Universidade Federal de Viçosa - *Campus Florestal*

Instituto de Ciências Exatas e Tecnológicas

Disciplina de Programação Orientada a Objetos

Professor Doutor Fabrício Aguiar Silva

TRABALHO PRÁTICO:
PROJETO DE APLICAÇÃO DE ACOMPANHAMENTO ACADÊMICO

Erian Alírio De Oliveira Alves - 3862

Guilherme Sérgio De Oliveira - 3854

Maria Theresa Arruda e Henriques - 3486

Florestal

2021

Lista de Tabelas

1. Introdução

O presente trabalho busca ser uma ferramenta de planejamento e acompanhamento da vida acadêmica do estudante de Ciência da Computação da Universidade Federal de Viçosa - *Campus Florestal*. Para isso, o programa apresenta as seguintes funcionalidades: acesso a informações sobre quais disciplinas são ofertadas por quais departamentos e quais professores; histórico de disciplinas já cursadas; disciplinas disponibilizadas por semestre, e, por fim, grade curricular do curso.

2. Desenvolvimento

Com o intuito de promover uma melhor organização do projeto e do código desenvolvido, fez-se uso de alguns aspectos do padrão de projeto Model-View-Controller (MVC).

Nas subseções seguintes serão explicadas as classes desenvolvidas para a representação de entidades do projeto e os elementos que realizam o controle entre as classes.

2.1 Classes modelo

2.1.1 Pessoa

A criação da classe abstrata [1] *Pessoa* tem por objetivo representar uma entidade de mesmo nome e assim agrupar alguns atributos comuns às classes *Aluno* e *Professor*. Assim, a superclasse *Pessoa* é composta por cinco atributos de tipo *String,* sendo eles: (1) *cpf*; (2) *nome*; (3) *email*; (4) *celular;* (5) *dataNascimento*. Todos os atributos mencionados são de caráter descritivo para com o objeto e são passados como herança as suas classes filhas, *Aluno* e *Professor*.

Em relação aos métodos, foram implementados apenas aqueles com funcionalidade assessora. Dentre os métodos presentes na classe, temos o construtor da classe, responsável por determinar quais ações devem ser executadas no momento de criação do objeto [2], e *getters* e *setters* para cada atributo, possibilitando assim o acesso e gerenciamento desses atributos [3].

A tabela 1 apresenta a lista de atributos e métodos presentes na classe. Já na tabela 2 é possível encontrar uma descrição mais detalhada a respeito do construtor. Por fim, a tabela 3

é composta por uma breve descrição dos métodos da classe, apresentando os tipos de retorno de cada método e os parâmetros que utilizam, quando os utilizam.

Tabela 1 - Classe modelo Pessoa

Pessoa
- cpf: String; - nome: String; - email: String; - celular: String; - dataNascimento: String;
+ Pessoa(): void; + getCpf(): String; + setCpf(): void; + getNome(): String; + setNome(): void; + getEmail(): String; + setEmail(): void; + getCelular(): String; + setCelular(): void; + getDataNascimento(): String; + setDataNascimento(): void;

Tabela 2 - Construtor classe Pessoa

Construtor e Descrição
Pessoa(String nome, String email**);** Pessoa recebe dois atributos por parâmetro, nome e email, que ajusta os valores para as variáveis nome e email,

respectivamente.

Tabela 3 - Métodos da classe Pessoa

Métodos	
Tipo	**Método e Descrição**
String	**getCpf();** Retorna o valor da variável de tipo *String cpf*.
void	**setCpf();** Atribui o valor passado por parâmetro a variável de tipo *String cpf*.
String	**getNome();** Retorna o valor da variável de tipo *String nome*.
void	**setNome();** Atribui o valor passado por parâmetro a variável de tipo *String nome*.
String	**getEmail();** Retorna o valor da variável de tipo *String email*.
void	**setEmail();** Atribui o valor passado por parâmetro a variável de tipo *String email*.
String	**getCelular();** Retorna o valor da variável de tipo *String celular*.
void	**setCelular();** Atribui o valor passado por parâmetro a variável de tipo *String celular.*
String	**getDataNascimento();**

	Retorna o valor da variável de tipo *String dataNascimento*.
void	**setDataNascimento();** Atribui o valor passado por parâmetro a variável de tipo *String dataNascimento*.

