

# Studify

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

**Disciplina:** INF01120 - Técnicas de  
Construção de Programas

**Professora:** Karina Kohl

## 1.2 Mudanças em relação à Etapa anterior

Algumas modificações foram feitas em relação à etapa anterior, entre elas:

- O tipo do atributo prioridade da classe Disciplina foi trocado para double;
- Atributo id de Atividade foi extinto, pois não apresentou utilidade;
- O nome do atributo dataEntrega foi trocado para dataLimite, pois dataEntrega limitava seu significado;
- Remoção da classe TimeSlot, que servia apenas para renomear LocalTime;
- Atributo slotOrigem, com tipo TimeSlot, de TimeSlotEstudo foi removido, pois a classe referente a seu tipo foi removida;
- Atributos LocalDate data e LocalTime time em TimeSlotEstudo foram substituídos por InicioEstudo, que é LocalDateTime;
- Atributos dataInicio e dataFim de Impedimento foram substituídos por dataHora, porque não faziam sentido;
- Atributo motivo de Impedimento foi descartado porque não seria usado;
- Atributo horariosFixos de AgendaEstudos foi substituído, pois classe TimeSlot foi removida, mas seu papel foi assumido pelo atributo diasSemana na classe ConfiguracaoAgenda;
- Atributo impedimentos de AgendaEstudos foi movido para ConfiguracaoAgenda para aumentar encapsulamento;
- Atributo disciplinas de ConfiguracaoAgenda foi movido para a nova classe Aluno para aumentar encapsulamento e melhorar a clareza;
- Classe Semestre e todos atributos do tipo Semestre foram excluídos, já que sua utilidade e papel não justificavam a existência da classe;
- Os novos atributos dataInicioVigencia e dataFimVigencia da classe ConfiguracaoAgenda substituíram os atributos da classe Semestre;

- Classe AgendaFactory foi separada em três novas classes: GeradorAgenda, AtribuidorAtividades e DistribuidorAtividades para respeitar o SRP;
- Nova classe CalculadoraPesoAtividades foi criada para respeitar o SRP, porque seu papel seria executado a partir de outras classes;
- Classe DiaSemana foi criada para guardar os horários configurados pelo usuário para um dia da semana;
- Nova classe auxiliar AlocacaoAtividade foi criada;

### 1.3 Implementação

Fortemente baseada no diagrama de classes modificado, assim como neste foram criadas as classes Disciplina, Impedimento, ConfiguracaoAgenda, Aluno, TimeSlotEstudo, AgendaEstudos, GeradorAgenda, DiaSemana, AtribuidorAtividades, DistribuidorAtividades, AlocacaoAtividade, CalculadoraPesoAtividades, Atividade e suas classes filhas Prova, Trabalho e Exercício.

(descrição do que foi implementado, relacionar código com classes definidas, seus relacionamentos e adaptações)

Na classe Disciplina, armazenou-se os atributos nome, prioridade, lista de atividades e total de disciplinas. Tendo seu propósito focado em armazenar informações referentes a cada disciplina inserida pelo usuário, de maneira a evitar que outra classe acumulasse responsabilidades em demasia, seus métodos são apenas construtores, getters e setters.

De maneira similar, a classe DiaSemana, que possui como atributos um dia da semana e uma lista de horários, tem como utilidade central guardar dados inseridos pelo usuário. Assim, sendo uma classe que encapsula informações provenientes de determinada tela da aplicação e que é posteriormente manipulada por outras classes, dispõem apenas de métodos getters, setters e construtores.

Por sua vez, a classe Impedimento tem o atributo data e horário, impondo neste período de tempo o significado de que nele o usuário estará ocupado, não podendo-o usar para estudos. Além de métodos construtores, getters e setters, ela também apresenta a validação conflitaCom, que determina se o horário do impedimento conflita com certa data e horário.

A classe Aluno foi utilizada como classe de dados, servindo para armazenar as disciplinas, a configuração da agenda e a própria agenda de estudos. Sendo assim, ela possui apenas construtores, getters e setters para estes atributos, fato necessário pois classes como GeradorAgenda e AtribuidorAtividades necessitavam exclusivamente destes dados, portanto foi criada esta classe para comportá-los. Também não compete a classe Aluno criar diretamente a sua AgendaEstudos, ela é gerada e preenchida usando GeradorAgenda e AtribuidorAtividades respectivamente.

