Projeto Final
Programação Orientada a Objetos

Uma simples descrição da documentação

Alunos:

Dayllon Vinícius Xavier Lemos - 201904200

Luan Cesar Dutra Carvalho - 201904231

1. Detalhando o problema a ser solucionado

A maratona de programação, promovida pela ICPC, é uma das maiores competições de programação no Brasil. Durante a competição, os times estão permitidos a levarem material impresso para consulta. Vários participantes, sabendo dessa regalia, selecionam, durante seus estudos, diversos tópicos, explicações, códigos, entre outros itens para desenvolverem um caderno de pesquisa, denominado comumente por "*Lib*".

Conforme os dias vão se aproximando da competição, os participantes que desejam levar sua *Lib* pessoal para a competição, devem construir um documento PDF (ou algum outro formato) com todo o conteúdo selecionado durante os treinos para ser impresso. Essa última ação é extremamente cansativa e repetitiva, de forma que os participantes que deixaram para montar sua *Lib* nos últimos dias sentem-se desmotivados, e acabam tendo de utilizar o pouco tempo disponível até a competição para a construção desse material.

Dessa forma, o *software*, denominado FastLib, apresentado neste trabalho, vem com o objetivo de automatizar a geração do documento *Lib*. O FastLib gera o código LaTeX do documento, utilizando as informações cadastradas em sua base de dados pelo estudante. Sendo assim, os participantes podem utilizar do FastLib para cadastrar os temas selecionados durante o treinamento, revisá-los no próprio *software* (uma vez que já estejam cadastrados) e com apenas um clique gerar o código LaTeX do PDF, necessitando apenas compilar esse código posteriormente em alguma outra ferramenta.

2. Modelagem do problema

O objetivo final do *software* é gerar o código Latex do PDF, dessa forma optou-se por desenvolver uma classe para cada componente do PDF. Inicialmente, especificou-se que os componentes essenciais de uma Lib genérica são: texto, código, título, tópico e capítulo.

Sendo assim, foi desenvolvido uma classe para cada um dos componentes citados acima. Todas essas classes foram construídas de forma a abstrair cada componente do PDF da melhor forma possível, sendo que ações como converter o componente em código Latex e converter o componente na tela do *software* foram tratadas de forma independente, por todas as classes, e modeladas como os métodos dessas classes. Essa abstração preza para que, quando deseja-se gerar todo o código Latex ou renderizar toda a Lib na tela do *software*, sejam gerados/renderizados cada componente de forma separada, e posteriormente agrupados, como o resultado final da ação. Portanto, como essas ações acontecem para cada classe, foi escolhido desenvolver uma classe abstrata denominada "Componente", sendo que todas as outras classes herdam dela, possibilitando assim um maior controle dos componentes, tanto de forma individual quanto em grupo.

O componente texto representa um ou mais parágrafos de texto (sequência de caracteres) de um PDF. Com isso, foi desenvolvido a classe "Texto", que possui o atributo "conteudo", que armazena todo o conteúdo dos parágrafos.

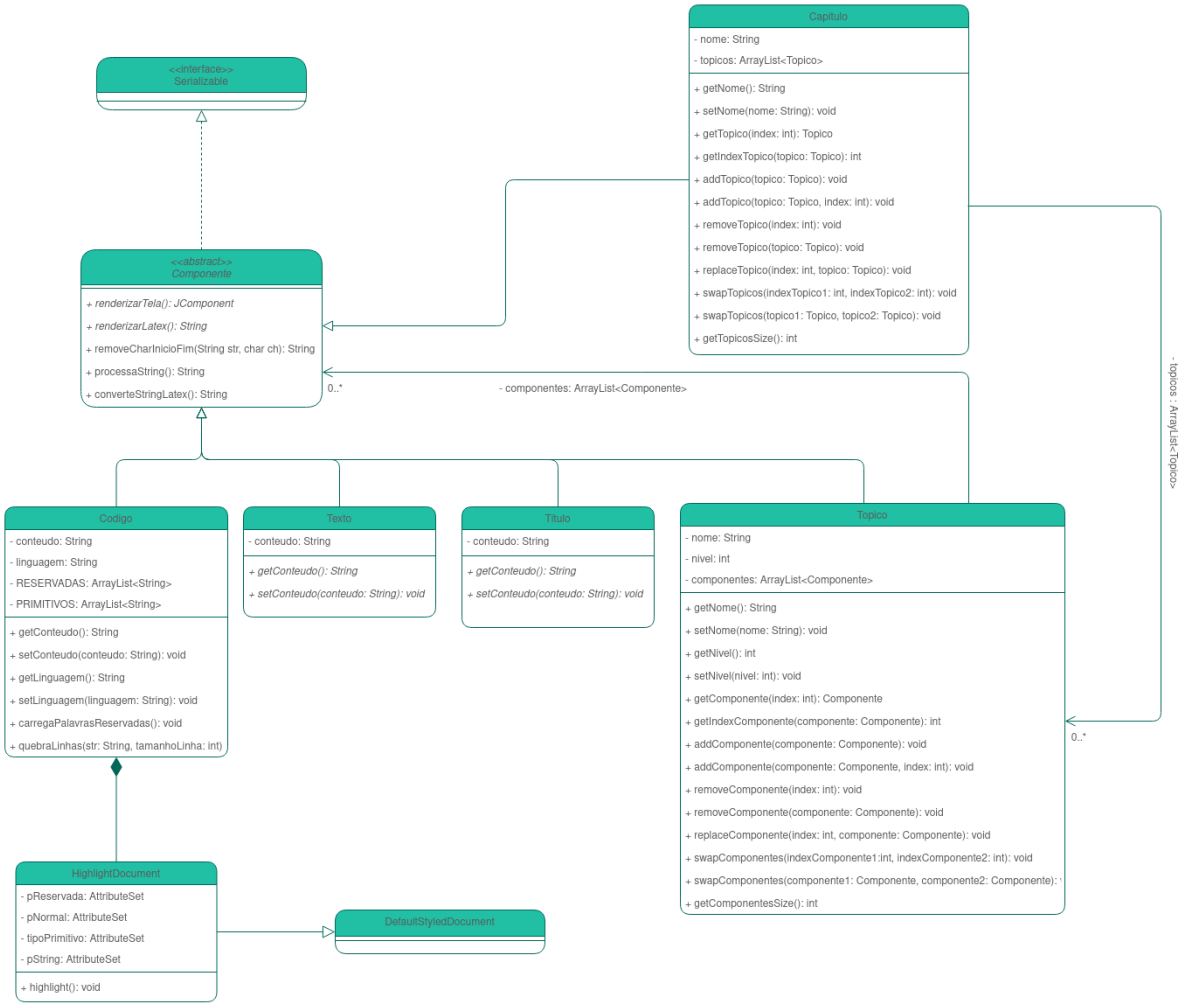
O componente código representa um código em alguma linguagem de programação. Esse código foi informado pelo usuário, e pode consistir tanto da implementação de um algoritmo, da resolução de algum problema, e entre outros códigos julgados necessários pelo usuário. Com isso, foi desenvolvido a classe "Código", que possui os atributos "conteudo" e "linguagem", que armazenam respectivamente o código informado e a linguagem de programação deste código. Para a versão atual deste trabalho, o atributo "linguagem" será definido sempre como "C++". Esse atributo, embora ocioso, não foi removido para facilitar a alteração de futuras versões do *software* que venham a permitir outras linguagens de programação.

