
                        System.out.println("Nova média calculada: " + novaMedia);
                    } else {
                        System.out.println("Nota adicionada com sucesso.");
                    }
                    jsonManager.salvarDadosAlunos(alunos);
                    return;
                }
            }
        }
    }
}
```

Figura 2 - Código para atribuição de nota ao discente

1. public void adicionarNotaAoAluno(int idAluno, float nota):
    - Este método adiciona uma nota a um aluno com o ID fornecido.
    - Se encontrar um aluno com o ID fornecido:
        - Verifica se o aluno já possui o número máximo de notas. Se sim, verifica se a média atual do aluno é inferior a 7.
            - Se for, encontra a menor nota atual do aluno.

        - Se a nova nota for maior que a menor nota atual, substitui a menor nota pela nova nota e recalcula a média.
        - Caso contrário, imprime “Nota não adicionada.”
    - Se não, adiciona a nova nota à lista de notas do aluno.
        - Se o número de notas atingir o máximo, calcula a nova média e imprime “Nova média calculada: {novaMedia}”.
        - Caso contrário, imprime “Nota adicionada com sucesso.”
    - Salva a lista atualizada de alunos.

2.1.3 Editar o nome de um discente já cadastrado

```
45
46      public void editarNomeAluno(int idAluno, String novoNome) {  no usages  Renef13
47          List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
48          if (alunos != null) {
49              for (AlunoModel aluno : alunos) {
50                  if (aluno.getIdAluno() == idAluno) {
51                      aluno.setNome(novoNome);
52                      jsonManager.salvarDadosAlunos(alunos);
53                      break;
54                  }
55              }
56          }
57      }
```

Figura 3 - Código para edição do nome de um discente já cadastrado

- public void editarNomeAluno(int idAluno, String novoNome):
    - Este método permite editar o nome de um aluno com o ID fornecido.
    - Se encontrar um aluno com o ID fornecido:
        - Define o novo nome para o aluno
        - Caso contrário, nada será feito

2.1.4 Busca discente por nome e por ID

```java
96      public List<AlunoModel> buscarAlunos(String nome) {  3 usages  Renef13
97          List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
98          List<AlunoModel> alunosEncontrados = new ArrayList<>();
99          if (alunos != null) {
100             for (AlunoModel aluno : alunos) {
101                 if (aluno.getNome().equals(nome)) {
102                     alunosEncontrados.add(aluno);
103                 }
104             }
105         }
106         return alunosEncontrados;
107     }
108     public AlunoModel buscarAlunosPorId(int idAluno) {  5 usages  Renef13
109         List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
110         AlunoModel alunoEncontrado = null;
111         if (alunos != null) {
112             for (AlunoModel aluno : alunos) {
113                 if (aluno.getIdAluno() == idAluno) {
114                     alunoEncontrado = aluno;
115                 }
116             }
117         }
118         return alunoEncontrado;
119     }
120 }
```

Figura 4 - Código para busca de discentes por nome(buscarAlunos()) e por ID(buscarAlunosPorId())

- public List<AlunoModel> buscarAlunos(String nome):
  - Este método busca alunos com base no nome fornecido.
  - Se encontrar um aluno cujo nome seja igual ao nome fornecido, adiciona-o à lista alunosEncontrados.
  - Retorna a lista de alunos encontrados.

- public AlunoModel buscarAlunosPorId(int idAluno):
  - Este método busca alunos com base no ID fornecido.
  - Se encontrar um aluno com o ID fornecido, retorna esse aluno
  - Caso contrário, retorna null

2.2.0 Turma Controller

2.2.1 Criação de Turmas

```
17      public void criarTurma(String nome) {  4 usages  Renef13
18          List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();
19
20          for (TurmaModel turma : turmas) {
21              if (turma.getNome().equalsIgnoreCase(nome)) {
22                  throw new RuntimeException("O nome da sala já existe! Por favor, insira um nome único para a sala.");
23              }
24          }
25
26          int idTurma = turmas != null ? turmas.size() + 1 : 1;
27          TurmaModel novaTurma = new TurmaModel(idTurma, nome,  qtdAlunos: 0,  mediaTurma: 0, new ArrayList<>());
28          turmas.add(novaTurma);
29          jsonManager.salvarDadosTurmas(turmas);
30      }
```

Figura 5 - Código para criação de turmas

- public List<AlunoModel> buscarAlunos(String nome):
    - A função criarTurma recebe um parâmetro nome do tipo String
    - Em seguida, ela verifica se já existe uma turma com o mesmo nome (ignorando maiúsculas e minúsculas) percorrendo a lista de turmas existentes.
    - Se encontrar uma turma com o mesmo nome, lança uma exceção com a mensagem “O nome da sala já existe! Por favor, insira um nome único para a sala.”
    - Caso contrário, gera um novo ID para a turma (incrementando o tamanho da lista de turmas) e cria uma nova instância de TurmaModel com o ID, nome, e outras informações padrão.

2.2.2 Edição e exclusão de turmas existentes

