Grupo 07

MATRICULADOR INF/UFRGS

Arthur Chagas, Mariana Burzlaff, Vicente Veiga,
João Clementel
Facilitador: Bruno Mengue

Técnicas de Construção de Programas – Turma A
Prof. Karina Kohl
2025/2

INSTITUTO
DE INFORMÁTICA
UFRGS

DESCRIÇÃO DO PROBLEMA

Contexto: Qual problema o projeto resolve?

- Alunos de graduação enfrentam dificuldade para montar horários válidos de matrícula, evitando choques e respeitando preferências pessoais.
- A função de gerar matrícula da universidade não é intuitiva.

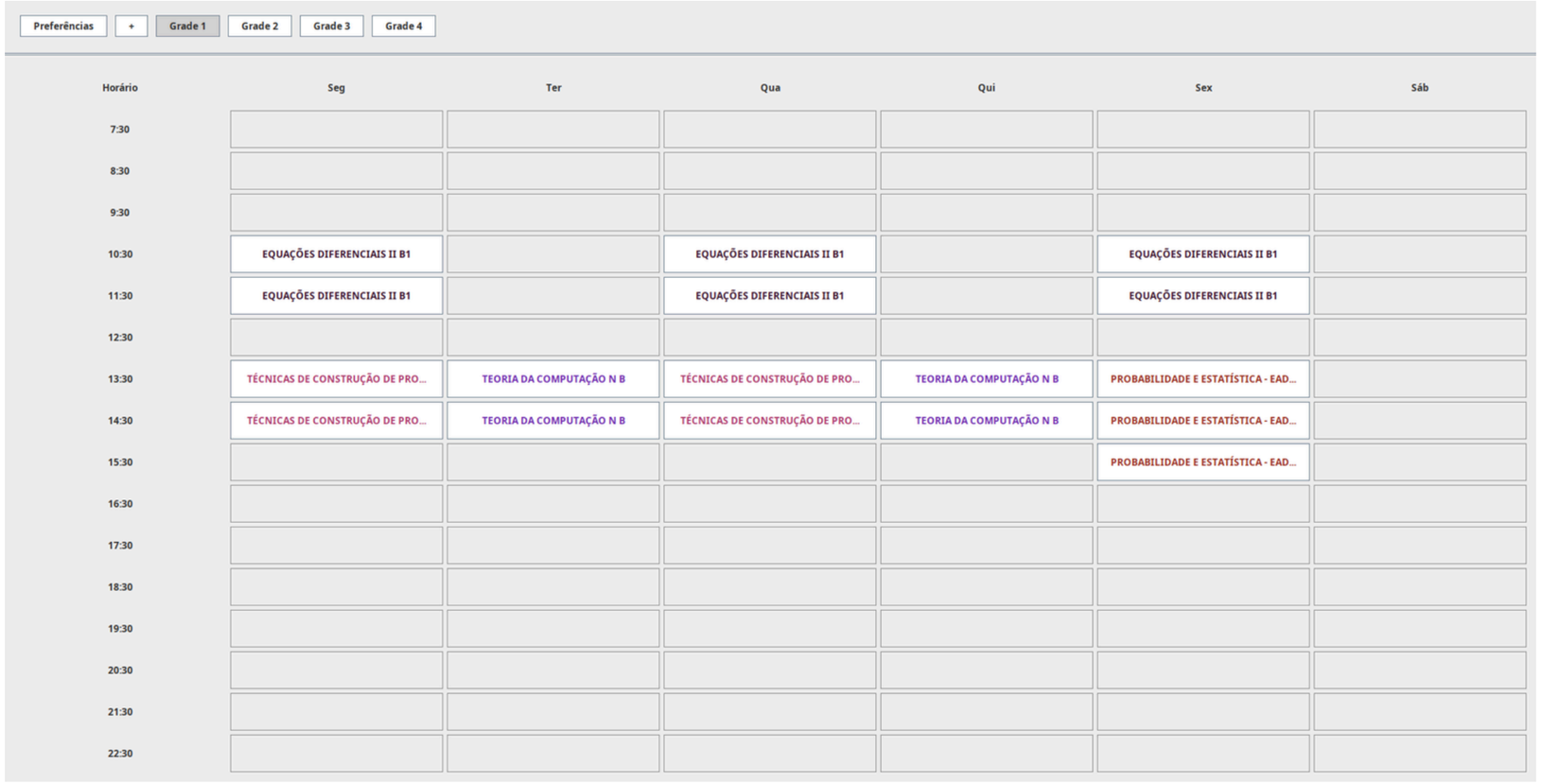
Motivação: Por que esse tema foi escolhido?