O componente título representa um título, uma frase destacada que tem um nível semântico maior que um texto, porém não aparece no sumário. Com isso, foi desenvolvido a classe “Título”, que possui o atributo “conteúdo”, o qual armazena a frase em destaque representada.

O componente tópico refere-se a uma área específica do PDF que retrata um assunto particular. Foi desenvolvido a classe “Topico” que representa esse componente. Essa classe possui o atributo “nome” que armazena o nome do tópico, o atributo “componentes” que é uma lista de componentes, podendo ser eles objetos das classes Texto, Código e Título (todos herdam de “Componente”, portanto podemos armazená-los em uma mesma lista de “Componentes” - propriedade de herança), estes objetos compõem o “Topico”. Um objeto do tipo “Topico” também pode estar na lista “componentes” de um outro objeto do tipo “Topico”, criando assim, semanticamente, um sub-tópico. Portanto, para diferenciar um tópico de um sub-tópico, a classe “Tópico” possui o atributo “nivel”, recebendo valor “0” se for um tópico externo, e recebendo “1” caso contrário.

O componente capítulo refere-se uma coleção de tópicos, e semanticamente é uma área do PDF destinada a um campo teórico específico (mais abrangente do que o tópico). Com isso, foi desenvolvido a classe “Capitulo” que representa um capítulo. Essa classe possui o atributo “nome”, que armazena o nome do capítulo, e uma lista de tópicos, atributo “topicos”, que armazena todos os objetos da classe “Topico” que fazem parte de um capítulo. O capítulo é a abstração mais externa da modelagem, dessa forma o PDF consiste de vários capítulos, que consistem de vários tópicos, que consistem de vários componentes, criando assim uma hierarquia.

a. Diagrama de Classes



A imagem acima apresenta o diagrama de classes do software modelado. O significado de cada classe juntamente com a explicação de cada atributo já foi informado nos parágrafos anteriores. Encontra-se no diagrama as relações entre as classes.

A classe componente “Componente” recebeu um “implements” da interface “Serializable”, do pacote “java.io” do Java. Isso foi necessário para que fosse possível salvar os objetos das classes em arquivos.

A classe código possui uma classe interna denominada “HighlightDocument” que herda da classe “DefaultStyledDocument”, do pacote “javax.swing.text”. Essa classe interna foi desenvolvida para que se pudesse realizar o “Syntax Highlight” (coloração das palavras-chave, tipos primitivos e funções de um código) no código informado pelo usuário na própria tela do Fastlib.

As implementações das classes modeladas encontram-se no repositório “FastLib” (precisamente na pasta “FastLib/src/br/com/fastlib/models/”. Além disso, o Javadoc do projeto também pode fornecer informações a respeito das classes e de cada uma dos métodos dessas classes.

b. Diagrama de casos de uso

A maneira que o usuário interage com a interface do software foi pensada de modo a ser simples, intuitiva e direta. As interações são feitas, basicamente, de três formas: Clicando em um botão, clicando em um elemento da JTree de modo a selecioná-lo ou preenchendo um campo de texto. Além disso, não é necessário que haja um agente intermediário que autentique ou viabilize as ações do usuário.

