**UNIVERSIDADE PAULISTA**

Lucas Eduardo Rosa Neves -- RA:G41414-4

Andressa Silva Da Costa -- RA:N8949C-7

**TRABALHO APS**

Estrutura de Dados

Goiânia

2023

Lucas Eduardo Rosa Neves -- RA:G41414-4

Andressa Silva Da Costa -- RA:N8949C-7

**TRABALHO APS**

Estrutura de Dados

Trabalho APS do curso Ciência da computação, noturno, com os orientadores: Nelson e Fernando, pela universidade Paulista.

Goiânia

2023

Sumário

1.0 Objetivo.......................................................................................................................................6

2.0 Introdução...................................................................................................................................7

3.0 Referencial teórico....................................................................................................................10

3.1 Bubble Sort................................................................................................................................10

3.2 Insertion Sort.............................................................................................................................10

3.4 Quick Sort..................................................................................................................................10

3.4 Integração com a Prática..........................................................................................................11

3.5 Referências Gerais....................................................................................................................12

4.0 Desenvolvimento.......................................................................................................................13

4.1 Pacote algorithm.......................................................................................................................13

4.1.1 BubbleSort..............................................................................................................................13

4.1.2 InsertionSort...........................................................................................................................14

4.1.3 QuickySort..............................................................................................................................14

4.2 Pacote backend.........................................................................................................................15

4.2.1 DeleteLogic............................................................................................................................15

4.2.2 InsertionLogic.........................................................................................................................16

4.2.3 UpdateLogic...........................................................................................................................17

4.2.4 Records..................................................................................................................................18

4.3 Pacote database.......................................................................................................................19

4.3.1 Db...........................................................................................................................................19

4.3.2 DbException...........................................................................................................................20

4.4 Pacote main..............................................................................................................................20

4.5 Pacote utils................................................................................................................................21

4.5.1 ConversorBlob........................................................................................................................21

4.5.2 CustomButton.........................................................................................................................22

4.5.3 RoundButton..........................................................................................................................23

4.5.4 UiHelper.................................................................................................................................23

4.6 Pacote ui...................................................................................................................................24

4.6.1 BaseFrame.............................................................................................................................24

4.6.2 HomeScreen..........................................................................................................................25

4.6.3 DeleteScreen..........................................................................................................................26

4.6.4 ImageScreen..........................................................................................................................26

4.6.5 InsertionScreen......................................................................................................................27

4.6.6 Ordering.................................................................................................................................28

4.6.7 PerformanceBubbleSort.........................................................................................................29

4.6.8 PerformanceInsertionSort......................................................................................................30

4.6.9 PerformanceQuickSort...........................................................................................................31

4.6.10 RecordsScreen.....................................................................................................................31

4.6.11 UpdateScreen\_1..................................................................................................................32

4.6.12 UpdateScreen\_2..................................................................................................................32

5.0 Resultado e Discussões............................................................................................................34

5.1 HomeScreen.............................................................................................................................34

5.2 RecordsScreen..........................................................................................................................35

5.3 InsertionScreen.........................................................................................................................36

5.4 UpdateScreen...........................................................................................................................37

5.5 ImageScreen.............................................................................................................................38

5.6 DeleteScreen.............................................................................................................................39

5.7 OrderingScreen.........................................................................................................................41

6.0 Considerações Finais................................................................................................................43

7.0 Referências Bibliográficas.........................................................................................................44

8.0 Código Fonte.............................................................................................................................45

8.1 Pacote algorithm.......................................................................................................................45

8.1.1 BubbleSort..............................................................................................................................45

8.1.2 InsertionSort...........................................................................................................................45

8.1.3 QuickySort..............................................................................................................................46

8.2 Pacote backend.........................................................................................................................48

8.2.1 DeleteLogic............................................................................................................................48

8.2.2 InsertionLogic.........................................................................................................................50

4.2.3 Records..................................................................................................................................54

8.2.4 UpdateLogic...........................................................................................................................57

8.3 Pacote database.......................................................................................................................63

8.3.1 Db...........................................................................................................................................63

8.3.2 DbException...........................................................................................................................66

8.4 Pacote main..............................................................................................................................67

8.4.1 Main........................................................................................................................................67

8.5 Pacote ui...................................................................................................................................67

8.5.1 BaseFrame.............................................................................................................................67

8.5.2 DeleteScreen..........................................................................................................................69

8.5.3 HomeScreen..........................................................................................................................73

8.5.4 ImageScreen..........................................................................................................................77

8.5.5 InsertionScreen......................................................................................................................81

8.5.6 Ordering.................................................................................................................................87

8.5.7 PerformanceBubbleSort.........................................................................................................92

8.5.8 PerformanceInsertionSort......................................................................................................99

8.5.9 PerformanceQuickSort.........................................................................................................106

8.5.10 RecordsScreen...................................................................................................................113

8.5.11 UpdateScreen\_1................................................................................................................120

8.5.12 UpdateScreen\_2................................................................................................................124

8.6 Pacote utils.............................................................................................................................132

8.6.1 ConversorBlob......................................................................................................................132

8.6.2 CustomButton.......................................................................................................................133

8.6.3 RoundButton........................................................................................................................136

8.6.4 UiHelper...............................................................................................................................137

**1.0 Objetivo**

A execução deste projeto teve como foco principal o aprimoramento das habilidades e conhecimentos em programação orientada a objetos, com ênfase na linguagem de programação Java. Para alcançar esse propósito, o projeto explorou diversas técnicas e recursos da linguagem, incorporando o uso de bibliotecas externas e aderindo às boas práticas de programação. O objetivo central era o desenvolvimento de uma aplicação robusta e de fácil manutenção.

Além da ênfase na programação orientada a objetos em Java, o projeto abordou conceitos fundamentais de banco de dados, abrangendo o armazenamento e a manipulação de informações. Também se dedicou à integração de recursos de estrutura de dados, ampliando os conhecimentos e habilidades relacionados aos algoritmos de ordenação.

Outro ponto crucial do projeto foi a busca constante pelo aprimoramento das habilidades de resolução de problemas e pela criação de interfaces completas e intuitivas. Durante o processo de desenvolvimento, enfrentou-se desafios significativos, incluindo a junção do back end com o front end, a manipulação de imagens no banco de dados e a manipulação dos algoritmos de ordenação para lista de vetores e não apenas vetores.

Por fim, o projeto incluiu como objetivo essencial o aprimoramento das habilidades de documentação e comunicação. Isso se concretizou por meio da elaboração deste texto introdutório, que descreve as principais características e funcionalidades da aplicação desenvolvida. Essas habilidades são vitais para uma comunicação eficaz e o compartilhamento de conhecimentos e soluções no ambiente dinâmico do desenvolvimento de software.

**2.0 Introdução**

O trabalho desenvolvido em Java constitui uma ampla abordagem no campo das estruturas de dados, com ênfase especial nos algoritmos de ordenação. O cerne deste projeto reside na análise detalhada do desempenho de três algoritmos específicos de ordenação: Bubble Sort, Insertion Sort e Quick Sort. Essa escolha não se restringe à mera implementação didática, mas visa avaliar, de forma prática, como cada um se comporta em termos de eficiência durante a execução.

O programa também incorpora a manipulação de dados em larga escala, utilizando uma base robusta com mais de 15 mil registros relacionados ao bioma cerrado. Essa base, originária de fontes confiáveis do IBGE, proporciona um contexto rico e relevante para a aplicação dos algoritmos de ordenação. A tabela "cerrado" serve como uma representação digital das características mapeadas por satélite desse bioma. Nesse sentido, a inovação deste projeto não está apenas na implementação de algoritmos de ordenação, mas também na escolha estratégica de trabalhar com o tipo BLOB no MySQL. Essa decisão proporcionou um meio eficaz de armazenar e recuperar dados complexos, como representações visuais do bioma. A presença da coluna "imagem" na tabela não só enriquece a experiência visual na interface gráfica do usuário (GUI) como também destaca a capacidade do programa de lidar com diferentes tipos de dados.

Em consonância com os princípios explorados na disciplina acadêmica (ALPOO), a concepção da interface gráfica transcende a mera exposição passiva de resultados, configurando-se como um ambiente interativo destinado a visualizar de maneira dinâmica dados e resultados oriundos dos algoritmos implementados. A inclusão de mais de 10 telas gráficas não representa apenas uma demonstração técnica, mas sim uma manifestação da habilidade em integrar princípios teóricos de forma prática e versátil.

Um outro desígnio essencial está na convergência entre o conhecimento técnico e a representação visual. A interface, como mencionado, não almeja apenas otimizar a apresentação dos resultados dos algoritmos, mas também propiciar uma interação dinâmica e acessível. Cada componente gráfico e operação executada na interface não se restringe apenas à funcionalidade, mas busca aplicar de maneira concreta os conceitos absorvidos na disciplina (ALPOO). Dessa forma, a aplicação transcende a simples exposição de resultados, proporcionando uma experiência intuitiva para os usuários. A inclusão de elementos gráficos do Java, selecionados e organizados com critério, não apenas aprimora a estética da aplicação, mas também intensifica a compreensão do usuário em relação aos dados e funcionalidades do programa.

Nesse contexto, a escolha deliberada de incorporar elementos gráficos no desenvolvimento, como botões, caixas de texto e rótulos, não atende apenas à estética visual, mas também à ergonomia e à usabilidade. Cada elemento gráfico foi estrategicamente posicionado e projetado para otimizar a experiência do usuário, tornando a interação com o programa mais intuitiva e eficiente. A manipulação desses elementos gráficos é uma expressão prática dos conceitos aprendidos em ALPOO, destacando a importância da análise cuidadosa e da abordagem orientada a objetos na construção de interfaces eficazes.

Para além da manipulação de dados e algoritmos, o programa oferece uma área dedicada para operações CRUD (Create, Read, Update, Delete). Essa inclusão amplia a funcionalidade da aplicação, permitindo a interação direta com o banco de dados e realçando a aplicação prática dos conceitos de programação visual em situações do mundo real.

Ainda no âmbito visual, a utilização do componente gráfico JTable para exibição da base de dados revela uma escolha perspicaz. Esse componente proporciona uma representação tabular dos dados contidos no MySQL, oferecendo uma visualização clara e estruturada. A integração do JTable não apenas simplifica a apresentação dos registros do cerrado, mas também concede aos usuários a capacidade de explorar, classificar e filtrar os dados de maneira eficiente.

Ao longo deste trabalho, detalha-se como a implementação dos elementos gráficos foi realizada, destacando a lógica por trás de cada escolha e a forma como esses elementos contribuem para a eficácia da interface. Além disso, explora-se a interação do programa com o componente JTable, fornecendo uma visão abrangente de como a representação visual dos dados é realizada.

O programa desenvolvido em Java tem como objetivo central aprofundar o entendimento em estruturas de dados, com um foco especial na avaliação do desempenho de algoritmos de ordenação. Esta iniciativa visa não apenas à implementação técnica desses algoritmos, como o Bubble Sort, Insertion Sort e Quick Sort, mas também à análise prática de sua eficiência em termos de tempo de execução. Além disso, o programa busca integrar esses conceitos com a manipulação do tipo BLOB no MySQL, fornecendo uma experiência prática no contexto de uma base de dados robusta relacionada ao bioma cerrado.

A avaliação comparativa do desempenho dos algoritmos de ordenação, quando aplicados a uma quantidade significativa de dados, oferece insights valiosos sobre suas vantagens e limitações na prática. O projeto busca, assim, transcender a teoria e trazer uma perspectiva pragmática à utilização dessas ferramentas fundamentais.

Além disso, a inclusão de uma área gráfica dedicada para operações CRUD não apenas expande as funcionalidades da aplicação, mas também acentua a aplicação prática dos conceitos de manipulação de dados em tempo real. O programa, composto por mais de 10 telas gráficas, fornece uma plataforma extensa para a prática e aprimoramento de habilidades técnicas, permitindo uma exploração completa dos temas abordados.

Ao longo deste documento, examina-se detalhadamente como o programa atinge seus objetivos, desde a implementação dos algoritmos de ordenação até a interação com a base de dados, mantendo constantemente uma visão integrada entre teoria e prática. O resultado final é uma aplicação robusta que não apenas demonstra proficiência técnica, mas também destaca a importância e aplicabilidade desses conhecimentos em um contexto mais amplo, especialmente em relação ao bioma cerrado.

Durante o desenvolvimento, explora-se minuciosamente a evolução do programa, analisando a lógica por trás da escolha dos algoritmos de ordenação, examinando o desempenho de cada um com base no tempo de execução e destacando a integração do tipo BLOB. Essa abordagem visa não apenas oferecer uma visão técnica da aplicação, mas também ressaltar a importância prática da programação e manipulação de dados em contextos do mundo real, evidenciando o bioma cerrado como uma temática central e relevante.

**3.0 Referencial teórico**

O escopo deste trabalho transcende a simples implementação de algoritmos de ordenação, adentrando uma análise mais abrangente. Embora a estrutura de dados permaneça como o foco primordial da aplicação, o projeto não se restringe apenas aos algoritmos Bubble Sort, Insertion Sort e Quick Sort. Ampliam-se as fronteiras do trabalho acadêmico ao explorar implicações mais amplas, como a integração de uma interface gráfica do usuário (GUI) e funcionalidades CRUD (Create, Read, Update, Delete). A escolha específica dos algoritmos não é determinada apenas pela sua inclusão nas listas dadas em sala de aula; ela é pautada na consideração cuidadosa da sua relevância na solução de problemas do mundo real e na sua aplicabilidade em projetos de software mais intrincados.

Os algoritmos de ordenação desempenham um papel de extrema importância na eficiente organização de dados, sendo essenciais para os desenvolvedores de software. A abordagem detalhada desses algoritmos neste contexto não se limita meramente a descrever suas operações básicas, mas procura compreender de que forma suas características intrínsecas podem impactar significativamente o desempenho de um sistema. Nesse sentido, torna-se imperativo fornecer uma breve explicação para cada algoritmo escolhido:

**3.1 Bubble Sort**

O Bubble Sort é reconhecido por sua simplicidade. Percorrendo repetidamente a lista, compara elementos adjacentes e os troca se estiverem fora de ordem. Este processo continua até que a lista esteja completamente ordenada. Apesar de sua natureza intuitiva, o Bubble Sort possui uma complexidade assintótica quadrática, O(n²), tornando-o menos eficiente para conjuntos de dados extensos. A literatura consultada, como "Introduction to Algorithms" de Cormen et al., é fundamental para compreender a base teórica desse algoritmo.

**3.2 Insertion Sort**

O algoritmo de ordenação por inserção (Insertion Sort), revela-se particularmente eficaz em listas parcialmente ordenadas. Dividindo a lista em uma parte ordenada e uma não ordenada, o Insertion Sort seleciona elementos da parte não ordenada e os insere na parte ordenada, deslocando elementos maiores para abrir espaço. Sua complexidade assintótica também é quadrática, O(n²), mas ele supera o Bubble Sort em eficiência, especialmente em casos de listas parcialmente ordenadas.

**3.3 Quick Sort**

O Quick Sort, terceiro algoritmo destacado neste estudo, foi selecionado das listas fornecidas em aula devido à sua eficiência com conjuntos de dados extensos. Baseado no paradigma de divisão e conquista, o Quick Sort escolhe um pivô e rearranja os elementos, dividindo a lista em duas partes. Este processo é repetido recursivamente. Sua complexidade assintótica média é O(n log n), tornando-o uma escolha eficiente para grandes volumes de dados.

**3.4 Integração com a Prática**

O programa desenvolvido inclui uma interface gráfica do usuário (GUI) e funcionalidades CRUD (Create, Read, Update, Delete), demonstrando a aplicação prática dos algoritmos no desenvolvimento de software funcional e interativo. A implementação desses recursos não apenas valida a compreensão teórica dos algoritmos, mas também destaca sua utilidade em ambientes do mundo real.

A interface gráfica proporciona uma interação mais intuitiva e amigável, permitindo aos usuários visualizarem e compreenderem o processo de ordenação. Além disso, a inclusão das operações CRUD amplia o escopo do programa, tornando-o mais versátil e aplicável em diferentes contextos. Essas adições não apenas consolidam o conhecimento teórico, mas também evidenciam a capacidade de traduzir conceitos complexos em soluções práticas.

A escolha específica dos algoritmos, provenientes das listas fornecidas em aula, reflete a importância do conteúdo acadêmico na formação do entendimento teórico. Os materiais de apoio não apenas orientaram a seleção dos algoritmos, mas também influenciaram diretamente a abordagem adotada para a implementação prática. Essa integração entre teoria e prática é fundamental para o desenvolvimento de habilidades sólidas em ciência da computação.

A literatura consultada durante o processo de pesquisa, além das fontes mencionadas anteriormente, incluiu material didático fornecido em aula, contribuindo para uma compreensão abrangente dos conceitos abordados. A sinergia entre os materiais de aula e as referências bibliográficas enriqueceu a análise teórica e prática, garantindo uma base sólida para o desenvolvimento do programa e para as conclusões apresentadas neste trabalho.

Em resumo, o estudo teórico dos algoritmos de ordenação é fundamental para compreender a eficiência e a escolha adequada em diferentes contextos. O Bubble Sort, o Insertion Sort e o Quick Sort, escolhidos com base nos materiais dados, representam uma variedade de abordagens, desde as mais simples até as mais eficientes. A literatura consultada forneceu uma base sólida para a compreensão desses algoritmos, sendo essencial tanto para o desenvolvimento do programa quanto para o aprimoramento contínuo das habilidades em ciência da computação.

**3.5 Referências Gerais:**

Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Java. Wiley.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley.

**4.0 Desenvolvimento**

O grupo desempenhou um papel exemplar na criação de um robusto e multifuncional sistema, dividido em seis pacotes distintos: algorithm, backend, database, main, ui e utils. Nesse contexto, será oferecida uma análise detalhada de cada pacote e suas respectivas classes.

**4.1 Pacote algorithm:**

No pacote "algorithm" desenvolvido em Java, três classes distintas desempenham papéis cruciais na implementação de algoritmos de ordenação. A primeira delas, "BubbleSort", representa uma implementação do algoritmo de ordenação conhecido como Bubble Sort. Essa classe é responsável por realizar a ordenação de elementos através de comparações consecutivas e trocas, movendo-se através da lista até que a mesma esteja devidamente ordenada.

A segunda classe, "InsertionSort", representa a implementação do algoritmo de ordenação chamado Insertion Sort. Nesse contexto, a classe é responsável por inserir elementos em suas posições apropriadas, construindo gradualmente uma sequência ordenada.

Por fim, a terceira classe, "QuickSort", implementa o algoritmo de ordenação Quick Sort. Essa classe se destaca por sua eficiência, utilizando uma estratégia de divisão e conquista para ordenar os elementos. O Quick Sort é conhecido por sua rapidez em comparação com outros algoritmos de ordenação, tornando-se uma escolha preferida em muitos cenários.

**4.1.1 BubbleSort:**

A classe "BubbleSort" é uma implementação do algoritmo de ordenação, destinado a ordenar uma lista de arrays de strings com base na coluna ‘fid’ da tabela cerrado.

O método público bubbleSort aceita uma lista de arrays de strings como parâmetro e realiza o processo de ordenação. Utilizando dois loops aninhados, o algoritmo compara elementos adjacentes e efetua trocas sempre que necessário para garantir que a lista seja ordenada de acordo com os valores presentes na coluna 'fid'.

Durante cada iteração do loop externo, o algoritmo percorre a lista, comparando os elementos adjacentes. Caso o valor na coluna 'fid' do elemento atual seja maior que o valor na coluna 'fid' do próximo elemento, ocorre uma troca, garantindo que os elementos estejam em ordem crescente.

A troca de elementos é realizada pelo método privado replace, que recebe como parâmetros a lista de arrays e as posições dos elementos a serem trocados. Esse método utiliza uma abordagem temporária para efetuar a troca de posição entre os elementos, contribuindo para a ordenação correta.

