

Studify

Grupo: 04

Integrantes: Cristopher dos Santos Filho,
Luís Filipe Moura,
Milena Silva,
Nickolas Xisto Machado,
Pedro Schuck de Azevedo.

Facilitador: Luís Filipe Moura

Disciplina: INF01120

Professora: Karina Kohl

1.2 Alterações

Nenhuma alteração relevante em relação à etapa anterior foi feita.

1.3 Requisitos

Requisitos Funcionais	Requisitos não funcionais
Rf-1: O usuário deve ser capaz de definir os horários disponíveis para estudo	Rnf-1: A interface do sistema deve ser intuitiva ao usuário, sem necessitar nenhum tipo de tutorial.
Rf-2: O usuário deve ser capaz de definir matérias para o qual deseja delimitar tempo, bem como sua prioridade	Rnf-2: O sistema deve identificar de forma clara matérias a serem estudadas em cada horário, bem como a duração.
Rf-3: O usuário deve ser capaz de adicionar provas e trabalhos em datas específicas	Rnf-3: O sistema deve identificar de forma clara datas importantes como provas e trabalhos.
Rf-4: O sistema deve gerar, através de um algoritmo simples, uma grade de horários de estudos para o usuário	Rnf-4: O sistema deve retornar o cronograma montado em até 2 segundos
Rf-5: O usuário deve ser capaz de verificar sua agenda para um dia o qual ele selecionar	Rnf-5: O sistema deve ser capaz de salvar um cronograma já gerado em até 2 segundos
Rf-6: O usuário deve ser capaz de salvar um cronograma gerado	Rnf-6: O sistema deve ser capaz de carregar um cronograma já gerado em até 2 segundos
Rf-7: O usuário deve ser capaz de carregar um cronograma já gerado	Rnf-7: O sistema deve ser capaz de lidar com alocação de pelo menos 10 matérias sem afetar seu desempenho

A ordem dos requisitos funcionais foi escolhida desta forma, pois sem a criação de um cronograma bem detalhado, para dar mais prioridade a um cronograma que possa ser gerado de forma mais completa.

Já para os requisitos não funcionais, é de suma importância que não haja desentendimento sobre quando o usuário possui uma data importante, ou que ele não se confunda com qual matéria ele deveria estudar.

1.4 Projeto

Classe Disciplina:

Atributos:

- nome: String // Nome da disciplinas definida pelo usuário
- prioridade: int // Prioridade definida pelo usuário em ranking de disciplinas
- atividades: ArrayList<Atividade> // Lista de atividades referente a esta disciplina
- _totalDisciplinas: int

Métodos:

- + Disciplina(nome: String, rank: int): Disciplina
- + updateDisciplina(nome: String, rank: int): void
- + setNome(nome: String): void
- + setPrioridade(rank: int): void
- + getNome(): String
- + getPrioridade(): int
- + adicionarAtividade(atividade: Atividade): void
- + excluirAtividade(id: int): void
- + getAtividades(): List<Atividade>
- + _getTotalDisciplinas(): int

Classe Abstrata Atividade:

Atributos:

- id: final int // Definido no construtor, único e imutável
- nome: String // Definido pelo usuário
- dataEntrega : LocalDate // Definida pelo usuário
- disciplina: Disciplina // Referência à disciplina a qual a atividade pertence
- _total_atividades: int // Contador geral de atividades, incrementado no construtor

Métodos:

- + Atividade(nome: String, dataEntrega: LocalDate, disciplina: Disciplina)
- + getId(): int
- + getNome(): String
- + setNome(nome: String): void
- + getDataEntrega(): LocalDate
- + setDataEntrega(data: LocalDate): void
- + getDisciplina(): Disciplina
- + setDisciplina(disciplina: Disciplina): void
- + getTotalAtividades(): static int
- + checkData(semestre: Semestre): boolean

// templates implementados na classe pai

```
+ toString(): String
+ equals(obj: Object): boolean

// cada subclasse implementa
+ getPeso(): abstract double
+ getTipoContador(): abstract int
+ incrementarContador(): abstract void
+ getTipoNome(): abstract String
```

Classe Prova extends Atividade:

Atributos:

```
+ _ final PESO: double
- _ totalProvas: int
```

Métodos:

```
+ Prova(nome: String, dataEntrega: LocalDate, disciplina: Disciplina)
// implementação dos métodos abstratos
+ getPeso(): double
+ getTipoContador(): int
+ incrementarContador(): void
+ getTipoNome(): String
//métodos específicos de prova
+ getTotalProvas(): _int
+ resetarContadorProvas(): _void
```

Classe Trabalho extends Atividade:

Atributos:

```
+ _ final PESO: double
- _ totalTrabalhos: int
```

Métodos:

```
+ Trabalho(nome: String, dataEntrega: LocalDate, disciplina: Disciplina)
// implementação dos métodos abstratos
+ getPeso(): double
+ getTipoContador(): int
- incrementarContador(): void
+ getTipoNome(): String
// métodos específicos de trabalho
+ getTotalTrabalhos(): _int
+ resetarContadorTrabalhos(): _void
```

Classe Exercicio extends Atividade:

Atributos:

```
+ _ final PESO: double
- _ totalExercicios: int
```

Métodos:

```
+ Exercicio(nome: String, dataEntrega: LocalDate, disciplina: Disciplina)
// implementação dos métodos abstratos
+ getPeso(): double
+ getTipoContador(): int
- incrementarContador(): void
+ getTipoNome(): String
```

```
// métodos específicos de trabalho
+ getTotalExercicios(): static int
+ resetarContadorExercicios(): static void
```

Classe TimeSlot:

Atributos:

- horaInicio: LocalTime
- horaFim: LocalTime
- diasSemana: Set<DayOfWeek>
- recorrente: boolean
- dataInicioVigencia: LocalDate
- dataFimVigencia: LocalDate

Métodos:

- + TimeSlot(inicio: LocalTime, fim: LocalTime, dias: Set<DayOfWeek>, recorrente: boolean)
- + TimeSlot(inicio: LocalTime, fim: LocalTime, dia: DayOfWeek)
- + isDisponivelEm(dataHora: LocalDateTime): boolean
- + conflitaCom(outro: TimeSlot): boolean
- + gerarSlotsEstudo(dataInicio: LocalDate, dataFim: LocalDate): List<TimeSlotEstudo>
- + contemDia(dia: DayOfWeek): boolean
- + contemHorario(hora: LocalTime): boolean
- + getHoraInicio(): LocalTime
- + getHoraFim(): LocalTime
- + getDiasSemana(): Set<DayOfWeek>
- + isRecorrente(): boolean
- + setDataVigencia(inicio: LocalDate, fim: LocalDate): void

Classe TimeSlotEstudo

Atributos:

- + _final DURACAO_PADRAO: Duration = Duration.ofMinutes(30)
- data: final LocalDate
- time: final LocalTime
- atividade: final Atividade
- slotOrigem: final TimeSlot

Métodos:

- + TimeSlotEstudo(data: LocalDate, time: LocalTime, atividade: Atividade)
- + getAtividade(): Atividade
- + getData(): LocalDate
- + getTime(): LocalTime
- + getDuracao(): _Duration
- + getHoraFim(): LocalTime
- + isDisponivel(): boolean
- + conflitaCom(outro: TimeSlotEstudo): boolean

Classe Impedimento

Atributos:

- dataInicio: LocalDateTime
- dataFim: LocalDateTime
- motivo: String

Métodos:

- + Impedimento(data: LocalDate)
- + Impedimento(dataInicio: LocalDateTime, dataFim: LocalDateTime)
- + Impedimento(dataInicio: LocalDateTime, duracao: Duration)
- + Impedimento(data: LocalDate, motivo: String)
- + Impedimento()
- + conflitaCom(timeSlot: TimeSlotEstudo): boolean
- + conflitaCom(periodo: LocalDateTime, duracao: Duration): boolean
- + cobreHorario(momento: LocalDateTime): boolean
- + isDiaInteiro(): boolean
- + getDuracao(): Duration
- + getDataInicio(): LocalDateTime
- + getDataFim(): LocalDateTime
- + getMotivo(): String
- + setMotivo(motivo: String): void
- + getSlotOrigem(): TimeSlot
- + setSlotOrigem(origem: TimeSlot): void

Classe AgendaEstudos

Atributos:

- + _final DURACAO_TIMESLOT: Duration = Duration.ofMinutes(30)
- horariosFixos: final List<TimeSlot>
- impedimentos: final List<Impedimento>
- estudos: final List<TimeSlotEstudo>
- disciplinas: final List<Disciplina>
- cronogramaCriado: boolean