```java
public void editarNomeTurma(String nomeAtual, String novoNome) {  no usages  Renef13
    List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();
    if (turmas != null) {
        for (TurmaModel turma : turmas) {
            if (turma.getNome().equalsIgnoreCase(nomeAtual)) {

                for (TurmaModel outraTurma : turmas) {
                    if (outraTurma.getNome().equalsIgnoreCase(novoNome)) {
                        throw new IllegalArgumentException("Nome existente: " + novoNome);
                    }
                }
                turma.setNome(novoNome);
                jsonManager.salvarDadosTurmas(turmas);
                return;
            }
        }
    }
    throw new NoSuchElementException("Turma não encontrada: " + nomeAtual);
}

public void excluirTurma(String nomeTurma) {  no usages  Renef13
    List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();
    if (turmas != null) {
        String nomeTurmaUppercase = nomeTurma.toUpperCase();

        turmas.removeIf(turma -> turma.getNome().toUpperCase().equals(nomeTurmaUppercase));

        jsonManager.salvarDadosTurmas(turmas);
    }
}
```

Figura 6 - Código para editar nome da turma e excluir turma, respectivamente.

1. editarNomeTurma(String nomeAtual, String novoNome):
    - Essa função recebe dois parâmetros: nomeAtual (o nome da turma que você deseja renomear) e novoNome (o novo nome que você deseja atribuir à turma).
    - Verifica se existe uma turma com o nome especificado em nomeAtual (ignorando maiúsculas e minúsculas).
    - Se encontrar a turma, verifica se o novoNome já está sendo usado por outra turma. Se sim, lança uma exceção com a mensagem “Nome existente: {novoNome}”.
    - Caso contrário, atualiza o nome da turma para novoNome e salva os dados atualizados.
    - Se a turma não for encontrada, lança uma exceção com a mensagem “Turma não encontrada: {nomeAtual}”.

2. excluirTurma(String nomeTurma):

- Essa função recebe um parâmetro: nomeTurma (o nome da turma que você deseja excluir).
- Em seguida, converte o nomeTurma para letras maiúsculas.
- Remove a turma da lista se o nome da turma (em letras maiúsculas) corresponder.
- Finalmente, salva os dados atualizados.

2.2.3 Adicionar Alunos na Turma

```java
public void adicionarAlunoNaTurma(String nomeTurma, AlunoModel aluno) throws Exception {  5 usages  Renef13 *
    List<AlunoModel> alunos = jsonManager.carregarDadosAlunos();
    List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();

    boolean alunoExisteNoJSON = alunos != null && alunos.stream()
            .anyMatch(a -> a.getNome().equalsIgnoreCase(aluno.getNome()));

    if (!alunoExisteNoJSON) {
        throw new Exception("O aluno não está cadastrado.");
    }

    boolean alunoJaNaTurma = turmas != null && turmas.stream()
            .filter(t -> t.getNome().equalsIgnoreCase(nomeTurma))
            .anyMatch(turma -> turma.getAlunos().stream()
                    .anyMatch(a -> a.getNome().equalsIgnoreCase(aluno.getNome())));

    if (alunoJaNaTurma) {
        throw new Exception("O aluno já está presente em uma turma.");
    }

    for (TurmaModel turma : turmas) {
        if (turma.getNome().equalsIgnoreCase(nomeTurma)) {
            turma.adicionarAluno(aluno);
            jsonManager.salvarDadosTurmas(turmas);
            System.out.println("Aluno adicionado à turma com sucesso.");
            return;
        }
    }

    throw new Exception("Turma não encontrada.");
}
```

Figura 7 - Código para editar nome da turma e excluir turma, respectivamente.

1. editarNomeTurma(String nomeAtual, String novoNome):
   - verifica se o aluno já existe nos dados carregados . Se o aluno não existir, imprime “O aluno não está cadastrado.” e retorna.

   - Verifica se o aluno já está presente em alguma turma:
     i. Carrega a lista de turmas existentes
     ii. Filtra as turmas pelo nome especificado em nomeTurma.
     iii. Verifica se o aluno já está presente em alguma dessas turmas. Se sim, imprime “O aluno já está presente em uma turma.” e retorna.

   - Se a turma for encontrada (com base no nome especificado), tenta adicionar o aluno à turma.

- Se a turma não for encontrada, imprime “Turma não encontrada.”

2.2.4 Remove Alunos e Busca turmas

