

Relatório Parcial

Escola

Augusto Samuel Modesto

Engenharia de Software

augusto.modesto@live.com

Tatielen Rodrigues Dutra Pereira

Engenharia Eletrônica

tatielen.rodrigues@hotmail.com

(Trancado)

Jamily Martins de Souza

Engenharia de Software

jamily.ms@gmail.com

Disciplina de Orientação a Objetos

Professora Dra. Milene Serrano

Gama, 18 de setembro de 2013

UnB - Campus Universitário Gama

Área Especial de Indústria Projeção A - UnB/Brasília, Setor Leste. CEP: 72.444-240

GAMA - DF

Relatório Parcial

Professora Dra. Milene Serrano Setembro, 2013

Relatório Parcial

Augusto Samuel Modesto, Tatielen Rodrigues Dutra Pereira, Jamily Martins de Souza

Engenharia de Software, Universidade de Brasília - Faculdade do Gama, Gama/DF  
Engenharia de Eletrônica, Universidade de Brasília - Faculdade do Gama, Gama/DF

Engenharia de Eletrônica, Universidade de Brasília - Faculdade do Gama, Gama/DF

[augusto.modesto@live.com](mailto:augusto.modesto@live.com), [tatielen.rodrigues@hotmail.com](mailto:tatielen.rodrigues@hotmail.com), jamily.ms@gmail.com

**Resumo**. Esse relatório é referente à Etapa 01 do trabalho da disciplina de Orientação a Objetos. O mesmo apresenta uma modelagem em UML, código fonte referente à implementação com testes de unidade, acesso ao repositório Git e relatório com os resultados obtidos do projeto de um software para uma escola de porte pequeno-médio, no qual possui o Ensino Infantil e Fundamental, chamada Colégio Século XXI.

**Palavras-chave**: Escola, Software, modelagem, código.

Sumário

1 Introdução 4

2 Breve Descrição 6

3 Modelagem 6

4 Implementação 11

Etapa I 11

Etapa II 12

Etapa III 13

Etapa IV 14

Etapa V 15

Etapa VI 16

5 Controle de Versão 17

# Introdução

O relatório será organizado em seções, ao qual pelo decorrer do curso, serão criadas. Isto é, o relatório será preenchido de acordo com cada atividade prática semanalmente. Todos os arquivos gerados serão disponibilizadas via GitHub além de serem registrados aqui.

**Seção 01** *(Semana 2 de setembro a 12 de setembro)*

Foram criadas as 7 principais classes do sistema proposto – podendo estás serem alteradas, excluídas ou trocadas ao decorrer do semestre. Além do mais, foram adicionadas as classes Aluno, Funcionario e Responsavel um construtor para que não seja possível a instância de objetos em que seriam necessários alguns dados importantes como nome e CPF por exemplo.

Alguns métodos foram adicionados para que as classes para cadastro de alunos, alteração de graduação dos professores entre outros. Vale ressaltar que, durante o semestre, esses métodos poderão sofrer alterações, substituições ou até mesmo exclusões a depender dos critério que forem adotados pelo grupo.

**Seção 02** *(Semana 13 de setembro a 18 de setembro)*

Ainda restarão 7 classes no projeto, no entanto foram adicionados novos métodos as classes, além do mais, foram alterados alguns atributos das classes para que se possa adequá-los ao desenvolvimento do sistema . Isso se deu por terem sido adicionados os relacionamentos de dependência e associação. Logo, para que o projeto pode-se ficar ainda mais robusto e com funcionalidade objetivas, foram efetivados novos teste de unidade e classe. Também foi adicionado na diagramação todas os novos métodos e alterações nos atributos.

**Seção 03** *(Semana 19 de setembro a 25 de setembro)*

Nessa semana foram abordados em sala de aula os conceitos de associação e agregação. Os conceitos são bem simulares, em que ambos são parte de um todo. No entanto, a agregação se difere da associação por possuir uma relação não tão forte quanto esta, portando uma semântica mais simples, TEM.