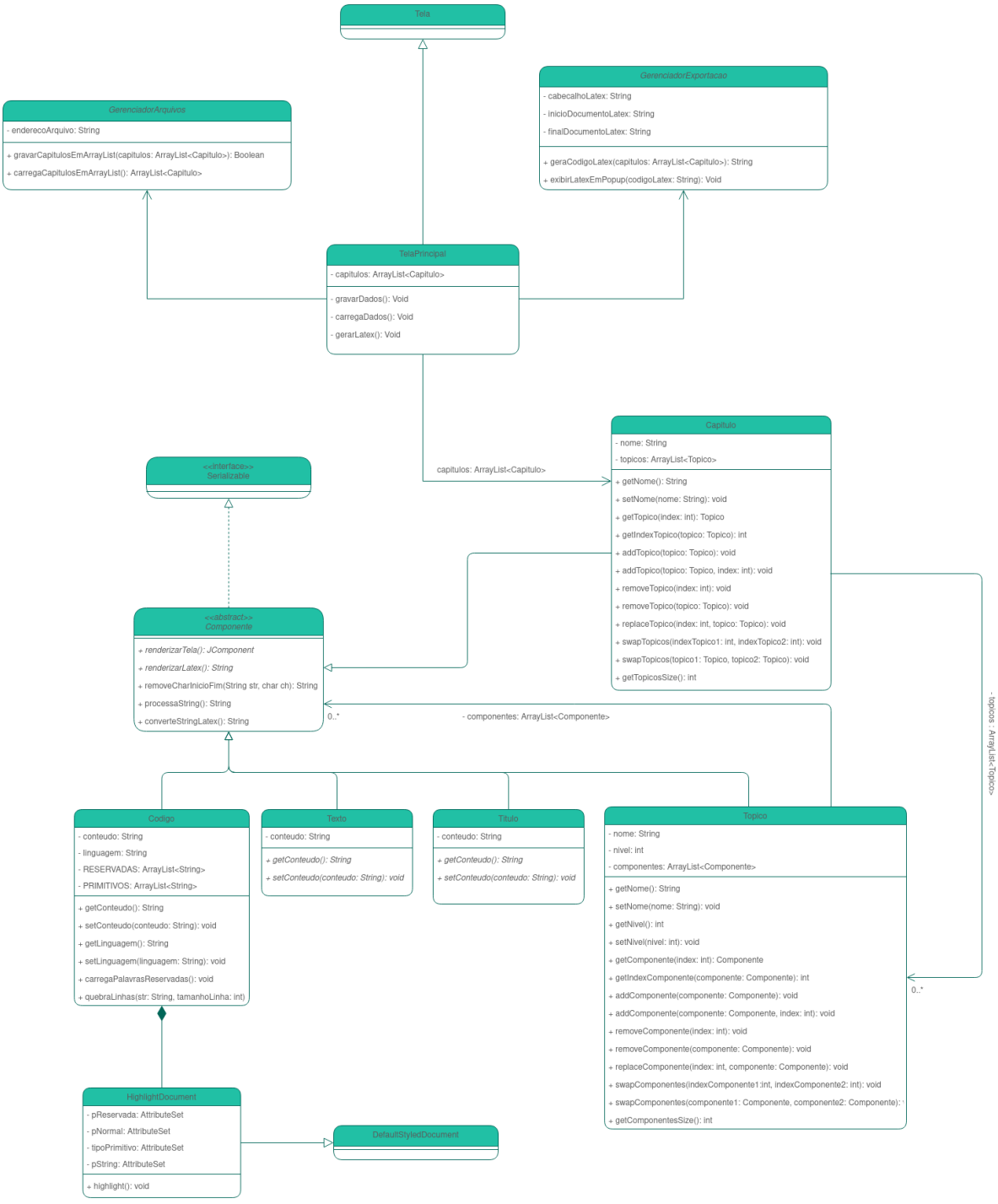
Como mostra o diagrama acima, o usuário pode realizar seis ações, são essas: Inserir, Editar ou Excluir um componente, Selecionar um elemento da JTree, fazer uma

pesquisa nos componentes e Gerar o código LaTeX da lib. Cada uma dessas ações, porém, exige do usuário certos requisitos. Para fazer uma pesquisa nos componentes, ou para inserir um novo componente, é necessário que se preencha as respectivas caixas de texto, caso contrário a pesquisa retornará o documento inteiro, e a inserção retornará uma exceção do tipo “CaixaTextoVaziaException”. Se o usuário quiser editar ou excluir um componente, deve selecioná-lo na JTree. Além disso, caso o usuário queira fazer uma inserção de um componente que não seja um capítulo, deve selecionar um componente na JTree de modo que o seu componente seja inserido no local correto. No caso da edição de um componente, pressupõe-se que o usuário irá alterar o conteúdo da caixa de texto, porém, se ele não o fizer, não ocorrerá nenhum erro.

Dessa forma, o caminho que o usuário percorre para a construção de sua lib, consiste concretamente de inserções, edições e remoções. As outras ações servem como suporte para essa construção. Ao final, o usuário irá executar a ação de gerar o código LaTeX da Lib e o software terá cumprido sua função.