- Fazer a matrícula é uma necessidade presente em nossas vidas como estudantes.
- Não existe nenhum "matrículador" voltado aos alunos do INF.

DESCRIÇÃO DO PROBLEMA

Objetivo: O que o trabalho pretende alcançar?

- Criar uma interface gráfica intuitiva e que gere matrículas baseadas nas necessidades e preferências do aluno.



INSTITUTO DE INFORMÁTICA UFRGS

REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Funcionais

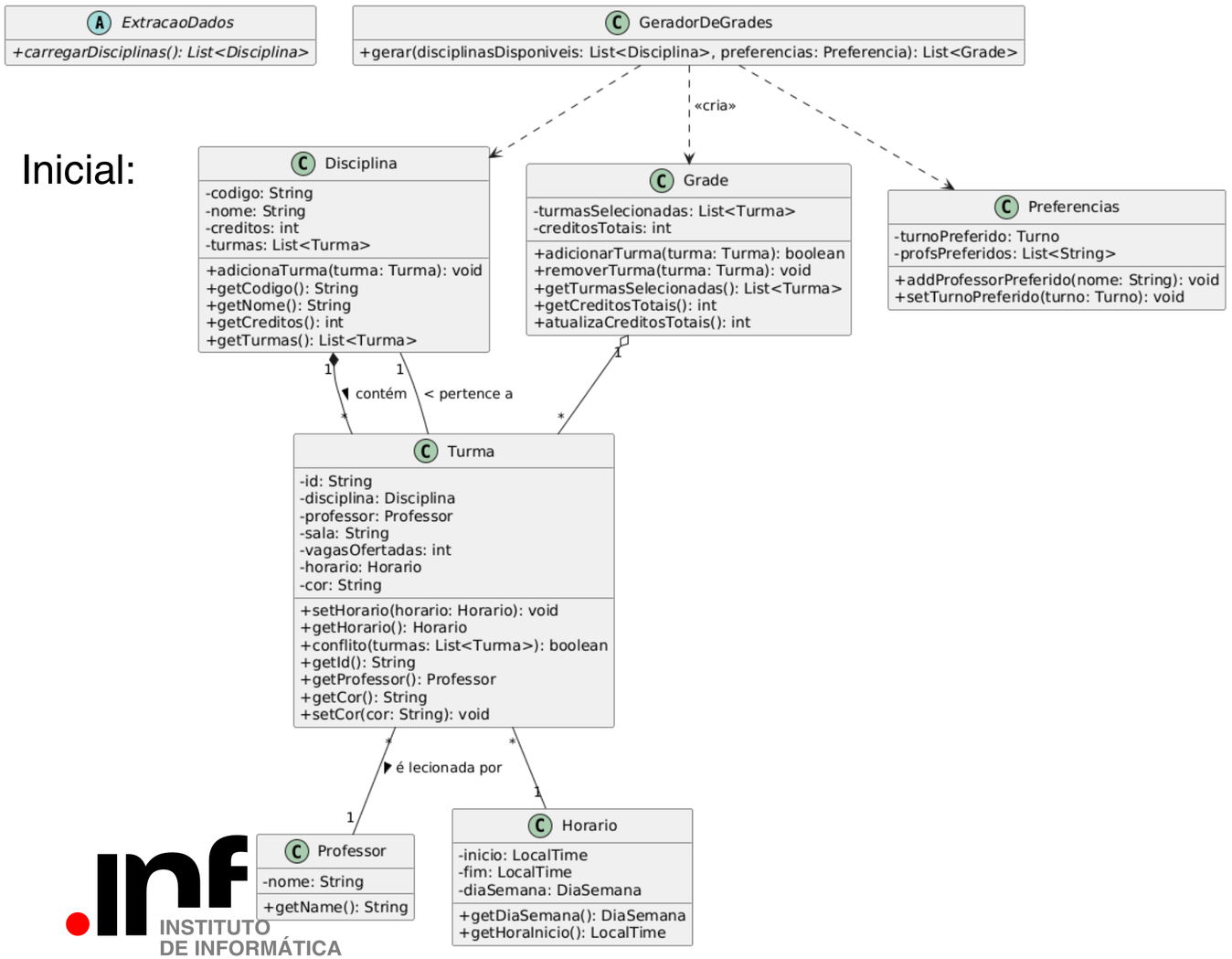
- Permitir o cadastramento de dados relevantes sobre as disciplinas, através de input manual e/ou importação de arquivos.
- Gerar e exibir as opções de grade de forma gráfica/calendário, com dia e horário e com indicadores de conflito.
- Exibir dados de cada disciplina.
- O usuário pode definir prioridades e parâmetros
- Ordenar e aplicar restrições às grades com base nos parâmetros fornecidos pelo usuário.
- As preferências do usuário devem ficar salvas localmente com opção de limpeza.