Métodos:

- + AgendaEstudos(horariosFixos: List<TimeSlot>, disciplinas: List<Disciplina>, impedimentos: List<Impedimento>)
- + criaEstudos(semestre: Semestre): void
- + distribuirEstudosPorPrioridade(): void
- + calcularQuantidadeTimeSlotsNecessarios(disciplina: Disciplina): int
- + calcularHorasNecessarias(disciplina: Disciplina): double
- gerarTimeSlotsDisponiveis(dataInicio: LocalDate, dataFim: LocalDate): List<TimeSlotEstudo>
- validarTimeSlot(timeSlot: TimeSlotEstudo): boolean
- isHorarioDisponivel(momento: LocalDateTime): boolean
- temImpedimentoEm(momento: LocalDateTime): boolean
- + getHorariosFixos(): List<TimeSlot>
- + getEstudos(): List<TimeSlotEstudo>
- + getDisciplinas(): List<Disciplina>
- + getImpedimentos(): List<Impedimento>
- + obterEstudosParaDisciplina(disciplina: Disciplina): List<TimeSlotEstudo>
- + isCronogramaCriado(): boolean
- + getDuracaoTimeslot(): _Duration
- + criarNovaAgenda(novasConfiguracoes: ConfiguracaoAgenda): _AgendaEstudos

Classe Abstrata ConfiguracaoAgenda:

Atributos:

- horariosFixos: List<TimeSlot>
- disciplinas: List<Disciplina>
- impedimentos: List<Impedimento>
- semestre: Semestre

Métodos:

- + ConfiguracaoAgenda(semestre: Semestre)
- + adicionarHorarioFixo(slot: TimeSlot): void
- + adicionarDisciplina(disciplina: Disciplina): void
- + adicionarImpedimento(impedimento: Impedimento): void
- + gerarAgenda(): AgendaEstudos
- + validarConfiguracao(): boolean
- + getHorariosFixos(): List<TimeSlot>
- + getDisciplinas(): List<Disciplina>
- + getImpedimentos(): List<Impedimento>
- + getSemestre(): Semestre

Classe AgendaFactory:

Métodos:

- + criarAgenda(configuracao: ConfiguracaoAgenda): _AgendaEstudos
- + criarAgendaPersonalizada(horariosFixos: List<TimeSlot>, disciplinas: List<Disciplina>, impedimentos: List<Impedimento>, semestre: Semestre): _AgendaEstudos
- validarParametros(configuracao: ConfiguracaoAgenda): boolean
- otimizarDistribuicao(configuracao: ConfiguracaoAgenda): List<TimeSlotEstudo>

Classe Semestre:

Atributos:

- dataInicio : LocalDate
- dataFim : LocalDate

Métodos:

- + Semestre(dataInicio: LocalDate, dataFim: LocalDate)
- + getDataInicio(): LocalDate
- + getDataFim(): LocalDate
- + getDuracao(): Period
- + contemData(data: LocalDate): boolean
- + isAtivo(): boolean

Relações:

AgendaEstudos	1..1	agrega	1..*	TimeSlot
AgendaEstudos	1..1	agrega	0..*	Impedimento.
AgendaEstudos	1..1	compõe	0..*	TimeSlotEstudo.
AgendaEstudos	1..1	agrega	1..*	Disciplina.
Disciplina	1..1	agrega	0..*	Atividade.

TimeSlotEstudo	0..n	associa-se com	1..1	Atividade.
----------------	------	----------------	------	------------

Prova	herda	de Atividade.
Trabalho	herda	de Atividade.

Exercício herda de Atividade.

AgendaFactory 1..1 cria 1..1 AgendaEstudos.
ConfiguracaoAgenda 1..1 cria 1..1 AgendaEstudos.
ConfiguracaoAgenda 1..1 agrega 1..1 Semestre

TimeSlotEstudo 0..1 associa-se com 1..1 TimeSlot

Existem outras relações de dependências mais sutis, como AgendaEstudos depende de semestre, mas outras relações já evidenciam essas dependências. Por isso, optamos por ocultar

Explicação:

Como o objetivo do programa é gerar uma agenda para o semestre, temos uma classe central AgendaEstudos. Outras classes como Impedimento, Disciplina, TimeSlot (horários fixos disponíveis), Semestre e Atividade existem para armazenar as respectivas entradas do usuário. A classe ConfiguracaoAgenda serve para guardar os parâmetros para gerar a agenda e AgendaFactory é responsável pela criação da agenda final. A classe TimeSlotEstudo será uma classe de saída com os horários agendados para estudo, vinculados a uma atividade específica. Na modelagem atual, uma vez criada a agenda, ela não pode ser modificada.