Já a classe ConfiguracaoAgenda tem como principal propósito salvar informações para a construção da agenda de estudos, sendo composta pelos atributos lista de impedimentos, lista de dias da semana, data final e data inicial de

vigência da agenda. Dessa forma, possui construtores, getters e setters para manipular tais atributos, mas também outros métodos para conferir a validade destes, como checar se a data de vigência inicial precede a final e se uma determinada data está entre ambas as datas de vigências.

A função da classe DiaSemana é guardar os horários de estudo configurados pelo usuário para um dia da semana. Posteriormente, esses horários servirão como moldes para objetos da classe TimeSlotEstudo.

O papel da classe Impedimento é sinalizar que não deve haver um TimeSlotEstudo com data e horário iguais aos do Impedimento, pois o usuário possui uma indisponibilidade para estudo naquele horário. Assim, por mais que seja uma data e horário válidos no cotidiano do usuário, sendo normalmente configurados para estudar, existe uma exceção nesta data específica, sendo impedida a criação de um TimeSlotEstudo nela.

Enquanto isso, a classe TimeSlotEstudo tem como funcionalidade central manter cada um dos períodos de estudo de 30 minutos que compõem toda a agenda, possuindo atributos como duração, disponibilidade, atividade, data e hora do início do estudo. Guardando em si construtores, getters e setters, esta classe dispõe de outros tipos de métodos, como um para verificar a validade de uma atividade e outro para analisar se certo impedimento conflita com a hora inicial do TimeSlotEstudo em questão.

A principal funcionalidade da classe GeradorAgenda é criar uma AgendaEstudos com base em ConfiguracaoAgenda. A AgendaEstudos gerada estará vazia, ou seja, os TimeSlotEstudo presentes no atributo estudos da AgendaEstudos gerada não terão Atividades associadas. Cada TimeSlotEstudo gerado terá sua data e horário em dia da semana com horários configurados em um dos diasSemana de ConfiguracaoAgenda. Também não haverão TimeSlotEstudo com a mesma data e hora que qualquer um dos impedimentos listados em ConfiguracaoAgenda. O processo executado por essa classe consiste em basicamente iterar pelas datas, começando na dataInicioVigencia e indo até dataFimVigencia de ConfiguracaoAgenda, verificando em qual dia da semana a data pertence, assim como se esse dia da semana tem horários de estudo configurados em diasSemana, e caso este fato mostrar-se verdadeiro, itera-se sobre eles, e na ocasião de não haver um impedimento nesta data e nesse horário, cria-se um TimeSlotEstudo com estes dados.

A responsabilidade central da classe AtribuidorAtividades é preencher uma AgendaEstudos com atividades, ou seja, atribuir objetos de classes que herdam de Atividade a classes TimeSlotEstudo presentes no atributo estudos da AgendaEstudos. Vários TimeSlotEstudo podem receber a mesma atividade, sendo que a decisão de qual TimeSlotEstudo vai receber qual atividade é tomada com base no pesoCalculado das Atividades. Esta classe usa um DistribuidorAtividades para decidir como será a distribuição das atividades para o conjunto de TimeSlotEstudo, também utilizando a classe auxiliar AlocacaoAtividade. O processo do método atribuir consiste, basicamente, em percorrer as "janelas de tempo", calcular porcentagem e posteriormente a quantidade de TimeSlotEstudo de dentro

da "janela" que devem receber a Atividade e, por fim, distribuir as Atividades para os TimeSlotEstudo usando um DistribuidorAtividades. As "janelas de tempo" servem para não atribuir Atividades a um TimeSlotEstudo com dataInicioEstudo posterior a dataLimite da atividade. As "janelas de tempo" são uma série de TimeSlotEstudo com datas entre a dataLimite das Atividades, por exemplo, a primeira janela vai do TimeSlotEstudo com a menor data até o último TimeSlotEstudo com data anterior a dataLimite da Atividade com prazo mais curto.

O propósito da classe AlocacaoAtividade é encapsular dados das atividades que são calculados e usados ao longo do processo realizado em AtribuidorAtividades como por exemplo a quantidade de TimeSlotEstudo em que atividade deve estar presente dado o seu pesoCalculado. Este valor é posteriormente utilizado em DistribuidorAtividades.

