Documentação de Projeto

Aluno: Matheus Mendes
Data de Criação: 28/10/2023

Qual versão de JDK e qual versão de IDE foi utilizada?
Java "20.0.1" 2023-04-18 – IDE intellij IDEA 2023.2.2

Como rodar o projeto: Preciso configurar alguma coisa?
Desde que esteja com o java e a ide nessas versoes, não precisa instalar nada, apenas abrir o código.

Explicação da arquitetura:

Minha arquitetura é a MVC.

Modelo (Model): Aqui é onde definimos como os dados do nosso aplicativo são organizados e processados. São as regras de negócios e as estruturas de dados que representam o estado da aplicação. Por exemplo, se estamos construindo um aplicativo de lista de tarefas, o Modelo poderia incluir como as tarefas são armazenadas, atualizadas e manipuladas.

Visão (View): Na Visão, decidimos como apresentar os dados aos usuários. Ela cuida da parte visual e interativa da aplicação. Se estivermos falando sobre um aplicativo de lista de tarefas, a Visão determinaria como as tarefas são exibidas na tela, qual é a aparência dos botões e como os usuários interagem com a interface.

Controlador (Controller): O Controlador age como um intermediário entre o Modelo e a Visão. Ele recebe as interações do usuário, como cliques em botões, e atualiza o Modelo conforme necessário. Se um usuário marca uma tarefa como concluída em um aplicativo de lista de tarefas, o Controlador capturaria essa ação, atualizaria o estado do Modelo e garantiria que a Visão refletisse essa mudança.

Liste as bibliotecas utilizadas:
JUnit:

- `org.junit.runner.JUnitCore`
- `org.junit.runner.Result`
- `org.junit.runner.notification.Failure`
- `org.junit.Before`, `org.junit.Test` (anotações usadas para escrever testes)
- `org.junit.jupiter.api.Assertions` (JUnit 5)

Java Standard Library:

- `java.util.Date` (utilizada para manipular datas em um dos códigos)

- `java.util.ArrayList`, `java.util.List` (utilizadas para criar listas e coleções)

Classe: CadastroUsuario

Descrição:
Esta classe representa um cadastro de usuários em um sistema. Ela fornece métodos para adicionar novos usuários ao cadastro e listar os usuários cadastrados.

Atributos:
- usuarios (List<Usuario>): Uma lista que armazena objetos do tipo Usuario, representando os usuários cadastrados.

Métodos:

1. adicionarUsuario(Usuario usuario):
   - Propósito: Adiciona um novo usuário à lista de usuários cadastrados.
   - Parâmetros:
     - usuario (Usuario): O objeto de usuário a ser adicionado ao cadastro.
   - Retorno: Nenhum.

2. listarUsuarios():
   - Propósito: Retorna a lista de todos os usuários cadastrados.
   - Retorno: Uma lista contendo objetos do tipo Usuario, representando os usuários cadastrados.

Exemplo de Uso:

Criar uma instância de CadastroUsuario
CadastroUsuario cadastro = new CadastroUsuario();

Criar um novo usuário
Usuario novoUsuario = new Usuario("João", "joao@example.com", "senha123");

Adicionar o novo usuário ao cadastro

```
cadastro.adicionarUsuario(novoUsuario);

Listar todos os usuários cadastrados
List<Usuario> usuariosCadastrados = cadastro.listarUsuarios();
        for (Usuario u : usuariosCadastrados) {
                System.out.println("Nome: " + u.getNome() + ", Email: " + u.getEmail());
```

Notas:

Esta classe é responsável por gerenciar o cadastro de usuários no sistema, mas não lida com a autenticação ou validação dos dados do usuário. A autenticação e a verificação dos dados do usuário devem ser realizadas em outras partes do sistema.

- Certifique-se de validar os dados do usuário antes de adicioná-los ao cadastro.

Classe: Usuario

Descrição:
Esta classe representa um usuário em nosso sistema. Ela é responsável por armazenar informações pessoais e credenciais do usuário, incluindo nome, email, CPF e senha.

Atributos:
- nome (String): Armazena o nome do usuário.
- email (String): Contém o endereço de email do usuário.
- cpf (String): Armazena o número de CPF do usuário.
- senha (String): Contém a senha do usuário para fins de autenticação.

