# JavaFX Studies/ Aula12: Simulando um Banco de Dados com a Classe File e seus métodos:

Vamos aprender usando um BufferedWriter a escrever e ler dados de um arquivo data.txt:

Primeiro vamos entender duas coisas principais, para essa forma de escrever dados em um arquivo.txt vamos utilizar uma File com o path do nosso banco de dados e um HashMap que tem como chaves String sendo elas o usuário e senha da pessoa que está fazendo o login:

```
File file = new File(pathname:"database/data.txt");
HashMap<String, String> loginInfo = new HashMap<>();
```

E nosso metodo pricipal será o updateUsernamesAndPasswords:

```
    }

    private void updateUsernamesAndPasswords() throws IOException {
        Scanner scanner = new Scanner(file);
        loginInfo.clear(); // Limpa o mapa existente
        while (scanner.hasNext()) {
            String[] splitInfo = scanner.nextLine().split(regex:",");
            loginInfo.put(splitInfo[0], splitInfo[1]);
        }
        scanner.close();
    }
```

Onde ele recebe tudo que está no file por meio do Scanner e passa para o nosso HashMap loginInfo.

Agora Vamos escrever dados no nosso data.txt:

```
public void write(ActionEvent event){
        if(!file.exists()){
            try {
                file.createNewFile();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

```
public void write(ActionEvent event){
    try {
        String username = textfield.getText();
        String password = passwordfield.getText();
        updateUsernamesAndPasswords();
        if(loginInfo.containsKey(username)){
            showAlert(string:"Error", string2:"Username already exists.");
        } else {
            showAlert(string:"Success", string2:"Account created successfully.");
            BufferedWriter writer = new BufferedWriter(new FileWriter(file, append:true));

            writer.write(username + "," + password + "\n");
            writer.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Primeiro vemos se o nosso “dataBase” existe, caso não exista criamos ele no diretório, após isso recebemos o usuário e senha e depois preenchemos nosso HashMap, checamos se tem alguém com aquele nome de usuário, caso exista, ele não permite criar outro, caso não exista ele usa o BufferedWriter para escrever o usuário e senha no nosso arquivo txt em uma nova linha.

Agora vamos ver o método de ler o nosso data.txt:

```java
public void read(ActionEvent event) {
    try {
        updateUsernamesAndPasswords();
        String username = textfield.getText();
        String password = passwordfield.getText();
        if (loginInfo.containsKey(username) && loginInfo.get(username).equals
        (password)) {
            showAlert(string:"Login Successful", "Welcome " + username + "!");
        } else {
            showAlert(string:"Login Failed", string2:"Invalid username or password.");
        }
    } catch (IOException e) {
        e.printStackTrace();
        showAlert(string:"Error", string2:"Failed to read from file.");
    }
}
```

- updateUsernamesAndPasswords();: Este método é chamado para atualizar o HashMap com os dados do arquivo. Ele é responsável por ler os dados do arquivo e preencher o HashMap com os usernames e senhas correspondentes.
- String username = textfield.getText();: Obtém o texto presente no TextField onde o usuário insere seu username.
- String password = passwordfield.getText();: Obtém o texto presente no PasswordField onde o usuário insere sua senha.
- if (loginInfo.containsKey(username) && loginInfo.get(username).equals(password)) { ... }: Verifica se o username fornecido está presente no HashMap e se a senha correspondente a esse username no HashMap é igual à senha fornecida pelo usuário.
- showAlert("Login Successful", "Welcome " + username + "!");: Se o login for bem-sucedido, exibe uma mensagem de boas-vindas ao usuário.
- showAlert("Login Failed", "Invalid username or password.");: Se o login falhar, exibe uma mensagem informando que o username ou a senha fornecidos são inválidos.

Temos formas de ler arquivos Txt, como criar uma tela pós login que acessa a tela e adiciona a um ListView as tasks do usuário respectivamente, mas para isso precisamos de outra abordagem de construção:

1. Controller01

A classe `Controller01` é responsável por lidar com a lógica da interface do usuário (GUI) e a interação com os usuários. Ela contém métodos para autenticar usuários, adicionar tarefas, ler tarefas e exibir alertas na interface do usuário.