3. Modelagem e interface gráfica

A integração das classes modeladas com as telas pode ser representada pelo seguinte diagrama (esse diagramação não é fiel à implementação, apenas tenta explicitar a relação entre a interface gráfica e a modelagem desenvolvida) :



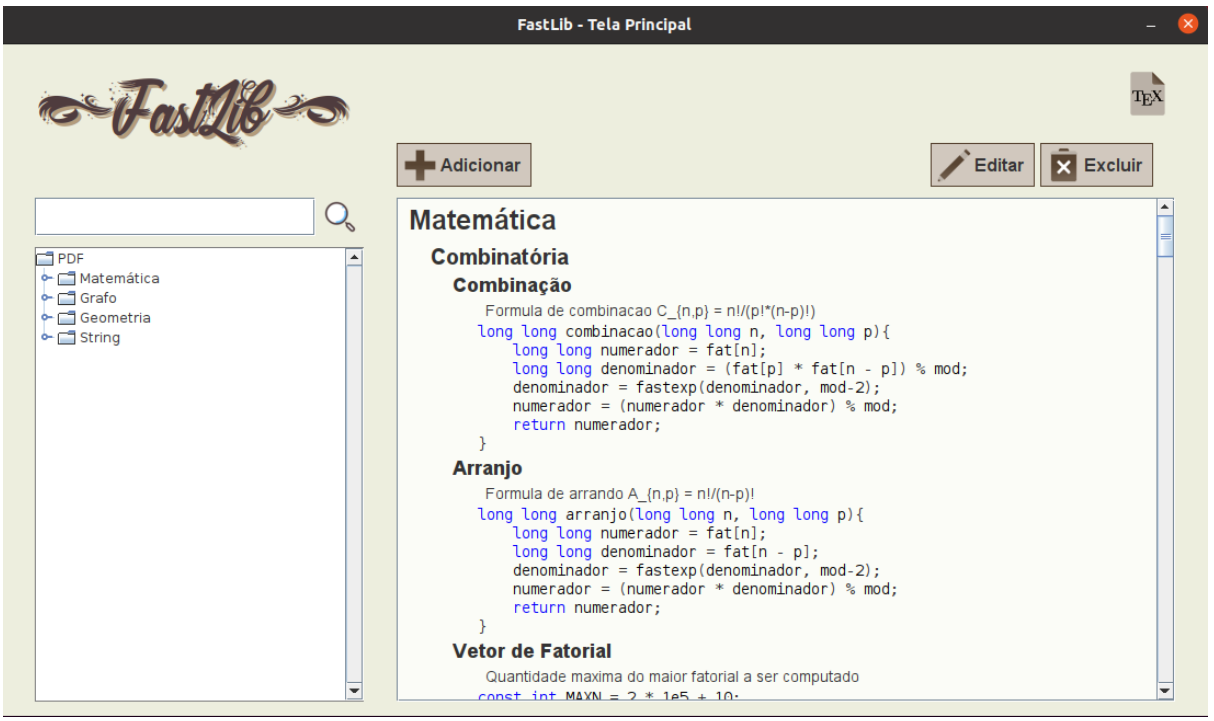
A classe “TelaPrincipal”, a qual não foi totalmente descrita neste diagrama, herda da classe abstrata “Tela”, e possui uma lista de “Capitulos”, que são os dados do *software*, fornecido pelo usuário. Essa lista é carregada quando o *software* é iniciado por meio do objeto “gerenciadorArquivo” da classe “GerenciadorArquivo”, e a lista de “capitulos” é salva em arquivo por esse mesmo objeto ao encerrar o software.

O atributo “gerenciadorExportação” possui a referência de um objeto da classe “GerenciadorExportação”. Essa última classe exporta o código Latex do PDF e o apresenta em um Popup.

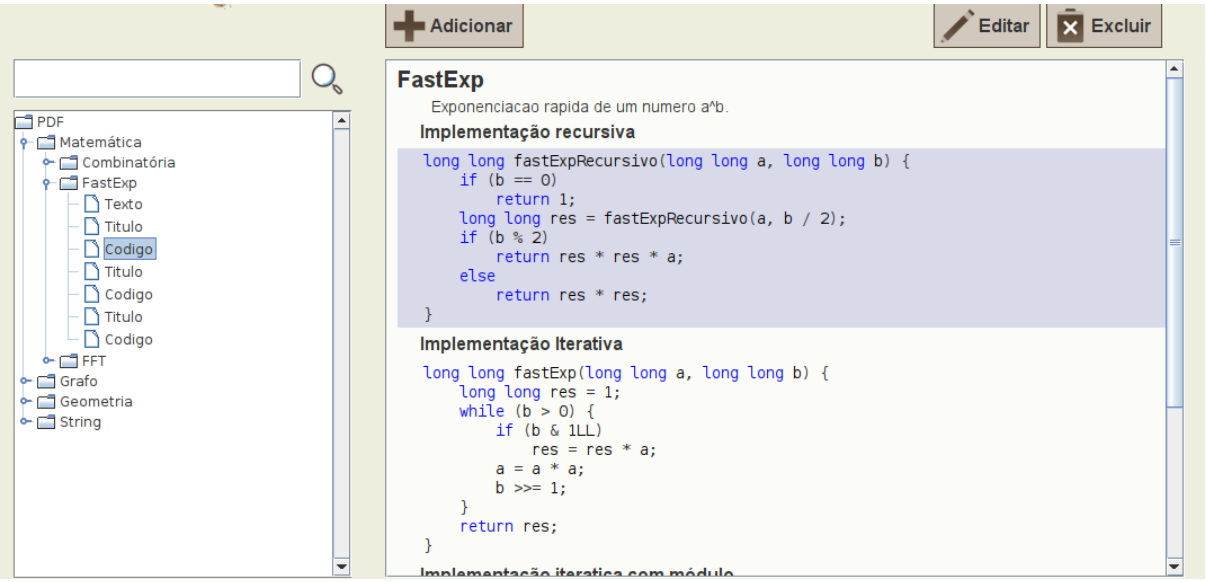
4. Apresentando a interface

Do ponto de vista do usuário, a interface da FastLib é limpa e intuitiva, todos os botões têm sua função clara e evidente através de descrições ou símbolos. Além disso, os campos reservados para a visualização da Lib e da hierarquia dos componentes são vastos e compreendem a maior parte da tela, facilitando ao usuário o gerenciamento de sua Lib.