O algoritmo continua iterando até que toda a lista esteja ordenada. A implementação é clara e direta, oferecendo uma solução eficaz para a ordenação dos dados com base nos valores da coluna 'fid'.

**4.1.2 InsertionSort:**

A classe "InsertionSort" representa uma implementação do algoritmo de ordenação conhecido como Insertion Sort. Este algoritmo se destaca por sua abordagem incremental, construindo uma sequência ordenada à medida que itera sobre a lista fornecida.

O método público insertionSort recebe como parâmetro uma lista de arrays de strings e inicia o processo de ordenação. Durante cada iteração do loop externo, o algoritmo seleciona um elemento como "chave" e o compara com os elementos à sua esquerda na lista.

A comparação é feita com base nos valores numéricos presentes na coluna 'fid' dos arrays de strings. Caso o valor 'fid' do elemento à esquerda seja maior que o da "chave", ocorre um deslocamento para a direita, abrindo espaço para a correta posição da "chave" na sequência ordenada.

Este deslocamento é realizado pelo loop interno, onde o algoritmo compara e move elementos até encontrar a posição adequada para inserir a "chave". Essa abordagem incremental é eficiente para listas que já estão parcialmente ordenadas, uma vez que requer menos movimentação de elementos.

Ao final da iteração, a "chave" é inserida na posição correta na sequência ordenada, e o processo continua até que todos os elementos da lista estejam ordenados. A implementação é concisa e eficaz, oferecendo uma solução estável para a ordenação dos dados com base nos valores numéricos da coluna 'fid'.

**4.1.3QuickySort:**

A classe "QuickSort" representa uma implementação do eficiente algoritmo de ordenação conhecido como Quick Sort, capaz de realizar a ordenação de uma lista de arrays de strings com base nos valores presentes na coluna 'fid'. O código reflete uma abordagem clássica do Quick Sort, utilizando um pivô para dividir a lista em partições menores e realizar comparações e trocas entre os elementos.

O método público quickSort é a porta de entrada para o algoritmo e chama o método privado quickSort, que realiza a ordenação recursiva da lista. A escolha do pivô, representado pela variável *pivot*, é feita considerando o elemento central da lista delimitada pelos índices *left* e *right*.

O loop principal do algoritmo compara os elementos à esquerda e à direita do pivô, movendo-os em direção ao centro até que o ponto de interseção seja encontrado. Quando isso acontece, ocorre a troca de posições entre os elementos, contribuindo para a correta ordenação.

As linhas de código ***temp = data.get(leftIndex);, data.set(leftIndex, data.get(rightIndex));***, e ***data.set(rightIndex, temp);*** são responsáveis pela troca efetiva dos elementos, enquanto *leftIndex++* e *rightIndex--* ajustam os índices para a próxima iteração do loop.

Os blocos condicionais finais ***(if (rightIndex > left) { ... }***e ***if (leftIndex < right) { ... })*** garantem que a ordenação continue nas partições restantes da lista. Essa abordagem de divisão e conquista do Quick Sort se mostra eficaz na ordenação rápida e eficiente de grandes conjuntos de dados.

Assim, a classe "QuickSort" fornece uma implementação sólida e eficiente do algoritmo Quick Sort, contribuindo para a ordenação eficaz dos dados com base nos valores da coluna 'fid'.

**4.2 Pacote backend:**

O pacote "backend" desempenha um papel fundamental nas operações diretas na base de dados, sendo composto por quatro classes especializadas. A classe "DeleteLogic" concentra-se na lógica de exclusão de registros, proporcionando a capacidade de remover dados de maneira eficaz. A "InsertionLogic" é responsável por gerenciar a lógica de inserção, permitindo a adição de novos registros na base de dados de forma organizada. A classe "UpdateLogic" assume o controle das operações de atualização, possibilitando a modificação de registros existentes. Complementando essas funcionalidades, a classe "Records" desempenha um papel essencial na eficiente gestão e armazenamento dos registros resgatados.

**4.2.1DeleteLogic:**

Na classe "DeleteLogic" do pacote "backend", uma lógica eficaz para exclusão de registros no banco de dados é implementada. A classe implementa a interface "AutoCloseable", possibilitando seu uso em um bloco try-with-resources, onde os recursos, neste caso a conexão com o banco de dados, são fechados automaticamente ao sair do bloco.

O método público deleteRecord é responsável por excluir registros com base em um identificador específico. A conexão com o banco de dados é gerenciada, sendo verificado se a mesma é nula ou está fechada antes de ser inicializada através do método Db.getConnection().

A consulta SQL de exclusão é definida na string query, e um PreparedStatement é preparado para execução da consulta. O valor do parâmetro "id" é então definido, e a consulta é executada utilizando o método executeUpdate(). O número de linhas afetadas é verificado, e o método retorna verdadeiro se pelo menos uma linha foi excluída, indicando o sucesso da operação.

Em caso de exceção do tipo SQLException, o erro é impresso no console e o método retorna falso, indicando que a exclusão não foi bem-sucedida. Finalmente, o método close da interface "AutoCloseable" é implementado para fechar a conexão com o banco de dados, garantindo uma gestão adequada dos recursos. O código reflete uma abordagem organizada e eficiente para a operação de exclusão de registros.

**4.2.2 InsertionLogic:**

Na classe "InsertionLogic" do pacote "backend", uma lógica de inserção de registros na base de dados é implementada de maneira robusta e eficiente. A classe também implementa a interface "AutoCloseable", permitindo seu uso em um bloco try-with-resources, onde os recursos, como a conexão com o banco de dados, são fechados automaticamente ao sair do bloco.

No construtor da classe, é realizada uma verificação e inicialização da conexão com o banco de dados, garantindo que esteja pronta para ser utilizada em operações subsequentes.

O método público insertRecord é responsável por inserir um novo registro na tabela "cerrado". Ele aceita diversos parâmetros, representando os campos do registro a serem inseridos. Antes de realizar a inserção, verifica se todos os campos obrigatórios estão preenchidos através do método areFieldsFilled.

A consulta SQL de inserção é definida na string insertSQL, e um PreparedStatement é preparado para a execução da consulta. Os parâmetros da consulta são então definidos com base nos valores fornecidos nos campos do registro.

Uma característica notável é o tratamento do campo "areaKm", que é convertido para um valor double antes de ser inserido no banco de dados, garantindo a consistência do tipo de dado.

O método também inclui a lógica para ler o conteúdo do arquivo de imagem associado ao registro e definir como um objeto BLOB na consulta, permitindo o armazenamento de imagens no banco de dados.

Em caso de sucesso na execução da inserção, uma mensagem de sucesso é exibida através do JOptionPane, e o método retorna true. Em situações de exceção, mensagens de erro são apresentadas e o método retorna false, indicando que a inserção não foi bem-sucedida.

Além disso, a classe possui métodos utilitários como areFieldsFilled, que verifica se os campos estão preenchidos, e showMissingFieldsMessage, que exibe uma mensagem de erro informando campos obrigatórios não preenchidos.

Assim, a classe "InsertionLogic" oferece uma implementação completa e cuidadosa para a inserção de registros no banco de dados, incorporando boas práticas de programação e garantindo a integridade das operações.

**4.2.3 UpdateLogic:**

Na classe "UpdateLogic" do pacote "backend", uma lógica de atualização de registros em um banco de dados é implementada de maneira eficiente e segura. A classe também implementa a interface "AutoCloseable", permitindo seu uso em um bloco try-with-resources, onde os recursos, como a conexão com o banco de dados, são fechados automaticamente ao sair do bloco.

O método público validateId é responsável por verificar se um registro com o ID fornecido existe no banco de dados. Ele realiza uma consulta SQL para recuperar o registro associado ao ID e retorna true se o resultado da consulta contiver pelo menos uma linha, indicando a existência do registro.

O método público updateRecord é encarregado de atualizar um registro no banco de dados. Ele aceita vários parâmetros, representando os novos valores dos campos do registro. Antes de executar a atualização, verifica se a conexão com o banco de dados está estabelecida. A consulta SQL de atualização é definida na string query, e um objeto PreparedStatement é preparado para a execução da consulta. Os parâmetros da consulta são então definidos com base nos valores fornecidos nos campos do registro, similar à lógica de inserção. A execução da consulta é realizada com executeUpdate().

O método getRegistroPorId obtém um registro específico do banco de dados com base no ID fornecido. Ele utiliza uma consulta SQL para recuperar os dados associados ao ID, cria um objeto "Records" com esses dados e retorna o objeto.

A classe também possui a capacidade de fechar a conexão com o banco de dados no método close, garantindo uma boa prática de gerenciamento de recursos.

Em caso de exceções durante a execução dos métodos, mensagens de erro são impressas no console e exibidas com JOptionPane, proporcionando um feedback claro sobre qualquer problema encontrado durante a operação. Além disso, a classe trata adequadamente exceções relacionadas à leitura de imagens, fornecendo mensagens específicas ao usuário.

Assim, a classe "UpdateLogic" oferece uma implementação completa e cuidadosa para a atualização de registros no banco de dados, incorporando boas práticas de programação e garantindo a integridade das operações.

**4.2.4 Records:**

A classe "Records" no pacote "backend" é uma representação eficiente e organizada dos registros associados a um banco de dados. Ela encapsula os dados de um registro, fornecendo métodos de acesso para cada campo específico.

A classe possui os seguintes campos privados:

* fid: Representa o campo FID do registro.
* state: Representa o campo STATE do registro.
* pathRow: Representa o campo PATH\_ROW do registro.
* mainClass: Representa o campo MAIN\_CLASS do registro.
* nameClass: Representa o campo CLASS\_NAME do registro.
* julianDay: Representa o campo JULIAN\_DAY do registro.
* imageDate: Representa o campo IMAGE\_DATE do registro.
* year: Representa o campo YEAR do registro.
* areaKm: Representa o campo AREA\_KM do registro.
* satellite: Representa o campo SATELLITE do registro.
* sensor: Representa o campo SENSOR do registro.
* uuid: Representa o campo UUID do registro.
* imagem: Representa o campo IMAGEM do registro.
* blob: Representa um objeto InputStream para o campo IMAGEM no formato BLOB.

O construtor da classe aceita argumentos para inicializar todos esses campos no momento da criação de um objeto "Records". Além disso, a classe fornece métodos de acesso público (getters) para recuperar os valores de cada campo individualmente.

Essa organização permite uma manipulação fácil e segura dos registros associados ao banco de dados, facilitando a passagem e o uso desses dados em outras partes do código. A inclusão do objeto InputStream (blob) proporciona a capacidade de trabalhar com dados binários, como imagens.

Em resumo, a classe "Records" desempenha um papel crucial na estrutura do código, garantindo uma representação consistente e acessível dos registros do banco de dados.

**4.3 Pacote database:**

O pacote "database" é fundamental para a funcionalidade eficiente e coesa do sistema, abrigando duas classes essenciais: "Db" e "DbException". A classe "Db" desempenha o papel de estabelecer a conexão com o banco de dados, fornecendo uma infraestrutura robusta para a execução de operações relacionadas ao banco. Essa classe permite que outras partes do código acessem a conexão de maneira organizada e segura, facilitando a realização de operações CRUD (Create, Read, Update, Delete) e outros processos relacionados.

A classe "DbException" desempenha um papel crucial no tratamento de exceções relacionadas ao banco de dados. Proporciona uma estrutura sólida para lidar com erros específicos de banco de dados, contribuindo para a robustez do sistema.

Assim, o pacote "database" emerge como uma peça-chave na arquitetura do sistema, facilitando a integração e a utilização consistente da conexão com o banco de dados em várias classes, especialmente aquelas envolvidas nos processos fundamentais de CRUD.

**4.3.1 Db:**

A classe "Db" no pacote "database" desempenha um papel crucial na gestão da conexão com o banco de dados, oferecendo métodos para obtenção da conexão, fechamento de conexão, fechamento de instrução SQL (Statement) e fechamento de conjunto de resultados (ResultSet).

Através do método estático getConnection, a classe permite a obtenção de uma conexão com o banco de dados. O método verifica se já existe uma conexão ativa (conn) e, caso contrário, carrega as configurações do banco de dados a partir do arquivo "db.properties" usando a classe FileInputStream. As propriedades carregadas incluem a URL de conexão e outras informações necessárias para estabelecer a conexão. Em seguida, utiliza a classe DriverManager para conectar-se ao banco.

Além disso, a classe oferece métodos para fechar a conexão (closeConnection), o statement (closeStatement) e o conjunto de resultados (closeResultSet). Estes métodos são projetados para liberar recursos após o uso, contribuindo para a eficiência e segurança da aplicação.

Em casos de exceções durante esses processos, a classe lança exceções personalizadas (DbException) que encapsulam as mensagens de erro específicas da exceção original (por exemplo, IOException ou SQLException). Isso facilita a identificação e tratamento de erros, melhorando a robustez do código.

Portanto, a classe "Db" é uma parte integral do sistema, proporcionando uma interface simplificada e eficiente para a gestão da conexão com o banco de dados e o tratamento de exceções associadas.

**4.3.2DbException:**

A classe "DbException" é uma exceção personalizada que estende a classe RuntimeException. Sua principal finalidade é encapsular mensagens de erro específicas relacionadas a operações no banco de dados.

Ao herdar de RuntimeException, essa classe pertence à categoria de exceções não verificadas. Ela não exige a declaração em nível de método ou manipulação explícita, tornando-a conveniente para situações em que o tratamento detalhado da exceção pode não ser essencial, mas ainda fornece informações significativas em caso de erro.

O construtor da classe recebe uma mensagem de erro como argumento (msg) e chama o construtor da classe pai (RuntimeException) usando o método super(msg). Isso estabelece a mensagem de erro associada à exceção.

Ao ser lançada, a "DbException" fornece uma forma eficaz de transmitir mensagens de erro específicas da camada de acesso a dados (banco de dados) para camadas superiores na aplicação. Isso facilita a identificação e o tratamento apropriado dos problemas que podem surgir durante a execução das operações do banco de dados.

**4.4 Pacote main:**

O pacote "main" é composto exclusivamente pela classe "Main", cuja função principal é iniciar a execução do programa. Essa classe serve como ponto de entrada. Essa estruturação facilita a manutenção e compreensão do código, além de proporcionar um ponto claro de partida para desenvolvedores que exploram o código-fonte pela primeira vez.

A classe "Main" desempenha um papel crucial na aplicação, sendo responsável por iniciar a execução do programa. No código da classe, que é o ponto de entrada principal para a aplicação Java, é criada uma instância da classe HomeScreen. Esta instância representa a tela inicial do programa, onde os usuários são recebidos ao iniciar o aplicativo.

Ao criar um objeto da classe HomeScreen, é instanciada uma janela do tipo JFrame, que serve como a interface gráfica do usuário. Em seguida, são realizadas configurações importantes para o comportamento dessa janela: setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) assegura que o programa será encerrado quando o usuário fechar a janela, setLocationRelativeTo(null) centraliza a janela na tela, e setVisible(true) torna a janela visível para o usuário.

O uso da classe Main para abrir a tela de abertura do programa indica uma abordagem organizada e orientada a objetos, onde a lógica principal de inicialização e exibição da interface gráfica é centralizada em uma classe dedicada. Isso promove a modularidade e a manutenção eficiente do código, facilitando futuras alterações ou expansões na aplicação.

**4.5 Pacote utils:**

O pacote "utils" compreende um conjunto de quatro classes auxiliares fundamentais para o funcionamento eficiente do programa. A classe "ConversorBlob" desempenha um papel crucial ao fornecer métodos para converter dados entre diferentes tipos, sendo especialmente útil na manipulação de objetos do tipo BLOB (Binary Large Object) no contexto do banco de dados. Além disso, as classes "CustomButton", "RoundButton" e "UiHelper" contribuem para a interface gráfica do usuário (UI) de maneira significativa. "CustomButton" e "RoundButton" oferecem opções personalizadas para a criação de botões, enquanto "UiHelper" fornece métodos utilitários para facilitar o desenvolvimento da interface do usuário, consolidando assim funcionalidades auxiliares essenciais para a robustez e usabilidade do programa. Essa abordagem modular do pacote "utils" reflete uma prática eficaz de design de software, promovendo a reutilização de código e simplificando a manutenção do sistema.

**4.5.1 ConversorBlob:**

A classe "ConversorBlob" desempenha um papel crucial na aplicação ao oferecer métodos para converter dados binários de um formato específico, representado por um InputStream, em uma imagem BufferedImage. O método estático "inputStreamToImage" aceita um InputStream como parâmetro e realiza a conversão, proporcionando flexibilidade no manuseio de dados binários, particularmente útil em cenários que envolvem operações com imagens armazenadas em formato binário, como aquelas provenientes de um banco de dados.

Internamente, a classe utiliza a biblioteca ImageIO para decodificar o InputStream em um objeto BufferedImage, que representa a imagem na memória do programa. A verificação de nulidade do InputStream é realizada para garantir a robustez da operação e evitar exceções desnecessárias.

O uso dessa classe se destaca em situações em que é necessário converter dados binários, como imagens, provenientes de fontes diversas, como bancos de dados, em uma representação visual manipulável pela aplicação. A modularidade e a simplicidade da classe "ConversorBlob" contribuem para a clareza e eficiência do código, facilitando a integração dessas funcionalidades de conversão em diversos contextos dentro do programa.

**4.5.2 CustomButton:**

A classe "CustomButton" desempenha um papel significativo na interface gráfica da aplicação, proporcionando uma experiência interativa ao usuário por meio de botões personalizados com efeitos visuais dinâmicos. Esse componente estende a classe JButton e incorpora funcionalidades específicas para manipular ícones em diferentes estados, como normal, hover e clique.

Ao ser instanciada, a classe recebe três ImageIcon, representando respectivamente o ícone regular (estado padrão), o ícone durante o hover (mouse sobre o botão) e o ícone durante o clique. O construtor configura inicialmente o ícone regular e define características visuais, como a remoção da borda padrão, a desativação do preenchimento automático e a alteração do cursor para uma versão de mão, indicando a interatividade.

A classe incorpora tratadores de eventos do mouse, como "mousePressed," "mouseReleased," "mouseEntered" e "mouseExited," para reagir aos diferentes estados do mouse. Por exemplo, quando o botão é clicado, o ícone correspondente ao clique é exibido; quando o mouse é liberado, o ícone de hover é mostrado se o mouse ainda estiver sobre o botão, senão, o ícone regular é restaurado.

Dessa forma, o "CustomButton" enriquece a interface do usuário ao oferecer botões visualmente atrativos e responsivos, contribuindo para uma experiência de usuário mais intuitiva e envolvente. Essa classe é especialmente útil em contextos nos quais se busca aprimorar a estética e a interatividade dos elementos gráficos em uma aplicação.

**4.5.3RoundButton:**

A classe "RoundButton" é uma extensão da classe JButton que oferece uma representação visualmente distinta dos botões, apresentando uma forma arredondada em vez da forma retangular convencional. Esse componente contribui para a estética e a personalização da interface gráfica da aplicação, proporcionando uma aparência mais suave e moderna aos botões.

Ao ser instanciada, essa classe aceita um rótulo (label) para o botão, que é passado ao construtor da superclasse JButton. A área de conteúdo do botão é configurada como não preenchida, resultando em uma aparência transparente para a área interna do botão.

A pintura do componente é personalizada por meio dos métodos paintComponent e paintBorder. O método paintComponent lida com a pintura da área interna do botão, ajustando a cor de fundo com base no estado do modelo do botão (pressionado ou não). A forma arredondada é desenhada usando g.fillRoundRect, proporcionando uma estética arrojada ao botão.

Por sua vez, o método paintBorder trata da pintura da borda do botão. Utilizando uma instância de Graphics2D, ele define a cor e a espessura da borda, desenhando um contorno arredondado ao redor do botão. Esse contorno contribui para enfatizar a forma arredondada do botão.

Adicionalmente, a classe implementa a lógica para evitar que o botão tenha um contorno quando recebe o foco, contribuindo para uma experiência de usuário mais coesa e esteticamente agradável.