2.1.2 Aluno

Tendo a classe *Aluno* a classe acima como classe mãe, esta terá os atributos herdados mais os seguintes atributos: (1) *matricula*; (2) *historico*; (3) *semestreAtual.* O primeiro atributo, de tipo *String*, tem caráter descritivo e identificatório, dado que será uma chave única para cada instância desta classe adicionada ao programa. O segundo atributo, criado para servir como uma lista de registro das disciplinas cursadas pelo estudante, é de tipo *Historico*, classe esta que será detalhada mais à frente. E por fim, o atributo *semestreAtual*, que tem a finalidade de armazenar o valor de tipo Semestre, classe que também será detalhada posteriormente, que o estudante está no momento que faz uso do programa. Importante ressaltar que todos os atributos da classe *Aluno* possuem modificador de acesso de tipo *private*, ou seja, protegem os atributos de acesso direto por parte do usuário [4].

Tabela 4 - Classe modelo Aluno

Aluno
- matricula: String; - historico: Historico; - semestreAtual: Semestre;
+ Aluno(): void; + ExibirDadosAluno(): void; + getMatricula(): String; + setMatricula(): void; + getHistorico(): Historico; + setHistorico(): void;

+ getSemestreAtual(): Semestre; + setSemestreAtual(): void;

Tabela 5 - Construtor classe Aluno

Construtor e Descrição
Aluno(String nome, String email, String matricula**);** Instância um objeto Aluno com os atributos nome, email e matrícula passados por parâmetro no momento da chamada. O método **super()** instanciará os atributos herdados, nome e email. O valor passado de matrícula será armazenado na variável de mesmo nome. E, por fim, a variável *historico* fará chamada do construtor da classe *Historico*.

Tabela 6 - Métodos da classe Aluno

Métodos	
Tipo	**Método e Descrição**
void	**ExibirDadosAluno();** Exibe os valores do objeto *Aluno*, que são: nome, email, total de créditos cursados e semestre atual.
String	**getMatricula();** Retorna o valor da variável de tipo *String matricula*.
void	**setMatricula();** Atribui o valor passado por parâmetro a variável de tipo *String matricula*.

Historico	**getHistorico();** Retorna o valor da variável de tipo *Historico historico*.
void	**setHistorico();** Atribui o valor passado por parâmetro a variável de tipo *Historico* a variável *historico*.
Semestre	**getSemestreAtual();** Retorna o valor da variável de tipo *Semestre* armazenada na variável *semestreAtual*.
void	**setSemestreAtual();** Atribui o valor passado por parâmetro a variável de tipo *Semestre* a variável s*emestreAtual*.

2.1.3 Professor

A classe *Professor* é outra classe que herda todos os atributos e métodos da classe mãe *Pessoa*. Para além dos atributos herdados, a classe *Professor* tem o atributo *disciplinasLecionadas* de tipo *HashMap*. Sendo a estrutura HashMap associada a valores em pares, ou seja, cada elemento de sua lista possui uma chave e um valor associado [5], o atributo *disciplinasLecionadas* terá uma chave de tipo *String* e um valor de tipo *Disciplina*.

Quanto aos métodos, foi implementado um construtor para a classe e *getters* e *setters* para atribuir e retornar os valores dos atributos presentes na classe. Além desses, foi adicionado o método *listarDisciplinas* para exibir as disciplinas ministradas por cada professor.

Tabela 7 - Classe modelo Professor

Professor
- disciplinasLecionadas: HashMap;

+ Professor(): void; + setDisciplinasLecionadas(): void; + getDisciplinasLecionadas(): HashMap; + listarDisciplinas(): void;

Tabela 8 - Construtor classe Professor

Construtor e Descrição
Professor(String nomeProf, String email**);** Instância um objeto *Professor* com os atributos *nomeProf*, email por parâmetro no momento da chamada. O método **super()** instanciará os atributos herdados, *nomeProf* e *email* nos atributos *nome* e *email* da classe *Pessoa*. A variável *disciplinasLecionadas* fará chamada do construtor da classe *HashMap*.

Tabela 9 - Métodos da classe Professor

Métodos	
Tipo	**Método e Descrição**
void	**ListarDisciplinas();** Exibe os valores do objeto *Professor* que são: nome do professor e as disciplinas por ele ministradas.
HashMap	**getDisciplinasLecionadas();** Retorna o valor da variável de tipo *HashMap disciplinasLecionadas* que é composta por uma chave de tipo *String* e um valor de tipo *Disciplina*.

void	**setDisciplinasLecionadas();** Atribui os valores passados por parâmetro, a chave de tipo *String* e o valor de tipo *Disciplina*, a variável de tipo *HashMap disciplinasLecionadas*.

2.1.4 Curso

A criação da classe *Curso* tem a finalidade de representar uma entidade de mesmo nome. Os atributos foram selecionados pensando que todo curso deve possuir um código de identificação que facilite sua manipulação no sistema, um nome que o identifique socialmente, uma grade de disciplinas e um núcleo de professores para ministrá-las. Assim, esta classe é composta por quatro atributos, sendo eles: (1) *codigo*; (2) *nome*; (3) *gradePadrao;* (4) *professor*.