```
103     public void removerAlunoDaTurma(int idTurma, int idAluno) {  no usages  Renef13
104         List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();
105         if (turmas != null) {
106             for (TurmaModel turma : turmas) {
107                 if (turma.getIdTurma() == idTurma) {
108                     turma.removerAluno(idAluno);
109                     jsonManager.salvarDadosTurmas(turmas);
110                     break;
111                 }
112             }
113         }
114     }
115
116     public TurmaModel buscarTurma(String nome) {  no usages  Renef13
117         List<TurmaModel> turmas = jsonManager.carregarDadosTurmas();
118         if (turmas != null) {
119             String nomeUpperCase = nome.toUpperCase();
120             for (TurmaModel turma : turmas) {
121                 if (turma.getNome().toUpperCase().equals(nomeUpperCase)) {
122                     return turma;
123                 }
124             }
125         }
126         return null;
127     }
128 }
```

Figura 8 - Código para remover aluno de uma turma e para buscar turma, respectivamente.

1. removerAlunoDaTurma(int idTurma, int idAluno):
    - Percorre a lista de turmas e verifica se o ID da turma corresponde ao idTurma especificado.
    - Se encontrar a turma, chama o método removerAluno(idAluno) nessa turma para remover o aluno com o ID especificado.
    - Finalmente, salva os dados atualizados das turmas

2. buscarTurma(String nome):
    - Percorre a lista de turmas e verifica se o nome da turma corresponde
    - Se encontrar a turma, retorna essa turma.
    - Caso contrário, retorna null.

3.0 Testes

Nesta etapa teremos os testes implementados para essa aplicação utilizando jUnit.

3.1 testCalcularMediaSimples()

```
9     @Test  Renef13
10    public void testCalcularMediaSimples() {
11        AlunoController alunoController = new AlunoController();
12        TurmaController turmaController = new TurmaController();
13
14        AlunoModel novoAluno = alunoController.criarAluno( nome: "Ingrid Coelho");
15        AlunoModel ingrid = alunoController.buscarAlunosPorId(novoAluno.getIdAluno());
16
17        turmaController.criarTurma( nome: "Qualidade de Software");
18        turmaController.adicionarAlunoNaTurma( nomeTurma: "Qualidade de Software", ingrid);
19
20        float[] notas = { 8f, 8f, 7f };
21        float media = (notas[0] + notas[1] + notas[2]) / notas.length;
22        alunoController.adicionarNotaAoAluno(ingrid.getIdAluno(), notas[0]);
23        alunoController.adicionarNotaAoAluno(ingrid.getIdAluno(), notas[1]);
24        alunoController.adicionarNotaAoAluno(ingrid.getIdAluno(), notas[2]);
25
26        ingrid = alunoController.buscarAlunosPorId(ingrid.getIdAluno());
27        float mediaIngrid = ingrid.getMedia();
28
29        System.out.println("ID do Aluno: " + ingrid.getIdAluno());
30        System.out.println("Nome do Aluno: " + ingrid.getNome());
31        System.out.println("Notas do Aluno: " + ingrid.getListaNotas());
32        System.out.println("Média do Aluno: " + mediaIngrid);
33
34        assertEquals(media, mediaIngrid,  message: "A média calculada não está correta.");
35    }
```

Figura 9 - Código para testar o cálculo da média.

3.1.1 Resultados do jUnit para média simples

- Parâmetros: notas = 8, 8, 7
- Resultado esperado: média = 7,6

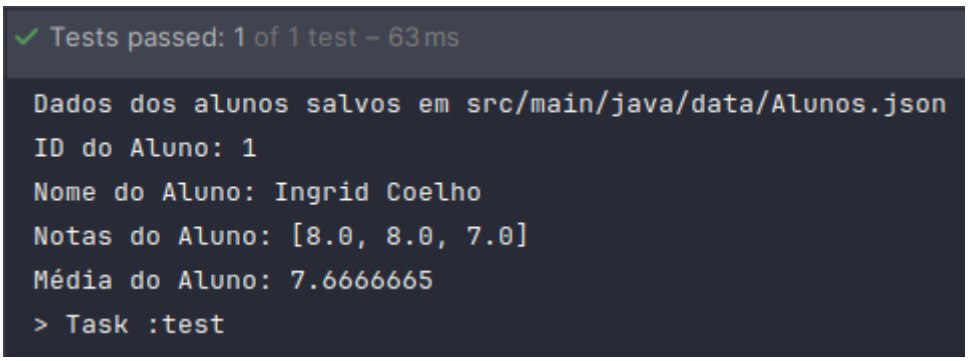

Figura 10 - Resultado do teste de média com valores válidos

3.2 testCalcularMediaNovaNota()

