



LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

ETEC DE HORTOLÂNDIA

CURSO TÉCNICO EM INFORMÁTICA INTEGRADO AO ENSINO MÉDIO

PROF. RALFE DELLA CROCE FILHO

CONTEÚDO

- Tipos de erros
- Tratamento de erros em Java
- Classe Error
- Classe Exception
 - Unchecked Exception
 - Checked Exception
- try-catch-finally
- throws
- throw

TIPOS DE ERROS

- Na execução de um programa podem ocorrer erros que produzem resultados incorretos, comportamentos inesperados ou até a interrupção do programa.
- Esses erros podem ser de lógica ou de execução.

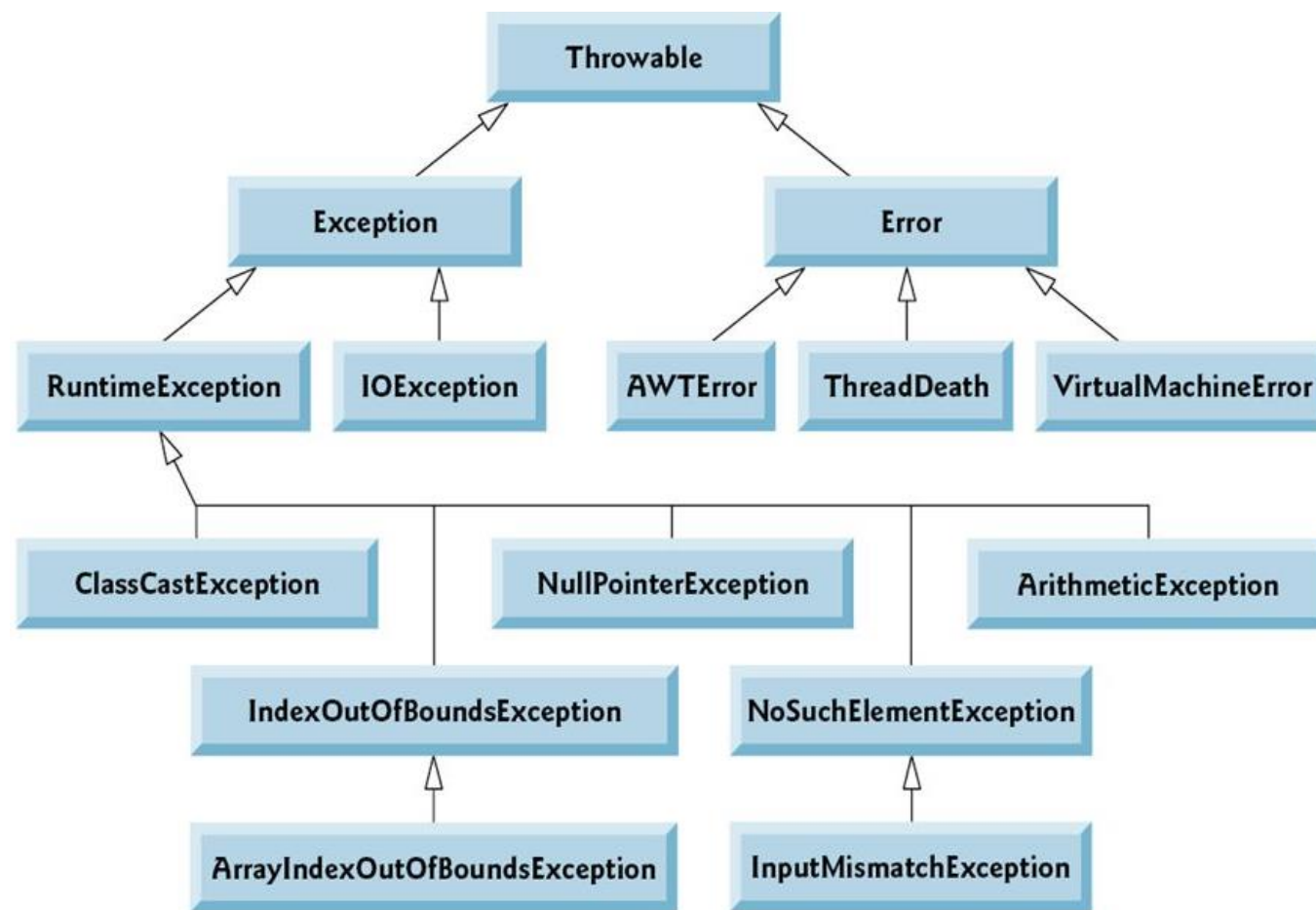
TIPOS DE ERROS

- Erros de lógica decorrem de algoritmos inconsistentes. Estes erros não necessariamente causam interrupção na execução do programa e são detectados em testes durante as etapas de desenvolvimento e as soluções estão na adequação e reorganização das instruções lógicas.
- Erros de execução decorrem de operações inválidas e causam interrupção na execução do programa.