Os métodos empregados na classe são os *getters* e *setters* dos atributos, e o construtor da classe. Abaixo seguem as tabelas com mais detalhes do que aqui foi exposto.

Tabela 10 - Classe modelo Curso

Curso
- codigo: String; - nome: String; - gradePadrao: GradeCurricular; - professores: HashMap;
+ Curso(): void; + setCodigo(): void; + getCodigo(): String; + setNome(): void; + getNome(): String; + setGradePadrao(): void;

+ getGradePadrao(): GradeCurricular; + setProfessores(): void; + getProfessores(): HashMap;

Tabela 11 - Construtor classe Curso

Construtor e Descrição
Curso(String id, String nome**);** Instância um objeto *Curso* com os atributos *id* e *nome* por parâmetro. O *id* confere valor ao código do curso e *nome* atribui valor a variável *nome*.

Tabela 12 - Métodos da classe Curso

Métodos	
Tipo	**Método e Descrição**
void	**setCodigo();** Atribui o valor de tipo *String* passado por parâmetro a variável *codigo*.
String	**getCodigo();** Retorna o valor de tipo *String* armazenado na variável *codigo.*
void	**setNome();** Atribui o valor de tipo *String* passado por parâmetro a variável *nome*.
String	**getNome();** Retorna o valor de tipo *String* armazenado na variável

	nome.
void	**setGradePadrao();** Atribui o valor de tipo *GradeCurricular* a variável *gradePadrao*.
GradeCurricular	**getGradePadrao();** Retorna o valor de tipo *GradeCurricular* armazenado na variável *gradePadrao*.
void	**setProfessores();** Atribui os valores passados por parâmetro, a chave de tipo *String* e o valor de tipo *Professor*, a variável de tipo *HashMap professores*.
HashMap	**getGradePadrao();** Retorna o valor da variável de tipo *HashMap professores* que é composta por uma chave de tipo *String* e um valor de tipo *Professor*.

2.1.5 Departamento

A criação da classe *Departamento* deveu-se à necessidade de representar uma entidade de mesmo nome. Os atributos foram selecionados sabendo-se que todo departamento deve possuir um código de identificação no sistema, um nome reconhecido socialmente, um email e um telefone para contato, um professor responsável por geri-lo, e deve ter participação em um ou mais cursos da instituição de ensino. Portanto, esta classe é composta por seis atributos, sendo eles: (1) *codigo*; (2) *nome*; (3) *email;* (4) *telefone; (5) cursos; (6) diretor*.

Dentre os métodos implementados tem-se um construtor para a classe, *getters* e *setters* para os atributos, e um método para exibir os detalhes presentes em cada departamento registrado.

Tabela 13 - Classe modelo Departamento

Departamento
- codigo: String; - nome: String; - email: String; - telefone: String; - cursos: HashMap; - diretor: Professor;
+ Departamento(): void; + ExibirDepartamento(): void; + setCodigo(): void; + getCodigo(): String; + setNome(): void; + getNome(): String; + setEmail(): void; + getEmail(): String; + setTelefone(): void; + getTelefone(): String; + setCursos(): void; + getCursos(): HashMap; + setDiretor(): void; + getDiretor(): Professor;

Tabela 14 - Construtor classe Departamento

Construtor e Descrição
Departamento(String nome, String codigo, String email, String telefone**);** Instância um objeto *Departamento* com os atributos *nome*, *codigo*, *email* e *telefone* por parâmetro. Os valores passados por parâmetro

conferem são instanciados nas variáveis de mesmo nome na classe.

Tabela 15 - Métodos da classe Departamento

Métodos	
Tipo	**Método e Descrição**
void	**setCodigo();** Atribui o valor de tipo *String* passado por parâmetro a variável *codigo*.
String	**getCodigo();** Retorna o valor de tipo *String* armazenado na variável *codigo*.
void	**setNome();** Atribui o valor de tipo *String* passado por parâmetro a variável *nome*.
String	**getNome();** Retorna o valor de tipo *String* armazenado na variável *nome*.
void	**setEmail();** Atribui o valor de tipo *String* passado por parâmetro a variável *email*.
String	**getEmail();** Retorna o valor de tipo *String* armazenado na variável *email*.
void	**setTelefone();** Atribui o valor de tipo *String* passado por parâmetro a variável *telefone*.