```
36      @Test  Renef13 *
37      public void testCalcularMediaNovaNota() {
38          AlunoController alunoController = new AlunoController();
39          TurmaController turmaController = new TurmaController();
40
41          AlunoModel novoAluno =alunoController.criarAluno( nome: "Thiago Amaral");
42          AlunoModel thiago = alunoController.buscarAlunosPorId(novoAluno.getIdAluno());
43
44          //turmaController.criarTurma("Qualidade de Software");
45          turmaController.adicionarAlunoNaTurma( nomeTurma: "Qualidade de Software", thiago);
46
47          float[] notas = { 5f, 5f, 5f, 10 };
48          float media = (notas[0] + notas[1] + notas[3]) / thiago.getNUMAXNOTAS();
49          alunoController.adicionarNotaAoAluno(thiago.getIdAluno(), notas[0]);
50          alunoController.adicionarNotaAoAluno(thiago.getIdAluno(), notas[1]);
51          alunoController.adicionarNotaAoAluno(thiago.getIdAluno(), notas[2]);
52          alunoController.adicionarNotaAoAluno(thiago.getIdAluno(), notas[3]);
53
54          thiago = alunoController.buscarAlunosPorId(thiago.getIdAluno());
55          float mediaThiago = thiago.getMedia();
56
57          System.out.println("ID do Aluno: " + thiago.getIdAluno());
58          System.out.println("Nome do Aluno: " + thiago.getNome());
59          System.out.println("Notas do Aluno: " + thiago.getListaNotas());
60          System.out.println("Média do Aluno: " + mediaThiago);
61
62          assertEquals(media, mediaThiago,  message: "A média calculada não está correta.");
63      }
```

Figura 11 - Código para testar o cálculo da nova média com quarta nota.

3.2.1 Resultados do jUnit com quarta nota adicionada

- Parâmetros: notas = 5, 5, 5, 10
- Resultado esperado: média = 6,66

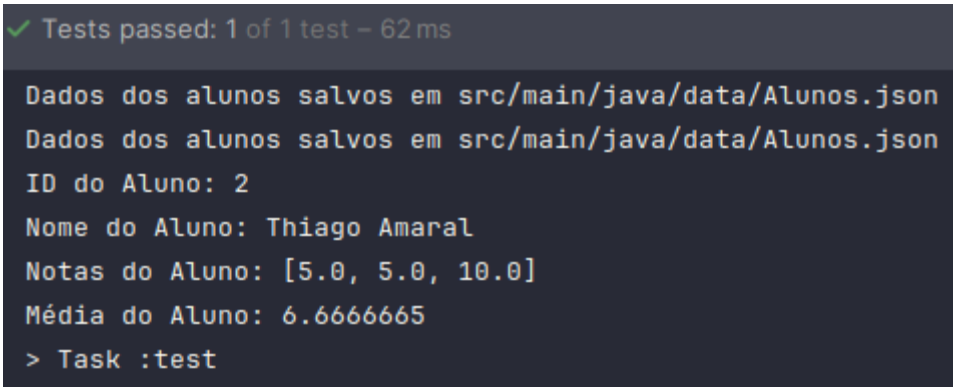

Figura 12 - Resultado do teste de média com adição de quarta nota.

3.2.2 Resultados do jUnit com quarta nota inferior a menor nota do aluno

- Parâmetros: notas = 5, 5, 5, 4
- Resultado esperado: média = 5

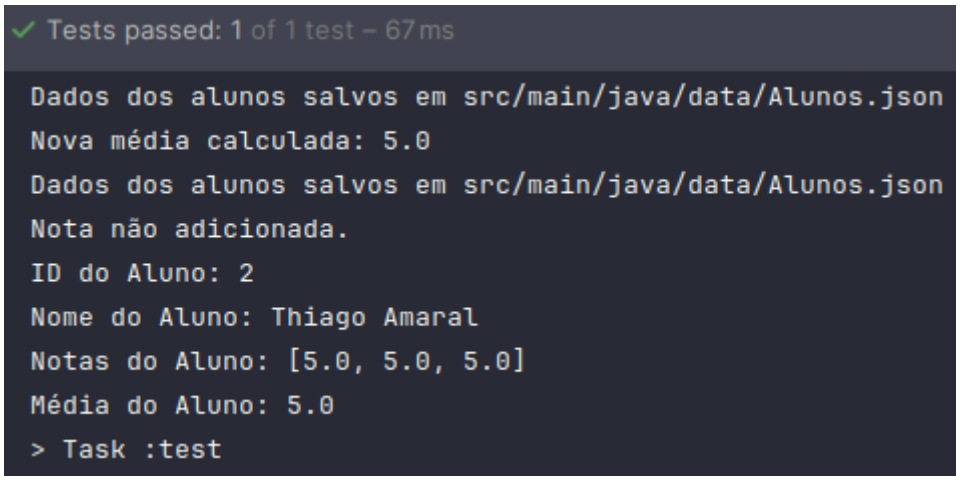

Figura 13 - Resultado do teste de média com adição de quarta nota menor que as já inseridas.

3.3 testInserirNotasForaDoPadrao()