A função da classe DistribuidorAtividades, que também faz uso de AlocacaoAtividade, é dada a quantidade de TimeSlotEstudo em que atividade deve estar, em quais e como devem estar distribuídos cada TimeSlotEstudo referente a essa Atividade. Na aplicação está implementado apenas um sistema que garante a alternância entre as Atividades, mas seria possível, por exemplo, implementar uma distribuição em que todo TimeSlotEstudo com a mesma Atividade fosse subsequente em data e hora.

Já em relação a classe CalculadoraPesoAtividades, sua utilidade está principalmente em calcular o peso de todas as atividades presentes na lista recebida. O atributo pesoCalculado de cada Atividade é calculado através da seguinte equação:  $pd * ca / qt$ , onde pd = prioridade da disciplina a qual a atividade está associada, ca = constante relativa ao tipo da atividade e qt = quantidade de TimeSlotEstudo com datas anteriores a dataLimite da atividade ("distância" de dataLimite). Quanto maior o pesoCalculado maior vai ser a quantidade de TimeSlotEstudo com certa atividade.

Importante destacar também, o uso das classes de controle de dados, responsáveis pelo fluxo de dados entre cada uma das telas, para que nenhum dado inserido pelo usuário seja perdido. Classes como CondadosEntreTelas, que armazena dados como o Aluno em questão e outros atributos importantes, ControladorRegistrar atividade, que foi responsável principalmente por tratar da criação das atividades que o usuário selecionou, o ControladorRegistrarSemana, que registra as datas de vigência, e serve como ponte para armazenamento dos Timeslots, e o ControladorRegistrarTimeSLot, que faz a maior parte do trabalho para o armazenamento dos timeslots.

## 1.4 Teste

Para testar as classes utilizamos uma série de testes unitários, entre eles testes de criação dos objetos das classes, de validação de métodos booleanos, de exceções, de fluxo de dados e etc. Na classe Disciplina empregamos os casos de teste em que uma é criada, seus dados são alterados e uma atividade é adicionada a ela. Para testar a classe ConfiguracaoAgenda, utilizou-se de testes referentes aos

métodos `isDataEntreVigencia`, com os casos em que a data está ou não entre as outras duas e `checaVigencia`, com o caso em que a data final precede a data inicial, além da exceção quando um DiaSemana é inserido em um dia diferente do próprio. Já em relação à classe `TimeSlotEstudo`, testou-se os métodos `atividadeValida`, com o caso de checar uma atividade existente e `confiltaComImpedimento`, com os casos em que o impedimento em questão ocorre ou não na mesma hora e dia do `TimeSlotEstudo`.

Em relação à classe `DiaSemana`, testou-se os eventos de criar uma nova e de adicionar dois horários distintos nela. Quanto à classe `Impedimento`, foi testada a criação de um, a alteração fazendo uso de seus setters e ambos os casos onde existe e não uma data e horário válido para conflitar com o impedimento. Já para as atividades como `Provas`, `Trabalhos` e `Exercício`, como todas eram subclasses de `Atividade`, e cada uma tinha apenas um método próprio, com exceção do próprio método de cada classe, todos os outros foram testados usando `Prova` como base. Os métodos próprios de `Trabalho` e `Exercício` foram devidamente testados em seus próprios testes, sendo a maioria dos testes relacionados à validação de entrada de dados, testando datas válidas, nomes válidos, prioridades que não fossem mirabolantes, etc..

Como grupo, acreditamos que o uso de testes auxiliou no desenvolvimento do projeto desde o início, ao demonstrar falhas e incoerências entre as implementações das classes de cada integrante. Além disso, os testes permitiram que o fluxo de dados necessário para a criação da agenda fosse devidamente verificado e que a ocorrência de exceções ao longo de toda a aplicação estavam de acordo com o esperado. Dessa forma, empregar testes não só facilitou o desenvolvimento do aplicativo, como também, muito possivelmente, nos poupará uma quantia considerável de tempo que sem eles teria sido gasto procurando e checando a existência de inúmeros bugs em toda a extensão do código.