String	**getTelefone();** Retorna o valor de tipo *String* armazenado na variável *telefone*.
void	**setCursos();** Atribui os valores passados por parâmetro, a chave de tipo *String* e o valor de tipo *Curso*, a variável de tipo *HashMap cursos*.
HashMap	**getCursos();** Retorna o valor da variável de tipo *HashMap* cursos que é composta por uma chave de tipo *String* e um valor de tipo *Curso*.
Professor	**getDiretor();** Retorna o valor de tipo *Professor* armazenado na variável *diretor.*
void	**setDiretor();** Atribui o valor de tipo *Professor* passado por parâmetro a variável *diretor*.

2.1.6 Disciplina

A classe *Disciplina* foi criada para representar uma entidade de mesmo nome. Assim, os atributos foram criados de acordo com as necessidades dessa representação, como do fato de que toda disciplina deve ter um nome, um professor para ministrá-la, a quantidade de créditos e carga horária que possui, e o código que a representa no sistema. Portanto, esta classe é composta por cinco atributos, sendo estes: (1) *codigo*; (2) *nome*; (3) *professor;* (4) *qntCreditos; (5) cargaHoraria.*

Sobre os métodos implementados, tem-se o construtor da classe, *getters* e *setters* de cada atributo, e um método que mostra os valores armazenados em cada instanciação de disciplina.

Tabela 16 - Classe modelo Disciplina

Disciplina
- codigo: String; - nome: String; - professor: String; - qntCreditos: int; - cargaHoraria: int;
+ Disciplina(): void; + ExibirDisciplina(): void; + getCodigo(): String; + setNome(): void; + getNome(): String; + setProfessor(): void; + getProfessor(): String; + setQntCreditos(): void; + getQntCreditos(): int; + getCargaHoraria(): int;

Tabela 17 - Construtor classe Disciplina

Construtor e Descrição
Disciplina(String nome, int qntCreditos, String codigo**);** Instância um objeto *Disciplina* com os atributos *nome*, *codigo*, *qntCreditos* passados por parâmetro. Os valores recebidos por parâmetro são instanciados nas variáveis de mesmo nome na classe. O valor instanciado no atributo *cargaHoraria* é dado pela multiplicação do valor recebido no parâmetro *qntCreditos* por 15.

Tabela 18 - Métodos da classe Disciplina

Métodos	
Tipo	**Método e Descrição**
String	**getCodigo();** Retorna o valor de tipo *String* armazenado na variável *codigo*.
void	**setNome();** Atribui o valor de tipo *String* passado por parâmetro a variável *nome*.
String	**getNome();** Retorna o valor de tipo *String* armazenado na variável *nome*.
void	**setProfessor();** Atribui o valor de tipo *String* passado por parâmetro a variável *professor*.
String	**getProfessor();** Retorna o valor de tipo *String* armazenado na variável *professor*.
void	**setQntCreditos();** Atribui o valor de tipo *int* passado por parâmetro a variável *qntCreditos*.
int	**getQntCreditos();** Retorna o valor de tipo *int* armazenado na variável *qntCreditos*.
int	**getCargaHoraria();** Retorna o valor de tipo *int* armazenado na variável *cargaHoraria*.

2.1.7 Grade Curricular

A classe *GradeCurricular* foi criada para representar a entidade de mesmo nome. A grade curricular é um componente importante na descrição de um curso e dado imprescindível ao se registrar um aluno na instituição. Assim, para representar uma grade curricular deve-se lembrar que um curso pode ter mais de uma grade curricular registrada ao longo de sua vigência na instituição, entretanto sendo registrada no máximo uma por ano. Com isso dito, os atributos criados para essa classe foram: (1) ano; (2) grade.

Quanto aos métodos, construtor da classe, *ExibirGradeCurricular*, *AdicionarSemestreGrade*, *RetornarSemestre*, *ObterTotalCreditosGrade*, *setAno, getAno, getSemestre*. Mais detalhes sobre os métodos podem ser encontrados nas tabelas abaixo.

Tabela 19 - Classe modelo GradeCurricular

GradeCurricular
- ano: int; - grade: ArrayList;
+ GradeCurricular(): void; + ExibirGradeCurricular(): void; + AdicionarSemestreGrade(): void; + RetornarSemestre(): Semestre; + ObterTotalCreditosGrade(): int; + setAno(): void; + getAno(): int; + getSemestres(): ArrayList;

Tabela 20 - Construtor classe GradeCurricular

Construtor e Descrição
GradeCurricular(int ano**);**

Instância um objeto *GradeCurricular* com o atributo ano passado por parâmetro. No construtor também é instanciado no atributo valor uma *ArrayList* do tipo *Semestre.*

Tabela 21 - Métodos da classe GradeCurricular

Métodos	
Tipo	**Método e Descrição**
void	**ExibirGradeCurricular();** Exibe os valores do objeto *GradeCurricular* que é a grade de disciplinas de todos os semestres presentes.
void	**AdicionarSemestreGrade();** Atribui o valor de tipo *Semestre* passado por parâmetro a variável de mesmo nome.
Semestre	**RetornarSemestre();** Retorna o valor resultado da busca pelo semestre quando compatível com o valor armazenado em *grade.* Caso não haja semestre com aquele valor o método retorna *null*.
int	**ObterTotalCreditosGrade();** O método percorre todos os semestres até o valor do semestre declarado pelo aluno e retorna a soma de todos créditos cursados até esse valor.
void	**setAno();** Atribui o valor de tipo *int* passado por parâmetro a variável ano ao atributo de mesmo nome.
int	**getAno();** Retorna o valor armazenado no atributo *ano*.