```
64      @Test  Renef13 *
65      public void testInserirNotasForaDoPadrao() {
66          AlunoController alunoController = new AlunoController();
67          TurmaController turmaController = new TurmaController();
68
69          AlunoModel novoAluno = alunoController.criarAluno( nome: "Renef Silva");
70          AlunoModel renef = alunoController.buscarAlunosPorId(novoAluno.getIdAluno());
71
72          turmaController.adicionarAlunoNaTurma( nomeTurma: "Qualidade de Software", renef);
73
74          alunoController.adicionarNotaAoAluno(renef.getIdAluno(),  nota: 12f);
75          alunoController.adicionarNotaAoAluno(renef.getIdAluno(),  nota: 8f);
76          alunoController.adicionarNotaAoAluno(renef.getIdAluno(),  nota: 7f);
77          alunoController.adicionarNotaAoAluno(renef.getIdAluno(),  nota: 6f);
78
79          renef = alunoController.buscarAlunos( nome: "Renef Silva").getFirst();
80          System.out.println("Notas do Aluno: " + renef.getListaNotas());
81          System.out.println("Média do Aluno: " + renef.getMedia());
82
83          assertEquals( expected: 7, renef.getMedia(),  message: "A média deve ser 7.0.");
84      }
```

Figura 14 - Código para testar a inserção de notas fora do padrão e média.

3.3.1 Resultados do jUnit com nota maior que 10

- Parâmetros: notas = 12, 7, 8, 6
- Resultado esperado: média = 7

```
✓ Tests passed: 1 of 1 test – 255 ms

A nota deve estar entre 0 e 10.
Nota adicionada com sucesso.
Dados dos alunos salvos em src/main/java/data/Alunos.json
Nova média calculada: 7.0
Dados dos alunos salvos em src/main/java/data/Alunos.json
Notas do Aluno: [7.0, 8.0, 6.0]
Média do Aluno: 7.0
> Task :test
```

Figura 15 - Resultado do teste de média com tentativa de inserção de nota maior que 10.

3.3.2 Resultados do jUnit com nota negativa

- Parâmetros: notas = 7, 8, -6, 6
- Resultado esperado: média = 7

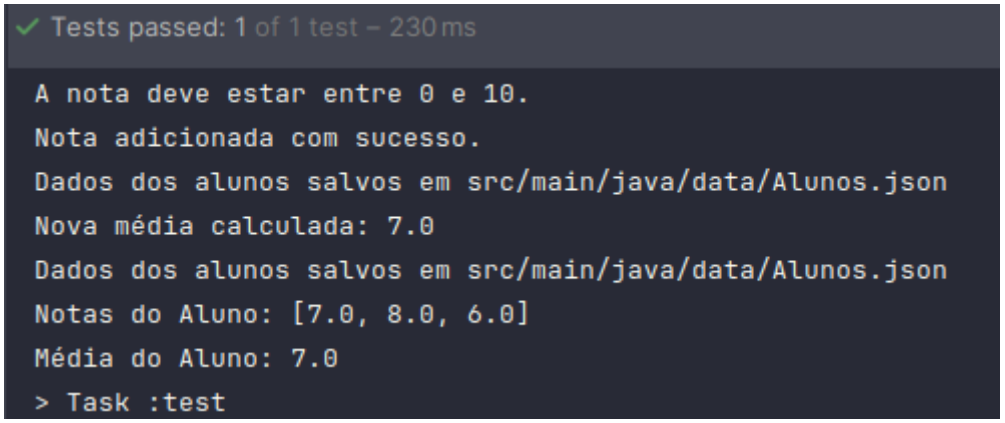

Figura 16 - Resultado do teste de média com tentativa de inserção de nota negativa.

3.4 testLimiteDeSala()