Foram feitas novas alterações no projeto, na diagramação e sua implementação. Algumas classes ganharam novos atributos e métodos, e em contrapartida, algumas classes perderão alguns atributos para se tornarem novas classes com novos atributos, novos métodos, novas funcionalidades. Além disso, foram adicionados a diagramação os novos relacionamentos implementados.

Foram criados novos testes de unidade nas classes Aluno, Professor e Direcao. Apesar de serem relativamente poucos tendo em vista o tamanho do projeto, o objetivo é implementar os testes aos poucos mesmo, e até o final, conseguir alcançar uma média considerável de teste de unidade.

**Seção 04** *(Semana 26 de setembro a 07 de outubro)*

Nessa semana foram abordados o conceito de herança, que possui semântica “é um” ou “é uma”. A herança possibilita um aumento na legibilidade do programa e na capacidade de escrita do programador, permitindo este a reutilização de códigos, ou seja, um dos principais objetivo da orientação a objeto. Sendo assim, atribuímos ao nosso projeto esse relacionamento da seguinte forma: A classe funcionário (classe pai) relacionada com duas classes filhas, a classe professor e a classe secretária.

Além do mais, foram feitos novos teste de unidade em duas classes para verificação do comportamento dos métodos em relação as classes.

**Seção 05** *(Semana 08 de outubro a 14 de outubro)*

Na semana anterior, abordamos o conceito de herança. Dentro deste relacionamento, encontramos o conceito de polimorfismo, portanto, polimorfismo está intimamente ligado à herança de classes. O polimorfismo é a técnica de modificar totalmente o código de um método herdado de uma superclasse, logo sobrescrever o método da classe pai na classe filha, sendo assim, esta semana implementamos esse conceito em nosso projeto de forma que, sobrescrevemos métodos da classe pai ( funcionário ) nas classes filhas ( professor, secretaria).

Além do mais, foram feitos novos teste de unidade em duas classes para verificação do comportamento dos métodos em relação as classes.

**Seção 06** *(Semana 15 de outubro a 22 de outubro)*

Nessa semana foram abordados o conceito de atributos e métodos abstratos e concretos e de atributos e métodos *static*, além de classe abstrata. A classe possibilita um aumento na legibilidade do programa e na capacidade de escrita do programador, além de trazer uma estática maior ao projeto. Os métodos e atributos de classe ou estáticos permitindo uma flexibilidade maior no código, no entanto, são necessários alguns cuidados na hora de sua utilização, pois este podem chamar métodos sem os mesmo possuírem uma instancia o que torna um “perigo” durante a montagem do código. Todos esses conceitos foram implementados no código essa semana e em sua UML.

Além do mais, foram feitos novos teste de unidade em uma classe para verificação do comportamento dos métodos em relação as classes. Também tiveram inicio aos teste de cobertura produzidos pelo framework EclEmma®.

# Breve Descrição

Desenvolvimento de um Sistema Orientado a Objeto sobre uma escola de porte pequeno-médio de ensino de Infantil e fundamental, em que foram consideradas, a princípio, as classes Aluno, Funcionário, Professor, Responsável, Disciplina, Direção e Sala.

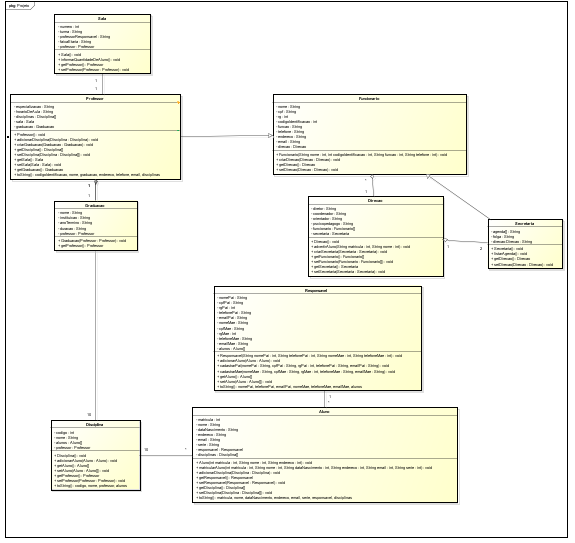
O objetivo principal desse projeto foi exercitar na prática como modelar e implementar um sistema baseado nos conceitos, princípios e fundamentos da Orientação a Objetos, tais como: classes, objetos, atributos, métodos, abstração e encapsulamento.