ArrayList	**getSemestre();** Retorna o valor armazenado na variável *grade*.

2.1.8 Historico

A classe *Historico* foi criada pensando na entidade histórico acadêmico de um estudante, onde a mesma deve conter uma lista de valores das disciplinas já cursadas pelo aluno durante sua vida universitária. Com isso dito, o único atributo da classe é do tipo *ArrayList* de *Semestre* denominado *historico*.

Os métodos empregados foram: construtor da classe; *ExibirHistorico*, *setHistorico, getHistorico, obterTotalCreditosGrade*. Os detalhes sobre os métodos encontram-se nas tabelas abaixo.

Tabela 22 - Classe modelo Historico

Historico
- historico: ArrayList;
+ Historico(): void; + ExibirHistorico(): void; + setHistorico(): void; + getHistorico(): ArrayList; + ObterTotalCreditosGrade(): int;

Tabela 23 - Construtor classe Historico

Construtor e Descrição
Historico(); Instância um objeto *Historico*. Internamente ao construtor é

realizada a instanciação do atributo *historico* de tipo *ArrayList Semestre.*

Tabela 24 - Métodos da classe Historico

Métodos	
Tipo	**Método e Descrição**
void	**ExibirHistorico();** Exibe os valores do objeto *Historico* que é o histórico de disciplinas de todos os semestres cursados pelo estudante.
int	**ObterTotalCreditosGrade();** O método percorre todos os semestres até o valor do semestre declarado pelo estudante e retorna a soma de todos créditos cursados até esse valor.
void	**setHistorico();** Adiciona o valor de tipo *Semestre,* passado por parâmetro como *semestreCursado,* ao *ArrayList historico*.
ArrayList	**getHistorico();** Retorna o valor armazenado no atributo *historico*.

2.1.9 Instituicao

Pensando que todo curso, disciplina, professor e departamento devam pertencer e estar ligados por uma instituição de ensino foi necessário criar uma classe que representasse essa entidade, classe essa que foi criada com o nome *Instituicao*. Da mesma forma, os atributos criados para esta classe também seguiram o padrão do que foi mencionado acima: um nome e lista de tipo *HashMap* de disciplinas, professores, cursos, departamentos, todos os atributos receberam os nomes equivalentes.

Dentre os métodos implementados tem-se um construtor para a classe, *getters* e *setters* para os atributos, e um método para exibir as disciplinas presentes nas instituições cadastradas.

Tabela 25 - Classe modelo Instituicao

Instituicao
- nome: String; - disciplinas: HashMap; - professores: HashMap; - cursos: HashMap; - departamentos: HashMap;
+ Instituicao(): void; + ExibirDisciplinasInstituicao(): void; + setNome(): void; + getNome(): String; + setProfessores(): void; + getProfessores(): HashMap; + setDisciplinas(): void; + getDisciplinas(): HashMap; + setCursos(): void; + getCursos(): HashMap; + setDepartamentos(): void; + getDepartamentos(): HashMap;

Tabela 26 - Construtor classe Instituicao

Construtor e Descrição
Instituicao(String nome**);** Instância um objeto *Intituicao* com o atributo *nome* passado por

parâmetro. Internamente ao construtor também será instanciado os atributos *disciplina, professores, cursos, departamentos*, todos do tipo *HashMap,* todos com chave do tipo *String* e os valores serão de suas classes com nome equivalente.