TRATAMENTO DE ERROS EM JAVA

- As linguagens de programação possuem formas diferentes para identificar e tratar os erros de execução, em Java eles são detectados pela JVM e um objeto referente ao erro é criado (a partir de classes específicas).
- A aplicação é notificada sobre o erro e, caso seja possível tratá-lo, a partir do objeto do erro pode-se tomar ações preventivas e/ou corretivas.
- Os erros são caracterizados por objetos de classes específicas que pertencem a hierarquia da classe Throwable.

TRATAMENTO DE ERROS EM JAVA



CLASSE ERROR

- A classe Error (e suas subclasses) reconhece erros graves que não podem ser tratados pela aplicação.
- Errors são menos comuns e não são de responsabilidade da aplicação.
- Exemplos de Errors:
 - Erros internos da JVM.
 - Perda de conexão de rede.
 - Problemas de hardware.

CLASSE EXCEPTION

- Os erros em Java são comumente chamados de exceptions (ou exceções) e essas podem ser tratadas dentro do programa.
- As classes referentes a exceções (Exception e suas subclasses) são divididas entre as que obrigam o programador a inserir um tratamento de erro (chamadas de checked ou verificadas) e as que o tratamento é “opcional” (chamadas de unchecked ou não verificadas).

CHECKED EXCEPTION

- A obrigatoriedade de tratamento existe porque as causas dessas exceções não estão dentro do escopo da aplicação e, portanto, são impossíveis de serem evitadas apenas por boas práticas de programação.
- Todos os tipos de exceção que herdam da classe `Exception`, mas não da `RuntimeException`, são exceções verificadas.
- Exemplo: abrir um arquivo para leitura/alteração (onde o arquivo pode não ser encontrado no caminho indicado ou estar corrompido, por exemplo).

UNCHECKED EXCEPTION

- A não obrigatoriedade de tratamento justifica-se porque o programador pode impedir a ocorrência dessas exceções por meio da codificação adequada (verificações prévias, formatações de entrada de dados, etc.).
- Essas exceções que são geradas a partir de subclasses da classe `RuntimeException`.
- Exemplos: entrada de tipos incompatíveis (leitura de uma `String` em um atributo `double`, por exemplo), acesso a índice inexistente em um array, chamada a um método de um objeto nulo, etc.

TRATAMENTO DAS EXCEÇÕES COM TRY-CATCH-FINALLY

- Métodos que geram uma checked exception apresentam um erro (mesmo estando sintaticamente corretos). Com trechos de código que podem gerar uma unchecked exception (como o próprio nome indica) isso não ocorre, porém, em ambos os casos a estrutura de try-catch-finally pode ser utilizada bastando delimitar esses trechos de código no bloco de try, dessa forma, caso ocorra uma exceção o objeto do erro gerado será capturado.
- Um catch identifica (por meio de um parâmetro) a classe que gerou o objeto de exceção e possibilita o tratamento. Podem ser definidos quantos catches forem necessários para cada try.

TRATAMENTO DAS EXCEÇÕES COM TRY-CATCH-FINALLY

```
try {  
    // Instruções que geram ou podem gerar exceções  
}  
catch (NullPointerException e) {  
    // Tratamento da exceção NullPointerException  
}  
catch (NumberFormatException e) {  
    // Tratamento da exceção NumberFormatException  
}  
catch (Exception e) {  
    // Exceção não prevista  
}
```

- O finally é opcional e se for usado é colocado depois do último catch. Essa última camada é executada ocorrendo ou não uma exceção.
- Obs.: Implementaremos essa camada nas aulas de conexão a banco de dados.

TRATAMENTO DAS EXCEÇÕES COM TRY-CATCH-FINALLY

- Sempre implemente os tratamentos específicos (para que ações direcionadas sejam executadas) prevendo as exceções de acordo com as classes e métodos utilizados.
- Mesmo assim, é interessante deixar o último catch com a classe Exception (superclasse de todas as exceções) para captura de exceções imprevistas. PORÉM, deve-se recuperar as informações dessa exceção imprevista por meio do parâmetro definido para posterior tratamento específico.

TRATAMENTO DAS EXCEÇÕES COM TRY-CATCH-FINALLY

```
try {  
    // Instruções que geram ou podem gerar exceções  
  
} catch (NullPointerException e) {  
    // Tratamento da exceção NullPointerException  
  
} catch (NumberFormatException e) {  
    // Tratamento da exceção NumberFormatException  
  
} catch (Exception e) { // Exceção não prevista  
    // Recupera a mensagem de erro completa.  
    e.printStackTrace();  
}
```