```
26      @Test  Renef13 *
27      public void testLimiteDeSala() throws Exception {
28          AlunoController alunoController = new AlunoController();
29          TurmaController turmaController = new TurmaController();
30
31          turmaController.criarTurma( nome: "Qualidade de Software");
32
33          for (int i = 0; i < 5; i++) {
34              AlunoModel alunoAluno = alunoController.criarAluno( nome: "Aluno " + (i + 1));
35              AlunoModel aluno = alunoController.buscarAlunosPorId(alunoAluno.getIdAluno());
36              turmaController.adicionarAlunoNaTurma( nomeTurma: "Qualidade de Software", aluno);
37          }
38
39          AlunoModel novoAluno6 = alunoController.criarAluno( nome: "Aluno 6");
40          AlunoModel aluno6 = alunoController.buscarAlunosPorId(novoAluno6.getIdAluno());
41
42          Exception exception = assertThrows(Exception.class, () -> {
43              turmaController.adicionarAlunoNaTurma( nomeTurma: "Qualidade de Software", aluno6);
44          });
45
46          assertEquals( expected: "A turma já possui o número máximo de alunos.", exception.getMessage());
47      }
```

Figura 17 - Código para testar o limite de alunos em uma sala.

3.4.1 Resultados do jUnit com tentativa de inserção de 6 alunos

- Parâmetros: alunos
- Resultado esperado: 5 alunos inseridos

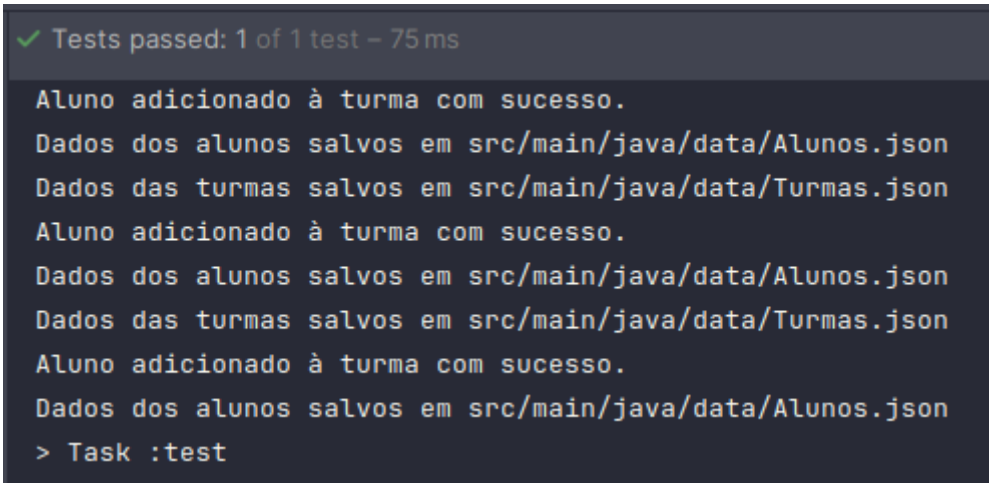

Figura 18 - Resultado do teste de tentativa de inserção de 6 alunos.

5.0 Conclusão

O projeto completo está disponível no GitHub. A implementação de novos testes e da interface de usuário será adicionada no futuro. Todos os testes realizados cobrem os requisitos da atividade. O código passou em todos os testes e não precisou de refatoração, tanto na parte sintática quanto nos requisitos e regras de negócio.