Não Funcionais

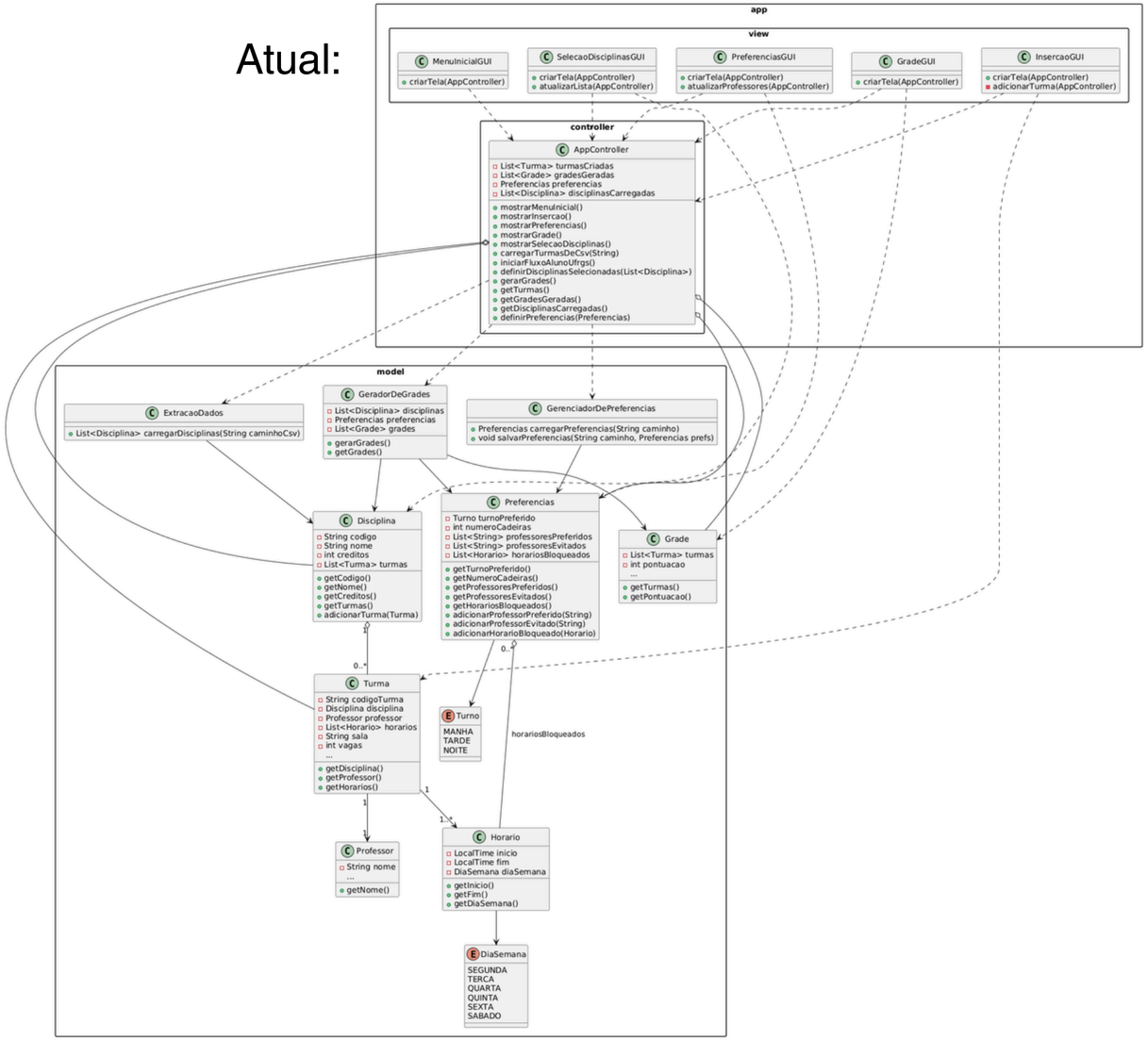
- O sistema deve ser capaz de operar de forma offline (salvo a extração dos dados das disciplinas caso necessário).
- A interface deve organizar as grades com uma indentação pré-definida, devendo ser simples e intuitiva.
- Ao gerar as grades, o sistema não deve demorar mais que poucos segundos.
- O sistema deve ser capaz de lidar com formatos de arquivo inválidos ou dados faltantes, fornecendo uma mensagem de erro clara ao usuário.
- O sistema deve garantir que as grades geradas são as solicitadas pelo usuário, assim como na ordem correta caso seja implementada a ordenação.
- Cada cadeira deve possuir uma cor diferente (arbitrária), facilitando a visualização.