Em resumo, a classe "RoundButton" é uma valiosa adição à aplicação, proporcionando botões com uma estética arredondada única, destacando-se como elementos visuais distintivos e modernos na interface do usuário.

**4.5.4UiHelper:**

A classe "UiHelper" é uma coleção de métodos utilitários que simplificam a criação e personalização de componentes de interface gráfica, como JLabels, JButtons, JTextAreas, JTextFields e JComboBoxes. Esses métodos são projetados para facilitar a manutenção do código, promover a coesão e melhorar a legibilidade ao encapsular as configurações comuns desses componentes.

1. createLabel: Este método cria e retorna um JLabel com texto, posição, dimensões, fonte e cor especificados. Essa função oferece uma maneira rápida e consistente de criar rótulos personalizados em diferentes partes da aplicação.
2. createRoundButton: Similar ao método anterior, este cria um JButton do tipo RoundButton com texto, posição, dimensões, fonte e cor definidos. A utilização do RoundButton contribui para uma estética visualmente distinta dos botões na interface.
3. createTextArea: Cria e retorna um JTextArea configurado com posição, dimensões, fonte e cor específicas. Este método é útil para criar áreas de texto personalizadas, sendo transparentes e permitindo quebras de linha.
4. createTextField: Retorna um JTextField com posição, dimensões, fonte e cor definidas. Este método simplifica a criação de campos de texto em diferentes partes da aplicação.
5. createComboBox: Cria um JComboBox preenchido com os itens fornecidos, configurado com posição, dimensões, fonte e cor. Facilita a criação de menus suspensos com consistência visual.

Esses métodos encapsulam detalhes de implementação comuns, como configurações de posição, dimensões, fonte e cor, permitindo que esses componentes sejam facilmente adicionados e personalizados em várias partes da aplicação. Essa abordagem modular promove a reutilização de código e contribui para uma manutenção mais eficiente e organizada da interface gráfica do programa.

**4.6 Pacote ui:**

O pacote "ui" desempenha um papel crucial no desenvolvimento do programa, abrigando as classes responsáveis por compor a interface gráfica. Nele, encontramos as implementações específicas de janelas, botões, campos de texto e outros componentes visuais que os usuários interagem ao utilizar o software. Essas classes não apenas fornecem a representação visual dos elementos da aplicação, mas também integram a lógica de interação do usuário com os processos subjacentes do programa. Organizando essas classes em um pacote dedicado, o "ui" contribui para uma estrutura modular e coesa, facilitando a manutenção, expansão e compreensão do código-fonte relacionado à interface gráfica. Essa abordagem promove uma clara separação de preocupações, melhorando a legibilidade e a escalabilidade do projeto.

**4.6.1 BaseFrame:**

A classe BaseFrame no pacote "ui" desempenha um papel fundamental na construção da estrutura da interface gráfica do programa. Como uma classe abstrata estendendo a classe JFrame do Swing, ela serve como uma base para a criação de outras janelas na aplicação. O construtor padrão configura várias propriedades essenciais do JFrame, como o título, o tamanho padrão e o encerramento da aplicação ao fechar a janela. Além disso, o método initializeComponents é declarado como abstrato, exigindo que as subclasses forneçam uma implementação específica para inicializar os componentes visuais da janela.

Há também um construtor adicional que aceita uma imagem de fundo como parâmetro. Esse construtor chama o construtor padrão para configurar as propriedades comuns do frame e, em seguida, define um painel personalizado como o conteúdo da janela. Esse painel, ao ser desenhado, utiliza a imagem fornecida como fundo, proporcionando uma aparência visual atraente e personalizada para a interface do usuário.

Assim, a BaseFrame representa uma abstração central para a construção de janelas na aplicação, promovendo uma estrutura modular e facilitando a extensão e manutenção do código relacionado à interface gráfica.

**4.6.2 HomeScreen:**

A classe HomeScreen no pacote "ui" é responsável por criar a interface gráfica da tela inicial do programa. Assim como outras classes no mesmo pacote, ela estende a classe BaseFrame, aproveitando suas configurações básicas de uma janela Swing e adicionando elementos específicos para a funcionalidade da tela inicial.

Ao ser instanciada, a HomeScreen carrega uma imagem de fundo personalizada para a interface, proporcionando um visual atraente. A tela é composta por vários elementos, como rótulos, um botão personalizado e uma área de texto.

O botão personalizado, denominado customButton, é criado com três imagens diferentes representando os estados normais, de hover e de clique. Um ouvinte de ação é adicionado a esse botão, e quando clicado, fecha a tela atual (HomeScreen) e abre a tela RecordsScreen, proporcionando uma transição suave entre as telas.

Além disso, a HomeScreen possui uma área de texto (textArea) que exibe informações sobre o Cerrado, destacando sua importância como bioma brasileiro e os desafios ambientais que enfrenta. A formatação do texto é cuidadosamente estilizada para proporcionar uma apresentação visualmente agradável.

Os rótulos, como titleLabel e nextLabel, complementam a interface, fornecendo títulos e instruções visuais. Esses elementos são posicionados manualmente no painel de conteúdo da janela, aproveitando o layout nulo para maior controle sobre o posicionamento.

Em resumo, a HomeScreen cria uma interface gráfica envolvente e informativa, conectando o usuário às funcionalidades do programa e apresentando visualmente o bioma do Cerrado e suas questões ambientais.

**4.6.3 DeleteScreen:**

A classe DeleteScreen no pacote "ui" representa a interface gráfica para a exclusão de registros no programa. Ao herdar da classe BaseFrame, ela adota a estrutura básica de uma janela Swing, com a adição de elementos específicos para a funcionalidade de exclusão.

A tela é construída com uma imagem de fundo, proporcionando uma aparência visual agradável e personalizada. Três imagens diferentes são carregadas para os estados normais, de hover e de clique de um botão personalizado chamado buttonimg1. Esse botão serve como um meio de retorno à tela principal quando pressionado, fechando a tela atual e abrindo a RecordsScreen.

Além disso, a DeleteScreen contém vários componentes gráficos, como rótulos, botões e campos de texto. O título "Deletar" é exibido com uma fonte estilizada e cor branca. Há um botão redondo com o texto "Apagar", permitindo a execução da exclusão quando acionado.

A parte crucial da funcionalidade é implementada no botão de exclusão (buttonSubmit). Quando acionado, este botão lê o ID inserido, verifica sua validade e tenta excluir o registro correspondente chamando o método deleteRecord da classe DeleteLogic. Mensagens de confirmação ou erro são exibidas ao usuário, proporcionando uma interação amigável.

No geral, a DeleteScreen contribui para uma experiência do usuário intuitiva ao fornecer uma interface visualmente atraente e funcional para a exclusão de registros no programa.

**4.6.4 ImageScreen:**

A classe ImageScreen no pacote "ui" é responsável por criar a interface gráfica para exibir imagens associadas aos registros do banco de dados. Como outras classes no mesmo pacote, ela estende a classe BaseFrame, herdando as configurações básicas de uma janela Swing e adicionando elementos específicos para essa funcionalidade.

Ao ser instanciada, a ImageScreen carrega uma imagem de fundo personalizada para a interface, proporcionando uma experiência visual agradável. A tela é composta por vários elementos, como rótulos, um botão personalizado e uma área para exibição da imagem.

O botão personalizado, denominado buttonimg1, é criado com três imagens diferentes representando os estados normais, de hover e de clique. Um ouvinte de ação é adicionado a esse botão, e quando clicado, fecha a tela atual (ImageScreen) e abre a tela UpdateScreen\_2, permitindo a atualização do registro associado à imagem.

Além disso, a ImageScreen exibe a imagem associada ao registro correspondente ao ID fornecido na tela anterior. Se a imagem estiver disponível, ela é exibida centralizada em um JLabel. Caso contrário, um painel branco com um rótulo indicando "NULL" é apresentado, indicando que a imagem não está disponível.

Os elementos da interface, como rótulos e botões, são posicionados manualmente no painel de conteúdo da janela, aproveitando o layout nulo para um controle preciso sobre o posicionamento.

Em resumo, a ImageScreen cria uma interface gráfica envolvente e interativa para visualizar e atualizar registros de imagens associadas a um ID específico. Essa tela conecta o usuário às funcionalidades do programa e proporciona uma experiência visualmente agradável.

**4.6.5 InsertionScreen:**

A classe InsertionScreen no pacote "ui" é responsável por criar a interface gráfica para inserção de novos registros no banco de dados. Assim como outras classes no mesmo pacote, ela estende a classe BaseFrame, herdando as configurações básicas de uma janela Swing e adicionando elementos específicos para essa funcionalidade.

Ao ser instanciada, a InsertionScreen carrega uma imagem de fundo personalizada para a interface, proporcionando uma experiência visual agradável. A tela é composta por vários elementos, como rótulos, botões, campos de texto e ComboBoxes.

O botão personalizado, denominado buttonimg1, é criado com três imagens diferentes representando os estados normais, de hover e de clique. Um ouvinte de ação é adicionado a esse botão, e quando clicado, fecha a tela atual (InsertionScreen) e abre a tela RecordsScreen, exibindo os registros existentes.

Os elementos de entrada, como campos de texto e ComboBoxes, são posicionados manualmente no painel de conteúdo da janela, aproveitando o layout nulo para um controle preciso sobre o posicionamento. Além disso, são definidas opções padrão para alguns ComboBoxes.

Um evento de ação é associado ao botão de salvar (buttonSave). Quando clicado, este evento chama o método insertRecord da classe InsertionLogic, responsável por realizar a inserção dos dados no banco. Se a inserção for bem-sucedida, os campos de texto são resetados, proporcionando uma experiência amigável ao usuário.

Em resumo, a InsertionScreen cria uma interface gráfica intuitiva para a inserção de novos registros no banco de dados. Essa tela é essencial para a funcionalidade do programa, conectando o usuário à capacidade de adicionar novos dados de maneira fácil e eficiente.

**4.6.6 Ordering:**

A classe Ordering no pacote "ui" é responsável por criar a interface gráfica para a escolha e visualização dos algoritmos de ordenação disponíveis. Como as outras classes no mesmo pacote, ela estende a classe BaseFrame, herdando configurações básicas de uma janela Swing e adicionando elementos específicos para essa funcionalidade.

Ao ser instanciada, a Ordering define uma imagem de fundo personalizada para a interface, proporcionando uma experiência visual agradável. A tela é composta por vários elementos, incluindo rótulos, botões e botões redondos (customizados).

O botão personalizado buttonimg1 é criado com três imagens diferentes representando os estados normais, de hover e de clique. Um ouvinte de ação é adicionado a esse botão, e quando clicado, fecha a tela atual (Ordering) e abre a tela RecordsScreen, exibindo os registros existentes.

A interface também contém rótulos para identificar a tela como "Ordering" e outro para "Voltar", posicionados em locais específicos da tela. Além disso, três botões redondos são criados para selecionar diferentes algoritmos de ordenação: "Insertion Sort", "Quick Sort" e "Bubble Sort".

Para cada botão de algoritmo, é configurado um ouvinte de ação. Quando o respectivo botão é pressionado, a tela de desempenho correspondente é aberta. Isso cria uma estrutura modular e permite a expansão fácil para outros algoritmos de ordenação, mantendo o código organizado e extensível.

A disposição dos elementos na tela é feita manualmente, aproveitando o layout nulo para um controle preciso sobre o posicionamento. O objetivo é criar uma interface amigável e intuitiva para o usuário, facilitando a escolha e a visualização dos algoritmos de ordenação disponíveis.

Em resumo, a classe Ordering oferece uma interface para selecionar diferentes algoritmos de ordenação e visualizar o desempenho de cada um, proporcionando ao usuário uma experiência interativa e informativa.

**4.6.7 PerformanceBubbleSort:**

A classe PerformanceBubbleSort no pacote "ui" é responsável por exibir o desempenho do algoritmo de ordenação Bubble Sort em uma interface gráfica. Ela estende a classe BaseFrame e implementa a interface AutoCloseable. A janela da interface contém informações sobre o tempo de execução do algoritmo, uma tabela com os dados ordenados e um botão para retornar à tela de escolha de algoritmos.

Aqui está uma explicação detalhada da classe:

1. Variáveis de Instância:

* table: Uma instância de JTable que exibirá os dados ordenados.
* connection: Uma conexão com o banco de dados.
* executionTime: Armazena o tempo de execução do algoritmo Bubble Sort.
* textArea: Uma instância de JTextArea para exibir informações sobre o tempo de execução.

1. Construtor:

* Configura a imagem de fundo da interface através do construtor da classe BaseFrame.
* Chama o método initializeComponents() para configurar os componentes da interface.
* Carrega os dados do banco de dados e calcula o tempo de execução do algoritmo Bubble Sort.

1. Método loadDataFromDatabase():

* Carrega dados do banco de dados (tabela "cerrado") e armazena-os em uma lista.
* Inicia o cronômetro, realiza a ordenação dos dados usando o algoritmo Bubble Sort e registra o tempo de execução.
* Atualiza a área de texto com o tempo de execução.
* Preenche a tabela com os dados ordenados.

1. Método initializeComponents():

* Configuração de botões, rótulos e elementos da interface.
* Cria um botão de retorno (buttonimg1) que, quando pressionado, abre a tela de escolha de algoritmos (Ordering) e fecha a tela atual.
* Cria um rótulo para exibir o título da página ("Desempenho").
* Cria uma área de texto (textArea) para exibir o tempo de execução do algoritmo.
* Cria uma tabela (table) para exibir os dados ordenados.
* Adiciona uma barra de rolagem à tabela.

1. Método updateTextArea():

* Atualiza a área de texto (textArea) com o tempo de execução do algoritmo Bubble Sort.
* Utiliza SwingUtilities.invokeLater para garantir que a atualização seja feita na thread de interface do usuário.

1. Método close():

* Implementa o método da interface AutoCloseable.
* Fecha a conexão com o banco de dados, se estiver aberta.

A classe oferece uma visualização gráfica do desempenho do algoritmo Bubble Sort em termos de tempo de execução. Ela integra funcionalidades de ordenação com elementos gráficos, proporcionando uma experiência interativa ao usuário. O código está estruturado para manter a clareza e a modularidade, facilitando a compreensão e manutenção.

**4.6.8 PerformanceInsertionSort:**

A classe PerformanceInsertionSort no pacote "ui" é parte do sistema que exibe informações sobre o desempenho do algoritmo de ordenação Insertion Sort em uma interface gráfica. Esta classe estende a classe BaseFrame e implementa a interface AutoCloseable. Ao ser instanciada, a janela da interface exibe o tempo de execução do algoritmo, uma tabela contendo os dados ordenados e um botão que permite ao usuário retornar à tela de escolha de algoritmos.

Esta classe possui variáveis de instância que incluem uma tabela para exibir os dados, uma conexão com o banco de dados, a informação sobre o tempo de execução do algoritmo e uma área de texto para apresentar essas informações. Seu construtor inicializa a imagem de fundo da interface, configura os componentes e carrega dados do banco de dados, enquanto o método loadDataFromDatabase() lida com o processo de ordenação, calculando o tempo de execução.

O método initializeComponents() é responsável por configurar botões, rótulos e outros elementos visuais. Há um botão de retorno que, ao ser pressionado, fecha a tela atual e abre a tela de escolha de algoritmos. Além disso, rótulos e uma área de texto são configurados para exibir informações relevantes. O método updateTextArea() é utilizado para atualizar dinamicamente a área de texto com o tempo de execução.

A classe segue uma estrutura lógica, organizando as funcionalidades de maneira clara. Ela proporciona uma visualização gráfica do desempenho do algoritmo Insertion Sort, tornando a experiência do usuário mais interativa e informativa. O código é estruturado de forma a manter a clareza e a modularidade, facilitando a compreensão e a manutenção ao longo do tempo.

**4.6.9 PerformanceQuickSort:**

A classe PerformanceQuickSort no pacote "ui" faz parte de um sistema que apresenta informações sobre o desempenho do algoritmo de ordenação Quick Sort em uma interface gráfica. Assim como as classes anteriores, esta classe estende a classe BaseFrame e implementa a interface AutoCloseable. Ao ser instanciada, a janela da interface exibe o tempo de execução do algoritmo, uma tabela contendo os dados ordenados e um botão que permite ao usuário retornar à tela de escolha de algoritmos.

A classe possui variáveis de instância, incluindo uma tabela para exibir os dados, uma conexão com o banco de dados, informações sobre o tempo de execução do algoritmo e uma área de texto para apresentar essas informações. Seu construtor inicializa a imagem de fundo da interface, configura os componentes e carrega dados do banco de dados, enquanto o método loadDataFromDatabase() lida com o processo de ordenação, calculando o tempo de execução.

O método initializeComponents() é responsável por configurar botões, rótulos e outros elementos visuais. Há um botão de retorno que, ao ser pressionado, fecha a tela atual e abre a tela de escolha de algoritmos. Além disso, rótulos e uma área de texto são configurados para exibir informações relevantes. O método updateTextArea() é utilizado para atualizar dinamicamente a área de texto com o tempo de execução.

A estrutura da classe é semelhante às anteriores, mantendo a clareza e a modularidade do código. Essa abordagem facilita a compreensão e a manutenção ao longo do tempo. O código fornece uma visualização gráfica do desempenho do algoritmo Quick Sort, contribuindo para uma experiência do usuário mais interativa e informativa.Reco

**4.6.10 RecordsScreen:**

A classe RecordsScreen no pacote "ui" é responsável por exibir registros de uma tabela chamada "cerrado" em uma interface gráfica. Ela estende a classe BaseFrame e implementa a interface AutoCloseable. Ao ser instanciada, a janela da interface é exibida, apresentando uma tabela com os dados do banco de dados e botões que permitem navegar entre telas para inserir, atualizar ou excluir registros.

A classe possui variáveis de instância, incluindo uma tabela para exibir os dados, uma conexão com o banco de dados e o caminho para a imagem de fundo da interface.

O construtor da classe inicializa a imagem de fundo, configura os componentes e carrega os dados do banco de dados chamando o método loadDataFromDatabase(). Este método utiliza uma conexão com o banco de dados para executar uma consulta SQL e preencher a tabela com os resultados.

O método initializeComponents() configura botões, rótulos e outros elementos visuais. Há dois botões de imagem para voltar ou avançar entre telas, e três botões redondos para inserir, atualizar ou excluir registros. Cada botão possui um evento que abre a tela correspondente quando pressionado. Além disso, há rótulos para indicar as ações dos botões.

A estrutura do código segue uma abordagem modular e clara, facilitando a manutenção e compreensão. A interface oferece uma experiência do usuário intuitiva e interativa para gerenciar registros no banco de dados.

**4.6.11 UpdateScreen\_1:**

A classe UpdateScreen\_1 no pacote "ui" é responsável por exibir uma interface gráfica para atualização de registros em uma tabela chamada "cerrado". A interface inclui um campo de texto para inserção do ID do registro que se deseja atualizar, um botão para pesquisar o ID e um botão de imagem para voltar à tela principal.

O construtor da classe inicializa a imagem de fundo e chama o método initializeComponents() para configurar os elementos visuais da interface.

O método initializeComponents() configura a aparência e posição dos componentes na interface, incluindo rótulos, botões e campos de texto. O botão de imagem "Voltar" tem um evento associado que abre a tela RecordsScreen quando pressionado.

O botão "Pesquisar" tem um evento associado que valida o ID inserido no campo de texto. Se o ID for válido, a tela UpdateScreen\_2 é aberta, passando o ID como parâmetro. Caso contrário, uma mensagem de erro é exibida.

A estrutura do código é modular e compreensível, seguindo boas práticas de programação. A interface oferece uma maneira intuitiva para iniciar o processo de atualização de registros na tabela "cerrado".

**4.6.12 UpdateScreen\_2:**

A classe UpdateScreen\_2 no pacote "ui" é responsável por exibir uma interface gráfica para a atualização detalhada de um registro na tabela "cerrado". Esta classe estende BaseFrame e é aberta após a pesquisa do registro na tela UpdateScreen\_1.

O construtor da classe inicializa a imagem de fundo, define a cor de fundo como branca, inicializa o ID recebido como parâmetro e chama o método initializeComponents() para configurar os elementos visuais da interface.