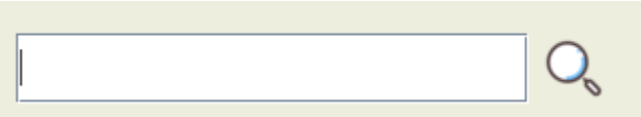
Todos os componentes são de fácil diferenciação, seja pelo tamanho da fonte, indentação na tela de preview, ou mesmo pela hierarquia vista na JTree.



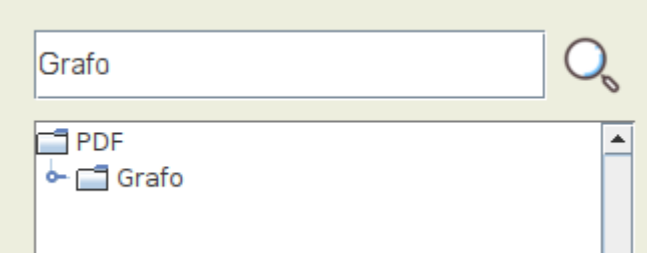
Apesar disso, para facilitar ainda mais a manipulação dos componentes, ao clicar em um nó qualquer da JTree à esquerda, o preview da Lib será reconstruído de modo a focar o componente selecionado, como mostra a imagem abaixo.



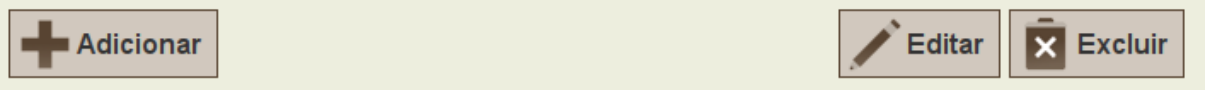
Mesmo com as facilidades apresentadas acima, conforme a Lib aumenta, se torna mais difícil encontrar partes específicas desta. Por esse motivo foi implementado na FastLib o seguinte mecanismo de pesquisa.



Ao escrever algo na caixa de texto e clicar no ícone da lupa, a pesquisa é feita e restarão na JTree apenas os componentes cujo conteúdo corresponde ao que foi pesquisado, como mostra a imagem a seguir.



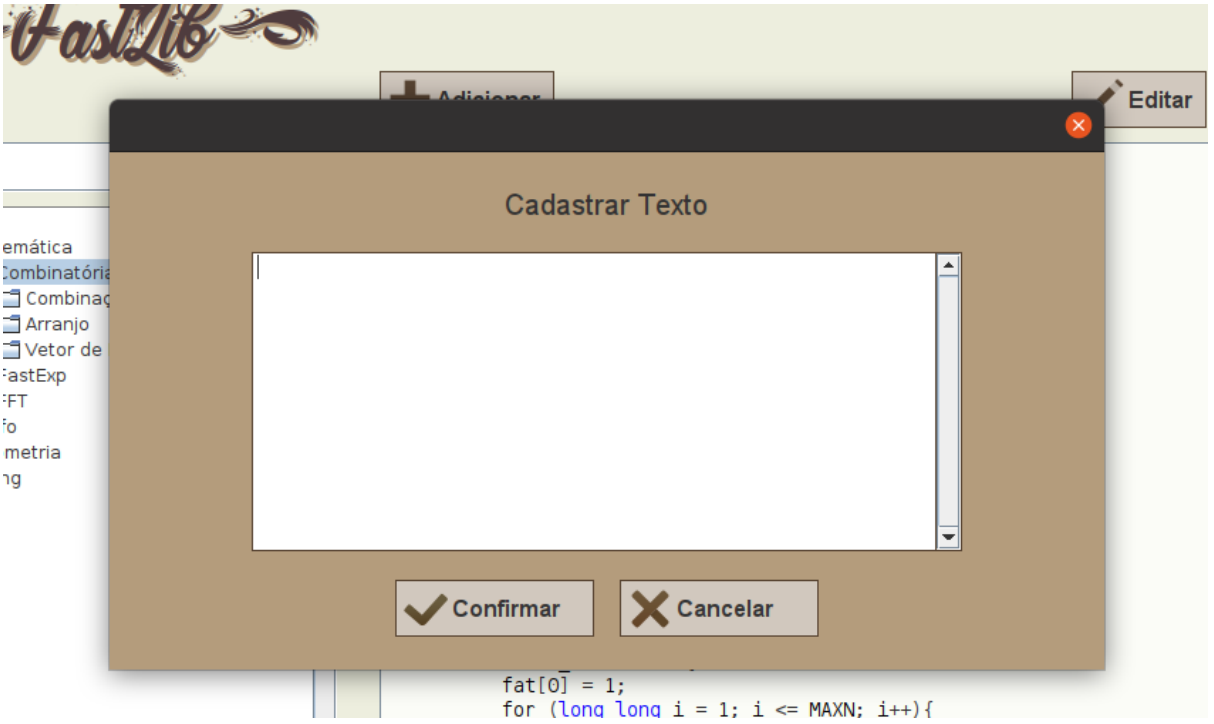
Apesar das facilidades na visualização e na busca dos componentes, antes disso a Lib deve ser criada de acordo com a vontade do usuário, para isso existem os botões Adicionar, Editar e Excluir mostrados na imagem.