Métodos:

1. Usuario(String nome, String email, String cpf, String senha):
   - Propósito: Construtor da classe para criar um novo objeto de usuário.
   - Parâmetros:
     - nome (String): O nome do usuário.
     - email (String): O endereço de email do usuário.
     - cpf (String): O número de CPF do usuário.
     - senha (String): A senha do usuário para autenticação.
   - Retorno: Nenhum.

Getters:
1. getNome():
   - Propósito: Obtém o nome do usuário.
   - Retorno: Uma String contendo o nome do usuário.

2. getEmail():
   - Propósito: Obtém o endereço de email do usuário.
   - Retorno: Uma String contendo o email do usuário.

3. getCpf():
   - Propósito: Obtém o número de CPF do usuário.
   - Retorno: Uma String contendo o CPF do usuário.

4. getSenha():
   - Propósito: Obtém a senha do usuário.
   - Retorno: Uma String contendo a senha do usuário.

Exemplo de Uso:
Criar um novo objeto de usuário
```
Usuario novoUsuario = new Usuario("João Silva", "joao@example.com", "123.456.789-00", "minhaSenhaSecreta");
```

Acessar informações do usuário
```
String nomeDoUsuario = novoUsuario.getNome();
String emailDoUsuario = novoUsuario.getEmail();
String cpfDoUsuario = novoUsuario.getCpf();
String senhaDoUsuario = novoUsuario.getSenha();
```

Notas:
Esta classe fornece uma estrutura básica para representar informações de um usuário, mas não lida com autenticação ou validação de dados do usuário. A verificação de dados do usuário deve ser realizada em outras partes do sistema.

Classe: Login

Descrição:
Esta classe fornece funcionalidades de autenticação para o sistema. Ela permite verificar se um usuário pode ser autenticado com base em seu email e senha.

Métodos:

1. boolean autenticarUsuario(String email, String senha, CadastroUsuario cadastro):
   - Propósito: Autentica um usuário com base em suas credenciais.
   - Parâmetros:
     - email (String): O endereço de email do usuário para autenticação.
     - senha (String): A senha do usuário para autenticação.
     - cadastro (CadastroUsuario): O objeto que contém a lista de usuários cadastrados.
   - Retorno: Um valor booleano que indica se a autenticação foi bem-sucedida (true) ou falhou (false).

Notas:
- Esta classe é responsável por percorrer a lista de usuários registrados no sistema, comparar as credenciais fornecidas (email e senha) com as credenciais de cada usuário e determinar se a autenticação é bem-sucedida ou não.

Exemplo de Uso:

```
Login autenticacao = new Login();
CadastroUsuario cadastro = new CadastroUsuario();

// Autenticar um usuário com email e senha
boolean autenticado = autenticacao.autenticarUsuario("usuario@example.com",
"minhaSenhaSecreta", cadastro);

if (autenticado) {
    System.out.println("Usuário autenticado com sucesso!");
} else {
    System.out.println("Falha na autenticação. Verifique suas credenciais.");
}
```

Classe: LoginTeste

Descrição:
Esta classe contém os testes unitários para a classe `Login`. Ela verifica o comportamento da autenticação de usuários em diferentes cenários, como autenticação bem-sucedida, falha de autenticação e credenciais incorretas.

Métodos de Teste:

1. testAutenticarUsuarioSucesso():
   - Propósito: Testa a autenticação de um usuário com credenciais válidas.
   - Verifica se o método `autenticarUsuario` retorna `true` quando as credenciais são válidas.

2. testAutenticarUsuarioFalha():
   - Propósito: Testa a falha na autenticação de um usuário com credenciais inválidas.
   - Verifica se o método `autenticarUsuario` retorna `false` quando as credenciais são inválidas.

3. testAutenticarUsuarioGmail():
   - Propósito: Testa a autenticação de um usuário com um email do Gmail e senha válidos.
   - Verifica se o método `autenticarUsuario` retorna `true` quando as credenciais são válidas.

4. testAutenticarUsuarioHotmail():
   - Propósito: Testa a autenticação de um usuário com um email do Hotmail e senha válidos.
   - Verifica se o método `autenticarUsuario` retorna `true` quando as credenciais são válidas.

5. testAutenticarUsuarioOutlook():
   - Propósito: Testa a autenticação de um usuário com um email do Outlook e senha válidos.
   - Verifica se o método `autenticarUsuario` retorna `true` quando as credenciais são válidas.