O método initializeComponents() configura a aparência e posição dos componentes na interface, incluindo rótulos, botões, campos de texto e um botão para visualizar a imagem associada ao registro.

Os campos de texto são preenchidos com os dados do registro correspondente ao ID fornecido, utilizando o método preencherCampos(int id).

O botão "Salvar" tem um evento associado que tenta atualizar o registro no banco de dados com os novos valores inseridos nos campos de texto. Se a atualização for bem-sucedida, uma mensagem de sucesso é exibida, e o usuário é redirecionado para a tela UpdateScreen\_1. Caso contrário, uma mensagem de erro é exibida.

O botão "Ver Imagem" abre a tela ImageScreen, permitindo que o usuário visualize a imagem associada ao registro.

A estrutura do código é modular, compreensível e segue boas práticas de programação, oferecendo uma interface intuitiva para a atualização detalhada de registros na tabela "cerrado".

**5.0 Resultados e Discussões**

No presente tópico, abordar-se-á o comportamento do programa em Java e a descrição detalhada de sua interface. De maneira objetiva, explora-se como o programa responde a diferentes inputs, destacando seu desempenho e características distintivas. Além disso, destaca-se os aspectos visuais e interativos da interface, oferecendo uma visão abrangente de como os usuários interagem e navegam neste sistema implementado em Java. A discussão tem como objetivo proporcionar uma compreensão completa do programa, abrangendo tanto sua funcionalidade quanto a experiência do usuário ao interagir com sua interface.

**5.1 HomeScreen**

Na tela inicial do programa, é possível vislumbrar uma composição visual cuidadosamente elaborada. O cenário é emoldurado por uma imagem de fundo que, de maneira representativa, captura a essência do cerrado. O tema é habilmente abordado por meio de um texto informativo estrategicamente posicionado, oferecendo uma introdução envolvente sobre a temática em questão.

Para dar continuidade à exploração do programa, um botão com a imagem de uma raposa é proeminente na tela. Este botão, além de contribuir para a estética do layout, desempenha um papel funcional ao direcionar o usuário para a próxima tela do programa quando acionado.

A disposição harmoniosa desses elementos na tela inicial não apenas reflete a atenção ao design, mas também visa envolver o usuário desde o início, proporcionando uma transição suave para a próxima etapa da interação.Uma observação a ser feita é que todos os botões raposas ficam acesos quando o mouse estiver em cima.

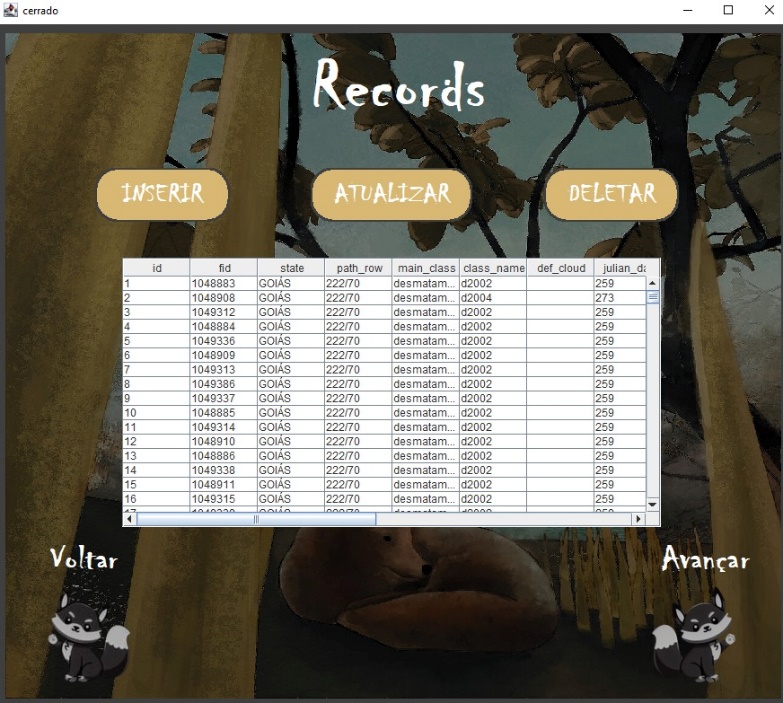




**5.2 RecordsScreen**

Na segunda tela do programa, os usuários são apresentados a uma visualização da tabela "cerrado" do MySQL, integrada à interface por meio de um componente JTable. Esta tabela exibe os dados de forma organizada e interativa. Três botões foram incorporados para facilitar as operações de CRUD (Create, Read, Update, Delete): "Inserir", "Atualizar" e "Deletar". Esses botões permitem ao usuário realizar ações específicas na base de dados, contribuindo para uma experiência de gerenciamento eficiente.

Além disso, dois botões com representações de raposas foram adicionados para proporcionar uma navegação fluída. O primeiro, "Voltar", permite retornar à tela inicial do programa, enquanto o segundo, "Avançar", conduz os usuários para a próxima tela. Essa abordagem intuitiva visa aprimorar a usabilidade, proporcionando ao usuário controle sobre as ações realizadas no programa.

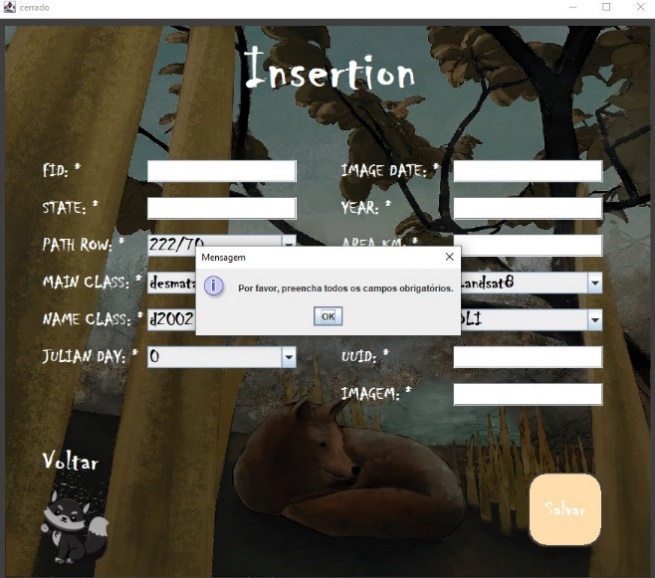
****

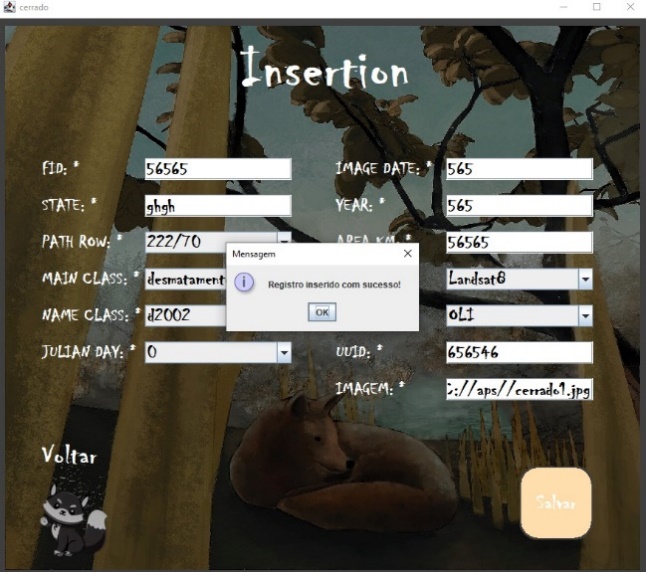
**5.3 InsertionScreen**

Na tela de inserção, os usuários têm a oportunidade de adicionar novos registros à tabela "cerrado" de maneira simples e eficaz. A interface oferece campos de digitação e caixas de opções, possibilitando que o usuário forneça os valores necessários para o novo registro. Uma característica notável é a validação rigorosa para garantir que nenhum campo obrigatório seja deixado em branco.

Se, porventura, o programa detectar a ausência de alguma informação vital, um aviso imediato é acionado, orientando o usuário a preencher todos os campos obrigatórios antes de prosseguir. Essa abordagem visa assegurar a integridade dos dados e uma inserção consistente na base de dados.

Ao preencher corretamente todos os campos, um aviso é exibido, informando ao usuário que o registro foi inserido com sucesso. Dois botões complementam essa tela: o primeiro, "Salvar", permite ao usuário confirmar e armazenar o novo registro após preencher os campos; o segundo, "Voltar", possibilita retornar à tela principal do CRUD, oferecendo flexibilidade de navegação e uma experiência amigável ao usuário.

****

****

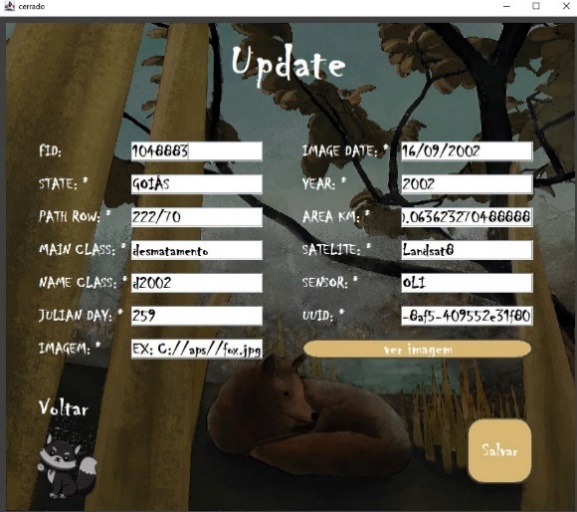
**5.4 UpdateScreen**

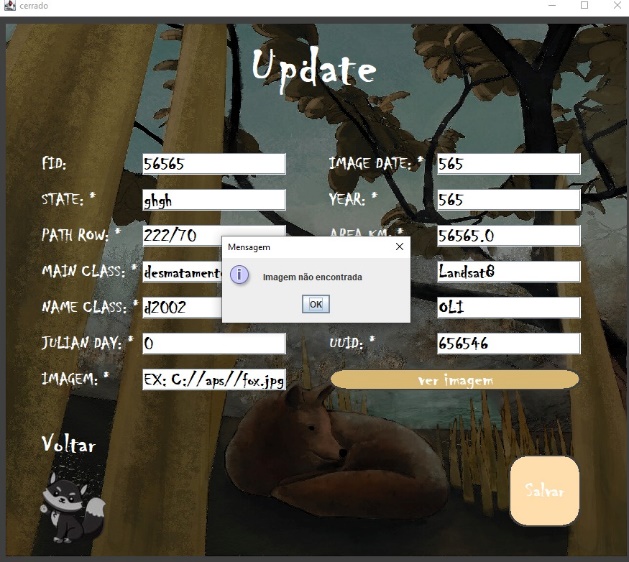
O processo de atualização acontece em duas etapas, proporcionando assim uma abordagem eficiente e organizada na tela de update. Inicialmente, ao clicar no botão "Atualizar", é apresentada uma tela com um único campo de digitação destinado à inserção do ID do registro a ser modificado. Ao acionar o botão "Pesquisar", uma transição ocorre, direcionando o usuário para a segunda tela de update. Caso o id não seja válido, é exibido um aviso dizendo que o id é inválido.

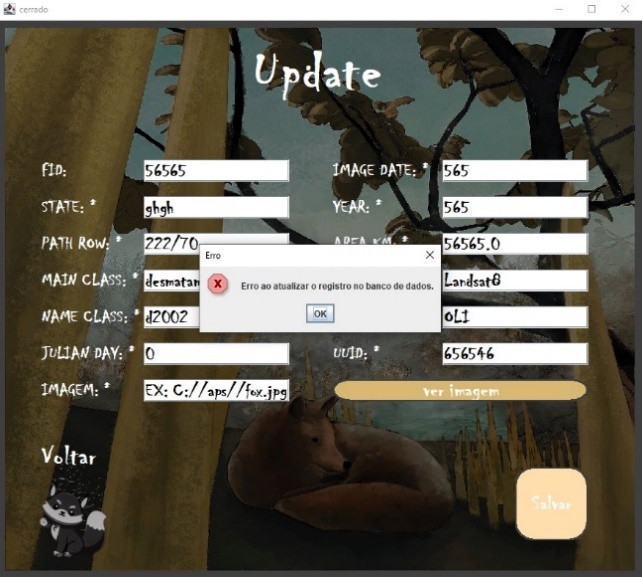
Nessa segunda tela, os campos de digitação e as caixas de opção, agora preenchidos com os dados atuais do registro, oferecem uma visão detalhada da informação a ser atualizada. Uma característica interessante é a inclusão do botão "Ver Imagem", que permite ao usuário visualizar o conteúdo da coluna BLOB do MySQL.

Assim como na tela de inserção, dois botões essenciais são disponibilizados: o botão "Salvar", que confirma e aplica as alterações efetuadas nos campos, e o botão "Voltar", permitindo ao usuário retornar à tela de busca por ID. Essa abordagem visa simplificar o processo de atualização, proporcionando uma experiência coesa e intuitiva ao usuário.

Parte superior do formulário

****

****

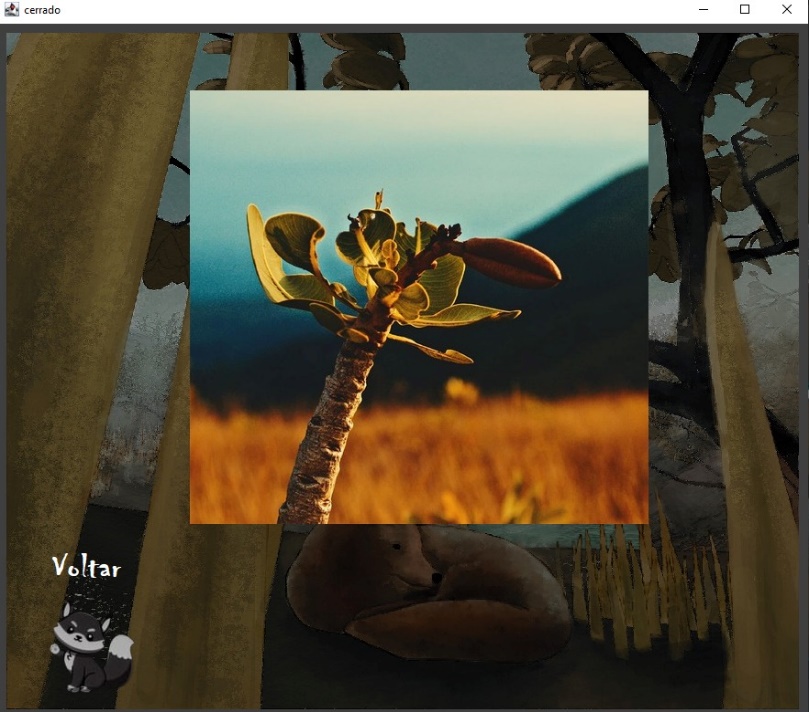
****

**5.5 ImageScreen**

Ao acionar o botão "Ver Imagem" na tela de atualização, uma nova janela é aberta para apresentar a imagem associada ao registro. Se o campo BLOB estiver preenchido, a imagem é exibida no centro da tela. No entanto, se o campo estiver nulo, um quadrado branco com a palavra "null" é apresentado. A presença desse elemento visual oferece ao usuário uma visão clara do conteúdo associado à coluna BLOB no MySQL.

Essa tela adicional também proporciona a opção de retorno à tela de atualização, possibilitando ao usuário ajustar e revisar as informações conforme necessário, garantindo uma experiência de navegação intuitiva e eficiente. O botão "Voltar" proporciona flexibilidade ao usuário durante a visualização da imagem, mantendo a coesão na interação com o sistema.



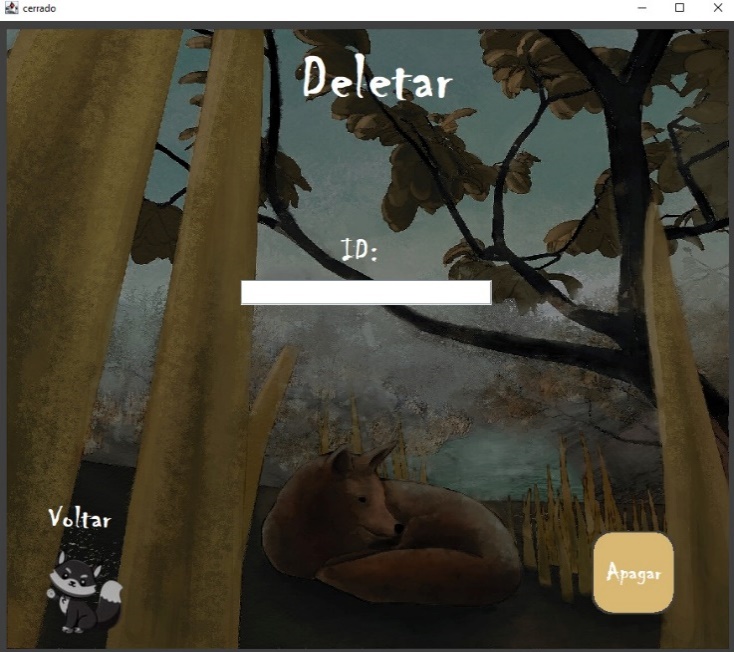
****

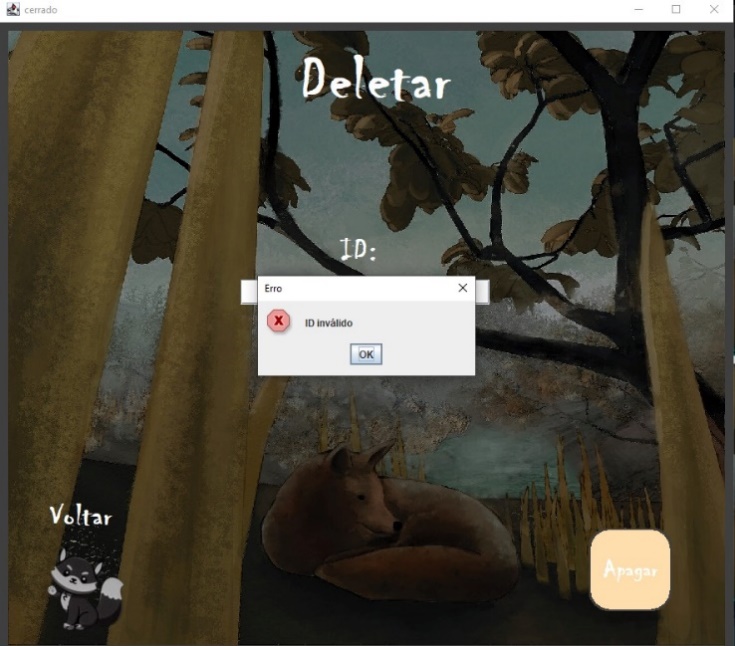
**5.6 DeleteScreen**

Ao clicar no botão "Deletar" na tela do CRUD, uma nova janela é apresentada para efetuar a exclusão de registros. Essa tela conta com um campo de digitação destinado à inserção do ID do registro a ser removido. Se o ID fornecido for inválido, uma mensagem de aviso é imediatamente exibida, alertando o usuário sobre a necessidade de inserir um ID válido.

Ao submeter um ID válido, o programa verifica se o registro correspondente existe no banco de dados. Se o registro não for encontrado, uma mensagem de erro é retornada, indicando que a exclusão não foi realizada com sucesso. Por outro lado, se o registro é localizado e removido com êxito, uma mensagem de confirmação é exibida, comunicando ao usuário que a exclusão foi bem-sucedida.

A presença do botão "Voltar" na tela proporciona ao usuário a flexibilidade de retornar à tela principal do CRUD, mantendo uma navegação coesa e intuitiva no sistema.