## 1.5 Executável

Inicialmente, o usuário abrirá o executável por meio do arquivo `.jar`, encontrando-se na primeira tela da aplicação (Imagen 1), devendo clicar no botão “Gerar Agenda” para prosseguir. Depois, ele observará a segunda tela (Imagen 2), onde deverá inserir uma data inicial e final de vigência para sua agenda de estudos, bem como impedimentos/compromissos em dias e horários específicos que não estarão disponíveis para estudo. Fora isso, é necessário clicar em sete botões distintos para abrir a terceira tela (Imagen 3) e nela preencher com horários livres para estudo referentes a cada dia da semana, considerando uma semana padrão para o usuário. Uma vez concluídos estes processos, retorna-se a segunda tela (através do botão “Salvar Time Slots”) e nela clica-se no botão “Próxima tela” para avançar para a quarta tela (Imagen 4), na qual devem ser inseridas as disciplinas cursadas, suas prioridades (de forma numérica) e atividades referentes a cada uma destas, estabelecendo suas datas de entrega, disciplina associada, se são provas, trabalhos ou exercícios e seus nomes. Fora isso, é possível retornar para a tela

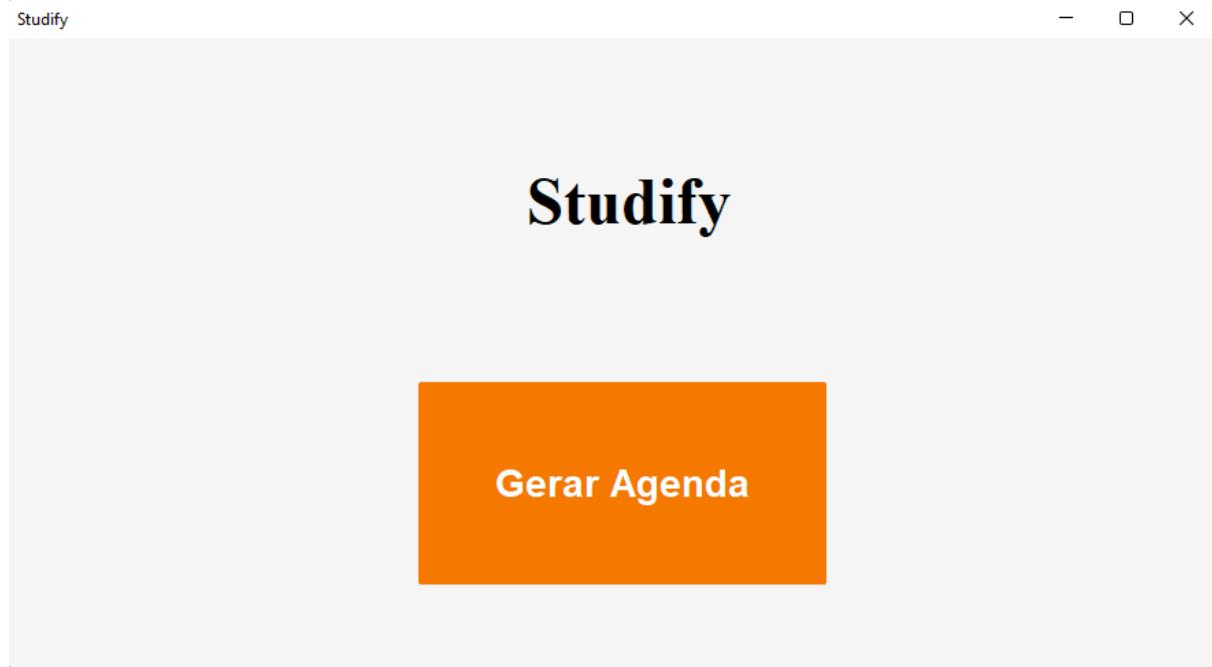
anterior por meio do botão “Retornar” caso deseje-se modificar algum dado e, finalmente, criar sua própria agenda de estudos ao clicar no botão “Concluir Registro”.

Uma vez originada uma agenda de estudos, o usuário observará a quinta tela (Imagen 5), onde poderá verificar sua agenda para cada dia dentro do horário de vigência estipulado.

No geral, a ideia original da interface sofreu algumas mudanças, sendo diferente em certos aspectos, como o botão de adicionar compromisso na segunda tela e a tela de exibição da agenda, onde optamos por unificar a tela de verificar os estudos no dia, com a de escolher uma data específica.