- **Necessidade**: Gerenciar a interação entre o usuário e o sistema de login e tarefas.
- **Métodos**:
  - `read(ActionEvent event)`: Lê os dados do arquivo e autentica os usuários com base nas informações fornecidas nos campos de texto da GUI.
  - `write(ActionEvent event)`: Escreve novos usuários e suas tarefas no arquivo.
  - `addTask(ActionEvent event)`: Adiciona uma nova tarefa ao usuário atual e atualiza o arquivo com a nova tarefa.
  - `readTasks(ActionEvent event)`: Lê as tarefas do usuário atual e exibe-as na interface do usuário.

2. User

A classe `User` representa um usuário do sistema, contendo seu username, password e as tarefas associadas.

- **Necessidade**: Armazenar informações sobre cada usuário, incluindo suas tarefas.
- **Métodos**:

  - `addTask(String task)`: Adiciona uma nova tarefa ao usuário.
  - `getTasks()`: Retorna as tarefas do usuário como um array de strings.
  - `getTasksAsString()`: Retorna as tarefas do usuário como uma única string.

3. Interface Gráfica do Usuário (GUI)

A GUI consiste nos campos de texto para inserir o username e a senha, bem como botões para realizar ações como login, adicionar tarefas e ler tarefas.

- **Necessidade**: Permitir que os usuários interajam com o sistema de login e tarefas de forma intuitiva.
- **Componentes**:
  - `textfield`: Campo de texto para inserir o username.
  - `passwordfield`: Campo de texto para inserir a senha.
  - Botões para executar as ações (login, adicionar tarefas, etc.).
  - Alertas para exibir mensagens de sucesso, falha ou erro.

4. Arquivo de Dados (.txt)

O arquivo de dados armazena as informações dos usuários, incluindo usernames, senhas e tarefas associadas.

- **Necessidade**: Persistir os dados dos usuários de forma que possam ser recuperados posteriormente.
- **Estrutura**:
  - Cada linha representa um usuário, com o username, a senha e as tarefas associadas, separadas por vírgulas.
  - Quando uma nova tarefa é adicionada para um usuário existente, ela é simplesmente adicionada ao final da linha correspondente a esse usuário, separada por vírgula.

5. Detalhes:

Mudanças no HashMap

**Adaptação do HashMap para armazenar objetos `User`**: Em vez de armazenar apenas usernames e senhas, o `HashMap` agora armazenará objetos `User`, que conterão não apenas a senha, mas também as tarefas associadas a cada usuário.

```java
HashMap<String, User> users = new HashMap<>();
```

Classe User

**Adição de uma lista de tarefas ao usuário**: A classe `User` agora possui uma lista de tarefas associadas a cada usuário.

```java
private List<String> tasks = new ArrayList<>();
```

Método para adicionar tarefas**: Adicionamos um método para permitir que novas tarefas sejam adicionadas ao usuário.

```java
public void addTask(String task) {
    tasks.add(task);
}
```

Método para obter tarefas como String**: Adicionamos um método para obter as tarefas do usuário como uma única String.

```java
public String getTasksAsString() {
    StringBuilder builder = new StringBuilder();
    for (String task : tasks) {
        builder.append(task).append(", ");
    }
    return builder.toString();
}
```

Vou explicar detalhadamente como os métodos `read`, `write`, `readTasks` e `addTask` devem ser construídos, juntamente com a forma como eles interagem com a estrutura de dados atualizada (usando um `HashMap` que armazena objetos `User`).

Método `read`

O método `read` é responsável por ler os dados do arquivo e carregar os usuários e suas tarefas associadas para o `HashMap`.

```java
public void read(ActionEvent event) {
    try {
        updateUsernamesAndPasswords(); // Atualiza o HashMap com os dados do arquivo
        String username = textfield.getText(); // Obtém o username do TextField
        String password = passwordfield.getText(); // Obtém a senha do PasswordField

        if (users.containsKey(username) && users.get(username).getPassword().equals(password)) {
            showAlert("Login Successful", "Welcome " + username + "!");
        } else {
            showAlert("Login Failed", "Invalid username or password.");
        }
    } catch (IOException e) {
        e.printStackTrace();
        showAlert("Error", "Failed to read from file.");
    }
}
```

- **Atualização do HashMap**: O método `updateUsernamesAndPasswords` é chamado para atualizar o `HashMap` com os dados do arquivo.
- **Autenticação do Usuário**: Verifica se o username fornecido está presente no `HashMap` e se a senha correspondente é válida.
- **Exibição de Alertas**: Exibe um alerta informando se o login foi bem-sucedido ou falhou.