DIAGRAMA DE CLASSES

Inicial:



Atual:



INSTITUTO DE INFORMÁTICA UFRGS

VISÃO DO REPOSITÓRIO

- Link do repositório no GitHub

Capturas de tela do README:

Matriculador INF-UFRGS

Matriculador INF-UfRGS

Geração automática de grades • MVC • Java • Swing

Java 17 Build Gradle Tested JUnit5

Como Executar

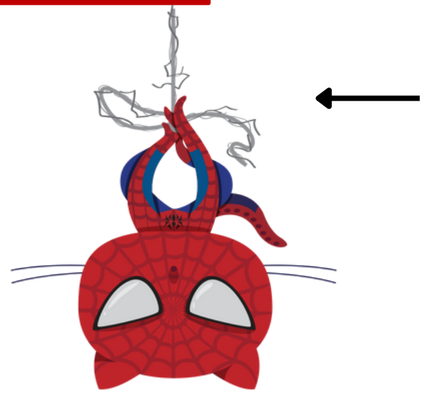
```
./gradlew run
```

Ou:

```
javac -d out src/main/java/**/*.java
java -cp out app.Main
```

Arquitetura do Sistema (MVC)

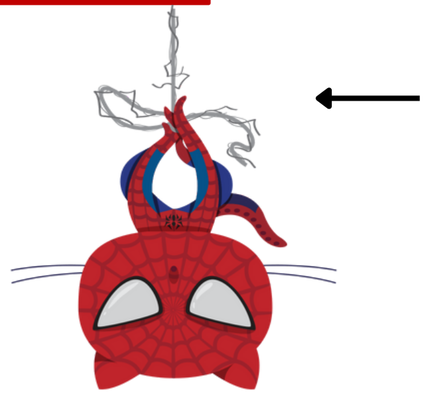
```
src/main/java/
│
├── model/
│      ├── Disciplina.java
│      ├── Turma.java
│      ├── Professor.java
│      ├── Horario.java
│      ├── Grade.java
│      ├── Preferencias.java
│      ├── ExtracaoDados.java
│      ├── GeradorDeGrades.java
│      └── GerenciadorDePreferencias.java
│
├── app/
│      ├── controller/
│      │       └── AppController.java
│      └── view/
│              ├── MenuInicialGUI.java
│              ├── InsercaoGUI.java
│              ├── SelecaoDisciplinasGUI.java
│              ├── PreferenciasGUI.java
│              └── GradeGUI.java
│
└── Main.java
```



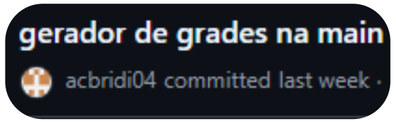
INSTITUTO DE INFORMÁTICA UFRGS

VISÃO DO REPOSITÓRIO

- Link do repositório no GitHub

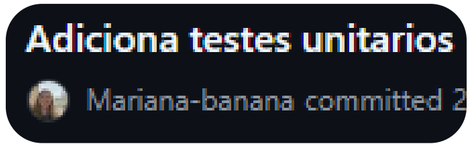


Exemplos de commits:



← testando o gerador na main

← criação da interface



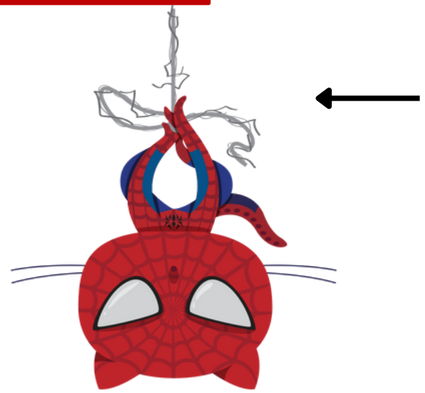
← criação dos testes

← commit inicial

Cada commit corrige ou adiciona uma funcionalidade especifíca, sem misturar responsabilidades!

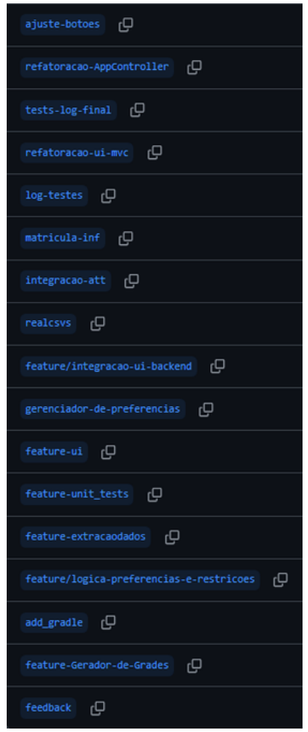
VISÃO DO REPOSITÓRIO

- Link do repositório no GitHub



Branches que começaram a deixar o projeto mais robusto:

Visão geral:



Quantidade aproximada de commits:

INSTITUTO DE INFORMÁTICA UFRGS

VISÃO GERAL DO CÓDIGO

Método responsável por gerar as grades:

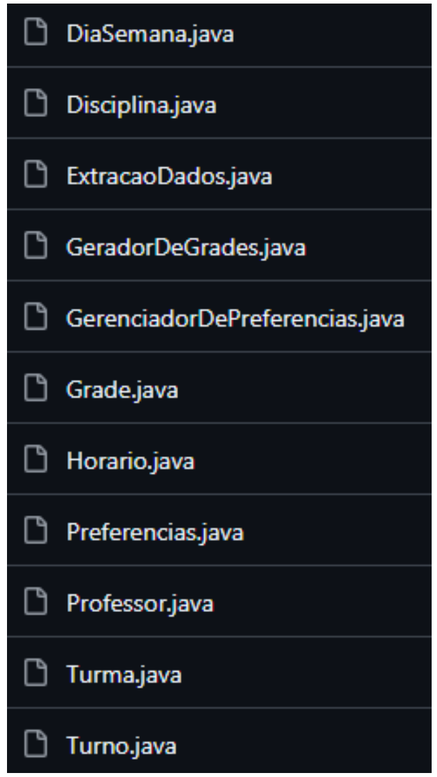
```java
private void buscarCombinacoes(Grade gradeAtual, int indexDisciplina) {
    int numCadeiras = preferenciasUsuario.getNumeroCadeiras();
    int qtdAtual = gradeAtual.getTurmasSelecionadas().size();

    if (qtdAtual == numCadeiras) {
        gradesGeradas.add(new Grade(gradeAtual));
        return;
    }

    if (indexDisciplina == disciplinasDisponiveis.size()) {
        return;
    }

    int disciplinasRestantes = disciplinasDisponiveis.size() - indexDisciplina;
    if (qtdAtual + disciplinasRestantes < numCadeiras) {
        return;
    }

    Disciplina disciplina = disciplinasDisponiveis.get(indexDisciplina);

    for (Turma turma : disciplina.getTurmas()) {
        if (violaRestricoes(turma, gradeAtual)) {
            continue;
        }

        if (gradeAtual.adicionarTurma(turma)) {
            buscarCombinacoes(gradeAtual, indexDisciplina + 1);
            gradeAtual.removerTurma(turma);
        }
    }

    buscarCombinacoes(gradeAtual, indexDisciplina + 1);
}
```