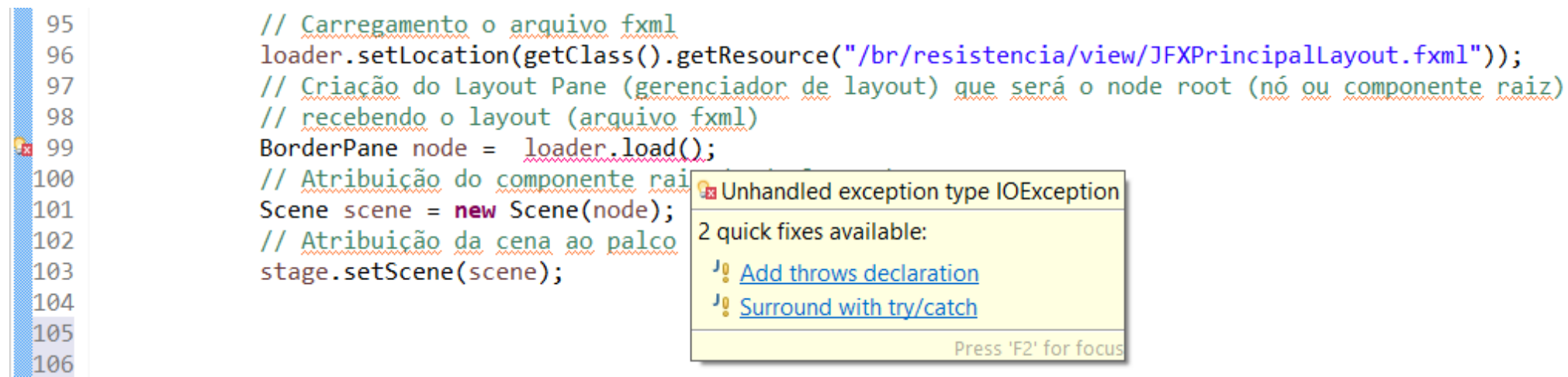
EXEMPLO TRY-CATCH-FINALLY

- No projeto Zion na classe JFXLoginControle.java, quando validado o login e senha, o formulário JFXPrincipalControle é carregado. Como ocorre em todas as chamadas de layout em JavaFX, é executada a chamada do método `loader.load()` que, por sua vez, recebe o caminho do arquivo de layout `.fxml`.

```
95 // Carregamento o arquivo fxml
96 loader.setLocation(getClass().getResource("/br/resistencia/view/JFXPrincipalLayout.fxml"));
97 // Criação do Layout Pane (gerenciador de layout) que será o node root (nó ou componente raiz)
98 // recebendo o layout (arquivo fxml)
99 BorderPane node = loader.load();
```

EXEMPLO TRY-CATCH-FINALLY

- Esse método gera uma checked exception e, portanto, seu tratamento é obrigatório.
- Parando com mouse sobre o erro (ou clicando sobre o ícone de erro na margem esquerda) é apresentado um menu com as opções de tratamento.



EXEMPLO TRY-CATCH-FINALLY

- Clicando sobre “Surround with try/catch” a estrutura de try-catch é inserida automaticamente.

```
98  
99  
100  
101  
102  
103  
104  
105  
  
// recebendo o layout (arquivo fxml)  
BorderPane node = null;  
try {  
    node = loader.load();  
} catch (IOException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```

```

JFXLoginControle.java
87 // Valida o contato e senha
88 if(contatoInformado.equals(contatoUsuario) && senhaInformada.equals(senhaUsuario)) { // Se for válido, invoca o formulário principal
89
90     try {
91
92         Stage stage = new Stage();
93         FXMLLoader loader = new FXMLLoader();
94         loader.setLocation(getClass().getResource("/br/resistencia/view/JFXPrincipalLayout.fxml"));
95         BorderPane node = loader.load();
96         Scene scene = new Scene(node);
97         stage.setScene(scene);
98         JFXPrincipalControle principalControle = loader.getController();
99         principalControle.setPalcoPrincipal(stage);
100         principalControle.setUsuarioLogado(usuario);
101         principalControle.setUsuarioLogado("Usuário: " + usuario.getNome());
102         stage.initStyle(StageStyle.UNDECORATED);
103         stage.setResizable(false);
104         stage.centerOnScreen();
105         stage.show();
106         this.getPalcoLogin().close();
107
108     } catch (IOException e) {
109         Alert alert = new Alert(AlertType.ERROR);
110         alert.setTitle("Falha ao iniciar");
111         alert.setHeaderText("Não foi possível abrir o formulário Principal.");
112         alert.setContentText("Entre em contato com o desenvolvedor.");
113         // Apresenta as informações da exceção no Console
114         e.printStackTrace();
115     }
116
117 } else {
118     // Se o contato e/ou senha estiverem incorretos
119     Alert alert = new Alert(AlertType.ERROR);
120     alert.setTitle("Login");
121     alert.setHeaderText("Validação de usuário.");
122     alert.setContentText("Contato e/ou senha não encontrados!");
123     alert.showAndWait();
124 }

```

- Daquela forma a exceção já havia sido tratada, mas, é possível melhorar a disposição dos códigos colocando toda a sequencia de instruções em comum no bloco try e acrescentar uma mensagem com orientações em caso de exceção.

EXEMPLO TRY-CATCH-FINALLY

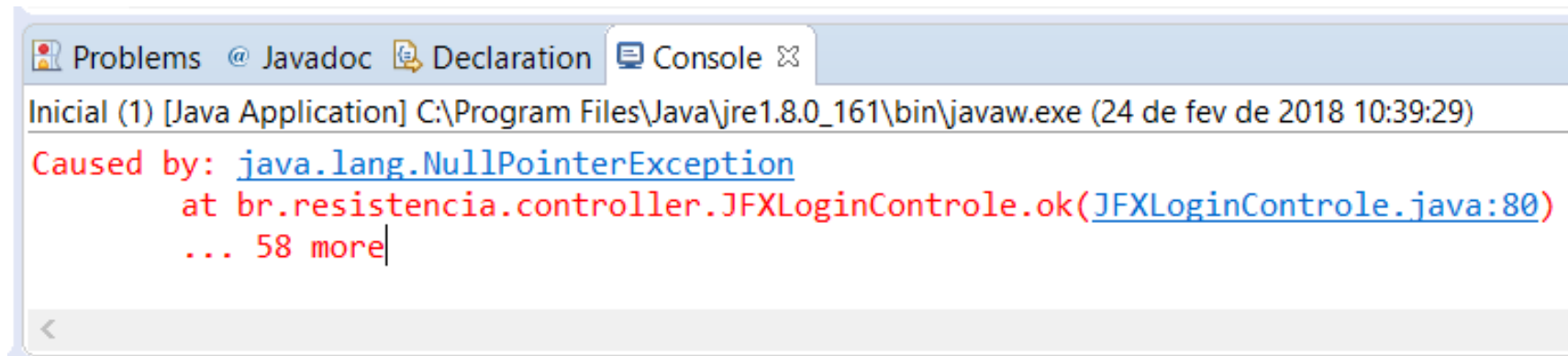
- Ainda na classe JFXLoginControle se o botão OK for clicado sem um tipo de usuário selecionado uma exceção será gerada.