Método `write`

O método `write` é responsável por adicionar um novo usuário e suas tarefas associadas ao arquivo.

```java
public void write(ActionEvent event) {
    try {
        String username = textfield.getText(); // Obtém o username do TextField
        String password = passwordfield.getText(); // Obtém a senha do PasswordField

        if (users.containsKey(username)) {
            showAlert("Error", "Username already exists.");
```

```
        } else {
            showAlert("Success", "Account created successfully.");
            User newUser = new User(username, password); // Cria um novo objeto User
            users.put(username, newUser); // Adiciona o novo usuário ao HashMap
            updateUserFile(); // Atualiza o arquivo com os novos dados
        }
    } catch (IOException e) {
        e.printStackTrace();
        showAlert("Error", "Failed to write to file.");
    }
}
```

- **Adição de um Novo Usuário**: Verifica se o username fornecido já existe no `HashMap`. Se não existir, cria um novo objeto `User` com o username e a senha fornecidos e adiciona ao `HashMap`.
- **Atualização do Arquivo**: Chama o método `updateUserFile` para atualizar o arquivo com os novos dados.

Método `readTasks`

O método `readTasks` é responsável por ler as tarefas do usuário atual e exibi-las.

```java
public void readTasks(ActionEvent event) {
    String username = textfield.getText(); // Obtém o username do TextField
    if (users.containsKey(username)) {
        showAlert("Tasks for " + username, users.get(username).getTasksAsString());
    } else {
        showAlert("Error", "Username does not exist.");
    }
}
```

- **Obtenção das Tarefas**: Obtém as tarefas associadas ao usuário atual usando o método `getTasksAsString` da classe `User`.
- **Exibição de Alertas**: Exibe um alerta contendo as tarefas do usuário atual.

Método `addTask`

O método `addTask` é responsável por adicionar uma nova tarefa ao usuário atual e atualizar o arquivo.

```java
public void addTask(ActionEvent event) {
    String username = textfield.getText(); // Obtém o username do TextField
    String task = passwordfield.getText(); // Obtém a tarefa do PasswordField
    if (users.containsKey(username)) {
        users.get(username).addTask(task); // Adiciona a nova tarefa ao usuário atual
        updateUserFile(); // Atualiza o arquivo com os novos dados
        showAlert("Success", "Task added successfully.");
    } else {
        showAlert("Error", "Username does not exist.");
    }
}
```

- **Adição de uma Nova Tarefa**: Obtém a tarefa fornecida e a adiciona ao usuário atual usando o método `addTask` da classe `User`.

- **Atualização do Arquivo**: Chama o método `updateUserFile` para atualizar o arquivo com os novos dados.
- **Exibição de Alertas**: Exibe um alerta informando se a tarefa foi adicionada com sucesso ou se ocorreu um erro.

Esses métodos são responsáveis por gerenciar a leitura, escrita e manipulação das tarefas associadas a cada usuário no sistema. Eles garantem que as informações sejam

6. Arquivo de Dados (.txt)

1. **Atualização da estrutura do arquivo para refletir as mudanças**: O arquivo `.txt` deve ser atualizado para refletir a nova estrutura de dados, que inclui agora não apenas usernames e senhas, mas também as tarefas associadas a cada usuário.

Essas são as principais mudanças que devem ser feitas no código para acomodar a nova estrutura de dados e funcionalidades. Basicamente, estamos estendendo a funcionalidade existente para incluir o gerenciamento de tarefas para cada usuário.

Resumo

A abordagem completa envolve a interação entre a GUI, o sistema de dados (representado pelo arquivo .txt) e a lógica de negócios (representada pelas classes `Controller01` e `User`). Cada classe desempenha um papel específico no sistema, e juntas elas fornecem funcionalidades completas de login e gerenciamento de tarefas para os usuários.

#ID