Tabela 27 - Métodos da classe Instituicao

Métodos	
Tipo	**Método e Descrição**
void	**ExibirDisciplinaInstituicao();** Exibe os valores do atributo *disciplinas.*
void	**setNome();** Atribui o valor de tipo *String* passado por parâmetro a variável *nome*.
String	**getNome();** Retorna o valor de tipo *String* armazenado na variável *nome.*
void	**setProfessores();** Atribui o valor de tipo *String* passado por parâmetro de nomenclatura *nomeProfessor* ao tipo *HashMap* de *professores.*
HasMap	**getProfessores();** Retorna o valor da variável de tipo *HashMap professores* que é composta por uma chave de tipo *String* e um valor de tipo *Professor*.
void	**setDisciplinas();** Atribui o valor de tipo *String* passado por parâmetro de nomenclatura *nomeDisciplina* ao tipo *HashMap* de

	disciplinas.
HasMap	**getDisciplinas();** Retorna o valor da variável de tipo *HashMap* disciplinas que é composta por uma chave de tipo *String* e um valor de tipo *Disciplina*.
void	**setCursos();** Atribui o valor de tipo *String* passado por parâmetro de nomenclatura *codigoCurso* ao tipo *HashMap* de *cursos.*
HasMap	**getCursos();** Retorna o valor da variável de tipo *HashMap* professores que é composta por uma chave de tipo *String* e um valor de tipo *Curso*.
void	**setDepartamentos();** Atribui o valor de tipo *String* passado por parâmetro de nomenclatura *codigo* ao tipo *HashMap* de *departamentos.*
HasMap	**getDepartamentos();** Retorna o valor da variável de tipo *HashMap* professores que é composta por uma chave de tipo *String* e um valor de tipo *Departamento*.

2.1.10 Semestre

Com o intuito de representar um semestre de algum curso, grade ou no histórico, uma classe para o semestre foi criada. Essa classe contém as disciplinas de um dado semestre, mapeadas de acordo com seu código. Além disso, a classe ainda conta com uma particularidade. Como foi necessário realizar ordenações de semestre em implementações posteriores, um comparador foi inserido. Para isso, a classe implementa a interface *Comparable,* que conta com um método específico para realizar a ordenação.

Tabela 28 - Classe modelo Semestre

Semestre
- semestre: HashMap; - numeroSemestre: int;
+ Semestre(): void; + ExibirSemestre(): void; + ObterCreditosSemestre(): int; + getnumeroSemestre(): int; + AdicionarDisciplinas(): void; + compareTo(): int;

Tabela 29 - Construtor classe Semestre

Construtor e Descrição
Semestre(int numeroSemestre**);** Instância um objeto *Semestre* com o valor passado por parâmetro que é atribuído a variável de mesmo nome. Internamente ao construtor é realizada a instanciação do atributo *semestre* de tipo *HashMap.*

Tabela 30 - Métodos da classe Semestre

Métodos	
Tipo	**Método e Descrição**
void	**ExibirSemestre();** Exibe as disciplinas do semestre com o código das mesmas e quantidade total de créditos dessas disciplinas.

int	**ObterCreditosSemestre();** O método percorre todas as disciplinas do semestre e retorna o valor da soma de todos créditos.
int	**getnumeroSemestre();** Retorna o valor armazenado no atributo *numeroSemestre*.
void	**AdicionarDisciplinas();** Adiciona disciplinas no semestre atual
int	**compareTo();** Método definido na interface *Comparable,* sendo seu método natural de comparação

2.2 Classes controle

2.2.1 Control_Geral

A classe *Control_Geral* foi criada com o intuito de conectar as referências dos objetos criados a partir dos dados inseridos. Como algumas classes modelos possuem atributos os quais são objetos de outras entidades do projeto, fez-se necessário esse controlador para garantir que uma determinada informação acessada por caminhos diferentes ainda fosse coerente. Essa classe controladora é composta pelos atributos *instituicao* e *aluno*, sendo os tipos *Instituicao* e *Aluno*, respectivamente.

Os métodos empregados na classe manipulam os dados cadastrados e também há o construtor da classe. Abaixo seguem as tabelas com mais detalhes do que aqui foi exposto.

Tabela 31 - Construtor classe Control_geral

Construtor e Descrição
Control_Geral(String nome**);** Instância um objeto *Control_Geral* com o atributo *nome* por parâmetro. A partir disso é instanciado *instituicao* com

o valor passado por parâmetro.

Tabela 32 - Métodos da classe Control_Geral

Métodos	
Tipo	**Método e Descrição**
void	**CadastrarAluno(**String nome, String email, String matricula**);** Instância o atributo *aluno* com os valores passados por parâmetro.
void	**RegistrarHistoricoAluno(**int numSemestreAtual**);** Adiciona todos os semestres anteriores a aquele o qual é referenciado pelo número passado por parâmetro no atributo *historico* da variável *aluno*.
void	**CadastrarSemestreAtual(**int numSemestreAtual**);** Recupera o semestre correspondente ao número passado por parâmetro e o ajusta como semestre atual do objeto *aluno*.
void	**ExibirDadosAlunoControle();** Mostra os dados do aluno cadastrado.
void	**ExibirHistoricoAlunoControle();** Mostra o histórico do aluno cadastrado.
Instituicao	**getInstituicao();** Retorna o valor de tipo *Instituicao* do atributo *instituicao*.
void	**AdicionarDepartamentoInstituicao(**String nome, String codigo, String email, String telefone**);** Cria um objeto *Departamento* com os dados passados por parâmetro e o adiciona ao atributo *instituicao*.