```
Problems @ Javadoc Declaration Console
Inicial (1) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (24 de fev de 2018 10:39:29)
Exception in thread "JavaFX Application Thread" java.lang.RuntimeException: java.lang.reflect.InvocationTargetException
    at javafx.fxml.FXMLLoader$MethodHandler.invoke(FXMLLoader.java:1774)
    at javafx.fxml.FXMLLoader$ControllerMethodEventHandler.handle(FXMLLoader.java:1657)
    at com.sun.javafx.event.CompositeEventHandler.dispatchBubblingEvent(CompositeEventHandler.java:86)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:238)
    at com.sun.javafx.event.EventHandlerManager.dispatchBubblingEvent(EventHandlerManager.java:191)
    at com.sun.javafx.event.CompositeEventDispatcher.dispatchBubblingEvent(CompositeEventDispatcher.java:59)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:58)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
    at com.sun.javafx.event.BasicEventDispatcher.dispatchEvent(BasicEventDispatcher.java:56)
    at com.sun.javafx.event.EventDispatchChainImpl.dispatchEvent(EventDispatchChainImpl.java:114)
```

EXEMPLO TRY-CATCH-FINALLY

- No painel Console será apresentada uma mensagem com informações da exceção, dentre elas, a classe da exceção gerada (NullPointerException), a classe e a linha onde a exceção foi gerada (JFXLoginControle.java:80).

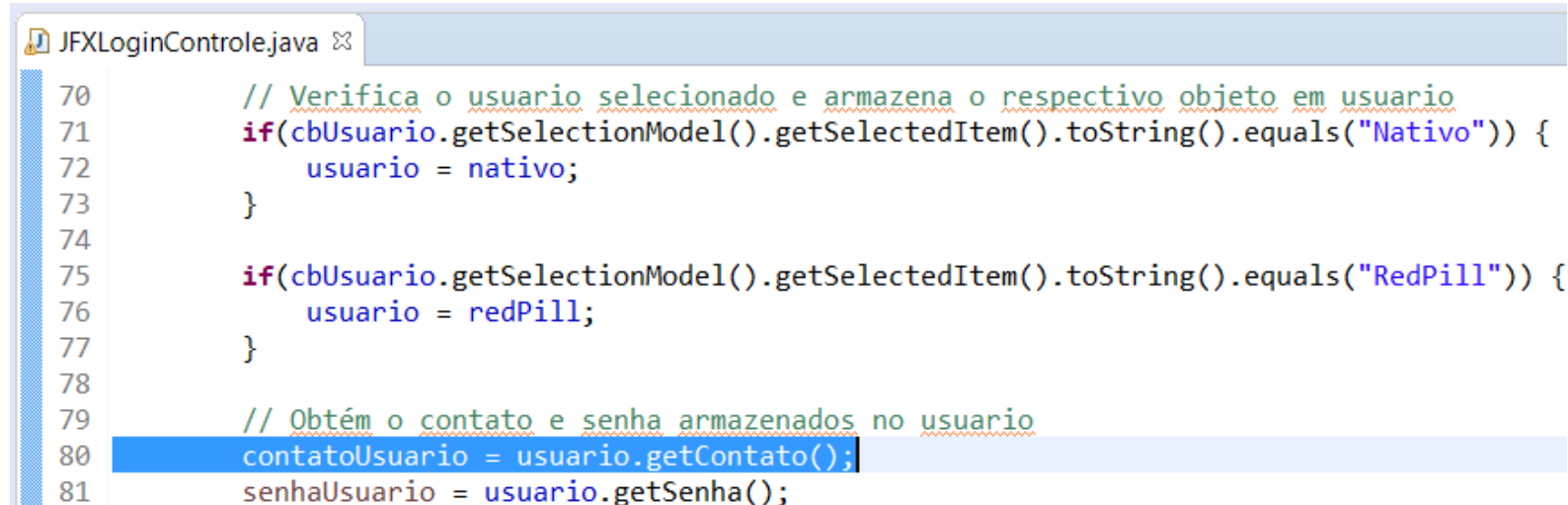


The screenshot shows the 'Console' tab in a Java IDE. The text in the console is as follows:

```
Problems  @ Javadoc  Declaration  Console  X
Inicial (1) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (24 de fev de 2018 10:39:29)
Caused by: java.lang.NullPointerException
           at br.resistencia.controller.JFXLoginControle.ok(JFXLoginControle.java:80)
           ... 58 more
```

EXEMPLO TRY-CATCH-FINALLY

- Clicando nas informações do erro (sobre o link da identificação da classe e linha) a classe será aberta para edição e a linha onde a exceção foi gerada ficará destacada.



```
JFXLoginControle.java
70 // Verifica o usuario selecionado e armazena o respectivo objeto em usuario
71 if(cbUsuario.getSelectionModel().getSelectedItem().toString().equals("Nativo")) {
72     usuario = nativo;
73 }
74
75 if(cbUsuario.getSelectionModel().getSelectedItem().toString().equals("RedPill")) {
76     usuario = redPill;
77 }
78
79 // Obtém o contato e senha armazenados no usuario
80 contatoUsuario = usuario.getContato();
81 senhaUsuario = usuario.getSenha();
```

EXEMPLO TRY-CATCH-FINALLY

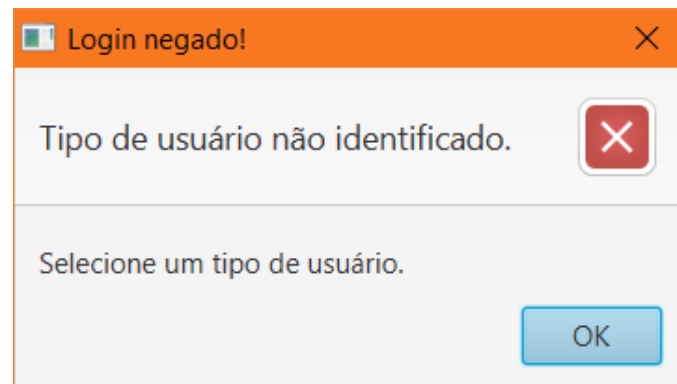
- Essa exceção ocorreu porque ao não se selecionar um tipo de usuário quando a linha 80 é executada não existe uma referência de objeto instanciado em usuário. Essa ação gerada uma unchecked exception que também pode ser tratada com try-catch.

```
JFXLoginControle.java
70 // Verifica o usuario selecionado e armazena o respectivo objeto em usuario
71 if(cbUsuario.getSelectionModel().getSelectedItem().toString().equals("Nativo")) {
72     usuario = nativo;
73 }
74
75 if(cbUsuario.getSelectionModel().getSelectedItem().toString().equals("RedPill")) {
76     usuario = redPill;
77 }
78
79 // Obtém o contato e senha armazenados no usuario
80 contatoUsuario = usuario.getContato();
81 senhaUsuario = usuario.getSenha();
```

EXEMPLO TRY-CATCH-FINALLY



```
JFXLoginControle.java
79
80
81 // Obtém o contato e senha armazenados no usuario
82 contatoUsuario = usuario.getContato();
83 senhaUsuario = usuario.getSenha();
84
85 } catch (NullPointerException e) {
86
87     Alert alert = new Alert(AlertType.ERROR);
88     alert.setTitle("Login negado!");
89     alert.setHeaderText("Tipo de usuário não identificado.");
90     alert.setContentText("Selecione um tipo de usuário.");
91     alert.showAndWait();
92
93 } catch (Exception e) {
94     // TODO: handle exception
95 }
```




```

JFXLoginControle.java
64 // Métodos para eventos FXML
65 @FXML public void ok() {
66
67     try {
68         String contatoUsuario = "", senhaUsuario = "";
69         String contatoInformado = "", senhaInformada = "";
70         if(cbUsuario.getSelectionModel().getSelectedItem().toString().equals("Nativo")) {
71             usuario = nativo;
72         }
73         if(cbUsuario.getSelectionModel().getSelectedItem().toString().equals("RedPill")) {
74             usuario = redPill;
75         }
76         contatoUsuario = usuario.getContato();
77         senhaUsuario = usuario.getSenha();
78         contatoInformado = tfLogin.getText();
79         senhaInformada = pfSenha.getText();
80         if(contatoInformado.equals(contatoUsuario) && senhaInformada.equals(senhaUsuario)) { // Se for
81             Stage stage = new Stage();
82             FXMLLoader loader = new FXMLLoader();
83             loader.setLocation(getClass().getResource("/br/resistencia/view/JFXPrincipallayout.fxml"));
84             BorderPane node = loader.load();
85             Scene scene = new Scene(node);
86             stage.setScene(scene);
87             JFXPrincipalControle principalControle = loader.getController();
88             principalControle.setPalcoPrincipal(stage);
89             principalControle.setUsuarioLogado(usuario);
90             principalControle.setlUsuarioLogado("Usuário: " + usuario.getNome());
91             stage.initStyle(StageStyle.UNDECORATED);
92             stage.setResizable(false);
93             stage.centerOnScreen();
94             stage.show();
95             this.getPalcoLogin().close();
96         } else {
97             // Se o contato e/ou senha estiverem incorretos
98             Alert alert = new Alert(AlertType.ERROR);
99             alert.setTitle("Login");
100             alert.setHeaderText("Validação de usuário.");
101             alert.setContentText("Contato e/ou senha não encontrados!");
102             alert.showAndWait();
103         }
104     }
105 }

```

PROF. RALFE DELLA CROCE FILHO

- Como os dois try-catch ficariam no mesmo método torna-se mais organizado e legível unifica-los.