A seguir apresentaremos a modelagem parcial obtida até o momento.

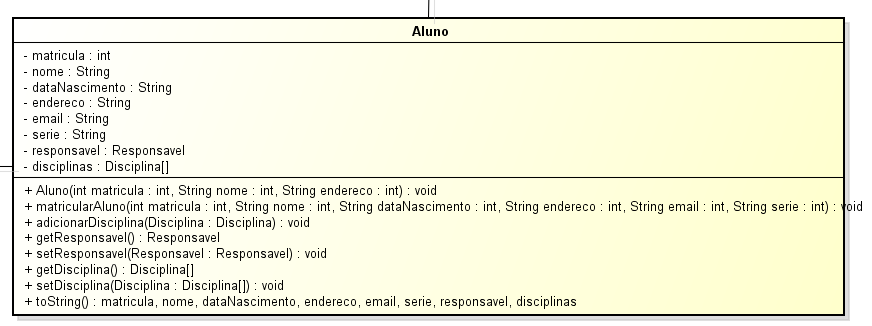
# Modelagem

Basicamente, na modelagem, utilizamos a UML (Unified Modeling Language) como base, na versão 2.0. Além disso, fizemos uso da GitHub, disponível em https://github.com/, na versão mais atual 10.0.

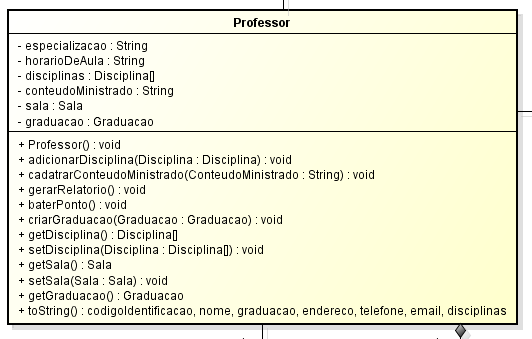
A Figura 01 apresenta o Diagrama de Classes obtido até o momento (*Seção 04*), o qual foi utilizado na implementação parcial de um Sistema Orientado a objeto sobre uma escola. Esse diagrama oferece uma visão estática (ainda parcial) das classes: aluno, professor, graduação, funcionário, direção, secretaria, responsável, sala, disciplina.



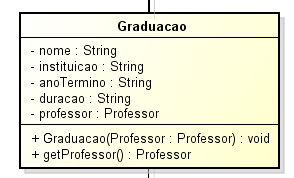
*Figura 01: Modelagem - Diagrama de Classes - Visão Estática de todas as classes e relacionamentos*



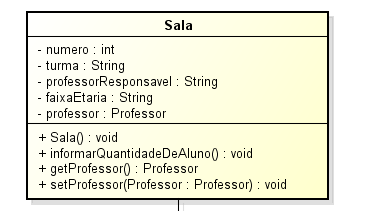
*Figura 02: Modelagem - Diagrama de Classes – Classe Aluno*