6. testAutenticarUsuarioInexistente():
   - Propósito: Testa a falha na autenticação de um usuário inexistente.
   - Verifica se o método `autenticarUsuario` retorna `false` quando o email não está cadastrado.

7. testAutenticarUsuarioSenhaIncorreta():
   - Propósito: Testa a falha na autenticação de um usuário com senha incorreta.
   - Verifica se o método `autenticarUsuario` retorna `false` quando a senha está incorreta.

8. testAutenticarUsuarioGmailSenhaIncorreta():
   - Propósito: Testa a falha na autenticação de um usuário com email do Gmail e senha incorreta.
   - Verifica se o método `autenticarUsuario` retorna `false` quando a senha está incorreta.

9. testAutenticarUsuarioOutlookSenhaIncorreta():

- Propósito: Testa a falha na autenticação de um usuário com email do Outlook e senha incorreta.
- Verifica se o método `autenticarUsuario` retorna `false` quando a senha está incorreta.

10. testAutenticarUsuarioHotmailSenhaIncorreta():
- Propósito: Testa a falha na autenticação de um usuário com email do Hotmail e senha incorreta.
- Verifica se o método `autenticarUsuario` retorna `false` quando a senha está incorreta.

Exemplo de Uso:
Este conjunto de testes é usado para garantir que a classe `Login` funcione corretamente ao autenticar usuários em várias situações. Os testes validam se a autenticação é consistente e lida adequadamente com diferentes cenários.

Notas:
- Certifique-se de que os dados de usuário fornecidos durante os testes correspondam aos dados esperados no ambiente de teste.
- Para executar esses testes, uma estrutura de teste adequada, como o framework JUnit, deve ser configurada no ambiente de desenvolvimento.

Classe: Conteudo

Descrição:
Esta classe representa um conteúdo audiovisual em nosso sistema. Ela armazena informações sobre o conteúdo, como sinopse, duração, título, gênero, diretor, status de conteúdo impróprio para menores e favoritismo do usuário.

Atributos:

- sinopse (String): Armazena a sinopse ou descrição do conteúdo.
- duracao (int): Contém a duração do conteúdo em minutos.
- titulo (String): Armazena o título do conteúdo.
- genero (String): Indica o gênero do conteúdo.
- diretor (String): Armazena o nome do diretor do conteúdo.
- impróprioParaMenores (boolean): Indica se o conteúdo é impróprio para menores.
- favorito (boolean): Indica se o conteúdo foi marcado como favorito pelo usuário.
- emReproducao (boolean): Indica se o conteúdo está em reprodução.

Métodos:

1. Conteudo(String sinopse, int duracao, String titulo, String genero, String diretor, boolean impróprioParaMenores):
   - Propósito: Construtor da classe para criar um novo objeto de conteúdo.
   - Parâmetros:
     - sinopse (String): A sinopse ou descrição do conteúdo.
     - duracao (int): A duração do conteúdo em minutos.
     - titulo (String): O título do conteúdo.
     - genero (String): O gênero do conteúdo.
     - diretor (String): O nome do diretor do conteúdo.
     - impróprioParaMenores (boolean): Indica se o conteúdo é impróprio para menores.
   - Retorno: Nenhum.

2. reproduzir():
   - Propósito: Inicia a reprodução do conteúdo, definindo o status de reprodução como verdadeiro.
   - Retorno: Nenhum.

3. pausar():
   - Propósito: Pausa a reprodução do conteúdo, definindo o status de reprodução como falso.
   - Retorno: Nenhum.

4. parar():
   - Propósito: Interrompe a reprodução do conteúdo, definindo o status de reprodução como falso.
   - Retorno: Nenhum.

5. favoritar():
   - Propósito: Marca o conteúdo como favorito, definindo o status de favorito como verdadeiro.
   - Retorno: Nenhum.

6. liberarConteudo():
   - Propósito: Libera o conteúdo para todos os públicos, se for marcado como impróprio para menores.
   - Retorno: Nenhum.

Getters:

1. getSinopse():
   - Propósito: Obtém a sinopse do conteúdo.
   - Retorno: Uma String contendo a sinopse.

2. getDuracao():
   - Propósito: Obtém a duração do conteúdo em minutos.
   - Retorno: Um inteiro representando a duração.