****

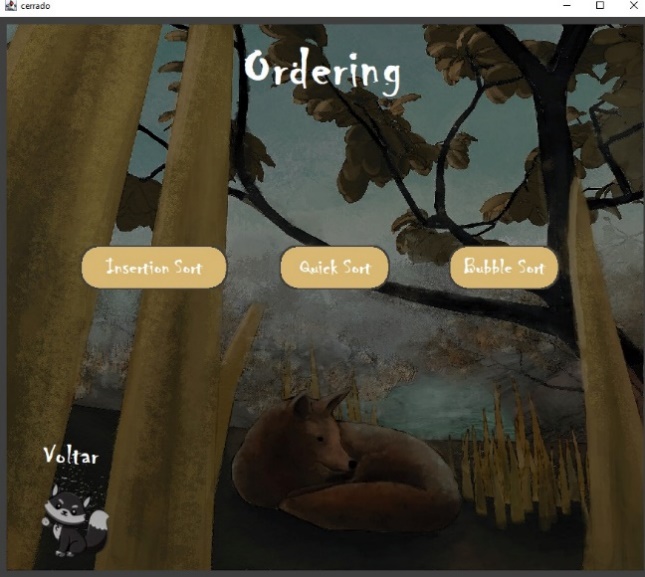
****

**5.7 OrderingScreen**

Após acessar a tela do CRUD, uma etapa subsequente possibilita a ordenação dos registros por meio de diferentes algoritmos. Nessa tela específica, são apresentados três botões redondos centrais, cada um representando um algoritmo de ordenação específico: Insertion Sort, Quick Sort e Bubble Sort.

Essa disposição facilita a escolha do usuário entre os algoritmos disponíveis, proporcionando flexibilidade para optar pelo método de ordenação desejado. Cada botão está associado à execução do respectivo algoritmo, permitindo visualizar os registros ordenados de acordo com a lógica de classificação escolhida.

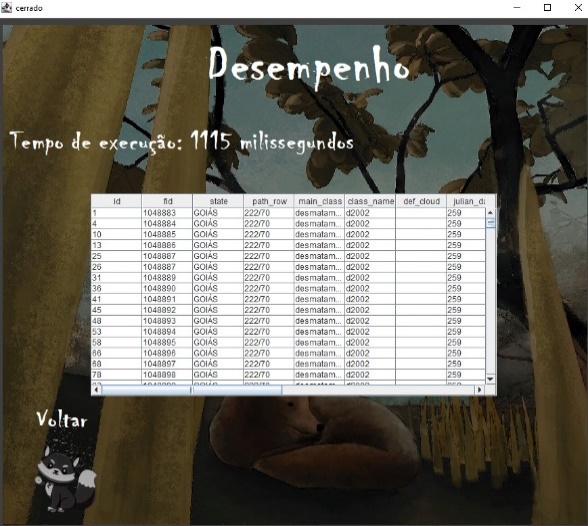
Além dos botões de ordenação, a tela inclui um botão "Raposa" que possibilita o retorno à tela principal do CRUD. Essa opção de voltar contribui para uma navegação fluida e intuitiva, permitindo que o usuário retorne à interface principal a qualquer momento.

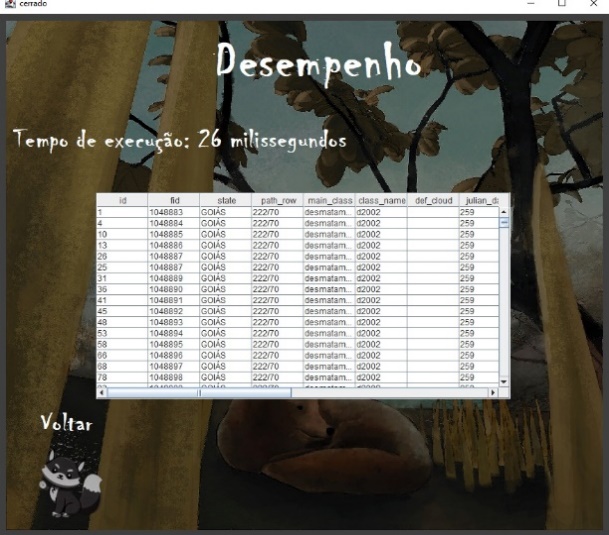


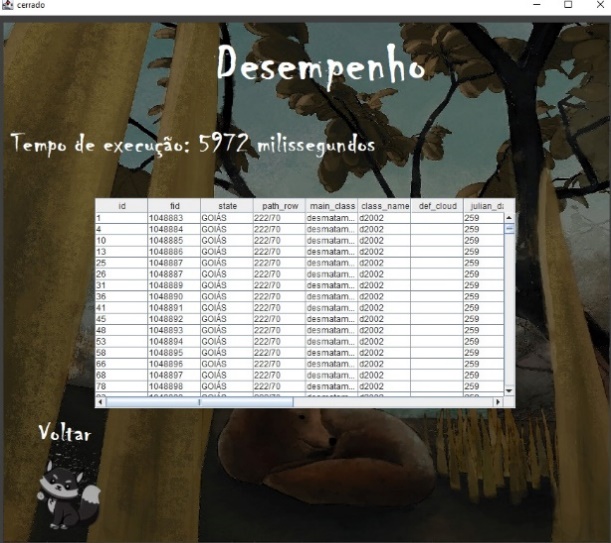
As telas de ordenação proporcionam uma análise detalhada da performance de diferentes algoritmos de ordenação. Ao clicar em cada botão representando um algoritmo específico, o usuário é direcionado para uma tela dedicada à execução desse algoritmo. Cada tela exibe informações cruciais para a avaliação da eficiência, incluindo o tempo de execução em milissegundos e a apresentação da tabela ordenada com base no campo "fid" em formato de JTable.

A disposição dessas informações permite uma comparação direta da eficácia de cada algoritmo, auxiliando na seleção do método mais adequado às necessidades específicas do usuário ou do sistema. A inclusão do botão "Raposa" em cada tela de ordenação facilita o retorno à tela principal de ordenação, proporcionando uma experiência de navegação coesa e intuitiva.

Essa abordagem estruturada das telas de ordenação não apenas oferece uma visão abrangente do desempenho de cada algoritmo, mas também permite uma comparação eficiente, facilitando a tomada de decisões informadas sobre qual algoritmo adotar com base nos requisitos específicos do usuário ou do sistema.

****

****

****

Parte inferior do formulário

**6.0 Considerações Finais**

A conclusão do trabalho desenvolvido reflete não apenas a culminação de esforços dedicados à programação orientada a objetos, mas também uma jornada de aprendizado e superação de desafios. A dupla, ao longo desse processo, demonstrou um comprometimento notável em adquirir conhecimentos em Java, explorando suas capacidades para criar uma aplicação robusta e funcional.

Durante o desenvolvimento do programa, a aplicação de conceitos avançados de programação orientada a objetos foi evidente. A estrutura do código em pacotes, o uso eficaz de bibliotecas externas e a implementação de boas práticas de programação são testemunhos da maturidade técnica alcançada. O trabalho não apenas buscou explorar as características intrínsecas da linguagem Java, mas também abordou desafios específicos, como a integração com banco de dados, manipulação de imagens e estrutura de dados.

O foco na manipulação de banco de dados revelou uma compreensão sólida dos princípios fundamentais de armazenamento e recuperação de dados. A interação com o MySQL para realizar operações CRUD (Create, Read, Update, Delete) demonstrou a capacidade do desenvolvedor de aplicar conceitos teóricos em um contexto prático.

A inclusão de recursos visuais, como a exibição de imagens armazenadas como blobs no MySQL, adicionou uma camada de complexidade ao projeto. A implementação dessa funcionalidade, juntamente com a integração de telas interativas, ilustra a habilidade em lidar com aspectos visuais e interativos de uma aplicação.

Ao enfrentar desafios, como a ordenação eficiente de dados, o grupo demonstrou criatividade e habilidade na implementação de algoritmos conhecidos, como o Insertion Sort, Quick Sort e Bubble Sort. A criação de telas específicas para cada algoritmo, resultando na exibição do tempo de execução e na tabela ordenada, é um exemplo claro do cuidado em oferecer uma experiência informativa e comparativa aos usuários.

Além dos aspectos técnicos, a conclusão do trabalho também enfatiza o desenvolvimento de habilidades interpessoais. A documentação clara e a comunicação eficaz através do texto introdutório evidenciam a importância atribuída à capacidade de comunicar ideias e soluções de maneira acessível.

Em resumo, a conclusão desta jornada reflete não apenas a conclusão de um projeto de programação em Java, mas o início de uma trajetória contínua de aprendizado e aprimoramento. O grupo, ao superar desafios e entregar um projeto significativo, evidenciou não apenas competências técnicas, mas também um comprometimento duradouro com a excelência na arte do desenvolvimento de software.

**7.0 Referências Bibliográficas**

- Udemy (curso de JAVA)

- Site Dev Media

- Site Alura

- Site Stack overflow

- Site [W3Schools](https://www.google.com/search?rlz=1C1ONGR_pt-PTBR1058BR1058&sxsrf=APwXEdfZshfG82hcCJFWzm94TbGpKqC9dA:1685015697149&q=W3Schools&si=AMnBZoEofOODruSEFWFjdccePwMH96ZlZt3bOiKSR9t4pqlu2HLBgqKEqdxf5upVafSDm1RWS5jZa0-Lq-z9gp-p8n7nCtd9z2L8OBkym0JxK8HgVS8ux2tratv8iEbrL1WRvSX25YD9uHIvsR1tzWOhNkQegHxcJmk2LyoSY-e2eeCJUf3EWHtIdppIWtexUwUqR73XQRiGFmrRJ-0NyEFnlfMqxIfTyoUPRUFQ0Rnbq95LsjQRVUzxgNEcMZDD9TiBhps1mDPIHoztR2GhmaV1JMDT3Omwrn1sNsTZTrqxrLazD9zdfKc%3D&sa=X&ved=2ahUKEwjo8cbQtJD_AhWGK7kGHWMzAfgQxA16BAhKEAk)

**8.0 Código Fonte**

**8.1 Pacote** [**algorithm**](https://github.com/andressa-silvaa/data-structure-cerrado/tree/main/src/algorithm)

**8.1.1 BubbleSort**

package algorithm;

import java.util.List;

public class BubbleSort {

public void bubbleSort(List<String[]> data) {

int n = data.size();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

// Compare as colunas 'fid' para ordenar pela coluna 'fid'

if (data.get(j)[1].compareTo(data.get(j + 1)[1]) > 0) {

// Troca os elementos se estiverem fora de ordem

replace(data, j, j + 1);

}

}

}

}

private void replace(List<String[]> data, int i, int j) {

String[] temp = data.get(i);

data.set(i, data.get(j));

data.set(j, temp);

}

}

**8.1.2 InsertionSort**

package algorithm;

import java.util.List;

public class InsertionSort {

// Método público para iniciar a ordenação por Insertion Sort.

public void insertionSort(List<String[]> data) {

for (int j = 1; j < data.size(); j++) {

String[] chave = data.get(j);

int i = j - 1;

while (i >= 0 && Integer.parseInt(data.get(i)[1]) > Integer.parseInt(chave[1])) {

data.set(i + 1, data.get(i));

i--;

}

data.set(i + 1, chave);

}

}

}

**8.1.3 QuickySort**

package algorithm;

import java.util.List;

public class QuickSort {

public static void quickSort(List<String[]> data) {

quickSort(data, 0, data.size() - 1);

}

private static void quickSort(List<String[]> data, int left, int right) {

int leftIndex = left;

int rightIndex = right;

String[] pivot = data.get((left + right) / 2);

String[] temp;

while (leftIndex <= rightIndex) {

while (data.get(leftIndex)[1].compareTo(pivot[1]) < 0) {

leftIndex++;

}

while (data.get(rightIndex)[1].compareTo(pivot[1]) > 0) {

rightIndex--;

}

if (leftIndex <= rightIndex) {

temp = data.get(leftIndex);

data.set(leftIndex, data.get(rightIndex));

data.set(rightIndex, temp);

leftIndex++;

rightIndex--;

}

}

if (rightIndex > left) {

quickSort(data, left, rightIndex);

}

if (leftIndex < right) {

quickSort(data, leftIndex, right);

}

}

}

**8.2 Pacote Backend**

**8.2.1 DeleteLogic**

package backend;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import dataBase.Db;

/\*A classe DeleteLogic é definida, e ela implementa a interface AutoCloseable.

\* Isso significa que a classe pode ser usada em um bloco try-with-resources,

\* onde os recursos serão fechados automaticamente ao sair do bloco.

\*/

public class DeleteLogic implements AutoCloseable{

private Connection connection;

//método para deletar registro do banco

public static boolean deleteRecord(int id) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

//Verifica se a variável connection é nula ou se a conexão está fechada.

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

//query contendo a consulta SQL de exclusão.

String query = "DELETE FROM cerrado WHERE ID = ?";

//é preparado um PreparedStatement associado a essa consulta

preparedStatement = connection.prepareStatement(query);

//o valor do parâmetro id é definido

preparedStatement.setInt(1, id);

//A consulta é executada usando o método executeUpdate(),

//que retorna o número de linhas afetadas.

int rowsDeleted = preparedStatement.executeUpdate();

//O método retorna true se pelo menos uma linha foi excluída

return rowsDeleted > 0;

} catch (SQLException e) {

//Em caso de exceção, o erro é impresso no console (usando e.printStackTrace())

//e o método retorna false para indicar que a exclusão não foi bem-sucedida.

e.printStackTrace();

return false;

} finally {

Db.closeStatement(preparedStatement);

}

}

@Override

public void close() throws Exception {

//Implementação do método close da interface AutoCloseable.

//Esse método é usado para fechar a conexão com o banco de dados.

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.2.2 InsertionLogic**

package backend;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import javax.swing.JComboBox;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

import dataBase.Db;

/\*A classe InsertionLogic é definida, e ela implementa a interface AutoCloseable.

\* Isso significa que a classe pode ser usada em um bloco try-with-resources,

\* onde os recursos serão fechados automaticamente ao sair do bloco.

\*/

public class InsertionLogic implements AutoCloseable {

private static Connection connection = null;

public InsertionLogic() {

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

//O método insertRecord é definido para inserir um registro no banco de dados.

public boolean insertRecord(

JTextField fid, JTextField state, JComboBox<String> pathRow,

JComboBox<String> mainClass, JComboBox<String> nameClass, JComboBox<String> julianDay,

JTextField imageDate, JTextField year, JTextField areaKm, JComboBox<String> satelite, JComboBox<String> sensor, JTextField uuid, JTextField imagem) {

try {

//É verificado se todos os campos passados para o método estão preenchidos.

if (areFieldsFilled(fid, state, pathRow, mainClass, nameClass, julianDay, imageDate, year, areaKm, satelite, sensor, uuid, imagem)) {

//É criada a consulta SQL para a inserção

String insertSQL = "INSERT INTO cerrado (FID, STATE, PATH\_ROW, MAIN\_CLASS, CLASS\_NAME, JULIAN\_DAY, IMAGE\_DATE, YEAR, AREA\_KM, satellite, SENSOR, UUID, IMAGEM) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,?)";

//é preparado um objeto PreparedStatement associado a essa consulta.

PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);

//Os valores dos campos são definidos nos parâmetros da consulta

String areaKmText = areaKm.getText().replace(",", ".");

preparedStatement.setString(1, fid.getText());

preparedStatement.setString(2, state.getText());

preparedStatement.setString(3, pathRow.getSelectedItem().toString());

preparedStatement.setString(4, mainClass.getSelectedItem().toString());

preparedStatement.setString(5, nameClass.getSelectedItem().toString());

preparedStatement.setString(6, julianDay.getSelectedItem().toString());

preparedStatement.setString(7, imageDate.getText());

preparedStatement.setString(8, year.getText());

preparedStatement.setDouble(9, Double.parseDouble(areaKmText));

preparedStatement.setString(10, satelite.getSelectedItem().toString());

preparedStatement.setString(11, sensor.getSelectedItem().toString());

preparedStatement.setString(12, uuid.getText());

// Ler o conteúdo do arquivo de imagem e definir como BLOB

String imagePath = imagem.getText();

try {

FileInputStream imagemStream = new FileInputStream(imagePath);

preparedStatement.setBinaryStream(13, imagemStream, imagemStream.available());

} catch (FileNotFoundException e) {

JOptionPane.showMessageDialog(null, "Imagem não encontrada");

e.printStackTrace();

} catch (IOException e) {

JOptionPane.showMessageDialog(null, "Erro ao inserir registro no banco de dados.");

e.printStackTrace();

}

//A consulta é executada usando o método executeUpdate()

//e o objeto PreparedStatement é fechado.

preparedStatement.executeUpdate();

preparedStatement.close();

//Se a inserção for bem-sucedida, uma mensagem de sucesso é exibida

//com JOptionPane, e o método retorna true.

JOptionPane.showMessageDialog(null, "Registro inserido com sucesso!");

return true;

} else {

//Se algum campo estiver vazio, a função showMissingFieldsMessage()

//é chamada para exibir uma mensagem de erro, e o método retorna false.

showMissingFieldsMessage();

return false;

}

} catch (SQLException ex) {

ex.printStackTrace();

//Em caso de exceção, o erro é impresso no console e uma mensagem de erro é exibida

//com JOptionPane, e o método retorna false.

JOptionPane.showMessageDialog(null, "Erro ao inserir registro no banco de dados.");

return false;

}

}

//O método areFieldsFilled verifica se os campos estão preenchidos.

//Ele aceita um número variável de campos como argumentos.

public static boolean areFieldsFilled(Object... fields) {

for (Object field : fields) {

if (field instanceof JTextField) {

if (((JTextField)field).getText().isEmpty()) {

return false;

}

} else if (field instanceof JComboBox) {

if (((JComboBox<?>)field).getSelectedItem() == null) {

return false;

}

}

}

return true;

}

//O método showMissingFieldsMessage exibe uma mensagem de erro

//informando que campos obrigatórios não estão preenchidos.

public static void showMissingFieldsMessage() {

JOptionPane.showMessageDialog(null, "Por favor, preencha todos os campos obrigatórios.");

}

@Override

public void close() throws Exception {

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.2.3 Records**

package backend;

import java.io.InputStream;

public class Records {

private String fid;

private String state;

private String pathRow;

private String mainClass;

private String nameClass;

private int julianDay;

private String imageDate;

private int year;

private String areaKm;

private String satellite;

private String sensor;

private String uuid;

private String imagem;

private InputStream blob;

public Records(String fid, String state, String pathRow, String mainClass, String nameClass, int julianDay,

String imageDate, int year, String areaKm, String satellite, String sensor, String uuid, String imagem, InputStream blob) {

this.fid = fid;

this.state = state;

this.pathRow = pathRow;

this.mainClass = mainClass;

this.nameClass = nameClass;

this.julianDay = julianDay;

this.imageDate = imageDate;

this.year = year;

this.areaKm = areaKm;

this.satellite = satellite;

this.sensor = sensor;

this.uuid = uuid;

this.imagem = imagem;

this.blob = blob;

}

public String getFid() {

return fid;

}

public String getState() {

return state;

}

public String getPathRow() {

return pathRow;

}

public String getMainClass() {

return mainClass;

}

public String getNameClass() {

return nameClass;

}

public int getJulianDay() {

return julianDay;

}

public String getImageDate() {

return imageDate;

}

public int getYear() {

return year;

}

public String getAreaKm() {

return areaKm;

}

public String getSatellite() {

return satellite;

}

public String getSensor() {

return sensor;

}

public String getUuid() {

return uuid;

}

public String getImagem() {

return imagem;

}

public InputStream getBlob() {

return blob;

}

}

**8.2.4 UpdateLogic**

package backend;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.sql.Blob;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.swing.JOptionPane;

import dataBase.Db;

import utils.ConversorBlob;

/\*A classe UpdateLogic é definida, e ela implementa a interface AutoCloseable.

\* Isso significa que a classe pode ser usada em um bloco try-with-resources,

\* onde os recursos serão fechados automaticamente ao sair do bloco.

\*/

public class UpdateLogic implements AutoCloseable {

private static Connection connection;

//O método validateId verifica se um registro existe no banco de dados.

public static boolean validateId(int id) {

connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

//O PreparedStatement é configurado com o ID a ser verificado.