void	**AdicionarCursoInstituicao(**String id, String nome**);** Cria um objeto *Curso* com os dados passados por parâmetro e o adiciona ao atributo *instituicao*.
void	**AdicionarDisciplinaInstituicao(**String nome, int qntCreditos, String codigo**);** Cria um objeto *Disciplina* com os dados passados por parâmetro e o adiciona ao atributo *instituicao*.
void	**AdicionarProfessorInsituicao(**String nomeProf, String email**);** Cria um objeto *Professor* com os dados passados por parâmetro e o adiciona ao atributo *instituicao*.
void	**AdicionarGradeCurso(**String codigoCurso, int ano**);** Cria um objeto *Grade* com o valor inteiro passado por parâmetro e após pesquisar o curso correspondente ao valor de *codigoCurso,* adiciona a grade criada ao curso dentro do atributo *instituicao*.
void	**AdicionarSemetreGradeCurso(**String codigoCurso, int numeroSemestre**);** Cria um objeto *Semestre* com o valor inteiro passado por parâmetro e após pesquisar o curso correspondente ao valor de *codigoCurso,* adiciona a o semestre a grade do curso, o qual está dentro do atributo *instituicao*.
void	**AdicionarDisciplinaSemetreGradeCurso(**String codigoCurso, String codigoDiscp, int numSemetre**);** Pesquisa uma disciplina no atributo *instituicao* a partir da *String codigoDiscp* e cria um objeto com essa referência. Após isso, recupera o semestre, a partir do inteiro no parâmetro, da grade do curso com código referente àquele do parâmetro e adiciona a disciplina no semestre.

void	**VincularProfessorDisciplina(**String nomeProf, String codigoDiscp**);** Recupera um professor e uma disciplina cadastrados no atributo *instituicao* a partir dos valores passados por parâmetro. Com essas referências recuperadas, é realizada uma conexão entre essas duas entidades, na qual é ajustado que determinado professor ministra essa disciplina e que a disciplina é ministrada por esse professor.
void	**listarDisciplinasProfessores();** Exibe dados de todos os professores cadastrados na instituição. Dentre esses dados está presente as disciplinas lecionadas.
void	**exibirGradeCurricularCurso(**String codigoCurso**);** A partir do valor passado por parâmetro é pesquisado o curso correspondente na instituição e é exibida a sua grade curricular padrão.
void	**exibirSemestreGrade(**int numSemestre, String codigoCurso**);** Com base nos valores fornecidos é pesquisado um semestre da grade curricular de um curso e são mostradas suas informações.
int	**porcentagemCreditosCursados(**String codigoCurso**);** É exibido e retornado a porcentagem de créditos cursados pelo aluno com base no semestre atual que está sendo cursado.
void	**ExibirDepartamentoControle();** Mostra as informações de um departamento pré-cadastrado.

2.2.2 Sistema

A classe *Sistema* foi criada com o propósito de cadastrar dados nas estruturas desenvolvidas e assim possibilitar a visualização das capacidades da aplicação. A referida classe possui um atributo, *controladorGeral*, do tipo *Control_Geral*, e a partir dele alimenta o sistema da aplicação com informações. A explicação sobre o trabalho realizado pelos métodos e pelo construtor está presente nas tabelas a seguir.

Tabela 33 - Construtor classe Sistema

Construtor e Descrição
Sistema(String nomeInstituicao**);** Instância um objeto *Sistema* com o atributo *nomeInstituicao* por parâmetro. A partir disso é instanciado *controladorGeral* com o valor passado por parâmetro.

Tabela 34 - Métodos da classe Sistema

Métodos	
Tipo	**Método e Descrição**
void	**CadastrarAlunoSistema(**int numeroSemestre**);** Instância um objeto *aluno* com dados arbitrários no sistema, para fins de teste, marcando que ele está cursando o período do número passado por parâmetro.
Control_Geral	**getControladorGeral();** Retorna uma referência para o atributo *controladorGeral* da classe.
Instituicao	**getInstituicaoControlador();** Retorna uma referência para o atributo *instituicao* presente

	dentro da classe *Control_Geral*.
void	**CadastrarProfessores();** Instância objetos *professores* dentro da instituição a partir de dados já fornecidos.
void	**CadastrarDisciplinas();** Instância objetos *disciplinas* dentro da instituição a partir de dados já fornecidos.
void	**CadastrarCurso();** Instância um objeto *curso* dentro da instituição a partir de dados já fornecidos.
void	**CadastrarDepartamento();** Instância um objeto *departamento* dentro da instituição a partir de dados já fornecidos.
void	**CadastrarGrade();** Instância um objeto *grade* dentro da instituição a partir de dados já fornecidos.
void	**CadastrarSemestre();** Instância objetos *semestre* dentro da instituição a partir de dados já fornecidos.
void	**CadastrarDisciplinaSemestre();** Realiza a conexão entre as disciplinas cadastradas na instituição com os seus respectivos semestres padrão, de acordo com o curso.
void	**Conectar();** Realiza a conexão entre as disciplinas e os professores cadastrados na instituição.