3. getTitulo():
   - Propósito: Obtém o título do conteúdo.
   - Retorno: Uma String contendo o título.

4. getGenero():
   - Propósito: Obtém o gênero do conteúdo.
   - Retorno: Uma String contendo o gênero.

5. getDiretor():
   - Propósito: Obtém o nome do diretor do conteúdo.
   - Retorno: Uma String contendo o nome do diretor.

6. isImpróprioParaMenores():
   - Propósito: Verifica se o conteúdo é impróprio para menores.
   - Retorno: Um valor booleano (true se for impróprio para menores, false caso contrário).

7. isFavorito():
   - Propósito: Verifica se o conteúdo foi marcado como favorito.
   - Retorno: Um valor booleano (true se for favorito, false caso contrário).

8. isEmReproducao():
   - Propósito: Verifica se o conteúdo está em reprodução.
   - Retorno: Um valor booleano (true se estiver em reprodução, false caso contrário).

Exemplo de Uso:
Esta classe é usada para representar conteúdo audiovisual em nosso sistema. Os métodos permitem iniciar, pausar e parar a reprodução do conteúdo, marcar como favorito e verificar o status do conteúdo. Os atributos armazenam informações detalhadas sobre o conteúdo.

Notas:
- A verificação de conteúdo impróprio para menores e a marcação de favorito são recursos opcionais e podem ser habilitados conforme necessário.

Classe: ConteudoTeste

Descrição:
Esta classe contém testes unitários para a classe Conteudo. Ela verifica o comportamento dos métodos de Conteudo, como reprodução, pausa, parada, favoritismo e conteúdo impróprio para menores.

Métodos de Teste:

1. testReproduzirConteudo():
   - Propósito: Verifica se o método de reprodução inicia a reprodução de conteúdo corretamente.
   - Resultado Esperado: Espera-se que o conteúdo esteja em reprodução após a chamada ao método reproduzir().

2. testPausarConteudo():
   - Propósito: Verifica se o método de pausa interrompe a reprodução do conteúdo corretamente.
   - Resultado Esperado: Espera-se que o conteúdo não esteja em reprodução após a chamada ao método pausar().

3. testPararConteudo():
   - Propósito: Verifica se o método de parada interrompe a reprodução do conteúdo corretamente.
   - Resultado Esperado: Espera-se que o conteúdo não esteja em reprodução após a chamada ao método parar().

4. testFavoritarConteudo():
   - Propósito: Verifica se o método de favoritar marca o conteúdo como favorito corretamente.
   - Resultado Esperado: Espera-se que o conteúdo seja marcado como favorito após a chamada ao método favoritar().

5. testLiberarConteudoImproprio():
   - Propósito: Verifica se o método de liberar conteúdo impróprio para menores funciona corretamente.
   - Resultado Esperado: Espera-se que o conteúdo não seja mais considerado impróprio para menores após a chamada ao método liberarConteudo().

6. testConteudoNaoFavoritoPorPadrao():
   - Propósito: Verifica se o conteúdo não é favorito por padrão.
   - Resultado Esperado: Espera-se que o conteúdo não seja marcado como favorito por padrão.

7. testConteudoNaoEmReproducaoPorPadrao():
   - Propósito: Verifica se o conteúdo não está em reprodução por padrão.
   - Resultado Esperado: Espera-se que o conteúdo não esteja em reprodução por padrão.

8. testConteudoNaoImpróprioParaMenoresPorPadrão():
   - Propósito: Verifica se o conteúdo não é considerado impróprio para menores por padrão.

- Resultado Esperado: Espera-se que o conteúdo não seja considerado impróprio para menores por padrão.

9. testLiberarConteudoImproprioAposFavoritar():
- Propósito: Verifica se o método de liberar conteúdo impróprio para menores funciona corretamente após marcar o conteúdo como favorito.
- Resultado Esperado: Espera-se que o conteúdo não seja mais considerado impróprio para menores após marcar como favorito e chamar o método liberarConteudo().

10. testReproduzirAposPausar():
- Propósito: Verifica se o conteúdo pode ser reproduzido novamente após a pausa.
- Resultado Esperado: Espera-se que o conteúdo esteja em reprodução após a chamada ao método reproduzir() após uma pausa.