String query = "SELECT \* FROM cerrado WHERE ID = ?";

preparedStatement = connection.prepareStatement(query);

preparedStatement.setInt(1, id);

//A consulta é executada com executeQuery(), e o resultado é verificado

//usando resultSet.next(). Se houver um próximo resultado,

//isso significa que o registro com o ID existe, e a função retorna true.

resultSet = preparedStatement.executeQuery();

return resultSet.next();

} catch (SQLException e) {

e.printStackTrace();

return false;

} finally {

Db.closeStatement(preparedStatement);

Db.closeResultSet(resultSet);

}

}

//método responsável por atualizar um registro no banco de dados.

public static boolean updateRecord(int id, int fid, String state, String pathRow, String mainClass, String nameClass, int julianDay, String imageDate, int year, String areaKm, String satellite, String sensor, String uuid, String imagem) {

Connection connection = null;

PreparedStatement preparedStatement = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

//O PreparedStatement é configurado com a query a ser executada.

String query = "UPDATE cerrado SET FID = ?, STATE = ?, PATH\_ROW = ?, MAIN\_CLASS = ?, CLASS\_NAME = ?, JULIAN\_DAY = ?, IMAGE\_DATE = ?, YEAR = ?, AREA\_KM = ?, satellite = ?, SENSOR = ?, UUID = ? , IMAGEM = ? WHERE ID = ?";

preparedStatement = connection.prepareStatement(query);

//Os valores dos campos são definidos nos parâmetros da consulta

String areaKmText = areaKm.replace(",", ".");

preparedStatement.setInt(1, fid);

preparedStatement.setString(2, state);

preparedStatement.setString(3, pathRow);

preparedStatement.setString(4, mainClass);

preparedStatement.setString(5, nameClass);

preparedStatement.setInt(6, julianDay);

preparedStatement.setString(7, imageDate);

preparedStatement.setInt(8, year);

preparedStatement.setDouble(9, Double.parseDouble(areaKmText));

preparedStatement.setString(10, satellite);

preparedStatement.setString(11, sensor);

preparedStatement.setString(12, uuid);

preparedStatement.setInt(14, id);

String imagePath = imagem;

try {

FileInputStream imagemStream = new FileInputStream(imagePath);

preparedStatement.setBinaryStream(13, imagemStream, imagemStream.available());

} catch (FileNotFoundException e) {

JOptionPane.showMessageDialog(null, "Imagem não encontrada");

e.printStackTrace();

} catch (IOException e) {

JOptionPane.showMessageDialog(null, "Erro ao inserir registro no banco de dados.");

e.printStackTrace();

}

//salva a quantidade de linhas afetadas pela execução da query

int rowsUpdated = preparedStatement.executeUpdate();

//retorna true se forem feitas mudanças no banco de dados

return rowsUpdated > 0;

} catch (SQLException e) {

//Caso haja uma execessão, é retornado false

e.printStackTrace();

return false;

} finally {

Db.closeStatement(preparedStatement);

}

}

public static Records getRegistroPorId(int id) {

Connection connection = null;

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

Records records = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

//string SQL que será usada para consultar o banco de dados.

String query = "SELECT \* FROM cerrado WHERE ID = ?";

preparedStatement = connection.prepareStatement(query);

//O valor do ID fornecido como argumento é definido no PreparedStatement

//para substituir o ponto de interrogação na consulta SQL.

preparedStatement.setInt(1, id);

//A consulta é executada no banco de dados

resultSet = preparedStatement.executeQuery();

String imagemString = "EX: C://aps//fox.jpg";

//verifica-se se o resultSet contém algum resultado.

if (resultSet.next()) {

//os dados do registro são recuperados do resultSet e

//são usados para criar um novo objeto Records.

records = new Records(

resultSet.getString("FID"),

resultSet.getString("STATE"),

resultSet.getString("PATH\_ROW"),

resultSet.getString("MAIN\_CLASS"),

resultSet.getString("CLASS\_NAME"),

resultSet.getInt("JULIAN\_DAY"),

resultSet.getString("IMAGE\_DATE"),

resultSet.getInt("YEAR"),

resultSet.getString("AREA\_KM"),

resultSet.getString("SATELLITE"),

resultSet.getString("SENSOR"),

resultSet.getString("UUID"),

imagemString,

resultSet.getBinaryStream("IMAGEM"));

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

Db.closeStatement(preparedStatement);

Db.closeResultSet(resultSet);

}

//retorna os registros do id fornecido

return records;

}

@Override

public void close() throws Exception {

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.3 Pacote database**

**8.3.1 Db**

package dataBase;

import java.io.FileInputStream;

import java.io.IOException;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Properties;

public class Db {

private static Connection conn = null;

private static Properties loadProperties() {

//leitura do arquivo db.properties usando um objeto FileInputStream

try (FileInputStream fs = new FileInputStream("db.properties")) {

//Cria um objeto Properties que será usado para armazenar as propriedades do banco de dados.

Properties props = new Properties();

//Carrega as propriedades do banco de dados a partir do arquivo db.properties.

props.load(fs);

return props;

}

catch (IOException e) {

//Lança uma exceção personalizada DbException passando a mensagem de erro da exceção IOException

throw new DbException(e.getMessage());

}

}

public static Connection getConnection() {

//Verifica se a conexão conn é nula.

if (conn == null) {

try {

//chama o método loadProperties para carregar as propriedades do banco de dados a partir do arquivo db.properties.

Properties props = loadProperties();

//Obtém a URL de conexão com o banco de dados a partir das propriedades carregadas.

String url = props.getProperty("dburl");

//Estabelece a conexão com o banco de dados usando a classe DriverManager e as informações de URL e propriedades.

conn = DriverManager.getConnection(url, props);

}

catch (SQLException e) {

//Lança uma exceção personalizada DbException passando a mensagem de erro da exceção SQLException.

throw new DbException(e.getMessage());

}

}

return conn;

}

public static void closeConnection(Connection conn) {

//Verifica se a conexão conn não é nula.

if (conn != null) {

try {

//Fecha a conexão usando o método close() da classe Connection.

conn.close();

} catch (SQLException e) {

//Lança uma exceção personalizada DbException passando a mensagem de erro da exceção SQLException.

throw new DbException(e.getMessage());

}

}

}

public static void closeStatement(Statement st) {

//Verifica se o objeto Statement st não é nulo.

if (st != null) {

try {

//Fecha o objeto Statement usando o método close().

st.close();

} catch (SQLException e) {

//Lança uma exceção personalizada DbException, passando a mensagem de erro da exceção SQLException.

throw new DbException(e.getMessage());

}

}

}

public static void closeResultSet(ResultSet rs) {

//Verifica se o objeto ResultSet rs não é nulo.

if (rs != null) {

try {

//Fecha o objeto ResultSet usando o método close().

rs.close();

} catch (SQLException e) {

//Lança uma exceção personalizada DbException, passando a mensagem de erro da exceção SQLException.

throw new DbException(e.getMessage());

}

}

}

}

**8.3.2 DbException**

package dataBase;

public class DbException extends RuntimeException {

private static final long serialVersionUID = 1L;

public DbException(String msg) {

super(msg);

}

}

**8.4 Pacote main**

**8.4.1 Main**

package main;

import java.io.IOException;

import javax.swing.JFrame;

import ui.HomeScreen;

public class Main {

public static void main(String[] args) throws IOException {

HomeScreen frame = new HomeScreen();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

}

**8.5 Pacote ui**

**8.5.1 BaseFrame**

package ui;

import javax.swing.\*;

import java.awt.\*;

public abstract class BaseFrame extends JFrame {

private static final long serialVersionUID = 1L;

public BaseFrame() {

super("cerrado");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(900, 800);

//centraliza o frame na tela do monitor

setLocationRelativeTo(null);

try {

//cria a borda do frame

getRootPane().setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 10));

} catch (Exception e) {

e.printStackTrace();

}

initializeComponents();

}

public BaseFrame(Image background) {

//chama o construtor padrão para as demais configurações do frame

this();

//um novo painel (JPanel) é definido como o conteúdo da janela

setContentPane(new JPanel() {

private static final long serialVersionUID = 1L;

@Override

//Esse método é chamado pelo sistema de gráficos Swing para desenhar o conteúdo do painel.

//Ele recebe um objeto Graphics chamado g,

//que é usado para desenhar elementos gráficos na tela.

protected void paintComponent(Graphics g) {

super.paintComponent(g);

//desenha a imagem no fundo do frame

g.drawImage(background, 0, 0, getWidth(), getHeight(), this);

}

});

}

//método geral para os componentes da tela

public abstract void initializeComponents();

}

**8.5.2 DeleteScreen**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import backend.DeleteLogic;

import utils.CustomButton;

import utils.UiHelper;

public class DeleteScreen extends BaseFrame {

//define a interface do usuário (UI) para a tela de exclusão de registros.

//Aqui, a imagem de fundo é carregada a partir de um arquivo.

public DeleteScreen() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

}

@Override

public void initializeComponents() {

//Aqui são carregadas três imagens diferentes para os estados normais,

//de hover e de clique de um botão personalizado.

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

//Um botão personalizado chamado buttonimg1 é criado usando as imagens

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

/\*Um ouvinte de ação (ActionListener) é adicionado ao botão buttonimg1.

\*Quando esse botão é clicado, ele fecha a tela atual (this) e abre a tela RecordsScreen.

\*/

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

RecordsScreen recordsScreen = new RecordsScreen();

recordsScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

recordsScreen.setLocationRelativeTo(null);

recordsScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//define a posição e o tamanho do buttonimg1 na tela.

buttonimg1.setBounds(10, 600, 150, 150);

// criação de alguns componentes de texto, botao..etc

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Deletar", 350, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JButton buttonSubmit = UiHelper.createRoundButton("Apagar", 700, 600, 100, 100, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JLabel idLabel = UiHelper.createLabel("ID: ", 400, 250, 100, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JTextField id = UiHelper.createTextField(280, 300, 300, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

/\*Um ouvinte de ação é adicionado ao botão buttonSubmit.

\* Quando esse botão é clicado, ele lê o ID inserido,

\* verifica se é válido, tenta excluir o registro com esse ID

\* chamando o método deleteRecord da classe DeleteLogic.

\* Em seguida, exibe mensagens de confirmação ou erro.

\*/

buttonSubmit.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String idStr = id.getText();

if (idStr.isEmpty() || !idStr.matches("\\d+")) {

JOptionPane.showMessageDialog(null, "ID inválido", "Erro", JOptionPane.ERROR\_MESSAGE);

return;

}

int idValue = Integer.parseInt(idStr);

if (DeleteLogic.deleteRecord(idValue)) {

JOptionPane.showMessageDialog(null, "Registro excluído com sucesso!");

id.setText("");

} else {

JOptionPane.showMessageDialog(null, "Erro ao excluir o registro no banco de dados.", "Erro", JOptionPane.ERROR\_MESSAGE);

}

}

});

//os componentes criados são adicionados ao painel de conteúdo da janela .

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(buttonimg1);

getContentPane().add(buttonSubmit);

getContentPane().add(idLabel);

getContentPane().add(id);

}

}

**8.5.3 HomeScreen**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextArea;

import javax.swing.text.DefaultStyledDocument;

import javax.swing.text.SimpleAttributeSet;

import javax.swing.text.StyleConstants;

import utils.CustomButton;

import utils.UiHelper;

public class HomeScreen extends BaseFrame {

public HomeScreen() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

initializeComponents();

}

@Override

public void initializeComponents() {

//três imagens diferentes são carregadas para definir os estados normais,

//de hover e de clique de um botão personalizado.

final ImageIcon regular = new ImageIcon("C:\\aps\\foxbutton1.png");

final ImageIcon hovering = new ImageIcon("C:\\aps\\foxbutton.png");

final ImageIcon clicking = new ImageIcon("C:\\aps\\foxbutton.png");

//Um botão personalizado chamado customButton é criado e posicionado na tela.

CustomButton customButton = new CustomButton(regular, hovering, clicking);

customButton.setBounds(700, 600, 150, 150);

//Um ouvinte de ação é adicionado ao customButton.

//Quando o botão é clicado, ele fecha a tela atual,

//e abre a tela RecordsScreen.

customButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

RecordsScreen recordsScreen = new RecordsScreen ();

recordsScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

recordsScreen.setLocationRelativeTo(null);

recordsScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//Um componente de área de texto chamado textArea é criado.

JTextArea textArea = UiHelper.createTextArea(10, 200, 800, 300,new Font("Chiller", Font.BOLD, 40), Color.WHITE);

//Um documento de estilo padrão é definido para o textArea.

DefaultStyledDocument doc = new DefaultStyledDocument();

textArea.setDocument(doc);

//O texto no textArea é alinhado à esquerda.

SimpleAttributeSet leftAlign = new SimpleAttributeSet();

StyleConstants.setAlignment(leftAlign, StyleConstants.ALIGN\_LEFT);

doc.setParagraphAttributes(0, doc.getLength(), leftAlign, false);

//Texto é definido no textArea.

textArea.setText("O Cerrado, "

+ "bioma brasileiro importante, "

+ "enfrenta problemas ambientais graves, "

+ "como desmatamento ilegal, "

+ "queimadas e perda de habitat de espécies ameaçadas."

+ " Sua preservação é crucial para o equilíbrio ambiental"

+ " e proteção da biodiversidade.");

//Um painel de conteúdo é obtido a partir do conteúdo da janela

//e seu layout é definido como nulo,

//permitindo a colocação manual de componentes na tela.

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//Dois rótulos (titleLabel e nextLabel) são criados

JLabel titleLabel = UiHelper.createLabel("Cerrado e Crime Ambiental", 80, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JLabel nextLabel = UiHelper.createLabel("Avançar", 730, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

//Os componentes criados são adicionados ao painel de conteúdo.

contentPane.add(customButton);

contentPane.add(titleLabel);

contentPane.add(nextLabel);

contentPane.add(textArea);

}

}

**8.5.4 ImageScreen**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.Image;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import backend.Records;

import backend.UpdateLogic;

import utils.ConversorBlob;

import utils.CustomButton;

import utils.UiHelper;

public class ImageScreen extends BaseFrame{

private int id;

public ImageScreen(int id) throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

this.id = id;

getContentPane().setBackground(Color.WHITE);

initializeComponents();

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

UpdateScreen\_2 updateScreen\_2 = new UpdateScreen\_2(id);

updateScreen\_2.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

updateScreen\_2.setLocationRelativeTo(null);

updateScreen\_2.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

buttonimg1.setBounds(10, 600, 150, 150);

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

Records records = UpdateLogic.getRegistroPorId(id);

if (records != null) {

InputStream imageData = records.getBlob();

if (imageData != null) {

// Converte os dados binários em uma imagem.

Image image = ConversorBlob.inputStreamToImage(imageData);

// Cria um JLabel para exibir a imagem centralizada.

JLabel imageLabel = new JLabel(new ImageIcon(image));

imageLabel.setBounds(200, 50, 500, 500);

getContentPane().add(imageLabel);

}else {

JPanel whiteSquare = new JPanel();

whiteSquare.setBackground(Color.WHITE);

whiteSquare.setBounds(200, 50, 500, 500);

whiteSquare.setLayout(null);

getContentPane().add(whiteSquare);

JLabel nullLabel = new JLabel("NULL");

nullLabel.setFont(new Font("Chiller", Font.BOLD, 70));

nullLabel.setForeground(Color.BLACK);

nullLabel.setBounds(200, 150, 150, 150);

whiteSquare.add(nullLabel);

}

}

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

getContentPane().add(goBackLabel);

getContentPane().add(buttonimg1);

}

}

**8.5.5 InsertionScreen**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.sql.Connection;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

import backend.InsertionLogic;

import dataBase.Db;

import utils.CustomButton;

import utils.UiHelper;

public class InsertionScreen extends BaseFrame {

//o construtor InsertionScreen é criado.

//Ele recebe uma imagem como plano de fundo,

//define o plano de fundo do painel principal como branco

//e chama o método initializeComponents()

//para configurar os componentes da tela de inserção.

public InsertionScreen() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

}

@Override

public void initializeComponents() {

//3 imagens para os botoes sao definidas de acordo com o estado do mouse

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

//um botao imagem é criado

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

//é definido uma ação a esse botão, quando clicado ele dispensa a tela atual,

//e abre uma nova tela do tipo RecordsScreen

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try{

RecordsScreen recordsScreen = new RecordsScreen();

recordsScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

recordsScreen.setLocationRelativeTo(null);

recordsScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//é definido a posição do botão

buttonimg1.setBounds(10, 600, 150, 150);

//componentes da tela(rótulos,botões)....

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Insertion", 320, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JLabel fidLabel = UiHelper.createLabel("FID: \*", 50, 180, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel stateLabel = UiHelper.createLabel("STATE: \*", 50, 230, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel pathRowLabel = UiHelper.createLabel("PATH ROW: \*", 50, 280, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel mainClassLabel = UiHelper.createLabel("MAIN CLASS: \*", 50, 330, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel nameCLassLabel = UiHelper.createLabel("NAME CLASS: \*", 50, 380, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel julianDayLabel = UiHelper.createLabel("JULIAN DAY: \*", 50, 430, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel imageDateLabel = UiHelper.createLabel("IMAGE DATE: \*", 450, 180, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel yearLabel = UiHelper.createLabel("YEAR: \*", 450, 230, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel areaKmLabel = UiHelper.createLabel("AREA KM: \*", 450, 280, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel sateliteLabel = UiHelper.createLabel("SATELITE: \*", 450, 330, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel sensorLabel = UiHelper.createLabel("SENSOR: \*", 450, 380, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel uuidLabel = UiHelper.createLabel("UUID: \*", 450, 430, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel imagemLabel = UiHelper.createLabel("IMAGEM: \*", 450, 480, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JButton buttonSave = UiHelper.createRoundButton("Salvar", 700, 600, 100, 100, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

//criação de textfields

JTextField fid = UiHelper.createTextField(190, 180, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

JTextField state = UiHelper.createTextField(190, 230, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

JTextField imageDate = UiHelper.createTextField(600, 180, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

JTextField year = UiHelper.createTextField(600, 230, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

JTextField areaKm = UiHelper.createTextField(600, 280, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

JTextField uuid = UiHelper.createTextField(600, 430, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

JTextField imagem = UiHelper.createTextField(600, 480, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

//definição das opções de escolha para os elementos ComboBox

String[] pathRowOptions = {"222/70"};

String[] mainClassOptions = {"desmatamento"};

String[] nameClassOptions = {"d2002"};

String[] sateliteOptions = {"Landsat8"};

String[] sensorOptions = {"OLI"};

String[] julianDayOptions = {"0"};

//criação dos elementos tipo ComboBox

JComboBox<String> pathRow = UiHelper.createComboBox(pathRowOptions, 190, 280, 200, 30, new Font("Chiller", Font.BOLD, 26),Color.BLACK);

JComboBox<String> mainClass = UiHelper.createComboBox(mainClassOptions, 190, 330, 200, 30, new Font("Chiller", Font.BOLD, 26),Color.BLACK);

JComboBox<String> nameClass = UiHelper.createComboBox(nameClassOptions, 190, 380, 200, 30, new Font("Chiller", Font.BOLD, 26),Color.BLACK);

JComboBox<String> satelite = UiHelper.createComboBox(sateliteOptions, 600, 330, 200, 30, new Font("Chiller", Font.BOLD, 26),Color.BLACK);

JComboBox<String> sensor = UiHelper.createComboBox(sensorOptions, 600, 380, 200, 30, new Font("Chiller", Font.BOLD, 26),Color.BLACK);

JComboBox<String> julianDay = UiHelper.createComboBox(julianDayOptions, 190, 430, 200, 30, new Font("Chiller", Font.BOLD, 26),Color.BLACK);

//evento para o botão de salvar, quando clicado, é feita uma inserção

//com os métodos de outras classes

buttonSave.addActionListener(e -> {

InsertionLogic insertionLogic = new InsertionLogic();

boolean inseridoComSucesso = insertionLogic.insertRecord(

fid, state, pathRow, mainClass, nameClass, julianDay,

imageDate, year, areaKm, satelite, sensor, uuid,imagem);

//caso a inserção ocorra, os campos de textos são resetados

if (inseridoComSucesso) {

fid.setText("");

state.setText("");

pathRow.setSelectedIndex(0);

mainClass.setSelectedIndex(0);

nameClass.setSelectedIndex(0);

julianDay.setSelectedIndex(0);

imageDate.setText("");

year.setText("");

areaKm.setText("");

satelite.setSelectedIndex(0);

sensor.setSelectedIndex(0);

uuid.setText("");

imagem.setText("");

}

});

//remoção de layout, para posicionar elementos manualmente

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//adicionando os elementos a janela.