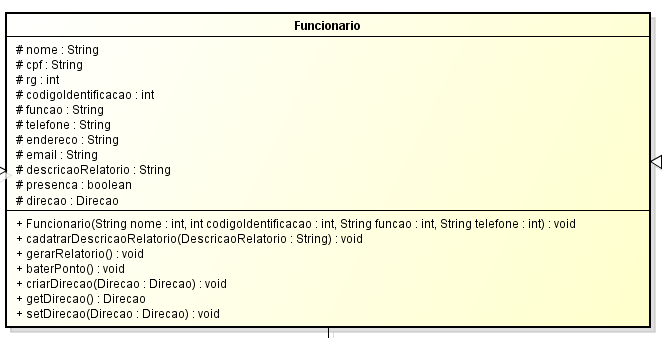
*Figura 03: Modelagem - Diagrama de Classes – Classe Professor*



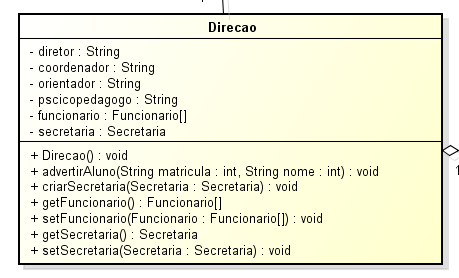
*Figura 04: Modelagem - Diagrama de Classes – Classe Graduação*



*Figura 05: Modelagem - Diagrama de Classes – Classe Sala*



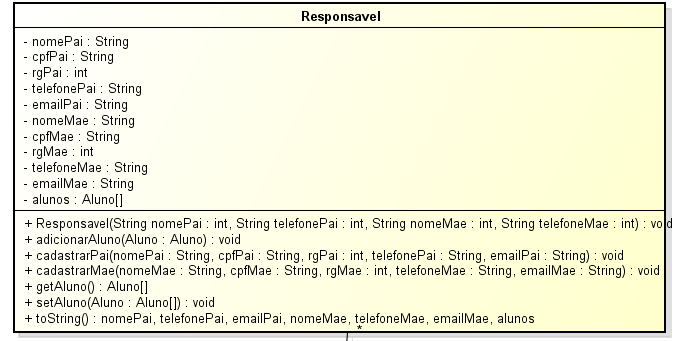
*Figura 06: Modelagem - Diagrama de Classes – Classe Funcionário*



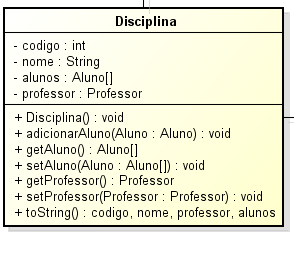
*Figura 07: Modelagem - Diagrama de Classes – Classe Direção*



*Figura 08: Modelagem - Diagrama de Classes – Classe Secretaria*



*Figura 09: Modelagem - Diagrama de Classes – Classe Responsável*



*Figura 10: Modelagem - Diagrama de Classes – Classe Disciplina*

A seguir apresentamos detalhes da implementação referente às etapas do Desenvolvimento do Sistema Orientado a Objeto da escola.

# Implementação

**Etapa 01**

Foram implementadas as classes: aluno, funcionário, professor, responsável, disciplina, direção e sala.

Todas as classes possuem construtores, no entanto as classes Aluno, Funcionario e Responsavel possuem seus construtores sobrecarregados para que não seja possível a instância de objetos em que seriam necessários alguns dados importantes como nome e CPF por exemplo.

Na classe Aluno fora implementado um método chamado matricularAluno no qual será possível fazer a matricular do aluno por meio da chamada deste método.

Na classe Professor fora implementado um método chamado atualizarGraduacao para que o professor, ao receber um novo título, possa alterar os dados de sua graduação.

Na classe Funcionario foi implementado um método chamado verificarFuncionario no qual, por meio da chamada do método, o funcionário poderá ver qual o valor de seu salário.

Na classe Sala foi implementado um método chamado informarQuantidadeDeAlunos para que seja possível, por meio da chamada do método, o objeto verificar a quantidade de alunos presentes numa determinada sala de aula. No entanto esse método só possui assinatura por enquanto, pois faremos uma implementação mas robusta quanto forem abordados em sala de aulas todos os tipos de relacionamento entre classes.