Exemplo de Uso:

```
Criar uma instância de Conteudo para teste
Conteudo filmeHomemDeFerro = new Conteudo("Homem de Ferro é um filme de super-herói baseado na Marvel Comics", 126, "Homem de Ferro", "Ação", "Jon Favreau", false);

Verificar a reprodução de conteúdo
filmeHomemDeFerro.reproduzir();
assertTrue(filmeHomemDeFerro.isEmReproducao());

Pausar o conteúdo
filmeHomemDeFerro.pausar();
assertFalse(filmeHomemDeFerro.isEmReproducao());

Marcar o conteúdo como favorito
filmeHomemDeFerro.favoritar();
assertTrue(filmeHomemDeFerro.isFavorito());

Liberar conteúdo impróprio para menores
filmeHomemDeFerro.liberarConteúdoImpróprio();
assertFalse(filmeHomemDeFerro.isImpróprioParaMenores());
```

Notas:
- Esta classe de teste verifica o comportamento dos métodos da classe Conteudo. Os resultados esperados foram descritos para cada teste.

Classe: Episodio

Descrição:
A classe Episodio representa um episódio de uma série. Ela armazena informações como título, descrição, temporada, número do episódio, duração, status de travamento e status de reprodução.

Métodos:

1. Episodio(String titulo, String descricao, int temporada, int episodio, int duracao):
   - Construtor da classe. Inicializa as informações do episódio.

2. isTravado():
   - Verifica se o episódio está travado.

3. travar():
   - Trava o episódio.

4. desbloquear():
   - Desbloqueia o episódio.

5. isEmAndamento():
   - Verifica se o episódio está em reprodução.

6. play():
   - Inicia a reprodução do episódio, desde que não esteja travado.

7. pause():
   - Pausa a reprodução do episódio.

8. getDuracao():
   - Obtém a duração do episódio.

9. toString():
   - Retorna uma representação em formato de string do episódio, incluindo título, descrição, temporada, episódio e duração.

Exemplo de Uso:

```
Episodio episodio1 = new Episodio("Título do Episódio", "Descrição do episódio", 1, 1, 30);
episodio1.play();
System.out.println(episodio1.toString());
```

Classe: Serie

Descrição:
A classe Serie representa uma série que contém episódios. Ela fornece métodos para gerenciar os episódios, como adicionar, listar, calcular a duração total, travar, desbloquear, reproduzir e pausar episódios.

Métodos:

1. Serie():
   - Construtor da classe. Inicializa a lista de episódios.

2. adicionarEpisodio(Episodio episodio):
   - Adiciona um episódio à série.

3. listarEpisodios():
   - Lista os episódios da série, exibindo suas informações.

4. getDuracaoTotal():
   - Calcula a duração total da série, excluindo episódios travados ou em reprodução.

5. episodioExiste(Episodio episodio):
   - Verifica se um episódio existe na série.

6. travarEpisodio(Episodio episodio):
   - Trava um episódio se ele existir e não estiver travado.

7. desbloquearEpisodio(Episodio episodio):
   - Desbloqueia um episódio se ele existir e estiver travado.

8. playEpisodio(Episodio episodio):
   - Inicia a reprodução de um episódio se ele existir, não estiver em reprodução e não estiver travado.

9. pauseEpisodio(Episodio episodio):
   - Pausa a reprodução de um episódio se ele existir e estiver em reprodução.

10. getEpisodios():
    - Obtém a lista de episódios da série.

Exemplo de Uso:

```
Serie minhaSerie = new Serie();
Episodio episodio1 = new Episodio("Episódio 1", "Descrição do Episódio 1", 1, 1, 45);
minhaSerie.adicionarEpisodio(episodio1);
minhaSerie.listarEpisodios();
int duracaoTotal = minhaSerie.getDuracaoTotal();
System.out.println("Duração Total: " + duracaoTotal + " minutos");
```

Notas:
- Esta documentação descreve as classes `Episodio` e `Serie`, seus métodos e fornece exemplos de uso.

- Certifique-se de preencher as informações do autor, data de criação, data da última revisão e nome do revisor com os detalhes apropriados.

Classe: SerieTeste

Descrição:
Esta classe contém testes unitários para a classe Serie. Ela verifica o comportamento dos métodos da classe Serie, como adicionar episódios, calcular duração total, travar, desbloquear, reproduzir e pausar episódios.