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(buttonimg1);

getContentPane().add(fid);

getContentPane().add(state);

getContentPane().add(pathRow);

getContentPane().add(mainClass);

getContentPane().add(nameClass);

getContentPane().add(julianDay);

getContentPane().add(imageDate);

getContentPane().add(year);

getContentPane().add(areaKm);

getContentPane().add(satelite);

getContentPane().add(sensor);

getContentPane().add(uuid);

getContentPane().add(fidLabel);

getContentPane().add(stateLabel);

getContentPane().add(pathRowLabel);

getContentPane().add(mainClassLabel);

getContentPane().add(nameCLassLabel);

getContentPane().add(julianDayLabel);

getContentPane().add(imageDateLabel);

getContentPane().add(yearLabel);

getContentPane().add(areaKmLabel);

getContentPane().add(sateliteLabel);

getContentPane().add(sensorLabel);

getContentPane().add(uuidLabel);

getContentPane().add(buttonSave);

getContentPane().add(imagemLabel);

getContentPane().add(imagem);

}

}

**8.5.6 Ordering**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import utils.CustomButton;

import utils.UiHelper;

public class Ordering extends BaseFrame {

//O construtor Ordering é criado. Ele define a imagem de fundo

public Ordering() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

// // Configuração do botão "Voltar" para retornar à tela de registros

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

RecordsScreen recordsScreen = new RecordsScreen();

recordsScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

recordsScreen.setLocationRelativeTo(null);

recordsScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//posicionamento do botão na tela

buttonimg1.setBounds(10, 600, 150, 150);

//criação de rótulos e botões

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Ordering", 320, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JButton insertionSort = UiHelper.createRoundButton("Insertion Sort", 100, 300, 200, 60, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JButton quickSort = UiHelper.createRoundButton("Quick Sort", 370, 300, 150, 60, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JButton bubbleSort = UiHelper.createRoundButton("Bubble Sort", 600, 300, 150, 60, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

//É configurado um ouvinte de ação para o botão "Merge Sort"

//para abrir uma tela de desempenho quando o botão é pressionado.

insertionSort.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

PerformanceInsertionSort performanceFrame = new PerformanceInsertionSort();

performanceFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

performanceFrame.setLocationRelativeTo(null);

performanceFrame.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//É configurado um ouvinte de ação para o botão "Quick Sort"

//para abrir uma tela de desempenho quando o botão é pressionado.

quickSort.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

PerformanceQuickSort performanceFrame = new PerformanceQuickSort();

performanceFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

performanceFrame.setLocationRelativeTo(null);

performanceFrame.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//É configurado um ouvinte de ação para o botão "Heap Sort"

//para abrir uma tela de desempenho quando o botão é pressionado.

bubbleSort.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

PerformanceBubbleSort performanceFrame = new PerformanceBubbleSort();

performanceFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

performanceFrame.setLocationRelativeTo(null);

performanceFrame.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//remoção de layout automático

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//adição de elementos a tela

getContentPane().add(titleLabel);

getContentPane().add(insertionSort);

getContentPane().add(quickSort);

getContentPane().add(bubbleSort);

getContentPane().add(goBackLabel);

getContentPane().add(buttonimg1);

}

}

**8.5.7 PerformanceBubbleSort**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;

import javax.swing.table.DefaultTableModel;

import javax.swing.text.DefaultStyledDocument;

import javax.swing.text.SimpleAttributeSet;

import javax.swing.text.StyleConstants;

import algorithm.BubbleSort;

import dataBase.Db;

import utils.CustomButton;

import utils.UiHelper;

//declaração da classe PerformanceHeapSort, que estende BaseFrame e

//implementa a interface AutoCloseable.

public class PerformanceBubbleSort extends BaseFrame implements AutoCloseable{

// variáveis de instância da classe.

private JTable table;

private Connection connection;

private long executionTime;

JTextArea textArea;

//construtor da classe PerformanceHeapSort. Ele configura a imagem de fundo

public PerformanceBubbleSort() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

loadDataFromDatabase();

}

//Este método é responsável por carregar dados do banco de dados,

//usando uma conexão com a tabela "cerrado".

private void loadDataFromDatabase() {

this.connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

statement = connection.createStatement();

resultSet = statement.executeQuery("SELECT \* FROM cerrado");

//Inicializa uma lista para armazenar os dados do banco de dados.

List<String[]> data = new ArrayList<>();

// Itera sobre as linhas do conjunto de resultados.

while (resultSet.next()) {

//Cria um array de strings com os valores da linha atual.

String[] rowData = new String[] {

resultSet.getString("id"),

resultSet.getString("fid"),

resultSet.getString("state"),

resultSet.getString("path\_row"),

resultSet.getString("main\_class"),

resultSet.getString("class\_name"),

resultSet.getString("def\_cloud"),

resultSet.getString("julian\_day"),

resultSet.getString("image\_date"),

resultSet.getString("year"),

resultSet.getString("area\_km"),

resultSet.getString("scene\_id"),

resultSet.getString("source"),

resultSet.getString("satellite"),

resultSet.getString("sensor"),

resultSet.getString("uuid")

};

// Adiciona o array de strings à lista de dados.

data.add(rowData);

}

// Registra o tempo de início da ordenação.

long startTime = System.currentTimeMillis();

BubbleSort bubbleSort = new BubbleSort();

// Ordena os dados usando o algoritmo Heap Sort.

bubbleSort.bubbleSort(data);

// Registra o tempo de término da ordenação.

long endTime = System.currentTimeMillis();

// Calcula o tempo de execução da ordenação.

this.executionTime = endTime - startTime;

// Atualiza a área de texto com o tempo de execução.

updateTextArea();

// Obtém o modelo da tabela.

DefaultTableModel model = (DefaultTableModel) table.getModel();

// Itera sobre os dados ordenados.

for (String[] rowData : data) {

model.addRow(rowData);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

Db.closeResultSet(resultSet);

Db.closeStatement(statement);

}

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

//cria um botao imagem e posiciona ele na tela

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

buttonimg1.setBounds(10, 600, 150, 150);

//define um evento para o botão,

//quando clicado, a tela Ordering é aberta

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Ordering orderingFrame = new Ordering ();

orderingFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

orderingFrame.setLocationRelativeTo(null);

orderingFrame.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//define alguns elementos

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

this.textArea = UiHelper.createTextArea(10, 150, 800, 100,new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Desempenho", 300, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

//cria um documento padrão e o vincula ao textarea

DefaultStyledDocument doc = new DefaultStyledDocument();

textArea.setDocument(doc);

//define o alinhamento a esquerda da formatação do documento

SimpleAttributeSet leftAlign = new SimpleAttributeSet();

StyleConstants.setAlignment(leftAlign, StyleConstants.ALIGN\_LEFT);

doc.setParagraphAttributes(0, doc.getLength(), leftAlign, false);

//define conteúdo de texto

textArea.setText("Tempo de execução: " + executionTime + " milissegundos");

//solicita uma repintura do componente textArea.

textArea.repaint();

//remove layout

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//adiciona os elementos do frame

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(textArea);

getContentPane().add(buttonimg1);

//define as colunas da tabela

DefaultTableModel model = new DefaultTableModel();

model.setColumnIdentifiers(new String[] { "id","fid", "state", "path\_row", "main\_class", "class\_name", "def\_cloud", "julian\_day", "image\_date", "year", "area\_km", "scene\_id", "source", "satellite", "sensor", "uuid" });

//cria a variável table para conter o padrão de tabela

table = new JTable(model);

//configura o comportamento de redimensionamento automático das colunas da tabela

table.setAutoResizeMode(JTable.AUTO\_RESIZE\_OFF);

//cria um painel de rolagem

JScrollPane scrollPane = new JScrollPane(table, JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

//define a posição

scrollPane.setBounds(130, 250, 600, 300);

//adiciona o elemento para a tela

getContentPane().add(scrollPane);

}

//Este método atualiza a área de texto com o tempo de execução do algoritmo Heap Sort,

//garantindo que a atualização seja feita na thread de interface do usuário.

private void updateTextArea() {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

textArea.setText("Tempo de execução: " + executionTime + " milissegundos");

}

});

}

@Override

public void close() throws Exception {

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.5.8 PerformanceInsertionSort**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;

import javax.swing.table.DefaultTableModel;

import javax.swing.text.DefaultStyledDocument;

import javax.swing.text.SimpleAttributeSet;

import javax.swing.text.StyleConstants;

import algorithm.InsertionSort;

import dataBase.Db;

import utils.CustomButton;

import utils.UiHelper;

//declaração da classe PerformanceHeapSort, que estende BaseFrame e

//implementa a interface AutoCloseable.

public class PerformanceInsertionSort extends BaseFrame implements AutoCloseable {

// variáveis de instância da classe.

private JTable table;

private Connection connection;

private long executionTime;

JTextArea textArea;

//construtor da classe . Ele configura a imagem de fundo

public PerformanceInsertionSort() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

loadDataFromDatabase();

}

//Este método é responsável por carregar dados do banco de dados,

//usando uma conexão com a tabela "cerrado".

public void loadDataFromDatabase() {

connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

statement = connection.createStatement();

resultSet = statement.executeQuery("SELECT \* FROM cerrado");

//Inicializa uma lista para armazenar os dados do banco de dados.

List<String[]> data = new ArrayList<>();

// Itera sobre as linhas do conjunto de resultados.

while (resultSet.next()) {

//Cria um array de strings com os valores da linha atual.

String[] rowData = new String[] {

resultSet.getString("id"),

resultSet.getString("fid"),

resultSet.getString("state"),

resultSet.getString("path\_row"),

resultSet.getString("main\_class"),

resultSet.getString("class\_name"),

resultSet.getString("def\_cloud"),

resultSet.getString("julian\_day"),

resultSet.getString("image\_date"),

resultSet.getString("year"),

resultSet.getString("area\_km"),

resultSet.getString("scene\_id"),

resultSet.getString("source"),

resultSet.getString("satellite"),

resultSet.getString("sensor"),

resultSet.getString("uuid")

};

// Adiciona o array de strings à lista de dados.

data.add(rowData);

}

// Registra o tempo de início da ordenação.

long startTime = System.currentTimeMillis();

InsertionSort insertionSort = new InsertionSort();

// Ordena os dados usando o algoritmo merge Sort.

insertionSort.insertionSort(data);

// Registra o tempo de término da ordenação.

long endTime = System.currentTimeMillis();

// Calcula o tempo de execução da ordenação.

this.executionTime = endTime - startTime;

// Atualiza a área de texto com o tempo de execução.

updateTextArea();

// Obtém o modelo da tabela.

DefaultTableModel model = (DefaultTableModel) table.getModel();

// Itera sobre os dados ordenados.

for (String[] rowData : data) {

model.addRow(rowData);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

Db.closeResultSet(resultSet);

Db.closeStatement(statement);

}

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

//cria um botao imagem e posiciona ele na tela

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

buttonimg1.setBounds(10, 600, 150, 150);

//define um evento para o botão,

//quando clicado, a tela Ordering é aberta

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Ordering orderingFrame = new Ordering();

orderingFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

orderingFrame.setLocationRelativeTo(null);

orderingFrame.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//define alguns elementos

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

this.textArea = UiHelper.createTextArea(10, 150, 800, 100,new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Desempenho", 300, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

//cria um documento padrão e o vincula ao textarea

DefaultStyledDocument doc = new DefaultStyledDocument();

textArea.setDocument(doc);

//define o alinhamento a esquerda da formatação do documento

SimpleAttributeSet leftAlign = new SimpleAttributeSet();

StyleConstants.setAlignment(leftAlign, StyleConstants.ALIGN\_LEFT);

doc.setParagraphAttributes(0, doc.getLength(), leftAlign, false);

//define conteúdo de texto

textArea.setText("Tempo de execução: " + executionTime + " milissegundos");

//solicita uma repintura do componente textArea.

textArea.repaint();

//remove layout

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//adiciona os elementos do frame

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(textArea);

getContentPane().add(buttonimg1);

//define as colunas da tabela

DefaultTableModel model = new DefaultTableModel();

model.setColumnIdentifiers(new String[] { "id","fid", "state", "path\_row", "main\_class", "class\_name", "def\_cloud", "julian\_day", "image\_date", "year", "area\_km", "scene\_id", "source", "satellite", "sensor", "uuid" });

//cria a variável table para conter o padrão de tabela

table = new JTable(model);

//configura o comportamento de redimensionamento automático das colunas da tabela

table.setAutoResizeMode(JTable.AUTO\_RESIZE\_OFF);

//cria um painel de rolagem

JScrollPane scrollPane = new JScrollPane(table, JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

//define a posição

scrollPane.setBounds(130, 250, 600, 300);

//adiciona o elemento para a tela

getContentPane().add(scrollPane);

}

//Este método atualiza a área de texto com o tempo de execução do algoritmo Heap Sort,

//garantindo que a atualização seja feita na thread de interface do usuário.

private void updateTextArea() {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

textArea.setText("Tempo de execução: " + executionTime + " milissegundos");

}

});

}

@Override

public void close() throws Exception {

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.5.9 PerformanceQuickSort**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;

import javax.swing.table.DefaultTableModel;

import javax.swing.text.DefaultStyledDocument;

import javax.swing.text.SimpleAttributeSet;

import javax.swing.text.StyleConstants;

import algorithm.QuickSort;

import dataBase.Db;

import utils.CustomButton;

import utils.UiHelper;

//declaração da classe PerformanceHeapSort, que estende BaseFrame e

//implementa a interface AutoCloseable.

public class PerformanceQuickSort extends BaseFrame implements AutoCloseable {

// variáveis de instância da classe.

private JTable table;

private Connection connection;

private long executionTime;

JTextArea textArea;

//construtor da classe. Ele configura a imagem de fundo

public PerformanceQuickSort() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

loadDataFromDatabase();

}

//Este método é responsável por carregar dados do banco de dados,

//usando uma conexão com a tabela "cerrado".

public void loadDataFromDatabase() {

connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

statement = connection.createStatement();

resultSet = statement.executeQuery("SELECT \* FROM cerrado");

//Inicializa uma lista para armazenar os dados do banco de dados.

List<String[]> data = new ArrayList<>();

//Cria um array de strings com os valores da linha atual.

while (resultSet.next()) {

String[] rowData = new String[] {

resultSet.getString("id"),

resultSet.getString("fid"),

resultSet.getString("state"),

resultSet.getString("path\_row"),

resultSet.getString("main\_class"),

resultSet.getString("class\_name"),

resultSet.getString("def\_cloud"),

resultSet.getString("julian\_day"),

resultSet.getString("image\_date"),

resultSet.getString("year"),

resultSet.getString("area\_km"),

resultSet.getString("scene\_id"),

resultSet.getString("source"),

resultSet.getString("satellite"),

resultSet.getString("sensor"),

resultSet.getString("uuid")

};

// Adiciona o array de strings à lista de dados.

data.add(rowData);

}

// Registra o tempo de início da ordenação.

long startTime = System.currentTimeMillis();

QuickSort quickSort = new QuickSort();

// Ordena os dados usando o algoritmo quick Sort.

quickSort.quickSort(data);

// Registra o tempo de término da ordenação.

long endTime = System.currentTimeMillis();

// Calcula o tempo de execução da ordenação.

this.executionTime = endTime - startTime;

// Atualiza a área de texto com o tempo de execução.

updateTextArea();

// Obtém o modelo da tabela.

DefaultTableModel model = (DefaultTableModel) table.getModel();

// Itera sobre os dados ordenados.

for (String[] rowData : data) {

model.addRow(rowData);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

Db.closeResultSet(resultSet);

Db.closeStatement(statement);

}

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

//cria um botao imagem e posiciona ele na tela

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

buttonimg1.setBounds(10, 600, 150, 150);

//define um evento para o botão,

//quando clicado, a tela Ordering é aberta

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Ordering orderingFrame = new Ordering();

orderingFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

orderingFrame.setLocationRelativeTo(null);

orderingFrame.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//define alguns elementos

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

this.textArea = UiHelper.createTextArea(10, 150, 800, 100,new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Desempenho", 300, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

//cria um documento padrão e o vincula ao textarea

DefaultStyledDocument doc = new DefaultStyledDocument();

textArea.setDocument(doc);

//define o alinhamento a esquerda da formatação do documento

SimpleAttributeSet leftAlign = new SimpleAttributeSet();

StyleConstants.setAlignment(leftAlign, StyleConstants.ALIGN\_LEFT);

doc.setParagraphAttributes(0, doc.getLength(), leftAlign, false);

//define conteúdo de texto

textArea.setText("Tempo de execução: " + executionTime + " milissegundos");

//solicita uma repintura do componente textArea.

textArea.repaint();

//remove layout

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//adiciona os elementos do frame

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(textArea);

getContentPane().add(buttonimg1);

//define as colunas da tabela

DefaultTableModel model = new DefaultTableModel();

model.setColumnIdentifiers(new String[] { "id","fid", "state", "path\_row", "main\_class", "class\_name", "def\_cloud", "julian\_day", "image\_date", "year", "area\_km", "scene\_id", "source", "satellite", "sensor", "uuid" });

//cria a variável table para conter o padrão de tabela

table = new JTable(model);

//configura o comportamento de redimensionamento automático das colunas da tabela

table.setAutoResizeMode(JTable.AUTO\_RESIZE\_OFF);

//cria um painel de rolagem

JScrollPane scrollPane = new JScrollPane(table, JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

//define a posição

scrollPane.setBounds(130, 250, 600, 300);

//adiciona o elemento para a tela

getContentPane().add(scrollPane);

}

//Este método atualiza a área de texto com o tempo de execução do algoritmo Heap Sort,

//garantindo que a atualização seja feita na thread de interface do usuário.

private void updateTextArea() {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

textArea.setText("Tempo de execução: " + executionTime + " milissegundos");

}

});

}

@Override

public void close() throws Exception {

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.5.10 RecordsScreen**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import dataBase.Db;

import utils.CustomButton;

import utils.UiHelper;

//declaração da classe PerformanceHeapSort, que estende BaseFrame e

//implementa a interface AutoCloseable.

public class RecordsScreen extends BaseFrame implements AutoCloseable {

// variáveis de instância da classe.

private JTable table;

private Connection connection;

private static final String BACKGROUND\_IMAGE\_PATH = "C:\\aps\\fox.jpeg.jpg";

//construtor da classe . Ele configura a imagem de fundo

public RecordsScreen() throws IOException {

super(ImageIO.read(new File(BACKGROUND\_IMAGE\_PATH)));

initializeComponents();

loadDataFromDatabase();

}

//Este método é responsável por carregar dados do banco de dados,

//usando uma conexão com a tabela "cerrado".

private void loadDataFromDatabase() {

this.connection = null;

Statement statement = null;

ResultSet resultSet = null;

try {

if (connection == null || connection.isClosed()) {

connection = Db.getConnection();

}

// Cria uma instrução SQL para consultar o banco de dados

statement = connection.createStatement();

// Executa a consulta SQL para selecionar todos os registros da tabela "cerrado"

resultSet = statement.executeQuery("SELECT \* FROM cerrado");

// Obtém o modelo da tabela onde os dados serão exibidos

DefaultTableModel model = (DefaultTableModel) table.getModel();