A AgendaEstudos agrega as configurações do usuário (disciplinas, horários, impedimentos) e compõe os TimeSlotEstudo como resultado final. As classes Prova, Trabalho e Exercício especializam Atividade usando Template Method Pattern para reduzir redundância de código.

1.5 Interface

Tela inicial (a esquerda). Botões para preencher os dias da semana (em verde), setar data atua e de fim do semestre, adicionar compromisso e próxima janela.

* = prova, trabalho ou exercício

A esquerda: Preencher os dias da semana com a grade de horários de estudos pretendidos.
A direita: Inserir disciplinas em sua ordem de prioridade e suas respectivas atividades.

semana 01/01 - 07/01

<

seg

ter

qua

qui

sex

sab

dom

>

disciplinas

1 - XXXXX

2 - YYYYY

3 - ZZZZZ

atividades nesta semana

seg - estudar P1, T3 e E2

ter - entrega E2, estudar P1, T3, P2

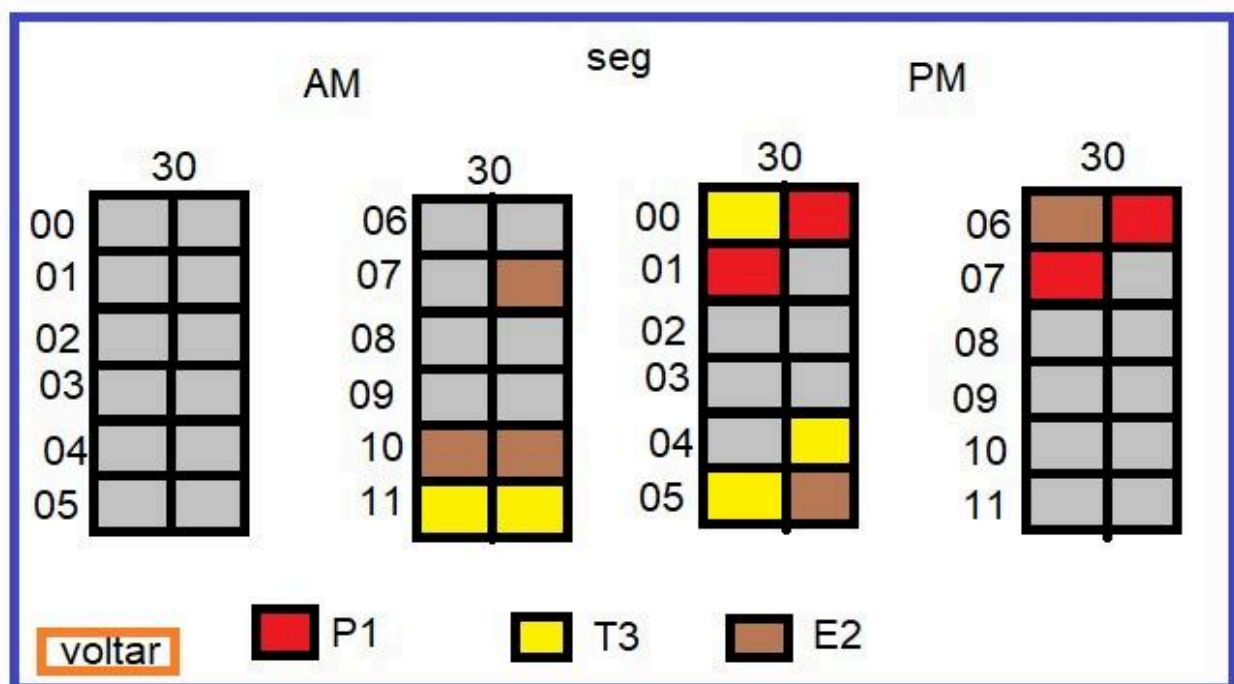
qua - entrega T3, estudar P2, P1, E3

qui - entrega P2 e E3, estudar P1

sex - entrega P1, estudar T1, P3

[...]

reiniciar



indisponível para estudo

Visualização da agenda recebida pelo usuário.