Métodos de Teste:

1. testAdicionarEpisodio():
   - Propósito: Verifica se é possível adicionar um novo episódio à série.
   - Resultado Esperado: Espera-se que o novo episódio seja adicionado com sucesso à série.

2. testDuracaoTotal():
   - Propósito: Verifica se o cálculo da duração total da série está correto.
   - Resultado Esperado: Espera-se que a duração total seja igual a 166 minutos.

3. testEpisodioInexistente():
   - Propósito: Verifica se a série reconhece corretamente que um episódio inexistente não faz parte dela.
   - Resultado Esperado: Espera-se que o episódio inexistente não exista na série.

4. testTravarEpisodio():
   - Propósito: Verifica se é possível travar um episódio da série.
   - Resultado Esperado: Espera-se que o episódio fique travado após a chamada ao método travarEpisodio().

5. testDesbloquearEpisodio():
   - Propósito: Verifica se é possível desbloquear um episódio travado na série.
   - Resultado Esperado: Espera-se que o episódio seja desbloqueado após a chamada ao método desbloquearEpisodio().

6. testPlayEpisodio():
   - Propósito: Verifica se é possível iniciar a reprodução de um episódio da série.
   - Resultado Esperado: Espera-se que o episódio esteja em reprodução após a chamada ao método playEpisodio().

7. testPauseEpisodio():
   - Propósito: Verifica se é possível pausar a reprodução de um episódio da série.
   - Resultado Esperado: Espera-se que o episódio não esteja mais em reprodução após a chamada ao método pauseEpisodio().

8. testTentarPlayEpisodioTravado():
   - Propósito: Verifica se não é possível iniciar a reprodução de um episódio travado.
   - Resultado Esperado: Espera-se que o método playEpisodio() retorne false quando tentar reproduzir um episódio travado.

9. testTentarPauseEpisodioNaoEmAndamento():
   - Propósito: Verifica se não é possível pausar a reprodução de um episódio que não está em andamento.

- Resultado Esperado: Espera-se que o método pauseEpisodio() retorne false quando tentar pausar um episódio não em andamento.

Exemplo de Uso:

```
Serie the100 = new Serie();
Episodio episodio1 = new Episodio("Piloto", "97 anos depois de um apocalipse...", 1, 1, 41);
the100.adicionarEpisodio(episodio1);
int duracaoTotal = the100.getDuracaoTotal();
System.out.println("Duração Total: " + duracaoTotal + " minutos");
```

Notas:
- Esta classe de teste verifica o comportamento dos métodos da classe Serie. Certifique-se de configurar adequadamente os episódios da série no método setUp() antes de executar os testes.

Classe: Suporte

Descrição:
A classe Suporte representa um tíquete de suporte ou solicitação de assistência. Ela armazena informações sobre o suporte, como o identificador, o assunto, a mensagem, o status, o responsável e a data de criação.

Atributos:
- idSuporte (int): O identificador único do tíquete de suporte.
- idUsuario (int): O identificador do usuário que criou o tíquete de suporte.
- assunto (String): O assunto da solicitação de suporte.
- mensagem (String): A mensagem detalhada da solicitação de suporte.
- status (String): O status atual do tíquete de suporte (Aberto ou Fechado).
- responsavel (String): O nome do responsável pelo tíquete (pode ser nulo inicialmente).
- dataSuporte (Date): A data de criação do tíquete de suporte.

Construtor:
- Suporte(int idSuporte, int idUsuario, String assunto, String mensagem): Cria uma instância de Suporte com os valores iniciais especificados. O status é definido como "Aberto", e o campo responsável é inicialmente nulo. A data de suporte é definida automaticamente na data de criação.

Métodos:
- fechar(): Define o status do tíquete de suporte como "Fechado".
- reabrir(): Define o status do tíquete de suporte como "Aberto".
- atribuir(String suporte): Define o responsável pelo tíquete de suporte como o nome especificado.

Getters (Métodos de Acesso):
- getIdSuporte(): Retorna o identificador único do tíquete de suporte.
- getIdUsuario(): Retorna o identificador do usuário que criou o tíquete de suporte.
- getAssunto(): Retorna o assunto da solicitação de suporte.
- getMensagem(): Retorna a mensagem detalhada da solicitação de suporte.
- getStatus(): Retorna o status atual do tíquete de suporte (Aberto ou Fechado).
- getResponsavel(): Retorna o nome do responsável pelo tíquete de suporte (pode ser nulo inicialmente).
- getDataSuporte(): Retorna a data de criação do tíquete de suporte.