Método que da pontos para as grandes:

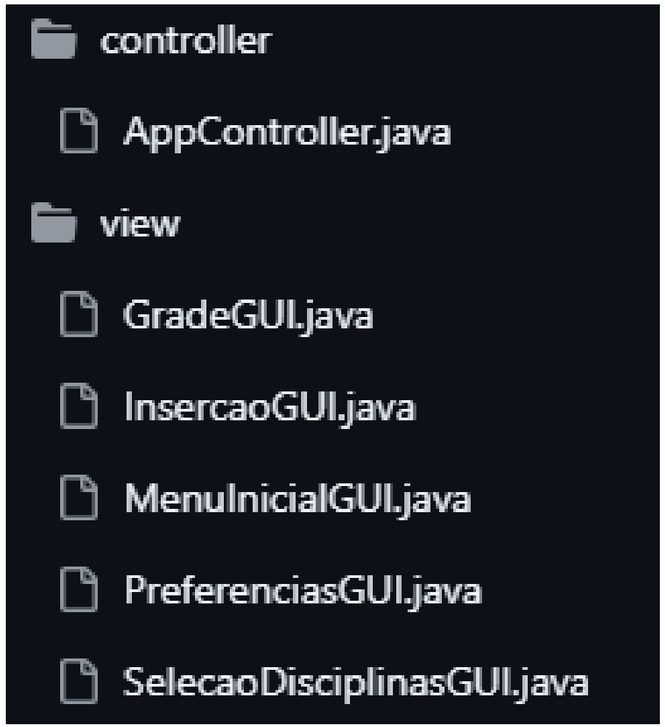
```java
private int calcularPontuacao(Grade grade) {
    int pontos = 0;

    for (Turma turma : grade.getTurmasSelecionadas()) {
        String nomeProf = turma.getProfessor().getNome().trim();
        for (String favorito : preferenciasUsuario.getProfessoresPreferidos()) {
            if (nomeProf.equalsIgnoreCase(favorito.trim())) {
                pontos += 100;
                break;
            }
        }

        Turno turnoPref = preferenciasUsuario.getTurnoPreferido();
        if (turnoPref != null) {
            for (Horario h : turma.getHorarios()) {
                if (ehDoTurno(h, turnoPref)) {
                    pontos += 10;
                }
            }
        }
    }

    return pontos;
}
```

VISÃO GERAL DO CÓDIGO

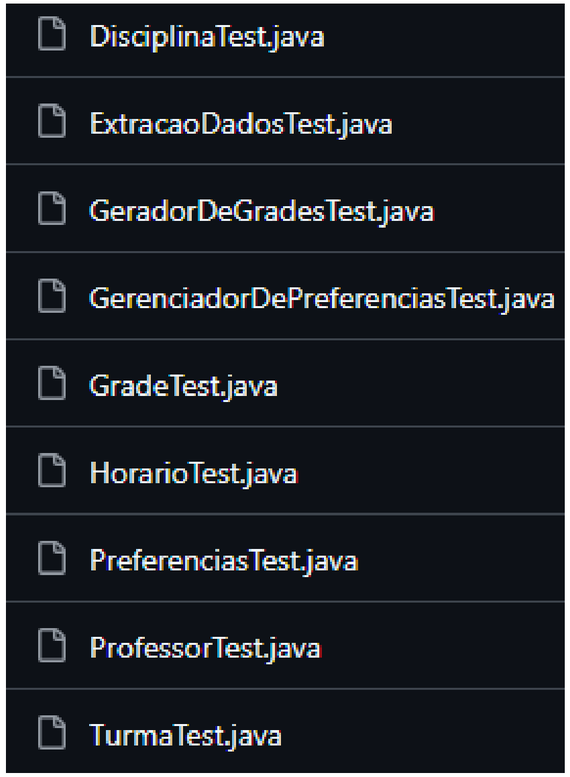
Pasta model:



Interface

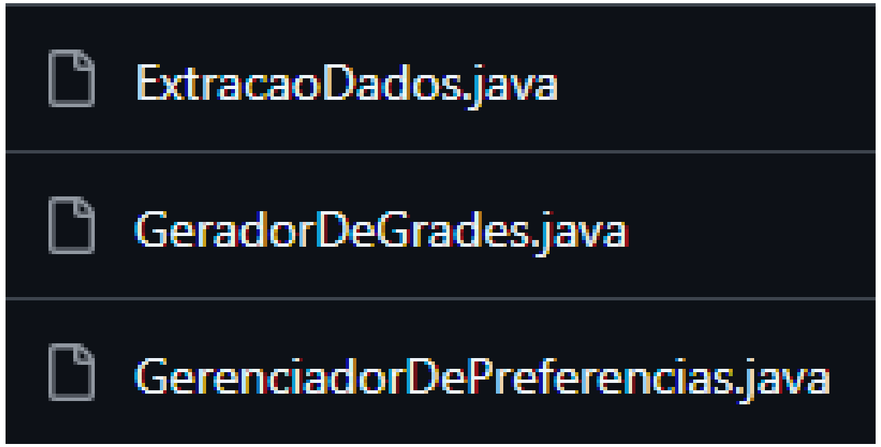


Pasta test/model



VISÃO GERAL DO CÓDIGO

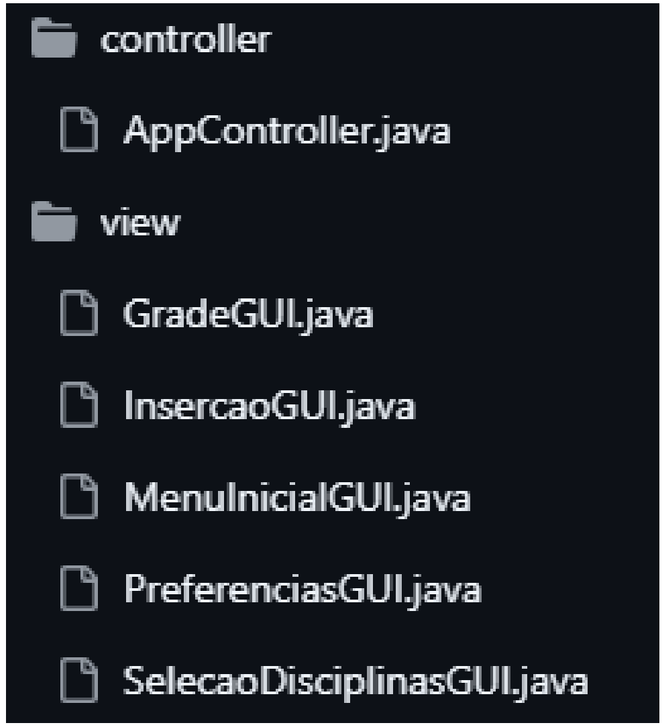
Arquivos que mais concentram a lógica do backend:



↑

Essas classes concentram as regras de negócio, ficando totalmente separadas da interface gráfica.

Arquitetura da interface:



A interface foi organizada segundo o padrão MVC, garantindo separação clara entre apresentação (view), controle (controller) e lógica (model).

TESTES

Test Summary

29	0	0	0.201s
tests	failures	ignored	duration

100%
successful

Para os testes foi usada a ferramenta Junit:

Class	Tests	Failures	Ignored	Duration	Success rate
model.DisciplinaTest	3	0	0	0.077s	100%
model.ExtracaoDadosTest	2	0	0	0.049s	100%
model.GeradorDeGradesTest	4	0	0	0.020s	100%
model.GerenciadorDePreferenciasTest	4	0	0	0.028s	100%
model.GradeTest	5	0	0	0.008s	100%
model.HorarioTest	3	0	0	0.004s	100%
model.PreferenciasTest	4	0	0	0.005s	100%
model.ProfessorTest	2	0	0	0.005s	100%
model.TurmaTest	2	0	0	0.005s	100%