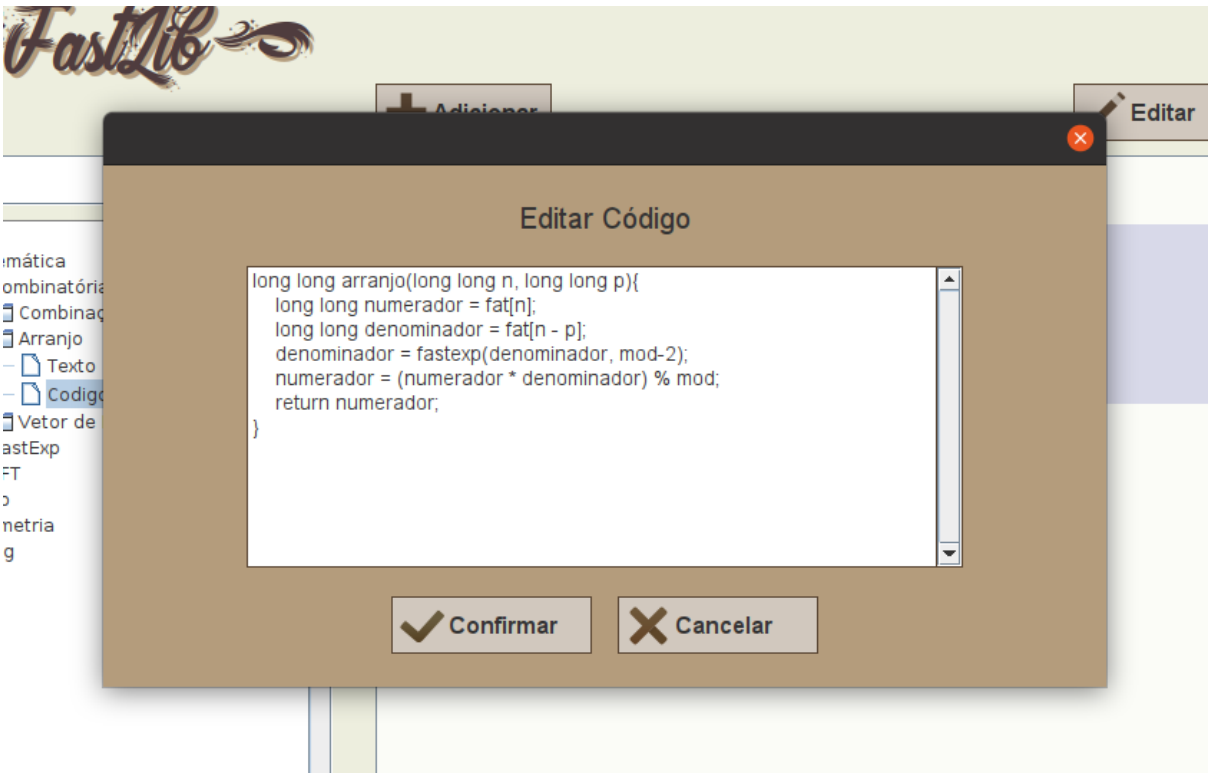
Ao clicar no botão adicionar, se o componente selecionado for o PDF não houver nenhum componente selecionado, só haverá a opção de se adicionar um componente do tipo capítulo. Se o componente selecionado for um capítulo, poder-se-á adicionar um tópico. Se o componente selecionado for um tópico que é filho direto de um capítulo, poder-se-á escolher entre adicionar um Título, um Texto, um Código, ou um subtópico. Por fim, se o componente selecionado for um subtópico, um Título, um texto ou um Código, só haverá a opção de se adicionar um Título, um Texto, ou um Código.

Ao escolher o que se quer adicionar, deve-se preencher a informação no Popup que irá aparecer. No caso de adicionar um tópico, será a seguinte.