// Itera pelos resultados da consulta e adiciona cada registro como uma nova linha na tabela

while (resultSet.next()) {

model.addRow(new Object[] { resultSet.getString("id"),resultSet.getString("fid"), resultSet.getString("state"), resultSet.getString("path\_row"), resultSet.getString("main\_class"), resultSet.getString("class\_name"), resultSet.getString("def\_cloud"), resultSet.getString("julian\_day"), resultSet.getString("image\_date"), resultSet.getString("year"), resultSet.getString("area\_km"), resultSet.getString("scene\_id"), resultSet.getString("source"), resultSet.getString("satellite"), resultSet.getString("sensor"), resultSet.getString("uuid") });

}

} catch (Exception e) {

e.printStackTrace();

} finally {

Db.closeResultSet(resultSet);

Db.closeStatement(statement);

}

}

@Override

public void initializeComponents() {

final ImageIcon regular = new ImageIcon("C:\\aps\\foxbutton1.png");

final ImageIcon hovering = new ImageIcon("C:\\aps\\foxbutton.png");

final ImageIcon clicking = new ImageIcon("C:\\aps\\foxbutton.png");

final ImageIcon regular2 = new ImageIcon("C:\\aps\\foxbuttonback1.png");

final ImageIcon hovering2 = new ImageIcon("C:\\aps\\foxbuttonback.png");

final ImageIcon clicking2 = new ImageIcon("C:\\aps\\foxbuttonback.png");

//cria 2 botao imagem e posiciona ele na tela

CustomButton buttonimg2 = new CustomButton(regular, hovering, clicking);

CustomButton buttonimg1 = new CustomButton(regular2, hovering2, clicking2);

//define um evento para o botão,

//quando clicado, a tela HomeScreen é aberta

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

HomeScreen homeScreen = new HomeScreen();

homeScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

homeScreen.setLocationRelativeTo(null);

homeScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

//define um evento para o botão,

//quando clicado, a tela Ordering é aberta

buttonimg2.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

Ordering ordering = new Ordering();

ordering.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

ordering.setLocationRelativeTo(null);

ordering.setVisible(true);

dispose();

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

//posiciona os botoes

buttonimg2.setBounds(700, 600, 150, 150);

buttonimg1.setBounds(10, 600, 150, 150);

//cria botoes tipo RoundButton

JButton button1 = UiHelper.createRoundButton("INSERIR", 100, 150, 150, 60, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JButton button2 = UiHelper.createRoundButton("ATUALIZAR", 340, 150, 180, 60, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JButton button3 = UiHelper.createRoundButton("DELETAR", 600, 150, 150, 60, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

//define um evento para o botão,

//quando clicado, a tela InsertionScreen é aberta

button1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

InsertionScreen insertion = new InsertionScreen();

insertion.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

insertion.setLocationRelativeTo(null);

insertion.setVisible(true);

dispose();

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

//define um evento para o botão,

//quando clicado, a tela UpdateScreen\_1 é aberta

button2.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

UpdateScreen\_1 updateScreen\_1 = new UpdateScreen\_1();

updateScreen\_1.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

updateScreen\_1.setLocationRelativeTo(null);

updateScreen\_1.setVisible(true);

dispose();

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

//define um evento para o botão,

//quando clicado, a tela DeleteScreen é aberta

button3.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

DeleteScreen deleteScreen = new DeleteScreen();

deleteScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

deleteScreen.setLocationRelativeTo(null);

deleteScreen.setVisible(true);

dispose();

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

//criação de rótulos

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Records", 340, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JLabel nextLabel = UiHelper.createLabel("Avançar", 730, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

//remoção de layout

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

//adição de elementos para a tela

getContentPane().add(titleLabel);

getContentPane().add(button1);

getContentPane().add(button2);

getContentPane().add(button3);

getContentPane().add(goBackLabel);

getContentPane().add(nextLabel);

getContentPane().add(buttonimg1);

getContentPane().add(buttonimg2);

//define as colunas da tabela

DefaultTableModel model = new DefaultTableModel();

model.setColumnIdentifiers(new String[] { "id","fid", "state", "path\_row", "main\_class", "class\_name", "def\_cloud", "julian\_day", "image\_date", "year", "area\_km", "scene\_id", "source", "satellite", "sensor", "uuid" });

//cria a variável table para conter o padrão de tabela

table = new JTable(model);

//configura o comportamento de redimensionamento automático das colunas da tabela

table.setAutoResizeMode(JTable.AUTO\_RESIZE\_OFF);

//cria um painel de rolagem

JScrollPane scrollPane = new JScrollPane(table, JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);

//define a posição

scrollPane.setBounds(130, 250, 600, 300);

//adiciona o elemento para a tela

getContentPane().add(scrollPane);

}

@Override

public void close() throws Exception {

if (connection != null && !connection.isClosed()) {

connection.close();

}

}

}

**8.5.11 UpdateScreen\_1**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import backend.UpdateLogic;

import utils.CustomButton;

import utils.UiHelper;

public class UpdateScreen\_1 extends BaseFrame {

public UpdateScreen\_1() throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

initializeComponents();

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

RecordsScreen recordsScreen = new RecordsScreen();

recordsScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

recordsScreen.setLocationRelativeTo(null);

recordsScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

buttonimg1.setBounds(10, 600, 150, 150);

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Update", 350, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JButton buttonSubmit = UiHelper.createRoundButton("Pesquisar", 700, 600, 150, 100, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JLabel idLabel = UiHelper.createLabel("ID: ", 400, 250, 100, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JTextField idTextField = UiHelper.createTextField(280, 300, 300, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

buttonSubmit.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String idText = idTextField.getText();

if (!idText.isEmpty()) {

try {

int id = Integer.parseInt(idText);

if (UpdateLogic.validateId(id)) {

try {

UpdateScreen\_2 updateScreen\_2 = new UpdateScreen\_2(id);

updateScreen\_2.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

updateScreen\_2.setLocationRelativeTo(null);

updateScreen\_2.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

} else {

JOptionPane.showMessageDialog(null, "ID inválido", "Erro", JOptionPane.ERROR\_MESSAGE);

}

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(null, "ID deve ser um número válido", "Erro", JOptionPane.ERROR\_MESSAGE);

}

} else {

JOptionPane.showMessageDialog(null, "Campo de ID vazio", "Erro", JOptionPane.ERROR\_MESSAGE);

}

}

});

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(buttonimg1);

getContentPane().add(buttonSubmit);

getContentPane().add(idLabel);

getContentPane().add(idTextField);

}

}

**8.5.12 UpdateScreen\_2**

package ui;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import backend.Records;

import backend.UpdateLogic;

import utils.CustomButton;

import utils.UiHelper;

public class UpdateScreen\_2 extends BaseFrame {

private int id;

private JTextField fid;

private JTextField state;

private JTextField pathRow;

private JTextField mainClass;

private JTextField nameClass;

private JTextField julianDay;

private JTextField imageDate;

private JTextField year;

private JTextField areaKm;

private JTextField satelite;

private JTextField sensor;

private JTextField uuid;

private JTextField imagem;

public UpdateScreen\_2(int id) throws IOException {

super(ImageIO.read(new File("C:\\aps\\fox.jpeg.jpg")));

getContentPane().setBackground(Color.WHITE);

this.id = id;

initializeComponents();

preencherCampos(id);

}

@Override

public void initializeComponents() {

ImageIcon regular = new ImageIcon("C:\\aps\\foxbuttonback1.png");

ImageIcon hovering = new ImageIcon("C:\\aps\\foxbuttonback.png");

ImageIcon clicking = new ImageIcon("C:\\aps\\foxbuttonback.png");

CustomButton buttonimg1 = new CustomButton(regular, hovering, clicking);

buttonimg1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

UpdateScreen\_1 updateScreen\_1 = new UpdateScreen\_1();

updateScreen\_1.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

updateScreen\_1.setLocationRelativeTo(null);

updateScreen\_1.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

buttonimg1.setBounds(10, 600, 150, 150);

JLabel goBackLabel = UiHelper.createLabel("Voltar", 50, 570, 150, 30, new Font("Chiller", Font.BOLD, 40), Color.WHITE);

JLabel titleLabel = UiHelper.createLabel("Update", 340, 10, 800, 100, new Font("Chiller", Font.BOLD, 80), Color.WHITE);

JLabel fidLabel = UiHelper.createLabel("FID: ", 50, 180, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel stateLabel = UiHelper.createLabel("STATE: \*", 50, 230, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel pathRowLabel = UiHelper.createLabel("PATH ROW: \*", 50, 280, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel mainClassLabel = UiHelper.createLabel("MAIN CLASS: \*", 50, 330, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel nameCLassLabel = UiHelper.createLabel("NAME CLASS: \*", 50, 380, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel julianDayLabel = UiHelper.createLabel("JULIAN DAY: \*", 50, 430, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel imageDateLabel = UiHelper.createLabel("IMAGE DATE: \*", 450, 180, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel yearLabel = UiHelper.createLabel("YEAR: \*", 450, 230, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel areaKmLabel = UiHelper.createLabel("AREA KM: \*", 450, 280, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel sateliteLabel = UiHelper.createLabel("SATELITE: \*", 450, 330, 150, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel sensorLabel = UiHelper.createLabel("SENSOR: \*", 450, 380, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel uuidLabel = UiHelper.createLabel("UUID: \*", 450, 430, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JLabel imagemLabel = UiHelper.createLabel("IMAGEM: \*", 50, 480, 100, 30, new Font("Chiller", Font.BOLD, 24), Color.WHITE);

JButton buttonSave = UiHelper.createRoundButton("Salvar", 700, 600, 100, 100, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

JButton imagemButton = UiHelper.createRoundButton("ver imagem", 450, 480, 350, 30, new Font("Chiller", Font.BOLD, 30), Color.WHITE);

imagemButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

ImageScreen imageScreen = new ImageScreen(id);

imageScreen.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

imageScreen.setLocationRelativeTo(null);

imageScreen.setVisible(true);

} catch (IOException ex) {

ex.printStackTrace();

}

dispose();

}

});

fid = UiHelper.createTextField(190, 180, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

state = UiHelper.createTextField(190, 230, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

pathRow = UiHelper.createTextField(190, 280, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

mainClass = UiHelper.createTextField(190, 330, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

nameClass = UiHelper.createTextField(190, 380, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

julianDay = UiHelper.createTextField(190, 430, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

imageDate = UiHelper.createTextField(600, 180, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

year = UiHelper.createTextField(600, 230, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

areaKm = UiHelper.createTextField(600, 280, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

satelite = UiHelper.createTextField(600, 330, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

sensor = UiHelper.createTextField(600, 380, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

uuid = UiHelper.createTextField(600, 430, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

imagem = UiHelper.createTextField(190, 480, 200, 30, new Font("Chiller", Font.BOLD, 26), Color.BLACK);

buttonSave.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

try {

boolean updateSuccessful = UpdateLogic.updateRecord(

id,

Integer.parseInt(fid.getText()),

state.getText(),

pathRow.getText(),

mainClass.getText(),

nameClass.getText(),

Integer.parseInt(julianDay.getText()),

imageDate.getText(),

Integer.parseInt(year.getText()),

areaKm.getText(),

satelite.getText(),

sensor.getText(),

uuid.getText(),

imagem.getText());

if (updateSuccessful) {

JOptionPane.showMessageDialog(null, "Registro atualizado com sucesso!");

UpdateScreen\_1 updateScreen\_1 = new UpdateScreen\_1();

updateScreen\_1.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

updateScreen\_1.setLocationRelativeTo(null);

updateScreen\_1.setVisible(true);

} else {

JOptionPane.showMessageDialog(null, "Erro ao atualizar o registro no banco de dados.", "Erro", JOptionPane.ERROR\_MESSAGE);

}

dispose();

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(null, "ID inválido", "Erro", JOptionPane.ERROR\_MESSAGE);

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Erro desconhecido", "Erro", JOptionPane.ERROR\_MESSAGE);

}

}

});

JPanel contentPane = (JPanel) getContentPane();

contentPane.setLayout(null);

getContentPane().add(titleLabel);

getContentPane().add(goBackLabel);

getContentPane().add(buttonimg1);

getContentPane().add(fid);

getContentPane().add(state);

getContentPane().add(pathRow);

getContentPane().add(mainClass);

getContentPane().add(nameClass);

getContentPane().add(julianDay);

getContentPane().add(imageDate);

getContentPane().add(year);

getContentPane().add(areaKm);

getContentPane().add(satelite);

getContentPane().add(sensor);

getContentPane().add(uuid);

getContentPane().add(fidLabel);

getContentPane().add(stateLabel);

getContentPane().add(pathRowLabel);

getContentPane().add(mainClassLabel);

getContentPane().add(nameCLassLabel);

getContentPane().add(julianDayLabel);

getContentPane().add(imageDateLabel);

getContentPane().add(yearLabel);

getContentPane().add(areaKmLabel);

getContentPane().add(sateliteLabel);

getContentPane().add(sensorLabel);

getContentPane().add(uuidLabel);

getContentPane().add(buttonSave);

getContentPane().add(imagemLabel);

getContentPane().add(imagem);

getContentPane().add(imagemButton);

}

private void preencherCampos(int id) {

Records records = UpdateLogic.getRegistroPorId(id);

if (records != null) {

fid.setText(records.getFid());

state.setText(records.getState());

pathRow.setText(records.getPathRow());

mainClass.setText(records.getMainClass());

nameClass.setText(records.getNameClass());

julianDay.setText(String.valueOf(records.getJulianDay()));

imageDate.setText(records.getImageDate());

year.setText(String.valueOf(records.getYear()));

areaKm.setText(records.getAreaKm());

satelite.setText(records.getSatellite());

sensor.setText(records.getSensor());

uuid.setText(records.getUuid());

imagem.setText(records.getImagem());

}

}

}

**8.6 Pacote utils**

**8.6.1 ConversorBlob**

package utils;

import java.awt.image.BufferedImage;

import java.io.IOException;

import java.io.InputStream;

import javax.imageio.ImageIO;

public class ConversorBlob {

public static BufferedImage inputStreamToImage(InputStream inputStream) {

try {

if (inputStream == null) {

return null;

}

// Converte o InputStream em uma imagem.

BufferedImage image = ImageIO.read(inputStream);

return image;

} catch (IOException ex) {

ex.printStackTrace();

return null;

}

}

}

**8.6.2 CustomButton**

package utils;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import java.awt.Cursor;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

public class CustomButton extends JButton {

private ImageIcon regularIcon;

private ImageIcon hoveringIcon;

private ImageIcon clickingIcon;

//Construtor que está atribuindo as imagens do botão raposinha

public CustomButton(ImageIcon regularIcon, ImageIcon hoveringIcon, ImageIcon clickingIcon) {

this.regularIcon = regularIcon;

this.hoveringIcon = hoveringIcon;

this.clickingIcon = clickingIcon;

setIcon(regularIcon);

//remove a borda padrão

setBorderPainted(false);

//desativa o preenchido automático do botçao, ele terá apenas o tamanho da imagem

setContentAreaFilled(false);

//habilita a versão mãozinha do mouse

setCursor(new Cursor(Cursor.HAND\_CURSOR));

//adiciona um ouvinte de eventos para o mouse

addMouseListener(new MouseAdapter() {

@Override

//define a imagem raposinha correspondente para quando estiver clicado

public void mousePressed(MouseEvent e) {

setIcon(clickingIcon);

}

@Override

//lógica para depois que o botão é clicado

public void mouseReleased(MouseEvent e) {

//se o mouse ainda estiver em cima do botão, ele continua com a imagem raposinha acesa

if (containsMouse(e.getX(), e.getY())) {

setIcon(hoveringIcon);

// senão, define a imagem da raposinha apagada

} else {

setIcon(regularIcon);

}

}

@Override

//verifica se o mouse está em cima do botão

public void mouseEntered(MouseEvent e) {

setIcon(hoveringIcon);

}

@Override

//define a imagem raposinha correspondente para quando nao tem cursor no botão

public void mouseExited(MouseEvent e) {

setIcon(regularIcon);

}

});

}

//verifica se o ponto definido pelas coordenadas x e y está dentro dos limites do botão.

private boolean containsMouse(int x, int y) {

return x >= 0 && x <= getWidth() && y >= 0 && y <= getHeight();

}

}

**8.6.3 RoundButton**

package utils;

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Stroke;

import javax.swing.JButton;

public class RoundButton extends JButton {

private int arcRadius = 50; // Raio de curvatura do botão

//Construtor recebe o nome do botão e o cria.

public RoundButton(String label) {

super(label);

//remove a cor de fundo da área de conteúdo, tornando a área interna transparente

setContentAreaFilled(false);

}

protected void paintComponent(Graphics g) {

//verifica se o modelo do botão está no estado "pressionado"

if (getModel().isArmed()) {

g.setColor(new Color(0xFFDEAD)); // Cor de fundo #D8B872 quando pressionado

} else {

g.setColor(new Color(0xD8B872)); // Cor de fundo #FFDEAD padrão

}

//um retângulo arredondado é desenhado para representar a forma do botão

g.fillRoundRect(0, 0, getSize().width - 1, getSize().height - 1, arcRadius, arcRadius); // Use g.fillRoundRect com o raio de curvatura

//desenha o conteúdo interno do botão, como o texto e os ícones.

super.paintComponent(g);

}

protected void paintBorder(Graphics g) {

//Verifica se o objeto g é uma instância de Graphics2D

if (g instanceof Graphics2D) {

//"g" é convertido para Graphics2D e atribuído a g2d

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.DARK\_GRAY); // Define a cor da borda.

Stroke oldStroke = g2d.getStroke(); // Salva a espessura da linha da borda atual

g2d.setStroke(new BasicStroke(3.0f)); // Define uma nova espessura para a linha da borda(3 pixels)

//desenha a borda

g2d.drawRoundRect(0, 0, getSize().width - 1, getSize().height - 1, arcRadius, arcRadius);

g2d.setStroke(oldStroke); // Restaura a espessura original da linha

}

}

// Usado para evitar que o botão tenha um contorno quando obtém foco

public boolean isFocusTraversable() {

return false;

}

}

**8.6.4 UiHelper**

package utils;

import java.awt.Color;

import java.awt.Cursor;

import java.awt.Font;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JLabel;

import javax.swing.JTextArea;

import javax.swing.JTextField;

public class UiHelper {

//cria um label a partir das definições dos parâmetros

public static JLabel createLabel(String text, int x, int y, int width, int height, Font font, Color color) {

JLabel label = new JLabel(text);

label.setFont(font);

label.setForeground(color);

label.setBounds(x, y, width, height);

return label;

}

//cria um Jbutto do tipo RoundButton a partir das definições dos parâmetros

public static JButton createRoundButton(String text, int x, int y, int width, int height, Font font, Color color) {

RoundButton button = new RoundButton(text);

button.setFont(font);

button.setForeground(color);

button.setCursor(new Cursor(Cursor.HAND\_CURSOR));

button.setBounds(x, y, width, height);

return button;

}

//cria um Jtextarea a partir das definições dos parâmetros

public static JTextArea createTextArea(int x, int y, int width, int height, Font font, Color color) {

JTextArea textArea = new JTextArea();

textArea.setFont(font);

textArea.setForeground(color);

textArea.setBounds(x, y, width, height);

textArea.setOpaque(false);

//permite a quebra de linha no texto

textArea.setLineWrap(true);

//define que a quebra de linha ocorre apenas entre palavras

textArea.setWrapStyleWord(true);

// define a área de texto como somente leitura

textArea.setEditable(false);

return textArea;

}

//cria um Jtextfield a partir das definições dos parâmetros

public static JTextField createTextField(int x, int y, int width, int height, Font font, Color color) {

JTextField textField = new JTextField();

textField.setFont(font);

textField.setForeground(color);

textField.setBounds(x, y, width, height);

return textField;

}

public static JComboBox<String> createComboBox(String[] items, int x, int y, int width, int height, Font font, Color color) {

JComboBox<String> comboBox = new JComboBox<>(items);

comboBox.setFont(font);

comboBox.setForeground(color);

comboBox.setBounds(x, y, width, height);

return comboBox;

}

}