2.3 Classes visão

2.2.1 Menu

Com o intuito de obter um retorno gráfico do sistema, uma classe menu foi criada. A classe tem como objetivo oferecer uma opção visual, pela linha de comando, para a utilização do software. Desta forma, a classe apresenta alguns atributos para facilitar a utilização das camadas modelo e controle, além de facilitarem também as escolhas das opções. Como não são atributos essenciais para definir um estado da classe, mas sim para facilitar a implementação, eles não serão citados.

Tabela 35 - Métodos da classe Menu

Métodos	
Tipo	**Método e Descrição**
void	**mostrarMenu();** Realiza toda a lógica da interface, selecionando opções e também recebendo opções do usuário.
void	**ExibirTelaInicio(**int numSemestreAtual**);** Exibe uma tela inicial. Criado para facilitar na criação do código.

2.2.1 OpcaoMenu

Para facilitar na seleção de opções do menu, uma classe do tipo *enum* foi criada. Dessa forma, foram criadas constantes para todas as opções possíveis do menu. Além disso, um método adicional foi inserido na classe, com o intuito de mostrar essas opções possíveis no Menu. Como a classe foi criada para auxiliar na utilização do menu, não será feita uma explicação mais detalhada.

3. Testes

A fim de realizar a execução do programa serão necessárias algumas configurações iniciais na máquina de testes. Devido ao uso da linguagem Java para a construção do trabalho, é necessário que o computador na qual ocorrerá a execução tenha instalado a versão 16 (ou superior) do *Java Development Kit* (*JDK*), a qual pode ser baixada no site da Oracle [6] com opções para diferentes sistemas operacionais.

Após a instalação do *JDK*, na pasta que contém a biblioteca haverá três arquivos os quais possibilitarão a execução: um arquivo com extensão *JAR,* um *BAT* e um *SH*. Este segundo nada mais faz do que rodar o arquivo .jar através do CMD do Windows. Já o terceiro facilita a utilização da biblioteca em sistemas operacionais Linux.

4. Conclusão

O trabalho foi um grande processo de aprendizado, desde o momento de sua idealização com descrição e os objetivos da aplicação até o momento de apresentá-lo em aula. Dos vários percursos durante o desenvolvimento do projeto, o momento de idealização e de comunicação quanto a lógica e a implementação foram certamente os mais desafiadores.

Traçar os *stories*, escolher as entidades e distribuir tarefas e funcionalidades foram as partes mais tranquilas do projeto. Já a falta de familiaridade e hábito em utilizar a ferramenta de desenvolvimento ágil para controle de tarefas como o *Trello* é algo que ainda deixa margem para melhorias pessoais e da equipe como um todo.

Por fim, e não menos importante, percebemos o quanto a linguagem Java, o paradigma da programação orientada a objetos e o padrão de projetos MVC facilitaram a comunicação entre os membros da equipe quanto às classes e métodos que estavam sendo discutidos em reuniões ou mensagens.

5. Bibliografia

[1] DevMedia; **Encapsulamento, Polimorfismo, Herança em Java.** Disponível em <Encapsulamento, Polimorfismo, Herança em Java>. Acesso em Maio de 2021

[2] Ricarte, Ivan. DCA- UNICAMP. **Construtores.** Disponível em <Construtores>. Acesso em Maio de 2021

[3] DevMedia; **Criando métodos get e set em Java.** Disponível em <Criando métodos get e set em Java>. Acesso em Maio de 2021

[4] Capítulo 05 -Caelum; **Modificadores de Acesso e Atributos de Classe.** Disponível em <Modificadores de Acesso e Atributos de Classe>. Acesso em Maio de 2021

[5] DevMedia; **HashMap Java: Trabalhando com Listas key-value.** Disponível em <HashMap Java: Trabalhando com Listas key-value>. Acesso em Maio de 2021

[6] ORACLE. **Java SE Development Kit 16 Downloads.** Disponível em <**Java SE Development Kit 16 - Downloads**>. Acesso em Março de 2021