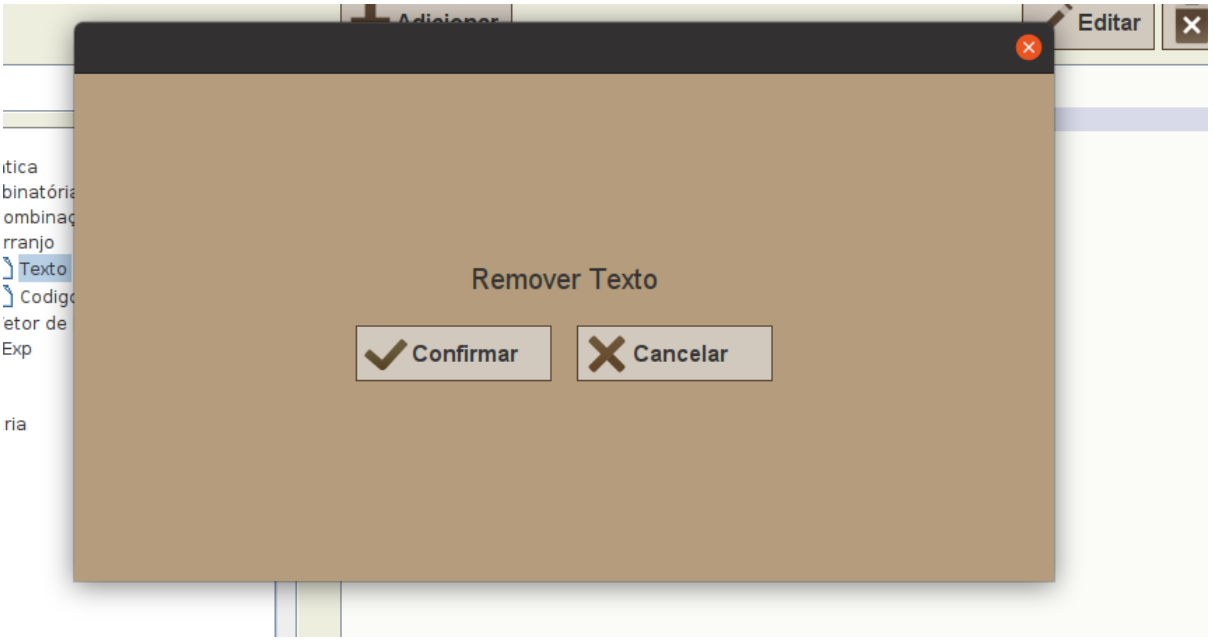
Para completar a adição, basta clicar no botão Confirmar. A popup varia de acordo com o que se quer adicionar, por exemplo, para um texto será a seguinte:



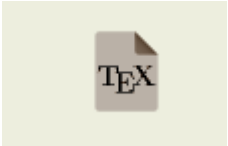
Após ter adicionado um componente, há a opção de editá-lo, basta selecionar o componente na JTree e clicar no botão Editar, e aparecerá uma Popup de edição. Por exemplo, para a edição de um código aparecerá a seguinte Popup.



Para completar a edição, basta clicar no botão confirmar. Por fim, para remover um componente basta selecionar o componente a ser removido na JTree e clicar no botão Remover. Aparecerá uma Popup de confirmação. Se quiser mesmo remover o componente, basta clicar em Confirmar, caso contrário clique em Cancelar. Por exemplo, ao tentar remover um texto, a popup será a seguinte:

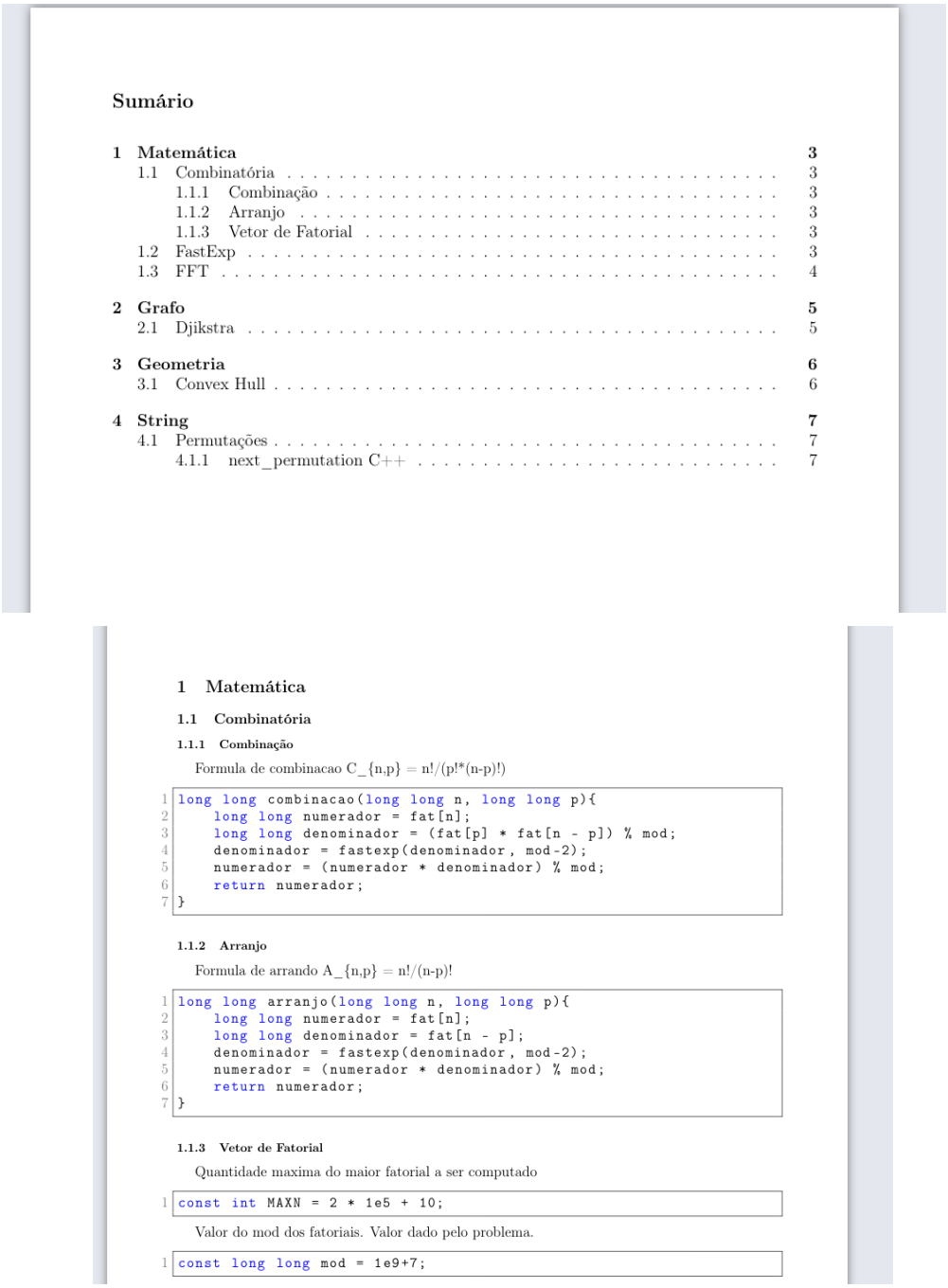


O objetivo final da FastLib é fornecer uma maneira fácil e rápida de montar uma Lib, dessa forma, uma ação imprescindível é gerar a Lib de uma forma que se possa facilmente transformá-la em um material físico. Para isso existe o botão de gerar o código LaTeX da Lib:



Ao clicar no botão acima, se abrirá uma tela com o código LaTeX da Lib, o qual poderá ser renderizado para PDF através de meios como o programa PDFLatex ou mesmo como o Overleaf. Um exemplo de código LaTeX gerado é o seguinte:

Após renderizado teremos um PDF como o seguinte



Sumário

1 Matemática

1.1 Combinatória

1.1.1 Combinação

Formula de combinacao C_{n,p} = n!/(p!*(n-p)!)

```
long long combinacao(long long n, long long p){
    long long numerador = fat[n];
    long long denominador = (fat[p] * fat[n - p]) % mod;
    denominador = fastexp(denominador, mod-2);
    numerador = (numerador * denominador) % mod;
    return numerador;
}
```

1.1.2 Arranjo

Formula de arrando A_{n,p} = n!/(n-p)!

```
long long arranjo(long long n, long long p){
    long long numerador = fat[n];
    long long denominador = fat[n - p];
    denominador = fastexp(denominador, mod-2);
    numerador = (numerador * denominador) % mod;
    return numerador;
}
```

1.1.3 Vetor de Fatorial

Quantidade maxima do maior fatorial a ser computado

```
const int MAXN = 2 * 1e5 + 10;
```

Valor do mod dos fatoriais. Valor dado pelo problema.

```
const long long mod = 1e9+7;
```

5. Apresentação da classe de teste

Foi desenvolvida uma classe de testes com o JUnit, denominada “ComponenteMetodosTest”, encontrada em anexo (caminho: “FastLib/src/br/com/fastlib/testes/ComponenteMetodosTest.java”) que realiza testes nos métodos “removeCharInicioFim”, “processaString” e “converteStringLatex” da classe “Componente”. Todos os testes foram executados com sucesso.

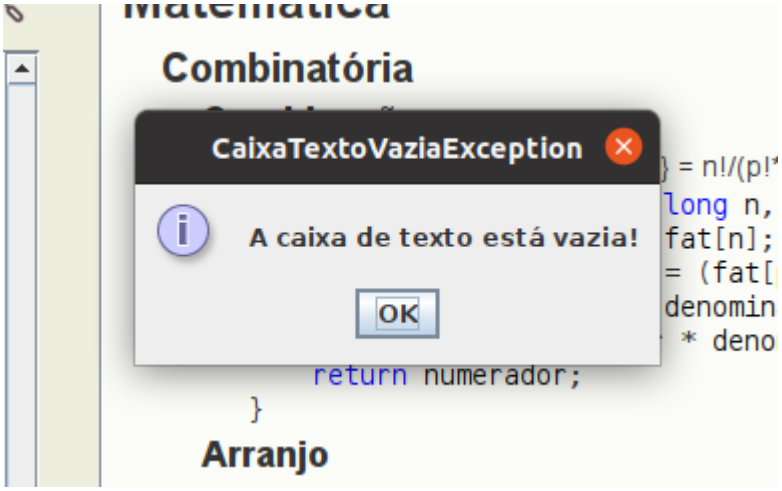
Como a classe “Componente” é uma classe abstrata, não é possível utilizar uma de suas instâncias nos testes, por isso foi instanciado um objeto da classe “Titulo”, que é uma subclasse de “Componente” e que não sobrescreve os métodos testados. Portanto, foi utilizado a instância da classe “Titulo” durante os testes dos métodos da classe “Componente”.

6. Apresentação da Exceção interna

Um erro comum durante o uso da FastLib é a tentativa de inserir um novo componente deixando a caixa de texto exigida vazia. Para tratar esse erro foi criada uma nova Exceção denominada “CaixaTextoVaziaException” a qual herda da classe Exception do Java.

A Exceção é tratada cancelando a inserção do componente e avisando ao usuário que deve sempre preencher os campos de textos durante a inserção.

Por exemplo, se clicarmos em confirmar com a caixa de texto vazia durante o cadastro de um capítulo, ocorrerá o seguinte:



Ou seja, quando a exceção criada é disparada, imediatamente inicia-se uma rotina de tratamento que cancela a operação de inserção e avisa ao usuário que não deve deixar a caixa de texto vazia durante a inserção.