Na classe direção foi implementado um método chamado advertirAluno para que seja possível um diretor advertir um aluno.

**Etapa 02**

Foram feitas modificações em quase todo o projetos por conta dos relacionamentos. Esse foram: Dependência e Associação.

Na classe Aluno foi relacionada com as classes Responsavel e Disciplina. Portanto, foi necessário criar novos métodos para a interação das classes. Foram criados os métodos adicionarDisciplina para que seja possível adicionar disciplinas para os alunos matriculados, e o método toString para exibição de dados via console. A classe também recebeu os novos atributos que são responsável e disciplina.

Na classe Responsavel também foi estabelecido uma associação com a classe aluno. Esta recebeu os métodos adicionarAluno no qual é possível cadastrar um aluno para um responsável, cadastrarPai e cadastrarMae em que é possível cadastrar os dados dos responsáveis e toString para exibição de dados via console. A classe também recebeu um novo atributo que foi alunos.

Na classe Sala foi estabelecido um relacionamento com a classe Professor. Além disso a classe Sala recebeu um novo atributo chamado professor do tipo Professor.

Na classe Professor foi estabelecido um relacionamento de associação entre Sala e também Disciplina. Recebeu também o métodos cadastrarDisciplina no qual é possível cadastrar a disciplina ministrar pela professor, e toString para exibição de dados via console. Além de ter sido adicionado dois atributos sala e disciplinas.

Na classe Disciplina também foi estabelecido uma relação de associação com Professor e Aluno. A classe Disciplina recebeu os novos métodos adicionarAluno para que seja possível adicionar um aluno a disciplina e o toString para exibição de dados via console. Foram criados os novos atributos professor e alunos.

Além de tudo, foram adicionados ainda a classe Escola os testes de relacionamento, neste caso, as associações. Também foram feitos novos teste de unidade com a ferramenta JUnit.

Nas demais classes não houvera alterações relevantes. Mas posteriormente sofrerão em suas estruturas. Vale resaltar que foram criados get’s e set’s para cada novo atributo gerado.

**Etapa 03**

Nessa etapa não foram grandes as modificações como na etapa anterior. No entanto, as modificações implementadas foram fundamentais a continuidade do projeto. Houveram três implementações, duas agregações e uma associação, sendo as seguintes:

No primeiro relacionamento que constitui uma agregação, fazem parte as classes Direcao e Funcionario. Após criar o relacionamento, foi feito também o espelhamento entre as classes, uma vez que sua multiplicidade é de um para muito, como pode ser visto na diagramação. Além disso, foi criado um método chamado criarDirecao para que seja possível adicionar funcionários ao vetor Direcao.

No segundo relacionamento que constitui também uma agregação, foi necessário reajustar o atributo secretaria da classe Direcao para se tornar uma nova classe com novos atributos e métodos, sendo feitas as devidas alterações. Logo, fazem parte desse relacionamento as classes Direcao e Secretaria. Após criar o relacionamento, foi feito também o espelhamento entre as classes, uma vez que sua multiplicidade é de um para um, como pode ser visto na diagramação. Além disso, foi criado um método chamado criarSecretaria para que seja possível criar uma secretaria a partir de Direcao.

E por ultimo, o relacionamento de associação, no entanto, para criar o relacionamento, foi necessário reajustar o atributo graduação da classe Professor para se tornar uma nova classe com novos atributos e métodos, sendo feitas as devidas alterações. Logo, fazem parte desse relacionamento as classes Professor e Graduação. Após criar o relacionamento, foi feito também o espelhamento entre as classes, uma vez que sua multiplicidade é de um para um, como pode ser visto na diagramação. Como se trará de um relacionamento com características muito fortes, foi necessário criar amarrações para que não seja possível criar ou alterar a classes graduação facilmente. Logo, só é possível criar um objeto da classe Graduacao, se for criar juntamente um objeto da classe Professor.