Exemplo de Uso:

```
Suporte ticket = new Suporte(1, 123, "Problema com login", "Não consigo fazer login no sistema.");
System.out.println("Status do tíquete: " + ticket.getStatus());
ticket.atribuir("SuporteTecnico");
System.out.println("Responsável pelo tíquete: " + ticket.getResponsavel());
ticket.fechar();
System.out.println("Status do tíquete após fechar: " + ticket.getStatus());
```

Notas:
- Esta classe fornece métodos para manipular o tíquete de suporte, como fechar, reabrir e atribuir um responsável.
- A data de suporte é definida automaticamente no momento da criação do tíquete.

Classe: SistemaSuporte

Descrição:
A classe `SistemaSuporte` é responsável por gerenciar tíquetes de suporte em um sistema. Ela permite abrir novos tíquetes, fechar, reabrir, atribuir responsáveis, listar os tíquetes existentes e obter informações sobre um tíquete específico.

Atributos:
- suportes (List<Suporte>): Uma lista de objetos `Suporte` que representam os tíquetes de suporte gerenciados pelo sistema.

Métodos:

1. `abrir(int idUsuario, String assunto, String mensagem, StreamService streamService)`: Permite abrir um novo tíquete de suporte.
   - Parâmetros:
     - `idUsuario` (int): O identificador do usuário que está criando o tíquete.
     - `assunto` (String): O assunto da solicitação de suporte.
     - `mensagem` (String): A mensagem detalhada da solicitação de suporte.
     - `streamService` (StreamService): Um serviço de streaming para uso futuro (não utilizado no código fornecido).

2. `fechar(int idSuporte)`: Fecha um tíquete de suporte existente com base no seu identificador.
   - Parâmetros:
     - `idSuporte` (int): O identificador único do tíquete de suporte a ser fechado.

3. `reabrir(int idSuporte)`: Reabre um tíquete de suporte fechado anteriormente com base no seu identificador.
   - Parâmetros:
     - `idSuporte` (int): O identificador único do tíquete de suporte a ser reaberto.

4. `atribuir(int idSuporte, String suporteAtribuido)`: Atribui um responsável a um tíquete de suporte com base no seu identificador.
   - Parâmetros:
     - `idSuporte` (int): O identificador único do tíquete de suporte ao qual será atribuído um responsável.
     - `suporteAtribuido` (String): O nome do responsável a ser atribuído ao tíquete.

5. `listarSuportes()`: Retorna uma lista de todos os tíquetes de suporte existentes no sistema.

6. `getSuporte(int idSuporte)`: Obtém informações detalhadas sobre um tíquete de suporte com base no seu identificador.
   - Parâmetros:
     - `idSuporte` (int): O identificador único do tíquete de suporte a ser obtido.
   - Retorna:
     - Objeto `Suporte`: As informações detalhadas do tíquete de suporte ou `null` se o tíquete não existir.

Exemplo de Uso:

```
SistemaSuporte sistema = new SistemaSuporte();
sistema.abrir(1, "Problema de Conexão", "Não consigo acessar o sistema.");
sistema.abrir(2, "Problema de Pagamento", "Minha assinatura expirou.");
sistema.atribuir(1, "SuporteTecnico");
sistema.fechar(2);
List<Suporte> suportes = sistema.listarSuportes();
Suporte suporte1 = sistema.getSuporte(1);
```

Classe: StreamService

Descrição:
A classe StreamService representa um serviço de streaming que permite iniciar, parar e verificar o status de um streaming.
Ela mantém o controle do estado do streaming (ativo ou inativo) e fornece métodos para interagir com ele.

Métodos e Funcionalidades:

1. Construtor:
    - `StreamService()`: Inicializa uma instância de StreamService com o streaming desativado.

2. Iniciar Streaming:
    - `iniciarStreaming()`: Tenta iniciar o streaming se ele estiver inativo.
    - Retorno: Retorna true se o streaming foi iniciado com sucesso, ou false se já estiver ativo.