```

104     } catch (IOException e) {
105         Alert alert = new Alert(AlertType.ERROR);
106         alert.setTitle("Falha ao iniciar");
107         alert.setHeaderText("Não foi possível abrir o formulário Principal.");
108         alert.setContentText("Entre em contato com o desenvolvedor.");
109         e.printStackTrace();
110     }
111     } catch (NullPointerException e) {
112         Alert alert = new Alert(AlertType.ERROR);
113         alert.setTitle("Login negado!");
114         alert.setHeaderText("Tipo de usuário não identificado.");
115         alert.setContentText("Selecione um tipo de usuário.");
116         alert.showAndWait();
117     }
118     } catch (Exception e) {
119         // Apresenta as informações da exceção no Console
120         e.printStackTrace();
121     }
122 }

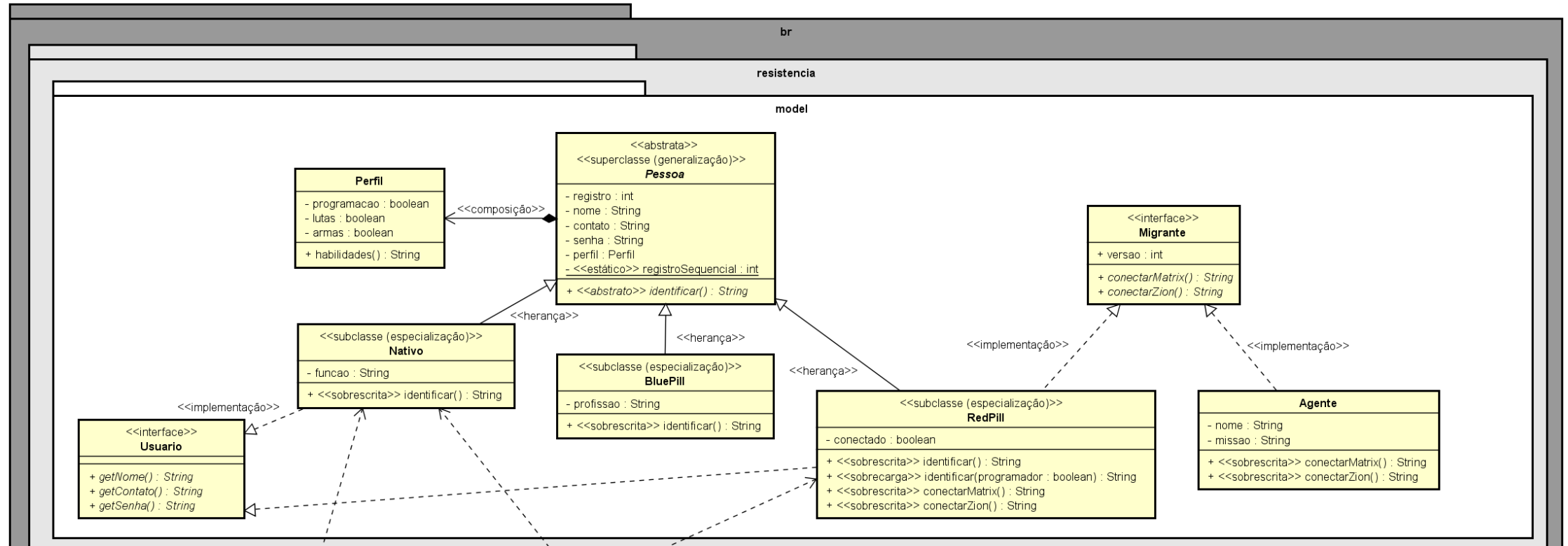
```


THROW

- Permite a criação de um objeto de exceção a partir de qualquer classe da hierarquia de Exception.
- Com ele é possível utilizar o mecanismo de exceções do Java para tratar regras da aplicação (que não gerariam exceções).

EXEMPLO THROW

- Considere uma regra para inserção de senha nos objetos da hierarquia Pessoa.



EXEMPLO THROW