**Etapa 04**

Nesta etapa tiveram mudanças significantes nas classes em que envolviam os relacionamentos. Com a utilização da ideia de reciclagem de códigos, por meio de herança, trouxe um grande poder a uma classe em especial.

A classe funcionário tornou-se a classe “pai” de duas novas classes: secretaria e professor. A classe pai pode “fatorar” todos os atributos em comum com as classes “filhas” (secretaria e professor). Isso fez com que o sistema não ficasse com redundâncias de linhas, nem de atributos.

Apesar disso, as classes filhas ganharam especificações. A classe secretaria ganhou dois atributos: folga como atributo simples e agendas como um vetor. Além disso, foi criado um novo método para que a classe pudesse exibir os valores que se encontravam dentro do vetor. A esse novo método, deu-se o nome de listarAgendas.

A classe professor também ganhou dois novos atributos simples: especialização e horarioDeAula. Esses dois atributos ajudam a especificar a classe professor, diferenciado-a de secretaria.

Além de tudo foram feitos novos testes de unidades nas classes filhas e seus principais atributos. Isto é, tanto secretaria quanto professor ganha uma classe no pacote teste chamado secretariaTest e professorTest respectivamente, em que são testados os métodos gets das classes e de alguns atributos em comum, no qual estão na classe pai também.

A diagramação foi atualizado e todos as novas atulizações commitadas no controle de versão.

Nessa etapa não foram grandes as modificações como na etapa anterior. No entanto, as modificações implementadas foram fundamentais a continuidade do projeto. Houveram três implementações, duas agregações e uma associação, sendo as seguintes:

No primeiro relacionamento que constitui uma agregação, fazem parte as classes Direcao e Funcionario. Após criar o relacionamento, foi feito também o espelhamento entre as classes, uma vez que sua multiplicidade é de um para muito, como pode ser visto na diagramação. Além disso, foi criado um método chamado criarDirecao para que seja possível adicionar funcionários ao vetor Direcao.

**Etapa 05**

Nesta etapa, as mudanças no sistema não foram muito grandes. Foram adicionados novos métodos e atributos as classes que estão envolvidas em relacionamento de herança, para que fosse possível gerar um polimorfismo de forma eficiente.

No código gerado essa semana, poderá ser visto um número maior de linhas no qual lembram a programação estrutura. Pois, foi necessário fazer algumas condições para que o código não ficasse promiscuo ou aberto demais a ponto de comprometer sua segurança.

Primeiramente foram adicionados os novos métodos na diagramação. A classe funcionário recebeu o atributo descricaoRelatorio no qual receberá informações acerca dos relatórios gerados diariamente pelos funcionários da escola e o atributo presenca , no qual é possível um funcionário registrar seu presença através de métodos. É importante dizer que o método isPresenca (que funciona como um get, mas para os tipos boolean) tem algumas condições de exibição. Além disso, recebeu também o método cadastrarDescricaoRelatorio em que é possível cadastrar esse novo atributo criado. Contudo, essas classe recebeu os métodos gerarRelatorio e baterPonto totalmente vazios em ser corpo. Estes serão usados posteriormente para polimorfismo.

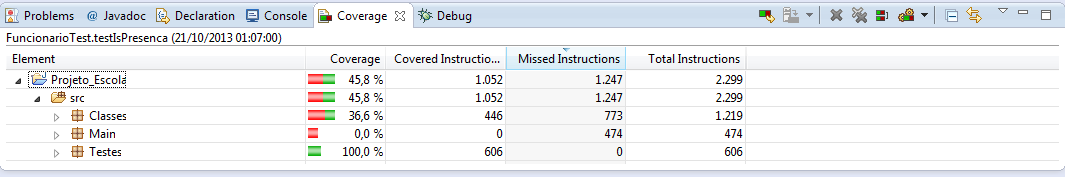
Nas classes filhas, temos em professor, um novo atributos chamado conteúdoMinistrado no servirá para que o professor registro o que foi ministrado naquele dia de aula. Além disso, foi criado o método cadastrarConteudoMinistrado para que fosse possível adicionar um valor ao atributo conteudoMinistrado. Os métodos gerarRelatorio e baterPonto foram sobrescritos passando a ter uma estrutura mais robusta e eficaz.