3. Parar Streaming:
    - `pararStreaming()`: Tenta parar o streaming se ele estiver ativo.
    - Retorno: Retorna true se o streaming foi parado com sucesso, ou false se já estiver inativo.

4. Verificar o Status de Streaming:
    - `estaStreaming()`: Verifica se o streaming está ativo ou inativo.
    - Retorno: Retorna true se o streaming está ativo, ou false se estiver inativo.

Uso Exemplo:

```
Criar uma instância de StreamService
StreamService streamService = new StreamService();

Iniciar o streaming
boolean iniciadoComSucesso = streamService.iniciarStreaming();

Verificar o status do streaming
boolean estaAtivo = streamService.estaStreaming();

Parar o streaming
boolean paradoComSucesso = streamService.pararStreaming();
```

Notas:
- A classe StreamService oferece uma maneira simples de controlar o estado do streaming, permitindo iniciar, parar e verificar seu status.
- Certifique-se de criar uma instância da classe antes de usar seus métodos.

Classe de Teste: SuporteTeste

Descrição:
A classe SuporteTeste contém métodos de teste unitários para a classe SistemaSuporte e suas interações com a classe StreamService. Ela verifica o comportamento de abrir, fechar, reabrir e atribuir suportes, além de lidar com situações como suportes inexistentes.

Métodos de Teste:

1. testAbrirSuporteParaProblemaDeConexao():
   - Propósito: Verifica se é possível abrir um novo suporte para um problema de conexão.
   - Resultado Esperado: Espera-se que o suporte seja criado com sucesso.

2. testFecharSuporteParaProblemaDeBug():
   - Propósito: Verifica se é possível fechar um suporte aberto para um problema de bug.
   - Resultado Esperado: Espera-se que o status do suporte seja "Fechado".

3. testReabrirSuporteParaCarregamentoDeFilme():
   - Propósito: Verifica se é possível reabrir um suporte fechado para um problema de carregamento de filme.
   - Resultado Esperado: Espera-se que o status do suporte seja "Aberto".

4. testAtribuirSuporteParaCarregamentoDeSerie():
   - Propósito: Verifica se é possível atribuir um suporte para um funcionário.
   - Resultado Esperado: Espera-se que o responsável pelo suporte seja o funcionário atribuído.

5. testAbrirSuporteParaTravamento():
   - Propósito: Verifica se é possível abrir um novo suporte para um problema de travamento.
   - Resultado Esperado: Espera-se que o suporte seja criado com sucesso.

6. testFecharSuporteInexistente():
   - Propósito: Verifica se a tentativa de fechar um suporte inexistente não afeta outros suportes.
   - Resultado Esperado: Espera-se que os outros suportes permaneçam inalterados.

7. testReabrirSuporteFechado():
   - Propósito: Verifica se é possível reabrir um suporte que já foi fechado.
   - Resultado Esperado: Espera-se que o status do suporte seja "Aberto".

8. testAtribuirSuporteFechado():
   - Propósito: Verifica se é possível atribuir um funcionário a um suporte que já foi fechado.
   - Resultado Esperado: Espera-se que o responsável seja nulo (sem atribuição).

9. testAtribuirSuporteInexistente():
   - Propósito: Verifica se a tentativa de atribuir um suporte inexistente não afeta outros suportes.
   - Resultado Esperado: Espera-se que os outros suportes permaneçam inalterados.

Exemplo de Uso:

A classe de teste verifica o comportamento dos métodos da classe SistemaSuporte, incluindo abrir, fechar, reabrir e atribuir suportes.
Certifique-se de configurar adequadamente os suportes no método setUp() antes de executar os testes.

Classe: Main * * Descrição: * Esta classe é responsável por executar testes unitários de diferentes classes usando o framework JUnit. Ela executa testes nas classes LoginTeste, ConteudoTeste, SerieTeste e SuporteTeste e exibe os resultados dos testes no console.

Métodos:
1. public static void main(String[] args):
- Este é o método principal da classe. Ele inicia a execução dos testes.
Exemplo de Uso:
Para executar os testes, basta executar o método main. Os resultados dos testes serão exibidos no console.
Notas:
- Esta classe funciona como um ponto de entrada para a execução de testes.
Certifique-se de que as classes de teste (LoginTeste, ConteudoTeste, SerieTeste, SuporteTeste) estejam corretamente configuradas e incluídas no classpath.