(manual de uso com imagens, explicações gerais, diferenças em relação à interface da etapa 1)

**adicionar mais imagens**



(Imagen 1 - Tela inicial da aplicação)

Studify

Data Início Vigência (DD/MM/AAAA):

Data Fim Vigência (DD/MM/AAAA):

Impedimentos (DD/MM/AAAA HH:MM):  Adicionar Impedimento

Segunda-feira Terça-feira Quarta-feira Quinta-feira

Sexta-feira Sábado Domingo

Próxima Tela

(Imagen 2 - Tela de registro de vigência)

Studify

Selecione os time slots disponíveis para estudo neste dia (considere uma semana padrão):

<input type="checkbox"/> 0:00	<input type="checkbox"/> 6:00	<input type="checkbox"/> 12:00	<input type="checkbox"/> 18:00
<input type="checkbox"/> 0:30	<input checked="" type="checkbox"/> 6:30	<input type="checkbox"/> 12:30	<input type="checkbox"/> 18:30
<input type="checkbox"/> 1:00	<input checked="" type="checkbox"/> 7:00	<input type="checkbox"/> 13:00	<input type="checkbox"/> 19:00
<input type="checkbox"/> 1:30	<input type="checkbox"/> 7:30	<input type="checkbox"/> 13:30	<input type="checkbox"/> 19:30
<input type="checkbox"/> 2:00	<input type="checkbox"/> 8:00	<input type="checkbox"/> 14:00	<input type="checkbox"/> 20:00
<input type="checkbox"/> 2:30	<input type="checkbox"/> 8:30	<input type="checkbox"/> 14:30	<input type="checkbox"/> 20:30
<input type="checkbox"/> 3:00	<input type="checkbox"/> 9:00	<input type="checkbox"/> 15:00	<input type="checkbox"/> 21:00
<input type="checkbox"/> 3:30	<input type="checkbox"/> 9:30	<input type="checkbox"/> 15:30	<input type="checkbox"/> 21:30
<input type="checkbox"/> 4:00	<input type="checkbox"/> 10:00	<input type="checkbox"/> 16:00	<input type="checkbox"/> 22:00
<input type="checkbox"/> 4:30	<input type="checkbox"/> 10:30	<input type="checkbox"/> 16:30	<input type="checkbox"/> 22:30
<input type="checkbox"/> 5:00	<input type="checkbox"/> 11:00	<input type="checkbox"/> 17:00	<input type="checkbox"/> 23:00
<input type="checkbox"/> 5:30	<input type="checkbox"/> 11:30	<input type="checkbox"/> 17:30	<input type="checkbox"/> 23:30

Salvar Time Slots

(Imagen 3 - Tela de escolher os horários de estudo)

Disciplina (nome e prioridade):

Data da Atividade (DD/MM/AAAA):

Nome da Atividade:

Adicionar Disciplina

Tipo de Atividade:

Disciplina da Atividade:

(Imagen 4 - Tela de registro de atividades)

Horário	Atividade
00:00	Livre
00:30	Livre
01:00	Livre
01:30	Livre
02:00	Livre
02:30	Livre
03:00	Livre
03:30	Livre
04:00	Livre
04:30	Livre
05:00	Livre
05:30	Livre
06:00	Livre
2025-12-16T06:30	bbb
2025-12-16T07:00	bbb
07:30	Livre
08:00	Livre
08:30	Livre
09:00	Livre
09:30	Livre

(Imagen 5-Tela de visualização da agenda)

## 1.6 Documentação e Reflexão sobre o Reuso

Para realizar o logging da aplicação, utilizamos a biblioteca Tinylog por sua simplicidade e eficiência em transmitir mensagens, sendo que para configurá-la bastou baixar os arquivos .jar presentes no site da biblioteca e inseri-los na pasta “lib” do projeto, não havendo qualquer dificuldade na integração. Sendo os principais itens reutilizados a biblioteca e sua classe Logger, alguns dos benefícios percebidos foram a clareza propiciada para os usuários sobre os eventos que ocorrem na interface, como por exemplo ao avisar por meio de mensagens erros de formatação

na escrita de dados, e uma melhora na manutenção do código ao servir como meio de debugar certos trechos, imprimindo valores de variáveis.

O nível adequado de log foi encontrado para cada mensagem ao verificar-se se determinado evento do sistema consistia em um erro, uma operação padrão ou parte de alguma detecção de bugs. Ainda que muitas informações sejam logadas, definiu-se que dados sensíveis e específicos do programa, como cálculos de peso para cada atividade, não deveriam ser logados. Outro aspecto percebido foi o de que o uso de logging auxiliou tanto na manutenção quanto em testes do sistema ao disponibilizar constantemente mensagens sobre os acontecimentos dentro da aplicação, sendo possível detectar algum fluxo indevido de ações ou valores inconsistentes de variáveis.