TESTES

Alguns casos testados:

```java
@Test
@DisplayName("Gera grades se não houver conflitos")
void geraGradeSemConflitos() {
    logger.info(message: "Teste: Gerar grade sem conflitos.");

    adicionarDisciplinaComTurma(codigo: "INF01120", nome: "TCP", codigoTurma: "A", nomeProf: "Karina Kohl", inicio: 8, fim… 10, DiaSemana.SEGUNDA);
    adicionarDisciplinaComTurma(codigo: "INF05516", nome: "Semantica Formal", codigoTurma: "A", nomeProf: "Alvaro Moreira", inicio: 10, fim… 12, DiaSemana.SEGUNDA);

    GeradorDeGrades gerador = new GeradorDeGrades(disciplinas, preferencias);
    gerador.gerarGrades();

    List<Grade> grades = gerador.getGrades();

    assertEquals(expected: 1, grades.size());
    assertFalse(grades.isEmpty(), message: "Deve gerar pelo menos uma grade válida.");
    assertEquals(expected: 2, grades.get(0).getTurmasSelecionadas().size(), message: "A grade deve conter 2 disciplinas.");

    logger.info(message: "Sucesso: Grade gerada com 2 turmas.");
}
```

```java
@Test
@DisplayName("Nao deve adicionar turma se houver conflito de horario")
void bloquearConflitoHorario() {
    logger.info(message: "Teste: Bloquear adicao de turma com conflito de horario.");

    Turma t1 = new Turma(codigo: "A", karina, tcp, vagasOfertadas: 30, sala: "108");
    t1.addHorario(new Horario(LocalTime.of(8,0), LocalTime.of(10,0), DiaSemana.SEGUNDA));
    grade.adicionarTurma(t1);

    Turma t2 = new Turma(codigo: "A", alvaro, semantica, vagasOfertadas: 30, sala: "109");
    t2.addHorario(new Horario(LocalTime.of(9,0), LocalTime.of(11,0), DiaSemana.SEGUNDA));

    boolean adicionou = grade.adicionarTurma(t2);

    assertFalse(adicionou, message: "Nao deveria adicionar turma com conflito de horario");
    assertEquals(expected: 1, grade.getTurmasSelecionadas().size());

    logger.info(message: "Sucesso: Adicao de turma bloqueada devido a conflito de horario.");
}
```

```java
@Test
@DisplayName("Nao deve adicionar uma mesma turma duas vezes")
void bloquearDuplicidadeTurma() {
    logger.info(message: "Teste: Bloquear adicao duplicada de turma.");

    Turma t1 = new Turma(codigo: "A", karina, tcp, vagasOfertadas: 30, sala: "108");
    t1.addHorario(new Horario(LocalTime.of(8,0), LocalTime.of(10,0), DiaSemana.SEGUNDA));
    grade.adicionarTurma(t1);

    boolean adicionouNovamente = grade.adicionarTurma(t1);

    assertFalse(adicionouNovamente, message: "Nao deveria permitir adicionar a mesma turma duas vezes");
    assertEquals(expected: 1, grade.getTurmasSelecionadas().size());

    logger.info(message: "Sucesso: Adicao duplicada de turma bloqueada.");
}
```

INSTITUTO DE INFORMÁTICA UFRGS

DEMONSTRAÇÃO

Agora iremos demonstrar como o matriculador funciona!

CONCLUSÃO

Principais Dificuldades:

- **Reunir o grupo inteiro para trabalhar juntos.**
  - Conciliar horários foi uma tarefa difícil no início, mas a partir do momento em que começamos a comunicar todas as mudanças e dividir as tarefas de forma mais clara, o grupo passou a funcionar muito melhor
- **Integração entre o backend e a UI sem quebrar o fluxo**
  - Criamos um controlador central (AppController) que gerencia toda a navegação e os dados carregados.

- O projeto reforçou os conceitos de POO e a importância de um código organizado.
- Além disso, foi uma experiência em conjunto muito proveitosa que mostrou o que se pode ou não fazer quando se está trabalhando em equipe

CONCLUSÃO

Próximos passos: O que pode ser melhorado?

- **Modularização mais forte da UI**
  - Separar melhor lógica de navegação e componentes visuais. Criar mais componentes reutilizáveis.
- **Adicionar validações mais completas**
  - Alertar quando disciplinas exigem pré-requisitos ou simultaneidade.
- **Expandir o modo Aluno INF/UFRGS para todos os cursos**
  - Com isso ninguém da universidade precisaria carregar um CSV ou fazer o input manualmente.

CONCLUSÃO

Obrigado pela atenção!! ;)

Perguntas?