```
Pessoa.java ✕
68 // Regra para inserção de senha:
69 // No mínimo 2 e no máximo 6 caracteres
70 public void setSenha(String senha) {
71
72     if (senha.length() >= 2 && senha.length() <= 6) {
73         this.senha = senha;
74     }else {
75         Alert alert = new Alert(AlertType.ERROR);
76         alert.setTitle("Erro de inserção!");
77         alert.setHeaderText("Tamanho da senha incorreto.");
78         alert.setContentText("A senha deve conter no mínimo 2 e no máximo 6 caracteres.");
79         alert.showAndWait();
80     }
81 }
```

Nativos

Registro: 3

Nome: Morpheus

Contato: morpheus@nabucodonosor.com

Senha:

.....

Função:

Programador

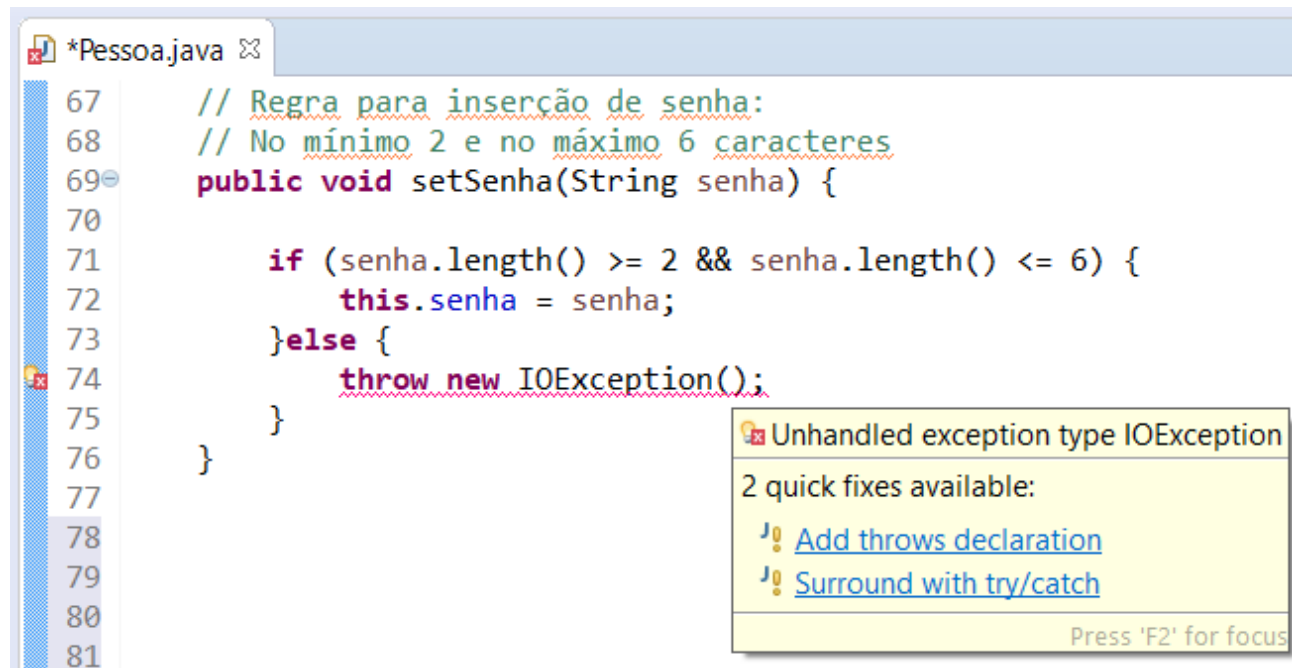
Erro de inserção!

Tamanho da senha incorreto.

A senha deve conter no mínimo 2 e no máximo 6 caracteres.

OK

EXEMPLO THROW



```
*Pessoa.java
67 // Regra para inserção de senha:
68 // No mínimo 2 e no máximo 6 caracteres
69 public void setSenha(String senha) {
70
71     if (senha.length() >= 2 && senha.length() <= 6) {
72         this.senha = senha;
73     } else {
74         throw new IOException();
75     }
76 }
77
78
79
80
81
```

Unhandled exception type IOException

2 quick fixes available:

- [Add throws declaration](#)
- [Surround with try/catch](#)

Press 'F2' for focus

EXEMPLO THROW

Pessoa.java

```
70 // Regra para inserção de senha:
71 // No mínimo 2 e no máximo 6 caracteres
72 public void setSenha(String senha) {
73
74     if (senha.length() >= 2 && senha.length() <= 6) {
75         this.senha = senha;
76     } else {
77
78         try {
79             throw new IOException();
80
81         } catch (IOException e) {
82
83             Alert alert = new Alert(AlertType.ERROR);
84             alert.setTitle("Erro de inserção!");
85             alert.setHeaderText("Tamanho da senha incorreto.");
86             alert.setContentText("A senha deve conter no mínimo 2 e no máximo 6 caracteres.");
87             alert.showAndWait();
88         }
89     }
}
```

Nativos

Registro: 3

Nome: Morpheus

Contato: morpheus@nabucodonosor.com

Senha:

.....

Função:

Programador

Erro de inserção!

Tamanho da senha incorreto.

A senha deve conter no mínimo 2 e no máximo 6 caracteres.

OK

THROWS

- O throws delega para o momento da chamada do método a responsabilidade de tratar a exceção, ou seja, a obrigatoriedade de tratamento é passada para a classe que fará a chamada ao método.

EXEMPLO THROWS

- No padrão MVC não é recomendado tratar exceções na camada de model. Tratamentos de exceções devem ser realizadas em controller.
- No exemplo anterior a criação da exceção por meio do throw ocorre na classe Pessoa (do pacote model), porém, com a utilização do throws é possível transferir/delegar o tratamento para a chamada do método setSenha que ocorre na classe JFXNativoControle (do pacote controller) resolvendo a quebra de recomendação do MVC.

EXEMPLO THROWS