Já a classe secretaria recebeu o atributo folga, no qual é possível registrar a folga da secretaria por meio de métodos. Recebeu também os métodos cadastrarFolga no qual é possível cadastrar um dia de folga para a secretaria, e o método exibirFolga que possui condições de exibição no console. Além de tudo isso, também recebeu os métodos gerarRelatorio e baterPonto sobrescrevendo-os de forma diferente da classe professor, obviamente.

Por fim, foram concluídos todos os teste de unidade possíveis para as classe criadas. Essa semana era previsto a instalação do plugin EMMA para que fosse possível fazer teste de cobertura. No entanto, não conseguimos instalar adequadamente o plugin ficando sem os resultados desejados. Mas, a próxima semana, possivelmente será instalado o programa para que seja possível tal feito.

**Etapa 06**

Nesta etapa, tivemos mudanças significativas no projeto. Primeiro lugar esta o adicionamento dos testes de cobertura no qual e possível verificar a cobertura dos testes que está sendo atribuída ao projeto. Utilizando o framework EclEmma® versão 2.2.1, download disponível na pagina: < <http://www.eclemma.org/>>, é possível fazer todos os testes de cobertura obtendo a porcentagem de cobertura do projeto. Em nosso projeto ainda está com uma porcentagem relativamente baixa de 45,8% como pode ser visto na imagem abaixo. No entanto, o objetivo é aumentar para até 90% a cobertura de teste até o final do projeto.



*Figura 11: Relatório do Teste de Cobertura – EclEmma.*

Alem disso, foi criado uma nova classe abstrata chamada Pessoa, por sua vez, carregada de atributos no qual são utilizados por sua classe filha, funcionário. Para este caso, a classe funcionário deixou de ter alguns atributos para ter atributos mais específicos de sua classe.

Também fora adicionados um atributo *static* na classe Sala, chamado quantidadeAluno, este atributo trata-se de uma constante o qual servirá para determinar a quantidade máxima de alunos dentro de uma sala, isto é, de uma turma. Além disso, fora criado novos métodos para a classe Sala. São eles: listarAluno no qual é possível verificar os nomes dos alunos que estão em uma determinada sala/turma, informarQuantidadeDeAlunos esse mostra a quantidade de alunos dentro de uma sala/turma e criarSala que trata de um método no qual é possível adicionar novos alunos em uma turma, porem não é possível adicionar mais do que 20 anos em uma turma, pois o método possui um restrição para isso.

E por sim fim, foi criado um novo relacionamento entre Aluno e Sala, isto é, uma associação simples de um para muitos. Foram adicionado também alguns teste de unidade, para aumentar a validação dos métodos.

# Controle de Versão

O controle de versão é uma importante ferramenta no desenvolvimento de um software. A eficácia do controle de versão de software é comprovada por fazer parte das exigências para melhorias do processo de desenvolvimento de certificações tais como [CMMI](http://pt.wikipedia.org/wiki/CMMI) e [SPICE](http://pt.wikipedia.org/wiki/SPICE).

O repositório com o histórico de modificações no sistema implementado está disponível no endereço contendo todas as informações necessárias: <https://github.com/tatielen/projetoOO>.

Na semana do dia 26 de setembro a 07 de outubro (etapa 4), uma das componentes do grupo trancou a disciplina. Então, o endereço do repositório na internet foi mudado, assumindo um novo “usuário máster”. No entanto, as informações anteriores (commits) não foram perdidas, apenas refatoradas. Portanto, o novo endereço do repositório è: <https://github.com/modestoo/projetoOO>.