```
Pessoa.java  JFXNativoControle.java
70 // Regra para inserção de senha:
71 // No mínimo 2 e no máximo 6 caracteres
72 public void setSenha(String senha) throws IOException{
73
74     if (senha.length() >= 2 && senha.length() <= 6) {
75         this.senha = senha;
76     }else {
77         throw new IOException();
78     }
79 }
```

```
Pessoa.java  *JFXNativoControle.java
59 @FXML public void inserir() {
60
61     nativo.setNome(tfNome.getText());
62     nativo.setContato(tfContato.getText());
63     nativo.setSenha(pfSenha.getText().toString());
64     nativo.setFuncao(cbFuncao.getSelectionModel().getSelectedItem());
65
66
67
68
69
70 }
```

Unhandled exception type IOException

2 quick fixes available:

- ⚡ Add throws declaration
- ⚡ Surround with try/catch

Press 'F2' for focus

EXEMPLO THROWS

```
Pessoa.java  JFXNativoControle.java ✕
59 // Métodos para eventos FXML
60 @FXML public void inserir() {
61
62     try {
63         nativo.setNome(tfNome.getText());
64         nativo.setContato(tfContato.getText());
65         nativo.setSenha(pfSenha.getText().toString());
66         nativo.setFuncao(cbFuncao.getSelectionModel().getSelectedItem().toString());
67     } catch (IOException e) {
68
69         Alert alert = new Alert(AlertType.ERROR);
70         alert.setTitle("Erro de inserção!");
71         alert.setHeaderText("Tamanho da senha incorreto.");
72         alert.setContentText("A senha deve conter no mínimo 2 e no máximo 6 caracteres.");
73         alert.showAndWait();
74     }
75 }
```

Nativos

Registro:

Nome:

Contato:

Senha: Função:

Erro de inserção!

Tamanho da senha incorreto.

A senha deve conter no mínimo 2 e no máximo 6 caracteres.

OK

EXCEPTIONS CUSTOMIZADAS

- Para criar uma classe de exceção basta extender (criar como subclasse) de Exception e todos o mecanismo de tratamento de exceções do Java passa a ficar disponível.
- Lembrando que toda subclasse de Exception que não for subclasse de RuntimeException é uma checked exception.



ADENDO

CONTEXTUALIZAÇÃO

CONTEXTUALIZAÇÃO

- A Orientação a Objetos possui uma cadeia de conceitos que estruturam, organizam e padronizam softwares. Todos esses conceitos são implementáveis, ou seja, eles foram (e são) criados para aplicação prática no desenvolvimento de softwares de acordo com esse paradigma.
- Contextualizar a aplicação prática desses conceitos é, didaticamente, o melhor caminho para a análise e entendimento de seus efetivos objetivos e importância. Com essa intenção, utilizarei dois caminhos...



- Na apresentação dos conceitos utilizarei exemplos baseados no universo da trilogia Matrix (1999, 2003 e 2003), filme dos irmãos Larry e Andy Wachowski (agora irmãs Lana e Lilly Wachowski). O motivo? Primeiro porque é um dos meus filmes favoritos, segundo porque, de forma extremamente visionária e competente, o filme aborda tanto a tecnologia (especificamente a área de softwares) quanto questões filosóficas fundamentais (pois é, se você achava que era “apenas” um filme de ação e ficção (ganhador de 4 óscares) acho que você não entendeu o filme).
- Obviamente, para o entendimento do conteúdo não é necessário assistir/conhecer o filme, mas, se você não assistiu, creio que quem está perdendo é você...

CONTEXTUALIZAÇÃO

- A outra forma de contextualização será por meio de vários exercícios utilizando simulações de softwares que gerenciam, por exemplo, uma livraria, um controle bancário, uma agência de turismo, um controle escolar e uma imobiliária.
- **Observação importante:** realizar as atividades práticas é fundamental e indispensável. Será MUITO mais difícil (senão impossível) a assimilação dos conteúdos sem a prática.

ESTEREÓTIPOS NO MODELO

- Um *stereotype* (anotação entre <<>>) é um elemento que identifica a finalidade de outros elementos do modelo. Existe um conjunto padrão de estereótipos que podem ser aplicados e são utilizados para refinar o significado de um elemento do modelo.
- Utilizarei os estereótipos fora dos padrões previstos na UML para identificar onde os conceitos de Orientação a Objetos estão sendo aplicados no modelo com intenções puramente didáticas.